RUCA: RUntime Configurable Approximate Circuits with Self-Correcting Capability

Jingxiao Ma
School of Engineering
Brown University
Providence, Rhode Island, USA

Sherief Reda
School of Engineering
Brown University
Providence, Rhode Island, USA

ABSTRACT

Approximate computing is an emerging computing paradigm that offers improved power consumption by relaxing the requirement for full accuracy. Since the requirements for accuracy may vary according to specific real-world applications, one trend of approximate computing is to design quality-configurable circuits, which are able to switch at runtime among different accuracy modes with different power and delay. In this paper, we present a novel framework RUCA which aims to synthesize runtime configurable approximate circuits based on arbitrary input circuits. By decomposing the truth table, our approach aims to approximate and separate the input circuit into multiple configuration blocks which support different accuracy levels, including a corrector circuit to restore full accuracy. Power gating is used to activate different blocks, such that the approximate circuit is able to operate at different accuracy-power configurations. To improve the scalability of our algorithm, we also provide a design space exploration scheme with circuit partitioning. We evaluate our methodology on a comprehensive set of benchmarks. For 3-level designs, RUCA saves power consumption by 43.71% within 2% error and by 30.15% within 1% error on average.

CCS CONCEPTS

• **Hardware → Circuit optimization**; *Circuits power issues*; *Reconfigurable logic and FPGAs.*

KEYWORDS

Approximate computing, Approximate design automation, Low Power, Dynamically configurable accuracy

ACM Reference Format:
Jingxiao Ma and Sherief Reda. 2023. RUCA: RUntime Configurable Approximate Circuits with Self-Correcting Capability. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567888

1 INTRODUCTION

As circuit customization is developed to meet the requirements of various applications, power consumption becomes a main factor limiting the scale of computational capacity. Approximate computing is one of the emerging low-power techniques, which aims to improve power consumption as well as circuit delay by relaxing the requirement for 100% accuracy. Approximate computing can be widely used in many application domains, such as machine learning, computer vision and signal processing, which have inherent resilience to small inaccuracies in the outputs [12]. Such resilience can originate from various sources including, noise in input data, inherent approximate calculations, or human tolerance to variations in the outputs, while different applications may have different resilience. Thus, one trend is to design approximate circuits which are able to dynamically switch among various accuracy levels (including full accuracy) at runtime, each of which is associated with different power consumption. By properly configuring accuracy levels at runtime, power consumption could be substantially saved.

The last few years have seen various techniques for approximate logic synthesis [3, 4, 7, 10]. Most of them focus on approximating circuits with "fixed" accuracy, while some other works start to explore the flexibility of runtime configuration [1, 2, 5, 6, 9, 14]. In this paper, we propose a novel *RUntime Configurable Approximate* (RUCA) methodology based on *truth tables factorization*, which generates approximate circuits with multiple accuracy levels, including full accuracy. The contributions of this paper are as follow.

- Using Boolean Matrix Factorization algorithm, RUCA approximates an *arbitrary* input circuit and separates it into multiple *configuration blocks* using truth tables decomposition. By activating different blocks at runtime using power gating, we can dynamically choose the expected accuracy-power configuration to optimize power and delay. A corrector circuit is introduced to restore 100% accuracy.
- To improve the scalability of our approach, a large input circuit is first *partitioned* into subcircuits, and a *design space exploration* scheme is used to choose the proper subcircuits to approximate in the runtime configurable manner.
- We evaluate RUCA framework on a number of commonly used arithmetic circuits from Benchmarks for Approximate Circuit Synthesis (BACS) [12]. We also compare our methodology against other accuracy-configurable frameworks, Approximate through Logic Isolation [6] and 8-bit QCM [9], showcasing that RUCA efficiently improves power and delay with the flexibility of operating under multiple modes.

The organization of this paper is as follow. In Section 2, we overview relevant previous work on Approximate Logic Synthesis (ALS). In Section 3, we discuss the problem of Boolean Matrix Factorization and its application in ALS. In Section 4, we introduce our novel RUCA methodology. We provide our experimental results in Section 5. Finally, we summarize our conclusion in Section 6.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567888

2 PREVIOUS WORK

Various approaches have been proposed for Approximate Logic Synthesis (ALS) [3, 4, 7, 10]. Compared to ALS that generates "fixed" approximate circuits, quality-configurable approximate design is less explored. One category of runtime configuration is Voltage Over-Scaling (VOS), where the power and accuracy of operation can be dynamically adjusted by tuning the voltage. However, the application of VOS is limited since it may cause uncontrollable errors that affect the most significant bits. Also, VOS increases delays on all timing paths, which may affect the performance of the whole system and even lead to the failure of operation [11].

