5-2019

Designing Approximate Computing Circuits with Scalable and Systematic Data-Driven Techniques

Ling Qiu
Clemson University, lingqiu33@gmail.com

DESIGNING APPROXIMATE COMPUTING CIRCUITS WITH SCALABLE AND SYSTEMATIC DATA-DRIVEN TECHNIQUES

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Ling Qiu
May 2019

Accepted by:
Dr. Yingjie Lao, Committee Chair
Dr. Jon Calhoun
Dr. William R. Harrell

Abstract

Semiconductor feature size has been shrinking significantly in the past decades. This decreasing trend of feature size leads to faster processing speed as well as lower area and power consumption. Among these attributes, power consumption has emerged as the primary concern in the design of integrated circuits in recent years due to the rapid increasing demand of energy efficient Internet of Things (IoT) devices. As a result, low power design approaches for digital circuits have become of great attractive in the past few years. To this end, approximate computing in hardware design has emerged as a promising design technique. It provides design opportunities to improve timing and energy efficiency by relaxing computing quality. This technique is feasible because of the error-resiliency of many emerging resource-hungry computational applications such as multimedia processing and machine learning. Thus, it is reasonable to utilize this characteristic to trade an acceptable amount of computing quality for energy saving.

In the literature, most prior works on approximate circuit design focus on using manual design strategies to redesign fundamental computational blocks such as adders and multipliers. However, the manual design techniques are not suitable for system level hardware due to much higher design complexity. In order to tackle this challenge, we focus on designing scalable, systematic and general design methodologies that are applicable on any circuits. In this paper, we present two novel approximate circuit design methods based on machine learning techniques. Both methods skip the complicated manual analysis steps and primarily look at the given input-error pattern to generate approximate circuits. Our first work presents a framework for designing compensation block, an essential component in many approximate circuits, based on feature selection. Our second work further extends and optimizes this framework and integrates data-driven consideration into the design. Several case studies on fixed-width multipliers and other approximate circuits are presented to demonstrate the effectiveness of the proposed design methods. The experimental results show that both of the proposed methods are able to automatically and efficiently design low-error approximate circuits.

Dedication

To my parents and friends for their love and companion and to my advisor, Dr. Yingjie Lao, for his patience and enlightenment in my research and life.

Acknowledgments

Firstly, I would like to thank my advisor Dr. Yingjie Lao, for his persistent encouragement and exceptional guidance throughout my M.S. study at the Clemson University. Dr. Yingjie Lao provided me with tremendous research inspiration and support in study and research. More importantly, he is more like a life mentor than a research advisor to me. I have learnt a lot of valuable life principles from him. Ancient Chinese litterateur, Yu Han, describes an ideal teacher as: one who could propagate the doctrine, impart professional knowledge, and resolve doubts. To me, he matches all these traits. I am really grateful and fortunate to meet Dr. Yingjie Lao at Clemson University and I wish him all the best for his future career.

I also would like express my special thanks to Dr. Jon Calhoun, Dr. William Harrell at Clemson University, for their support as members of my M.S. committee and the valuable reviews on this dissertation.

My sincere thank also goes to my research group. I am grateful to Joseph Clements, Bingyin Zhao, Jianchi Sun, Azadeh Gholamrezazadeh-Famili, Antian Wang, Xiaojia Wang, Weihang Tan, at Clemson University, for their collaborations and valuable feedback to my research/study. I would also like to thank Minghui Ji, Zhanrui Liao, Shuangying Zhang for their support during my M.S. life.

Lastly, I am truly grateful to my parents and all the other family members for their continuous encouragement and gratuitous care.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Moor's law predicts that the numbers of transistors in a dense integrated circuit doubles every two years [1]. Shrinking feature size enables transistors to consume less power so that they can conduct more operations without dissipating much heat, which often is the constraint of CPU performance. At the same time, a higher transistor density indicates smaller chips and in return smaller devices. The 7nm process of TSMC achieves 20% performance gain and 40% power reduction comparing to its previous 10nm technology [2]. This motivates the semiconductor industry to push the limit of the feature size. In 2016, Samsung became the first company to start their production of 10nm mobile chips for its flagship smartphone, Galaxy S8 [3]. Two years later, Apple released its 7nm mobile processor, A12 Bionic chip which is the first chip using 7nm technology for mass market use [4]. Furthermore, the tremendous computational power improvement resulted from significant feature size reduction also facilitates the evolution of many other computational intensive applications, such as the prevailing deep learning. Neural network, a technique first introduced in the 1950s, is a class of machine learning that imitate the activity of layers of neurons, which compose around 80% of the brain [5]. Early neural networks were built very simple mainly because of the computational limitations. As a result, they cannot be used to capture complex patterns. The stumbling progress of neural network did not see a significant development until nearly 20 years ago, when computers started to become more powerful at computational tasks and GPU (graphics processing units) were developed [6]. However, as the size of technology reaches the deep nanometer realm, the advancement of feature size is seeing an end. The improvements in area, power, and timing resulting from scaling have started to see a decrease. In addition, due to the exponential demand growth of portable IoT devices, energy efficiency in hardware design has become a major design bottleneck.

Under this circumstance, approximate computing in hardware design has become a promising paradigm in recent years, which explores design space to achieve energy-efficient hardware circuits [7–26]. Specifically, this technique relaxes the output accuracy of a circuit to an acceptable extent in exchange for area/power saving. The underlying reason behind this approximation is that many modern computational tasks are error resilient or the errors can be tolerated internally and not perceived by the end user. For example, applications such as machine learning and image processing sometimes do not require a completely accurate result. Therefore, approximate computing essentially takes advantage of this computing characteristic to achieve hardware consumption reduction. In addition, in order to meet the rising performance demands confronting with plateauing resource budgets, approximate computing has become, not merely attractive, but even imperative.

The main research focus of approximate computing at the logic level has been the basic arithmetic elements of a processor: adder [11–23] and multiplier [24–43]. In most of these prior works, approximate versions of the arithmetic elements are manually designed to exploit the trade-off space to achieve the best performance. Manual design strategies are suitable on these basic arithmetic elements since their structures are organized and well studied. These basic arithmetic elements are then used to build large approximate hardware systems. However, such scheme may be inefficient and erroneous due to the unpredictable error propagation within the large system, which will also be illustrated in a case study in our research. Meanwhile, many computational tasks nowadays are data-oriented (data-driven) such that they mainly operate on specific data patterns. In fact, capturing input distribution currently is a very popular problem in machine learning [44]. In additional, most of the existing approximate logic designs in the literature assume a uniform input distribution, which might not always be the case in the real-world scenario. Therefore, it is also imperative to develop data-driven approximate logic design methods, which may create better opportunity to tradeoff between computation accuracy and power consumption.

With the advent of machine learning and data mining algorithms, it is plausible to utilize these algorithms to help design approximate circuits, as projected in Fig 1.1. In this dissertation, we specifically consider the design of error compensation blocks in approximate logic circuits. Error compensation is a commonly used scheme in approximate computing, which aims at correcting the errors generated from logic simplification or voltage over-scaling. For example, in the context of a truncated approximate multiplier, we can save a significant amount of area and power by completely eliminating the computing units for lower bits in the multiplier and then use a small number of logic gates as error compensation blocks to mitigate the resulted errors. Conventional compensation blocks are designed either theoretically or empirically according

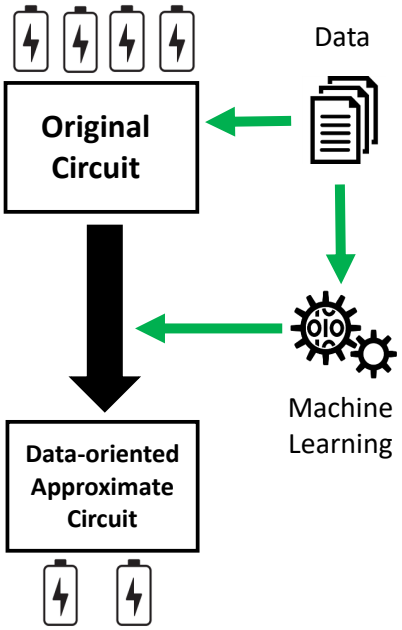


Figure 1.1. Design data-oriented approximate circuits using machine learning techniques

to the circuits' structures and error distribution. In comparison, our goal in this work is to generate data-driven compensation circuits in a systematic manner with the help of machine learning and data mining techniques.

We propose two novel methods in this dissertation that can automatically generate data-oriented compensation circuits for a given approximate circuit under different input distributions. Comparing to the previous works, both of the proposed methods skip all the complicated theoretical analysis steps and solely look into the input and error patterns to design the corresponding error compensation circuits. We use hardware complexity saving and error performance as the design metrics to evaluate our methods.

In our first work, we propose a novel systematic and scalable method of using feature selection methods to design compensation blocks for approximate circuits [45]. We employ the χ^2 feature selection, which is a well known univariate feature selection approach. We illustrate the proposed design flow on a case study of radix-4 modified booth multipliers, which are one of the most popular schemes for signed multiplication. The experimental results show that the proposed approach could achieve better area/power saving and comparable error performance comparing with the existing manually designed approximate multipliers. However, the design complexity and work load are significantly reduced using our approach.

Our second work builds upon the first work's idea of using feature selection and introduces a more robust method while incorporating the consideration of input data distribution into the approximate circuit

design. Compared to the first work, the second one yields a less hardware overhead especially from the timing perspective. We also improve the compensation accuracy significantly by proposing a modified Forward Stepwise Selection (FSS). Lastly, as opposed to the first work that only uses primary inputs of a circuit as features, we expand the candidate features to the circuit's internal wires to further optimize the performance. Our experimental results show that the proposed method achieves significantly better accuracy than the prior data-independent designs, while maintaining relatively minimal hardware overheads for the compensation block.

Chapter 2

Related Work

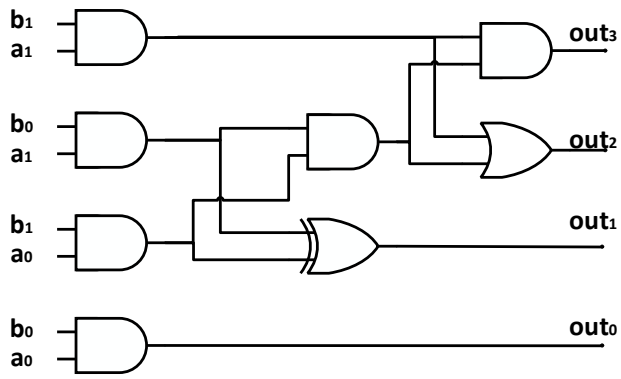
2.1 Approximate Computing

Reliability and accuracy are the elementary principles in all engineering design. However, at certain circumstances such pursuit can be redundant and unnecessary. The reason is that many modern applications may tolerate some extent of errors [46–50]. For example machine learning applications have accepted and incorporated some inaccuracy into model training and inference. There are several reasons behind this. First of all, the data we import into these models can not be completely precise and contains outliers inevitably [51–53]. As a result, the first procedure for data mining is always data cleaning [53], which usually takes a majority of the overall design time. However, with plenty of time and effort spent on this task, error data or outliers still cannot be eliminated due to the nature of man-made mistakes and the limitations of data cleaning techniques. Under such circumstance, error resilient applications are built to tolerate some amount of error. Secondly, some computational applications do not require a complete accurate result for every single data element [54, 55]. Large scale data analytic algorithms aim to get an overall trend instead of the complete correctness of an individual data, e.g., the model of stock market prediction. For such cases, values within a certain range rather than a fixed value are considered as acceptable results.

Since many nowaday applications can be error resilient, we may not need every step of their operations to be completely accurate or with equal importance. In other words, we can possibly trade some applications' computational accuracy for potential run time or energy reduction. A popular design paradigm for such error resilient applications is approximate computing, which includes both software [56–58] and hardware [7–26]. We take the design of an approximate 2-bit multiplier in [8] as an introductory example

a_1a_0 \ b_1b_0	00	01	11	10
00	0000	0000	0000	0000
01	0000	0001	0011	0010
11	0000	0011	1001	0110
10	0000	0010	0110	0100

(a)



(b)

Figure 2.1. Accurate 2-bit multiplier (a) Karnaugh map (b) circuit schematic.