To design stable and predictable circuits, methodologies based on logic synthesis have been proposed. SASIMI [14] proposed the first methodology to generate quality-configurable design from an arbitrary input circuit by identifying similar signals and substituting one for the other to simplify the logic. However, when full accuracy is required, the approximate circuit may need an additional clock cycle to detect errors and re-compute the substituted signals. Thus, SASIMI turns a combinational circuit into a variable latency circuit, which may not be applicable to large systems.

To mitigate the possibly doubled delay, an approximation approach through logic isolation is proposed [6], which aims to isolate parts of circuit that significantly contribute to power consumption while having less effect on overall accuracy, where power gating is then used to control the activation of these parts.

Another method based on clock gating has been proposed [1], where multiple approximate designs of input circuit are first instantiated. Then area-saving gating mechanisms are used to exploit synthesis relaxation, which leads to total energy saving. This approach was further extended using cross-layer approach [2]. While such methodologies reduce dynamic power consumption significantly, a large amount of area overhead is introduced. Also, clock gating does not reduce leakage power, which is significant in today's technology node. Besides methodologies that take arbitrary circuits as input, some runtime configurable designs with arithmetic circuits have been proposed [5, 9]. For example, QCM [9] designs configurable multipliers using genetic algorithms.

3 PRELIMINARIES

In this section, we describe the problem of Boolean Matrix Factorization (BMF) and its application in approximate logic synthesis.

3.1 Boolean Matrix Factorization

A Boolean matrix is a special matrix where all elements are limited to Boolean values, *i.e.*, '0's or '1's. Boolean Matrix Factorization aims to factorize an input Boolean matrix $\mathbf{M}$ of size $p \times q$ into two Boolean matrices: matrix $\mathbf{A}$ of size $p \times f$, and matrix $\mathbf{B}$ of size $f \times q$, such that $\mathbf{M} \approx \mathbf{AB}$, where f is called *factorization degree*. In many applications, factorization degree f is required to be smaller than q. The multiplications are carried out using the logical AND operation, while the additions can be performed by logical OR operation. Note that one can interpret the columns of $\mathbf{A}$ as *basis* vectors, which are linearly combined using $\mathbf{B}$. BMF has been proved to be NP-hard [8], which can also be formulated as an optimization problem to minimize errors resulted from factorization,

$$\text{argmin}_{\mathbf{A},\mathbf{B}} ||\mathbf{M} - \mathbf{AB}||_0 \quad (1)$$

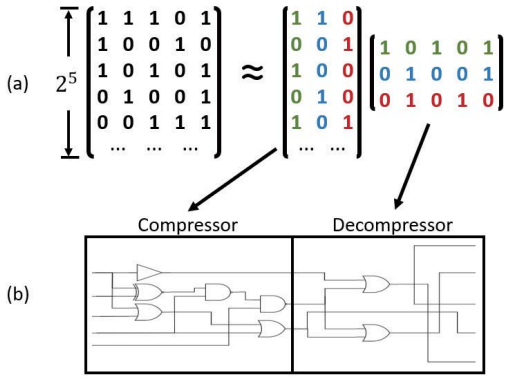

Figure 1: (a) An example of Boolean matrix factorization, where green pair of column and row is derived at first iteration, then blue pair, finally red pair. (b) An example of approximate logic synthesis using BMF.

where $\mathbf{M}$, $\mathbf{A}$ and $\mathbf{B}$ are Boolean matrices, and $||\mathbf{M}-\mathbf{AB}||_0$ is hamming distance between matrices $\mathbf{M}$ and $\mathbf{AB}$. Due to its NP-hardness, many algorithms solve BMF using heuristic approaches. In our work, we considered a heuristic algorithm based on association rule mining (ASSO) [8], which *greedily* solves column in $\mathbf{A}$ and row in $\mathbf{B}$ pair-by-pair in order to cover as many '1's as possible in input matrix $\mathbf{M}$. For an input matrix of size $p \times q$, the time complexity of ASSO algorithm is $O(pq^2)$. Figure 1a demonstrates an example of factorizing a $2^5 \times 5$ input matrix with factorization degree 3, where hamming distance $||\mathbf{M} - \mathbf{AB}||_0$ keeps reducing after each column-row pair.