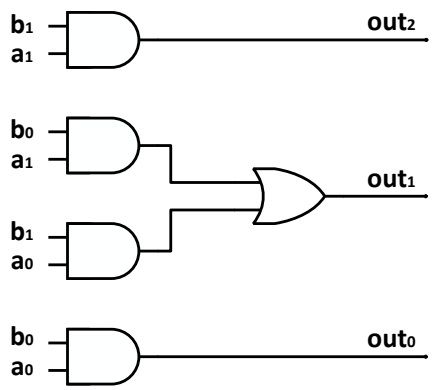
of approximate computing in hardware design. Fig. 2.1 shows the accurate 2-bit multiplier's Karnaugh map and schematic. We notice that there is only 1 minterm whose most significant bit (MSB) output bit is 1, indicated in red. We can take advantage of this characteristic and simplify this circuit with a small cost of output accuracy. If we change this minterm from 1001 to 111 (creates an error of 2) and eliminate the MSB as shown in Fig. 2.2(a), we obtain its approximate version with 37.5% saving on gate number while only introducing a 6.25% error rate. The approximate 2-bit multiplier is shown in Fig. 2.2(b).

2.1.1 Approximate Computing in Hardware Design: Logic Simplification and Voltage Over Scaling

Approximate computing in hardware design can be achieved using various techniques, which can be broadly classified into: logic simplification [21–43] and voltage over scaling (VOS) [59–66]. In this section, we will introduce several existing approximate logic design methods. .

a_1a_0 \ b_1b_0	00	01	11	10
00	000	000	000	000
01	000	001	011	010
11	000	011	111	110
10	000	010	110	100

(a)



(b)

Figure 2.2. Approximate 2-bit multiplier [8] (a) Karnaugh map (b) circuit schematic.

2.1.1.1 Logic Simplification

Logic simplification reduces power/area consumption by redesigning the conventional computing circuits, which can be at the transistor-level [9, 10] or the gate-level [11–20]. Designing approximate circuit at transistor level especially requires sufficient knowledge of the target circuit's architecture. The work in [9] proposes an approximate XOR/XNOR-based adder, which is based on the accurate design of a 10-transistor full adder in [67], as shown in Fig 2.3. The approximate design is shown in Fig 2.4. This design achieves the goal of logic simplification while not introducing much error.

Logic simplification at the gate-level achieves hardware cost reduction by relaxing the accuracy of the circuit's functionality. Most prior works focus on manually redesigning fundamental computation units (e.g., adder [11–23] and multiplier [24–43]) at the gate level. We first take the design of an approximate adder in [11] as an example. This scheme first truncates the computing units that calculate n least significant bits (LSBs). The selection of number n is a tradeoff between output accuracy and hardware consumption. The larger the n is, the more hardware consumption the approximate adder can save. The preserved bit length

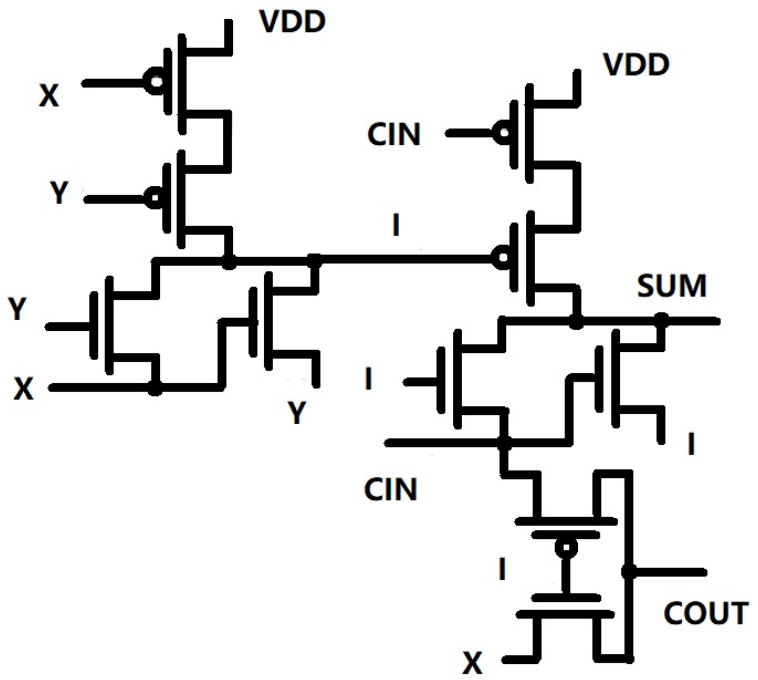


Figure 2.3. Accurate full adder with 10 transistors [67]

is m, while the total bit length is p (i.e. $p = m + n$). In order to compensate the truncation error, [11] applies bitwise OR to approximate the truncated LSB's output value, while the carry in from the n-th bit is approximated using an AND gate. The overall structure is shown in Fig. 2.5.

Next, we move on to the design of an approximate fixed-width multiplier, whose output bit-length is the same as the input bit-length. Similar to the multi-bit approximate adder design, in most approximate n-bit fixed-width multiplier designs, we first truncate the logic cells computing the least significant $n-2$ bits of the product, and then introduce an error compensation block to reduce the errors due to truncation. One approach to compensate the error is to manually add a constant error-compensation bias [33]. This methods leads to very simple logic and small implementation cost; however, the compensated error is still considerably large as the bias cannot adjust according to the input signals. An improved approach is to add an adaptive error-compensation bias to the retained adder cells. In [25], the Booth encoded inputs are transformed into a new set of variables which are then used to design the compensation circuit. The compensation circuit designed in [25] is shown in Fig. 2.6. The inputs of the compensation circuits are the transformation of booth encoded results, the outputs are two carries that are added to the $n-1$ adder cells. In [28], an adaptive conditional-

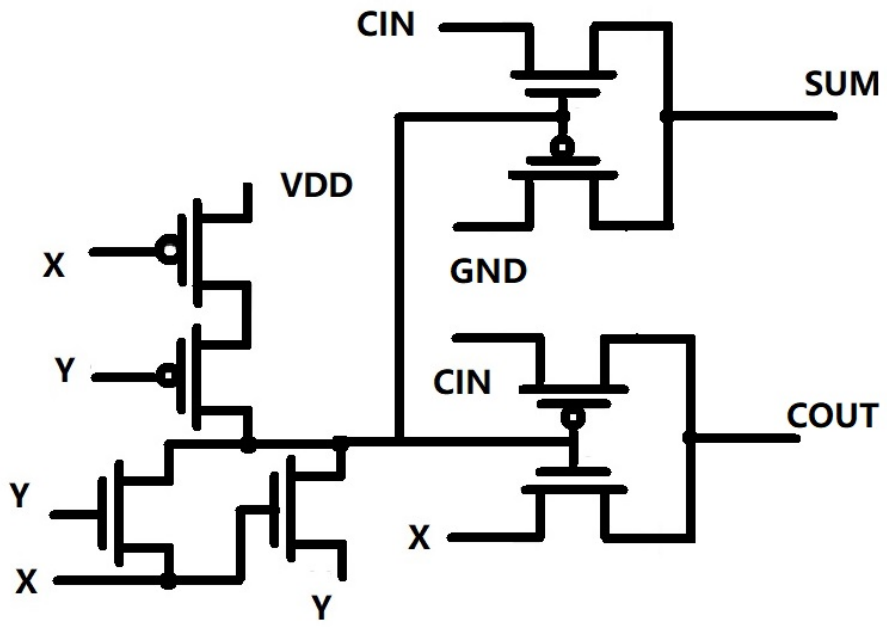


Figure 2.4. Approximate full adder with 8 transistors [9]

probability estimator is proposed to compensate the fixed-width Booth multiplier error. In [26], the sign bit of the Booth encoded multiplier is applied to conditional probability to generate compensation values. For the non-truncated scheme, the approximate designs can be achieved by altering internal components [24]. For example, an approximate 2×2 multiplier is utilized as the building block to construct a larger multiplier to elevate computation efficiency in [8]. In [24], the Booth encoders are approximated to a simpler logic by selectively complementing minterm to reduce circuit complexity.

2.1.1.2 Voltage Over Scaling (VOS)

The second type of approximation is VOS, which dynamically reduces the supply voltage of hardware near threshold to achieve reduction on power at the cost of accuracy [59–66]. As shown in Equation 2.1, a transistor's dynamic power is proportional to the supply voltage square. Thus, lowering only a small number of supply voltage can reduce hardware power consumption significantly. However, when the supply voltage is lowered to an extent such that the critical path is longer than the clock period, timing violation will be introduced to the circuits, which needs to be compensated. The work in [59] introduces the algorithmic

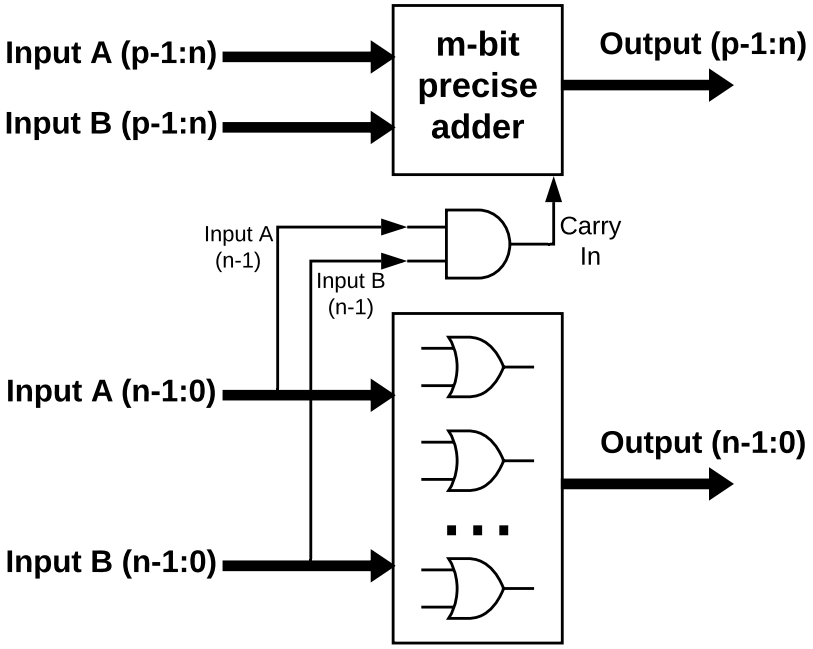


Figure 2.5. Lower-Bit-OR approximate adder [11]

noise-tolerance (ANT) specifically targeting at digital signal processing (DSP) circuits to compensate the error due to VOS. [65] proposes dynamic segmentation with multi-cycle error compensation and delay budgeting for chained data path components to scale computing functions in a constraint manner. [66] explores the trade-off between image processing quality and power consumption under VOS. While VOS can save a great amount of power, it will increase the area overhead due to the introduction of compensation circuits.

$$Power = C_L * Vdd^2 * f \tag{2.1}$$

2.1.2 Approximate Computing in Hardware Design: Manual Design and Automatic Design

The design of approximate computing hardware can also be broadly divided into two categories according to design strategies: manual designs and automatic approaches. Manual design strategies have achieved excellent performance on arithmetic elements ((e.g. adder [11–23] and multiplier [24–43])). These design strategies usually require analyzing the resiliency of each component in a circuit, which are difficult

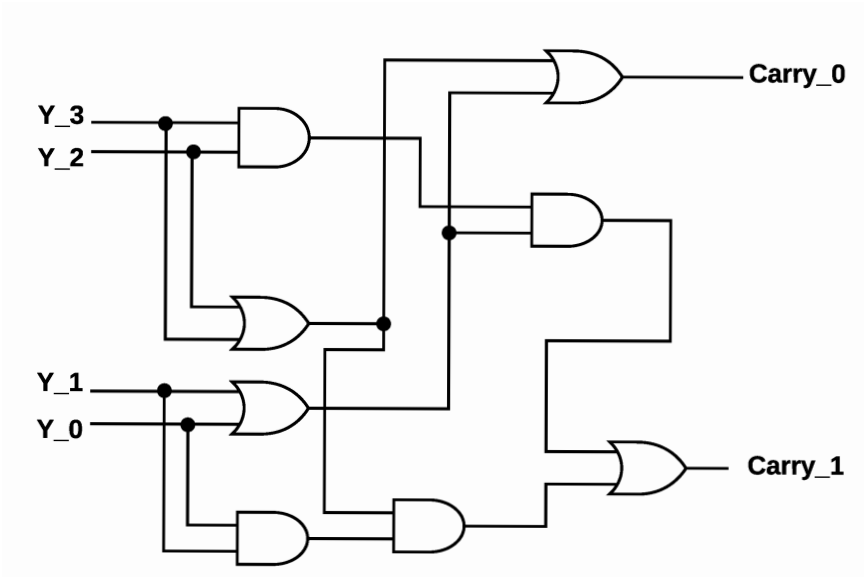


Figure 2.6. Compensation block for 10-bit fixed-width approximate booth multiplier [25]