3.2 BMF and Approximate Logic Synthesis

Truth tables of combinational logic are effectively Boolean matrices. Thus, as proposed in BLASYS [7], BMF can be used to approximate an arbitrary circuit with n inputs and m outputs. The exact input circuit is simulated to obtain the truth table $\mathbf{M}$ of size $2^n \times m$, which is then given as input to a BMF algorithm together with a factorization degree $1 \leq f < m$. $\mathbf{M}$ is then factorized into a $2^n \times f$ matrix $\mathbf{A}$, and a $f \times m$ matrix $\mathbf{B}$. Figure 1b illustrates an example of approximate logic synthesis using BMF, where $n = 5$, $m = 5$ and $f = 3$. Using existing logic synthesis techniques, first Boolean matrix $\mathbf{A}$ is used to synthesize the first part of approximate circuit with n inputs and f outputs, which is referred to as *compressor* circuit. The second part receives $f < m$ inputs from *compressor* circuit and maps them back to m outputs. This subcircuit is referred to as *decompressor* circuit, which can be generated using a network of OR gates according to each column in $\mathbf{B}$. By changing the factorization degree f, we are able to obtain numerous approximate designs with different trade-off between accuracy, design area and power consumption.

4 PROPOSED METHODOLOGY

In this section, we describe our proposed methodology of designing RUntime Configurable Approximate (RUCA) circuit by factorizing and separating truth table, together with the method of self-correcting by corrector circuit. Due to the complexity of BMF algorithm, we partition a large input circuit into subcircuits and use design space exploration to improve the scalability of our methodology. We also discuss the approaches of reducing design overhead.

4.1 RUCA with Corrector Circuit

According to the rule of matrix multiplication, after factorizing a matrix **M** into **A** and **B**, we may separate them into individual columns and rows, as Equation 2,

$$\mathbf{M} \approx \mathbf{AB} = (\mathbf{a_1} \cdots \mathbf{a_f}) \begin{pmatrix} \mathbf{b_1} \\ \vdots \\ \mathbf{b_f} \end{pmatrix} = \mathbf{a_1 b_1} + \mathbf{a_2 b_2} + \cdots + \mathbf{a_f b_f} \quad (2)$$

where $\mathbf{a_i}$ is the i^{th} column in matrix **A** and $\mathbf{b_j}$ is j^{th} row of matrix **B**. As discussed in Section 3, due to the *heuristic* property of BMF algorithm, as we add terms from $\mathbf{a_1 b_1}$ to $\mathbf{a_f b_f}$, the hamming distance $||\mathbf{M} - \mathbf{AB}||_0$ keeps decreasing in a *greedy* manner. To enable runtime configuration, our goal is to factorize the input matrix **M** and separate into multiple terms which may satisfy different error thresholds. For example, suppose that we want to factorize **M** such that there exists two configurable accuracy (e.g., error 2% and 1%). Starting from factorization degree $f = 1$ with only the first term $\mathbf{a_1 b_1}$, we gradually increment f and sum $\mathbf{a_i b_i}$ terms, until the approximation error becomes no larger than 2%. Assume the current factorization degree is $f = k_1$. In this case, we can stack vectors from $\mathbf{a_1}$ to $\mathbf{a_{k_1}}$ as $\mathbf{A_1}$, and stack vectors from $\mathbf{b_1}$ to $\mathbf{b_{k_1}}$ as $\mathbf{B_1}$, such that the error between **M** and $\mathbf{A_1 B_1}$ is no larger than 2%. We then keep incrementing f until 1% error threshold is met. Assuming now $f = k_1 + k_2$, vectors from $\mathbf{a_{k_1+1}}$ to $\mathbf{a_{k_2}}$ are stacked as $\mathbf{A_2}$, and vectors from $\mathbf{b_{k_1+1}}$ to $\mathbf{b_{k_2}}$ are stacked as $\mathbf{B_2}$, such that the error between **M** and $\mathbf{A_1 B_1} + \mathbf{A_2 B_2}$ is no greater than 1%. In other words, factorized matrices **A** and **B** are separated as

$$\mathbf{M} \approx \mathbf{AB} = \mathbf{A_1 B_1} + \mathbf{A_2 B_2} = (\mathbf{a_1} \cdots \mathbf{a_{k_1}}) \begin{pmatrix} \mathbf{b_1} \\ \vdots \\ \mathbf{b_{k_1}} \end{pmatrix} + (\mathbf{a_{k_1+1}} \cdots \mathbf{a_f}) \begin{pmatrix} \mathbf{b_{k_1+1}} \\ \vdots \\ \mathbf{b_f} \end{pmatrix} \quad (3)$$

If more accuracy levels are needed, this procedure is repeated until we obtain matrices $\mathbf{A_i}$ and $\mathbf{B_i}$ for each accuracy level. We propose to synthesize each $\mathbf{A_i B_i}$ term into its own circuit blocks. To implement binary addition, bitwise OR gates (gate *a* in Figure 2b) are used to connect each block of $\mathbf{A_i B_i}$ term. Therefore, starting from block of $\mathbf{A_1 B_1}$, as we activate more $\mathbf{A_i B_i}$ blocks, the error between original truth table **M** and summation of $\mathbf{A_i B_i}$ terms keeps decreasing, where different error thresholds can be achieved.

In order to support critical applications which require full accuracy, we propose to use a corrector circuit to restore the original functionality when needed. Here, field modulo-2 algebra (logic XOR) is used to correct flipped bits, where '1's can be used to flip bits such that $1 \oplus 1 = 0$ and $1 \oplus 0 = 1$. After input truth table **M** is factorized and separated into summation of terms $\mathbf{A_i B_i}$, bitwise XOR is computed between **M** and $\Sigma_i \mathbf{A_i B_i}$ to obtain the corrector matrix **C**. This matrix can be used to restore input truth table **M** by $\mathbf{M} = (\Sigma_i \mathbf{A_i B_i}) \oplus \mathbf{C}$. Figure 2a demonstrates a factorization algebra with three accuracy levels. Corrector matrix **C** is computed to restore the input matrix by XOR operation. Figure 2b demonstrates structure of a 3-level runtime configurable circuit. Firstly, an input circuit with n inputs and m outputs is simulated to obtain the $2^n \times m$ truth table **M**. Then, **M** is factorized into two matrices **A** and **B**, which are then separated into $\mathbf{A_1 B_1}$ and $\mathbf{A_2 B_2}$. All matrices are synthesized into corresponding parts of the circuit. Corrector matrix **C**, which is used to synthesize the corrector circuit, is computed for restoring input truth table **M**. As Boolean algebra indicates, $\mathbf{A_1 B_1}$

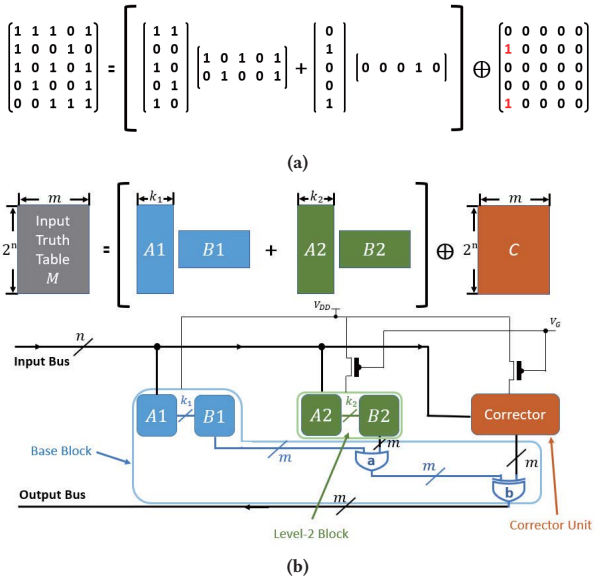

Figure 2: Example of a 3-level approximate circuits using RUCA. (a) BMF with multiple accuracy levels. (b) Runtime configurable circuit design, where power gating is used to activate different blocks.

and $\mathbf{A_2 B_2}$ are connected using bitwise OR (gates *a*), which is then connected to the corrector circuit using bitwise XOR (gates *b*). Thus, if all parts are activated, it will produce equivalent functionality as original circuit, where circuit operates in full-accuracy mode:

$$\mathbf{M} = (\mathbf{A_1 B_1} + \mathbf{A_2 B_2}) \oplus \mathbf{C} \quad (4)$$

In order to enable runtime configuration, we allocate these parts into different *configuration blocks*, and use power gating to control their activation. In this example, $\mathbf{A_1}$, $\mathbf{B_1}$ and all connecting gates compose the *base* block, which is always activated by default. When only activating the *base* block, the circuit operates in approximate mode with lowest accuracy, where the output matrix $\mathbf{M'}$ is

$$\mathbf{M'} = \mathbf{A_1 B_1} \quad (5)$$

$\mathbf{A_2}$ and $\mathbf{B_2}$ compose the *level-2* block, which can be additionally activated for higher-accuracy mode. And the output matrix $\mathbf{M''}$ is

$$\mathbf{M''} = \mathbf{A_1 B_1} + \mathbf{A_2 B_2} \quad (6)$$

Following this framework, we are able to design runtime configurable circuits with arbitrary number of accuracy levels.