to generalize to a wide variety of circuits. On the other hand, automatic design strategies focus on developing general methodologies for designing approximate circuits [68–75]. For instance, the concept of approximate logic synthesis (ALS) [69–73] has been developed to automatically synthesize a Boolean function into either a two-level [70] or multi-level [69, 71] approximate version under given error constraints. [70] introduces some algorithms to identify the minterm complements that result in an approximate circuit with least number of literals. [69] utilizes Boolean network simplifications allowed by external don't cares and effectively synthesizes the approximate networks in an error-constrained approach. In [71], the problem of approximate synthesis is mapped into a conventional don't care based optimization problems. Other techniques such as probabilistic pruning has also been proposed to obtain approximate circuits by iteratively pruning circuits' internal nodes at the gate-level [75–79].

2.1.3 Our Contribution in Approximate Computing in Hardware Design

Our work focuses on the automatic design of approximate computing circuits for both logic simplification and VOS. We propose two novel approaches that can systematically generate compensation blocks for any given approximate circuits, while incorporating with data-driven considerations. Our work treats the approximate circuit prior compensation as a black box and use machine learning based methods to systematically design its compensation circuit. Our methodologies offer significant design load reduction as a step toward the design of optimized system-level approximate circuits.

2.2 Feature Selection

Feature selection is the process of selecting a subset of relevant features to reduce the size of the structure without significantly decreasing the accuracy of the computation, and is widely-used in machine learning and statistics [80–82]. Feature selection can even improve the prediction accuracy by eliminating irrelevant features. This technique can be mainly divided into three categories [83]: filter methods [84–86], wrapper methods [87–89], and embedded methods [90, 91]. Filter methods select features regardless of the model. They are solely based on the correlation between each individual feature and the response. On the other hand, wrapper methods evaluate subsets of features instead of a single feature. In other word, they take potential feature interaction into consideration when selecting feature subsets. Embedded methods are the combination of both feature selection techniques, which leverage the advantages from both methods.

For an approximate circuit, the error is dependent on the input data pattern. In addition, we assume that the size of the compensation block is proportional to the number of inputs to this block. As a result, the quality and quantity of the compensation block inputs are critical to the error mitigation performance of the approximate logic circuit design. In general, the desired performance can be achieved by using only a few but highly correlated inputs to design the error compensation block.

2.2.1 χ^2 feature selection

In our first proposed approach, we use the χ^2 feature selection, which is a commonly used univariate feature selection approach, to detect the most informative features [92]. Univariate feature selection examines each feature separately to determine the degree of correlation between the feature and the given response labels. Thus it is one of the filter methods. The χ^2 algorithm can automatically discretize the continuous attributes and removes irrelevant attributes based on the χ^2 statistic and the inconsistency found in the data [93]. The calculation of χ^2 is shown in Equation 2.2, where n is the number of observations; k is the number of mutually exclusive classes; O_k is the observed frequency and E_k is the expected frequency. The value χ^2 is used to indicate the correlation between feature and response under the hypothesis that they are independent. If the value of χ^2 is comparably large, it indicates that this feature and response is highly correlated statistically, and vice versa.

$$\chi^2 = \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k} \tag{2.2}$$

2.2.2 Foward Stepwise Selection

In our second proposed approach, we use and modify the Foward Stepwise Selection (FSS) [94] in the feature selection process. FSS belongs to the wrapper method, since adding a new feature is determined by the previously selected features. The pseudo code of the algorithm is shown in Algorithm 1. At the beginning of the algorithm, FSS starts with an empty feature subset. Then, for the rest of the steps, FSS repeats the process of adding a new feature that help to reduce the cost function the most by joining with the selected features. In Algorithm 1, the stopping criteria for FSS is a pre-defined maximum number of features to be selected. Notice that a threshold of cost function can also be the stopping criteria.

Algorithm 1 Forward Stepwise Selection

INPUT: Candidate feature pool **F**, number of selected features m, cost function $\mathcal{C}(\cdot)$;
OUTPUT: Selected feature subset $\hat{F}$;

let $\hat{F}_0$ denote the null subset;
for $y \leftarrow 0$ ***to*** $m-1$ **do**
  $ft = \arg\min_{\mathbf{F}}(\hat{F}_y \cup ft)$;
  $\hat{F}_{y+1} \leftarrow \hat{F}_y \cup ft$;
end
return $\hat{F}$

In contrast to FSS, Backward Stepwise Selection (BSS) uses the opposite feature selection steps of the FSS. It starts with all the features included in the feature subset, and then eliminates a least significant feature at each step. Similar to Algorithm 1, the pseudocode of BSS, as shown in Algorithm 2, also uses the number of selected features as the stopping criteria.

Algorithm 2 Backward Stepwise Selection

INPUT: Candidate feature pool **F**, total number of feature N, number of selected features m, cost function $\mathcal{C}(\cdot)$;
OUTPUT: Selected feature subset $\hat{F}$;

let $\hat{F}_N$ denote the full feature subset;
for $y \leftarrow N$ ***to*** $N-m$ **do**
  $ft = \arg\min_{\mathbf{F}}(\hat{F}_y \setminus ft)$;
  $\hat{F}_{y-1} \leftarrow \hat{F}_y \setminus ft$;
end
return $\hat{F}$

The reason behind adopting FSS instead of BSS in our second proposed approach is that the former algorithm has significantly shorter running time comparing to that of the later one. In our case of hardware design, we usually have a large amount of candidate features. Thus, it would be very time consuming to

use BSS in the feature selection process. Secondly, since we aim to design a compensation circuit with low hardware overhead, we seek to minimize the number of features. Therefore, we use and modify FSS in our design flow, which will be presented in Chapter 4.

Chapter 3

Design Approximate Circuit Using Feature Selection

3.1 Introduction

In this work, we propose a systematic and general design flow for designing approximate circuits with specific focus on designing their compensation circuits. Our method uses feature selection, which aims at generating compensation circuits with low hardware cost. Unlike aforementioned manual design techniques, this method is able to skip all the theoretical analysis on circuits' structures, while only relying on given input-error data to automatically design compensation circuits to mitigate the error.