4.2 Partitioning and Design Space Exploration

The number of rows in a truth table grows exponentially with the number of primary inputs in the circuit, which makes BMF algorithm computationally expensive for circuits with large number of inputs. To scale our approach, we propose a *divide-and-conquer* method using circuit partitioning and design space exploration technique. As illustrated in Figure 3a, to begin with, a given circuit is partitioned into a number of subcircuits with limited number of inputs and outputs, each of which is approximated using RUCA approach as Figure 2. A design space exploration technique is used to find proper subcircuits and factorization degrees, where the priority is to approximate the subcircuits that consume more power while having less impact on final accuracy. Figure 3b demonstrates a 2-level configurable circuit, where base blocks of the approximate

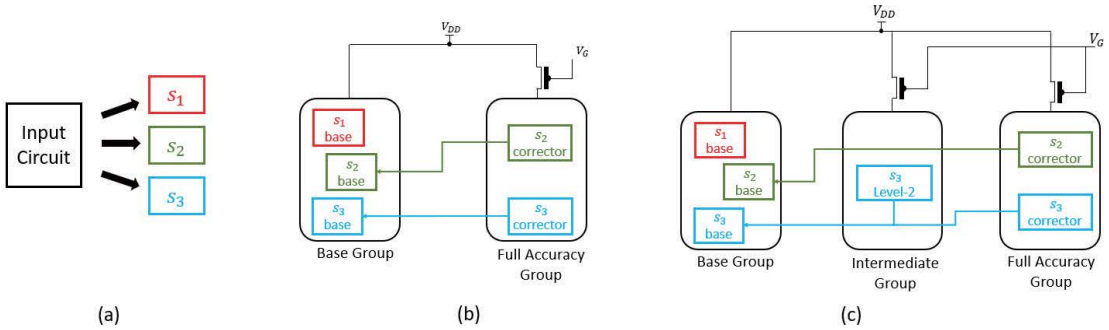

Figure 3: An example of runtime configurable designs for a large input circuit. (a) Input circuit is partitioned into three subcircuits. (b) *Subcircuits* are approximated into 2-level runtime configurable designs. Base blocks of 3 *subcircuits* are synthesized together as the base group of the *top-level circuit*. Corrector circuits are grouped together as the full-accuracy group of the *top-level circuit*. (c) Additional accuracy levels can be introduced by re-arranging RUCA blocks of *subcircuits* into intermediate group(s).

Algorithm 1: Runtime Configurable Approximate Circuit with Design Space Exploration

Input : $ICir$: input circuit
ϵ: list of error thresholds, sorted in ascending order
Output: $RCir$: list of groups in output RUCA circuit

1 S_{Cir} = Partition $ICir$ into list of subcircuits
2 **for** *each subcircuit* s_i *in* S_{Cir} **do**
3 Factorize truth table for s_i
4 Set current factorization degree $f_i = m_i$ where m_i is the number of primary outputs of subcircuit s_i
5 **end**
6 n = 0 // Indexing $RCir$ list
7 **while** ϵ *is not empty* **do**
8 **for** *each subcircuit* s_i *in* S_{Cir} **do**
9 $TCir_i$ = RUCA($s_i, f_i - 1$)
10 $loss_i = Err_i \cdot [P_{acc} + P_{app}]$
11 **end**
12 $k = \arg\min_i \ loss_i$
13 $f_k = f_k$ - 1
14 **if** $QoR \geq \epsilon[0]$ **then**
15 Replace s_i with $TCir_i$ for each subcircuit s_i
16 Add all new blocks (except *base blocks*) into $RCir[n]$
17 n = n + 1
18 ϵ.pop(0)
19 **end**
20 **end**
21 Add all *base blocks* of subcircuits into $RCir[n]$
22 **return** $RCir$

subcircuits are grouped in an individual power domain and synthesized together as the *base group* of the top-level design. Corrector circuits of approximate subcircuits are also grouped together into *full-accuracy group*, which enables the top-level design to restore full accuracy. Moreover, if additional accuracy levels are expected, more *intermediate groups* can be created by re-allocating different blocks of approximate subcircuits, as illustrated in Figure 3c.

Algorithm 1 describes the overall procedure of approximating subcircuits and re-allocating blocks into groups. To begin with, the input circuit is partitioned into subcircuits (line 1) using hypergraph partitioning algorithm [13]. Then the truth table of each subcircuit is factorized as Equation 2 to prepare for RUCA design (line 3).

In design space exploration scheme (line 7-19), we approximate and replace subcircuits with RUCA design, whose configuration blocks are re-allocated into groups of top-level design. In other words, for each subcircuit s_i, we search proper factorization degrees f's to separate factorized truth tables as Equation 3. Factorization degree f_i for each subcircuit is initialized as the number of primary outputs (line 4) and decrement by 1 at each iteration (line 9,13), where RUCA($s_i, f_i - 1$) denotes a runtime configurable design based on subcircuit s_i with *factorization degree* $f_i - 1$. For each candidate design, a loss is computed based on accuracy Err_i, power in full-accuracy mode P_{acc} and in approximate mode P_{app} (line 10,12). Whenever an error threshold is reached, all new added blocks are placed into corresponding group (line 14-19).

4.3 Reducing Design Overhead

Design overhead is considered as an important criterion in accuracy-configurable designs, which is defined as additional chip area, power and delay running in full-accuracy mode compared to original input circuit. In RUCA, the design overhead mainly comes from two sources: (1) As Figure 2b shows, additional bitwise OR and XOR gates are used to connect blocks, which is inevitable in RUCA design. To mitigate such overhead in design space exploration, we should limit number of levels for each approximate subcircuit. In our practice, each subcircuit is approximated with no more than *three levels*, such that each of these contains at most *one* bitwise OR gate and *one* bitwise XOR gate.

(2) Since each block is synthesized and optimized individually, we may lose the opportunity of logic optimization across different blocks, which leads to logic redundancy. Such logic redundancy becomes severe with a dense corrector matrix. As discussed in subsection 4.1, the corrector circuit is synthesized from corrector matrix to flip wrong bits in approximate truth table. Normally, the difference between approximate and original truth table is not too large, and the corrector matrix is sparse as shown in Figure 2a. In this case, the overhead caused by corrector circuit is small. However, if input circuit is partitioned and design space exploration is performed, for some subcircuits, the difference between approximate and original truth table may be large. In this situation, the corrector matrix is dense, which will be synthesized into large corrector circuit, sometimes even larger than the original subcircuit. In this case, rather than using a corrector circuit to achieve full accuracy, we

use the original subcircuit instead. In our design space exploration algorithm, once the corrector circuit is synthesized, we compare the power consumption between corrector circuit and original one. If a corrector circuit consumes less power, we follow the algorithm described in Section 4.2. However, if the corrector circuit consumes more power than the original subcircuit, we directly include original subcircuit for full-accuracy mode. In this case, instead of XOR gates, a multiplexer is used to choose between original subcircuit and the approximate versions.

5 EXPERIMENTAL RESULTS

In this section, we evaluate our proposed methodology on a number of arithmetic circuits which are commonly deployed in approximate computing from Benchmarks for Approximate Circuit Synthesis (BACS) [12]. Table 1 summarizes the characteristics of evaluated benchmarks. To begin with, we directly generate runtime configurable designs of 8-bit adder, where the trade-off between design overhead and choices of error thresholds is discussed. The remaining benchmarks are first partitioned into subcircuits, and then design space exploration is performed as Algorithm 1.

For hardware metrics, all designs are implemented in Verilog and synthesized with a 7*nm* predictive process design kit. Cadence Genus is used to synthesize each design and estimate chip area, circuit delay and power consumption under the maximum clock frequency of original circuit. For QoR metric, we report normalized mean absolute error (MAE) defined as

$$\text{MAE} = \frac{1}{N}\Sigma_{i=1}^{N}\frac{|R_i - R_i'|}{2^m}, \tag{7}$$

where N denotes the size of the test vectors while R_i and R_i' denote the accurate and approximate numerical results.

In the first set of experiment, we analyze the trade-off between design accuracy, power consumption and design overhead. RUCAs are generated for 8-bit adder with different error thresholds. Besides full accuracy, only *one* approximate level is considered for each design in this experiment. Since the original circuit has 9 primary outputs, after factorizing its truth table, first f pairs of columns and rows are synthesized into *base* block as approximate mode, where f ranges from 1 to 8. For each RUCA design, an associated corrector circuit is created to restore errors in full-accuracy mode. In Figure 4, we report the power consumption of the corrector circuit, and the RUCA design in both approximate mode and full-accuracy mode. In approximate mode, power consumption keeps reducing while error increases. However, in full-accuracy mode, the corrector circuit becomes more substantial and power-consuming, especially when factorization degree f is small. As MAE exceeds 5%, where factorization degree $f < 4$, power consumption of full-accuracy mode increases substantially due to the corrector circuit. Therefore, to limit the overhead in full-accuracy mode, error thresholds in approximate mode need to be limited, *e.g.*, below 5% MAE.

In the second set of experiments, for the remaining six benchmarks in BACS in Table 1, we generate three RUCA designs with 2 levels, 3 levels and 4 levels respectively. We use 0.1%, 1% and 2% as error thresholds. In order to highlight the benefits of our methodology, we report *relative power* as the ratio between power of RUCA design (under certain accuracy level) and power of the original circuit. Figure 5 illustrates relative power of RUCAs for

Table 1: Characteristics of evaluated benchmarks.