The overall design flow of the proposed methodology for designing approximate logic circuit using feature selection is shown in Fig. 3.1, which consists of five steps: Error Generation, Compensation Bits Insertion, Compensation Input Selection, Reduced Truth Table Generation and Compensation Circuit Synthesis.

In our proposed method, we assume the truncation error is given. In other words, error generation is achieved either due to logic simplification or voltage overscaling. We use the ideal output, truncated output and compensated output to represent the outputs of the original circuit, the primitive approximate circuit (PAC), and the compensated approximate circuit, respectively. The final circuit essentially consists of the PAC and the generated compensation circuit.

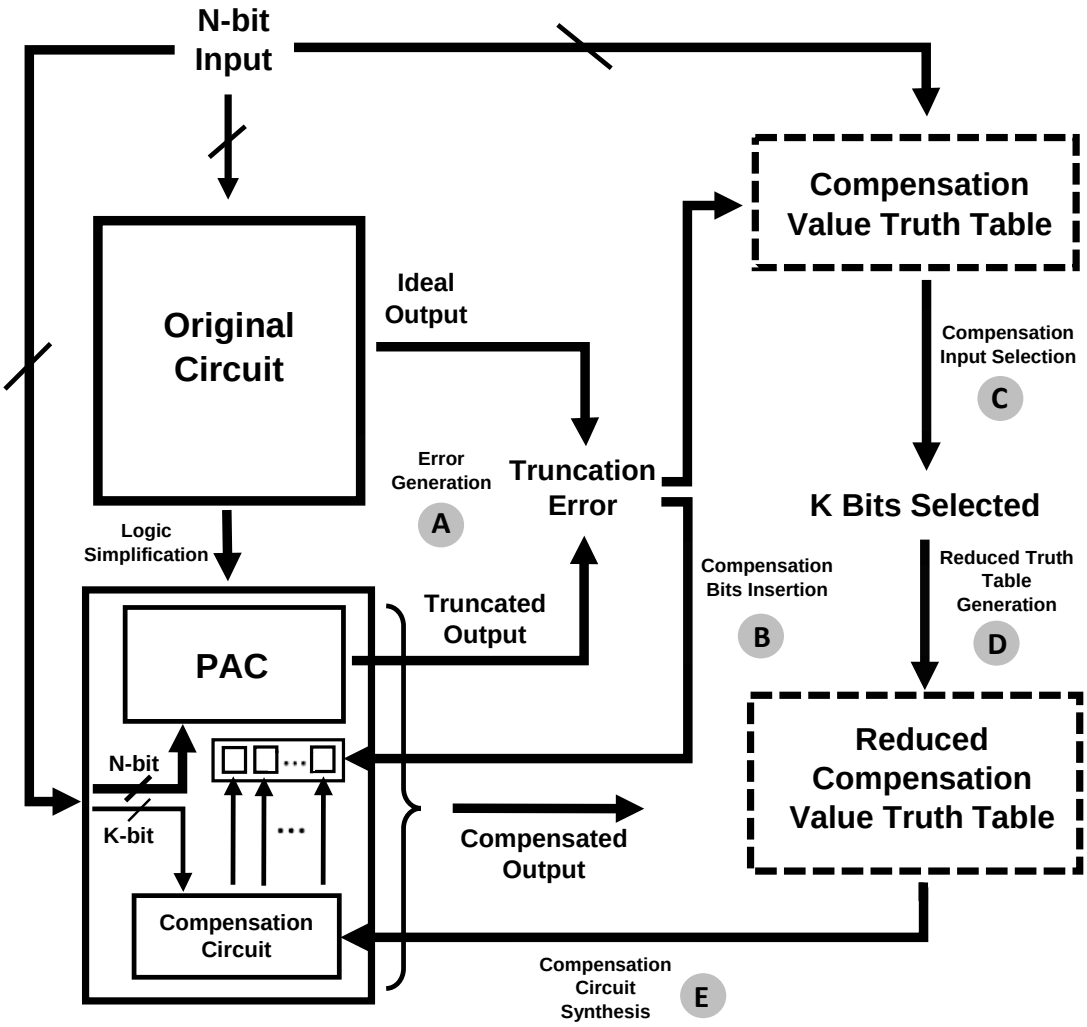


Figure 3.1. Compensation circuit design flow.

3.2 Design Flow

3.2.1 Error Generation

The first step is to compute the pre-compensation error of the approximate circuit. The errors are generated due to the logic simplification of the original circuit or the timing errors in voltage over-scaling applications. If we consider a truncated scheme in approximate circuit design, errors are basically calculated by taking the differences between the ideal output and truncated output without adding error compensation, i.e., the output of PAC, as shown in Fig. 3.1. Choosing the appropriate methods to provide the best simplification opportunity is another important research topic in approximate computing, which is peripheral to this work. Intuitively, if the truncation in a truncated approximate circuit is too large, it would be difficult to implement a simple compensation circuit to offset the errors; on the other hand, if the truncation is too small, the area/power saving would be minimal. In this work, we consider the error patterns are given and focus on the design of the error compensation block. Future work will be directed towards using machine learning algorithms to automatically determine the appropriate logic simplification without significantly compromising the accuracy of the computation.

3.2.2 Compensation Bits Insertion

We insert compensation bits into the approximate circuit at the second stage. After logic simplification, there are 2^N input-error pairs for an approximate circuit with an N-bit input, if every input bit is independent. We determine the number and the locations of compensation bits to insert based on the error distribution. Generally, at least $\lceil log_2 R_e \rceil$ compensation bits are required for an error dynamic range of R_e. However, since the errors are usually distributed unevenly across the dynamic range, it is possible to reduce the number of compensation bits to $log_2 R_e$ and approximate the $R_e - 2^{log_2 R_e}$ least frequent error values to the nearest value in the reduced dynamic range, which can be considered as another step of logic simplification. For example, most truncation-based logic simplification methods lead to all positive errors and tend to have less probability for larger magnitude errors. Therefore, we can assign compensation value cp_i to each input-error pair according to Equation 3.1, where the threshold T is equal to $2^{log_2 R_e} - 1$.

$$cp_i = \begin{cases} e_i & e_i \leq T \\ T & e_i > T \end{cases} \tag{3.1}$$

Consequently, the overall complexity of the compensation block is also reduced, while the errors would increase only slightly. In this work, we construct the compensation bits in different bit positions, i.e., N compensation bits could generate 2^N values. However, it is important to note that it is not necessary to always put all compensation bits in distinct bit positions. For example, if the error only ranges from 0 to 2, we can put two compensation bits both in the LSB of the circuit, which may result in better compensation block design compared to inserting the bits into two consecutive positions.