Bench-mark	Name	Function	I/O	Area (um^2)	Power (uW)	Delay (ns)
BACS	adder8	8-bit adder	16/9	47.58	24.70	0.81
	abs_diff	absolute difference	16/9	67.41	22.68	0.90
	adder32	32-bit adder	64/33	167.03	32.20	2.83
	buttfly	butterfly structure	32/34	174.26	42.30	3.05
	mac	multiply-add	12/8	94.48	33.76	1.14
	mult8	8-bit multiplier	16/16	364.61	82.21	1.97
	mult16	16-bit multiplier	32/32	1084.52	245.06	3.74

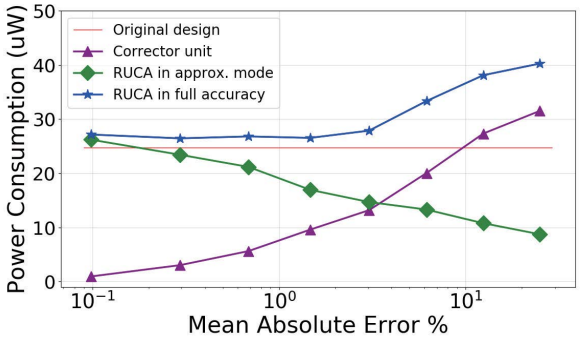

Figure 4: 2-level approximate design of 8-bit adder: Power consumption with different error thresholds.

each benchmark. Compared to original circuit, RUCA substantially saves power in approximate mode, and use slightly extra power to enable corrector circuit for full-accuracy mode. However, as the number of accuracy levels increases, RUCA approximates an input circuit into more configurable blocks, which potentially reduces opportunities to optimize logic synthesis and increases power consumption in full-accuracy mode.

In Table 2, we thoroughly evaluate all benchmarks and compare the performance against another runtime configurable framework named Approximation through Logic Isolation [6]. Under each accuracy level, we report *total area*, *power* and *delay* as ratio against original input circuit. We use 3-level runtime configurable designs and set error thresholds as 1% MAE and 2% MAE. Red numbers represent better performance between two methodologies. On average, we are able to reduce 30.15% power and 22.96% delay with 1% error threshold; and reduce 43.71% power and 52.08% delay with 2% error threshold. To run in full accuracy mode, RUCA consumes 7.81% additional power than the original circuit. However, it is expected that with approximate computing, the circuits will run approximately most of the time, and only in a few occasions, full accuracy will be needed and enabled. Compared to Logic Isolation, our RUCA framework has smaller total area in 4 designs out of 6 benchmarks, which on average saves 2.22% area compared to Logic Isolation. In terms of power consumption, our approach has 3 better results under 2% error level, and 5 better results under 1% error level and full-accuracy level respectively. In general, compared to Logic Isolation, RUCA is able to use smaller chip area and consumes less power to implement the same functionality of runtime configurable design, especially in higher-accuracy mode.

Finally, in Table 3, using 8-bit multiplier, we compare a 2-level QCM [9] against 3-level RUCA, where RUCA has more accuracy levels with even smaller total area, demonstrating that the design overhead of RUCA is relatively lower.

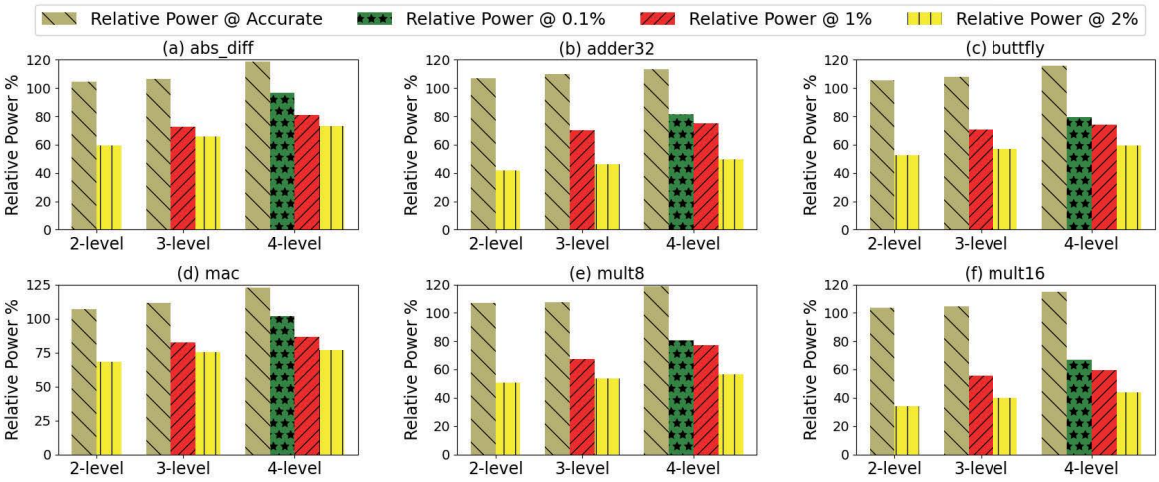

Figure 5: Relative power of RUCAs for each benchmark, with 2-4 levels.