3.2.3 Compensation Input Selection

We then apply feature selection algorithms to identify the inputs that are most correlated with the compensation values. We use the χ^2 univariate feature selection to select the inputs for each compensation bit. The total number of compensation input bits K should be picked based on the accuracy requirements or the error constraints. We can either select the inputs for all compensation bits simultaneously based on the overall feature selection rankings and scores or use logic synthesis tools to optimize the multiple-output combinational error compensation circuit automatically. The more features selected, the more accurate result the circuit will generate, however, the hardware cost and the power consumption of the error compensation block will be higher.

In this work, we apply feature selection directly on the primary inputs of a logic circuit. Ongoing work includes the investigation of using feature extraction algorithms to generate higher-order features as the inputs to the compensation blocks, which might be more informative and non-redundant and hence facilitate the subsequent compensation block synthesis steps to produce better performance. In fact, the Booth encoded inputs can be considered as higher-order features for designing an approximate multiplier.

3.2.4 Reduced Truth Table Generation

After we determined the highest correlated K inputs by using feature selection, the truth table of the input and the compensation value will be reduced from 2^N rows to 2^K rows. Thus, each row in the reduced truth table corresponds to 2^{N-K} rows in the original truth table. The compensation value cp_j for each row in the reduced truth table should be selected such that the overall error for this row is minimized, as expressed in Equation 3.2:

$$cp_j = \arg\min_{cp_j}(\sum_{i \in R_e} ef(cp_i, cp_j) \times p_i), \tag{3.2}$$

where p_i is the percentage appearance of cp_i in the corresponding 2^{N-K} rows of the original truth table and $ef()$ represents the error metric. For example, if we use the mean squared error as the error metric, Equation 3.2 can be reduced to

$$cp_j = \arg\min_{cp_j}(\sum_{i \in R_e}(cp_i - cp_j)^2 \times p_i) \tag{3.3}$$

We repeat this for all 2^K reduced input combinations to generate the complete reduced truth table. When K is large (i.e., 2^{N-K} is small), it is possible to have multiple cp_j with the same minimum value of Equation 3.2 for certain inputs. In this case, we can assign don't care terms to these rows in the reduced truth table, which could help to minimize the logic in the final step.

3.2.5 Compensation Circuit Synthesis

The last stage is to use logic synthesis tool to optimize the combinational error compensation circuit that is specified by the reduced truth table from the previous step. State-of-the-art approximate logic synthesis (ALS) algorithms [69–71] can also be applied on top of the error compensation circuit to further improve the hardware implementation efficiency.

3.3 Case Study and Experimental Results

Our proposed methodology will be extremely suitable for large and complex approximate circuit designs that are not feasible for manual analysis or simplifications, as only the input and error patterns are required for the design flow described in Section 3.1. In this section, we apply our methodology to design a relatively small but well-studied circuit (i.e., an approximate multiplier) for the purposes of demonstration and comparison.

3.3.1 Experimental Setup

In our experiment, we use the Scikit-learn toolkit to perform the χ^2 univariate feature selection, which is an open source machine learning library in Python that includes a wide range of machine learning tools [95]. All the circuits are synthesized using Synopsys Design Compiler and mapped to a 32 nm standard cell library to evaluate the performance of their hardware implementations.

3.3.2 Approximate Multiplier Architectures

We employ the proposed methodology on a 10-bit fixed-width multiplier and use the radix-4 modified Booth encoding scheme to reduce the number of partial products. We consider a truncated scheme where the adder cells for calculating the 9 LSBs are deleted from the original circuit, as shown in Fig. 3.2(a). The corresponding truncation error distribution is shown in Fig. 3.2(b). It can be seen that the dominant errors are 1 and 2, and the maximum error is 3. Therefore, we insert two compensation bits, λ_1 and λ_0, to the 2 LSBs of the PAC output product. The two compensation bits can be easily integrated into the Wallace tree or Dadda tree architecture. Next, we use feature selection to rank the inputs. We select 4, 6, 8, 10 features respectively in our experiment and then generate the reduced truth table based on the selected inputs accordingly.

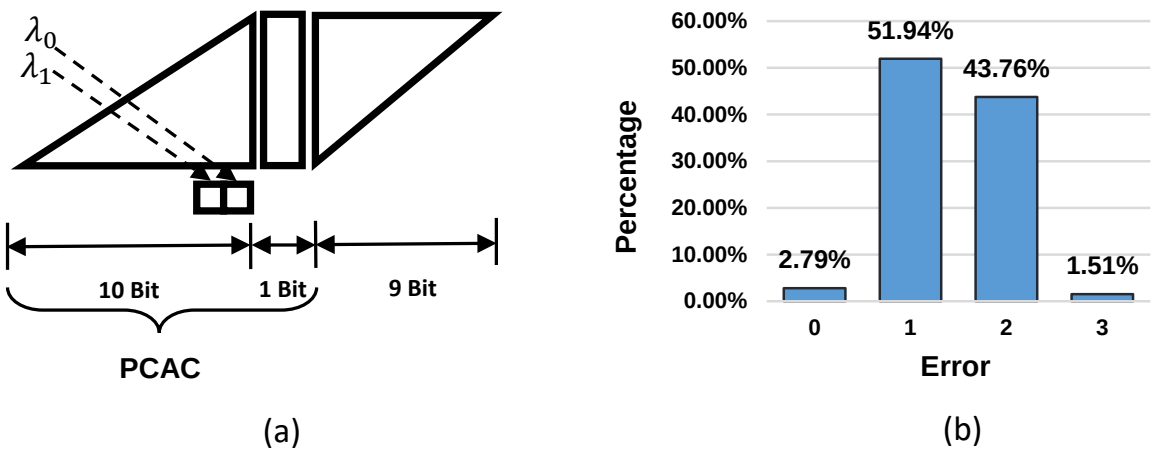


Figure 3.2. 10-bit fixed-width approximate multiplier: (a) truncation and bit insertion scheme, (b) error distribution after truncation