Table 2: Comparison of total area, power and delay between RUCA and Approximation through Logic Isolation (ISO) [6] (using 3-level runtime configurable design)

Name	Total Area		Power						Delay					
	RUCA	ISO	Under 2% MAE		Under 1% MAE		Full Accuracy		Under 2% MAE		Under 1% MAE		Full Accuracy	
			RUCA	ISO	RUCA	ISO	RUCA	ISO	RUCA	ISO	RUCA	ISO	RUCA	ISO
abs_diff	138.38%	152.09%	65.79%	55.82%	72.73%	82.54%	105.68%	109.85%	56.19%	64.90%	85.40%	88.94%	116.34%	124.92%
adder32	142.74%	134.47%	46.17%	51.89%	70.13%	60.93%	109.86%	113.10%	62.37%	68.50%	84.92%	82.56%	129.73%	124.99%
buttfly	147.73%	136.10%	56.84%	51.82%	70.95%	72.34%	107.93%	107.38%	42.27%	74.24%	76.08%	92.03%	121.05%	137.70%
mac	133.43%	138.22%	75.31%	69.50%	82.39%	83.38%	111.45%	117.42%	52.73%	65.10%	74.20%	87.76%	134.79%	129.53%
mult8	129.23%	133.09%	53.70%	71.29%	67.41%	87.23%	107.41%	112.49%	40.91%	71.46%	72.00%	91.26%	138.84%	134.26%
mult16	117.78%	128.66%	39.90%	54.16%	55.47%	72.45%	104.52%	109.73%	33.05%	62.07%	71.62%	79.41%	126.50%	136.83%
Average	134.89%	137.11%	56.29%	59.08%	69.85%	74.81%	107.81%	111.66%	47.92%	67.71%	77.04%	86.99%	127.87%	131.37%

Table 3: Comparison between RUCA and QCM [9]

RUCA				QCM			
Area	MAE	Power	Delay	Area	MAE	Power	Delay
129.33%	2.00%	53.70%	40.91%	135.01%	1.40%	53.93%	47.61%
	1.00%	67.41%	72.00%				
	0.00%	107.41%	138.84%		0.00%	112.40%	128.46%

6 CONCLUSION

In this paper, we proposed a novel methodology RUCA to design runtime configurable approximate circuit with Boolean matrix factorization. Factorized matrices are separated to synthesize each configuration block, while a corrector circuit is created to restore full accuracy. Moreover, we integrated our methodology with design space exploration scheme to improve scalability. We evaluated RUCA on a set of benchmarks, and demonstrated that RUCA significantly reduce power and delay, while providing flexibility to balance the accuracy-power trade-off. Comparing against other approaches, on average RUCA additionally saves power consumption by 3.87% and circuit delay by 11.08% while reducing chip area by 2.22%, which highlights the state-of-the-art performance.

ACKNOWLEDGMENTS

This work is partially supported by NSF grant 1814920 and DoD ARO grant W911NF-19-1-0484.

REFERENCES

[1] T. Alan, A. Gerstlauer, and J. Henkel. 2020. Runtime accuracy-configurable approximate hardware synthesis using logic gating and relaxation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1578–1581.

[2] T. Alan, A. Gerstlauer, and J. Henkel. 2021. Cross-layer approximate hardware synthesis for runtime configurable accuracy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 6 (2021), 1231–1243.

[3] S. Boroumand, C. S. Bouganis, and G. A. Constantinides. 2021. Learning boolean circuits from examples for approximate logic synthesis. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 524–529.

[4] J. Castro-Godinez, H. Barrantes-Garcia, M. Shafique, and J. Henkel. 2021. AxLS: A framework for approximate logic synthesis based on netlist transformations. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 8 (2021), 2845–2849.

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

[6] S. Jain, S. Venkataramani, and A. Raghunathan. 2016. Approximation through logic isolation for the design of quality configurable circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 612–617.

[7] J. Ma, S. Hashemi, and S. Reda. 2021. Approximate Logic Synthesis Using Boolean Matrix Factorization. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[8] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. 2008. The discrete basis problem. *IEEE transactions on knowledge and data engineering* 20, 10 (2008), 1348–1362.

[9] V. Mrazek, Z. Vasicek, and L. Sekanina. 2018. Design of quality-configurable approximate multipliers suitable for dynamic environment. In *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 264–271.

[10] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. 2014. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[11] S. Reda and M. Shafique. 2019. *Approximate Circuits*. Springer.

[12] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda. 2020. Approximate logic synthesis: A survey. *Proc. IEEE* 108, 12 (2020), 2195–2213.

[13] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. 2016. k-way Hypergraph Partitioning via *n*-Level Recursive Bisection. In *18th Workshop on Algorithm Engineering and Experiments*. 53–67.

[14] S. Venkataramani, K. Roy, and A. Raghunathan. 2013. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1367–1372.