3.3.3 Experimental Results

We use the mean error ε_{mean} and mean squared error ε_{mse} to evaluate the arithmetic performances of different approximate multiplier designs, which are calculated by:

$$\varepsilon_{mean} = mean(P_{oc} - P_{cac})/2^n, \tag{3.4}$$

$$\varepsilon_{mse} = mean(P_{oc} - P_{cac})^2/2^n, \tag{3.5}$$

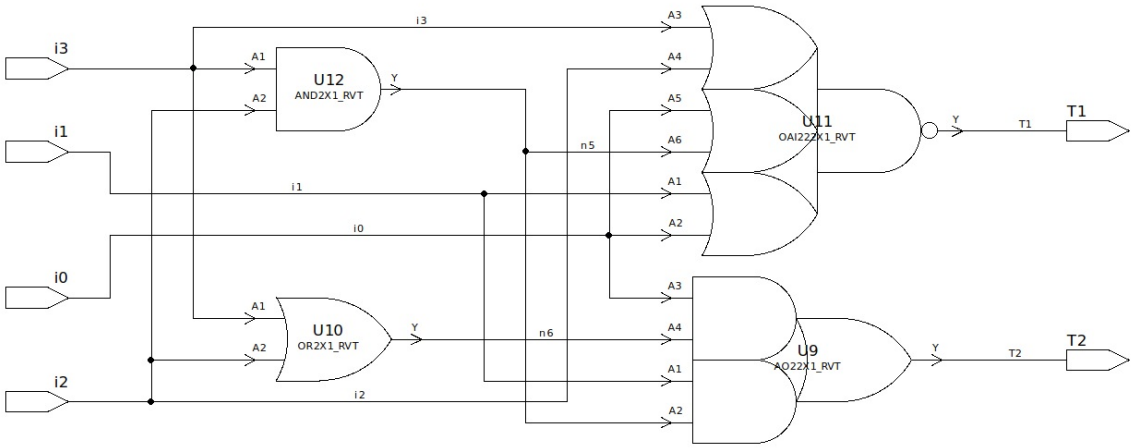


Figure 3.3. Synthesized compensation circuit for PAC (k=4)

where P_{oc} and P_{cac} represent the output products of the original booth modified multiplier and the compensated approximate booth modified multiplier, respectively. The mean error is an important metric for approximate multiplier design which captures the performance of systems that consist of a large number of multipliers, especially for multimedia and digital signal processing (DSP) applications where the final output results are usually accumulated by a series of multiplication products.

The performances are summarized in Table 3.1 and Table 3.2, along with comparisons to existing approximate fixed-width multiplier designs [26, 27, 96]. The area and power consumptions are normalized to the original circuit It can be seen that the error can be significantly reduced from PAC by using the proposed methodology, while achieving 30% to 40% saving in power consumption of the original circuit when we select 4, 6, 8, or 10 input bits for designing the compensation circuit. When we use more features, error is decreased, while the area/power consumption will be increased. For example, when we increase the number of input bits K from 4 to 10, the mean squared error is decreased by 15.7%; however, the area consumption is increased significantly, i.e., from 59.97% to 82.56% of the original circuit. In this particular case of approximate multiplier, $K = 4$ already achieves a very good performance, which only involves a very simple combinational circuit. The mean error is almost zero when $K = 4$. Fig. 3.3 shows the synthesized circuit. The input of this compensation circuit should be connected with corresponding selected features (inputs); the output of this compensation circuit is connected to the λ_1 and λ_0 shown in Fig. 3.2(a).

Compared to existing approximate fixed-width multiplier designs, the proposed methodology generally leads to slightly worse mean squared error than the designs in [26, 27, 96] but comparable mean error,

Table 3.1. Comparison of the error performance of different fixed-width multipliers

multiplier	ε_{mean}	ε_{mse}
PAC	1.4375×2^{-9}	2.3166×2^{-18}
[96]	-0.0039×2^{-9}	0.1542×2^{-18}
[27]	0.0689×2^{-9}	0.1544×2^{-18}
[26]	-0.0039×2^{-9}	0.1498×2^{-18}
proposed (K = 4)	0.0000×2^{-9}	0.2714×2^{-18}
proposed (K = 6)	0.0313×2^{-9}	0.2549×2^{-18}
proposed (K = 8)	0.0078×2^{-9}	0.2394×2^{-18}
proposed (K = 10)	0.0098×2^{-9}	0.2287×2^{-18}

Table 3.2. Normalized area and power consumptions of different fixed-width multipliers

multiplier	$NormArea$	$NormPower$
Original Circuit	100%	100%
[96]	66.95%	64.91%
[27]	65.45%	64.31%
[26]	66.10%	63.66%
proposed (K = 4)	59.97%	60.14%
proposed (K = 6)	61.64%	60.79%
proposed (K = 8)	65.30%	62.69%
proposed (K = 10)	82.56%	68.46%

while achieving better performance in area/power reduction for K from 4 to 8. Further more, the main advantage of the proposed methodology is that it significantly reduces the design complexity and provides a large design space of approximate circuit architectures with different values K. The number of features used for designing the compensation circuit needs to be selected based on the specific application requirement. Besides, it is important to note that compared to other existing manual design techniques, it is much easier to apply the proposed methodology on large-scale systems.

Chapter 4

Data-Driven Approximate Circuit Design

4.1 Introduction

In this chapter, we propose an automatic data-driven compensation circuit design technique, which takes circuits' input data into design consideration. The reason is that in the real-world scenario, we expect input patterns to vary across different tasks. Approximate circuits should also be adapted accordingly to better fit specific computational tasks. We also demonstrate the necessity of data-driven design via experimental results.

In addition, the proposed novel error correction structure is able to minimize the timing overhead comparing to our first method [45]. The motivation is that our previous work takes the difference between the erroneous output and the original output as the compensation value, which yields an additive compensation circuit that may incur non-negligible timing overhead. Therefore, it is not suitable for applications where speed is a major concern.

The other contributions of this work include: we propose a modified Forward Stepwise Selection (FSS) technique for selecting feature subset, which significantly improves the accuracy of feature selection; we also include internal wires into constructing compensation circuits, which achieves better performance in terms of error mitigation.

In our proposed approximate logic design methodology, both the input samples and the netlist of the

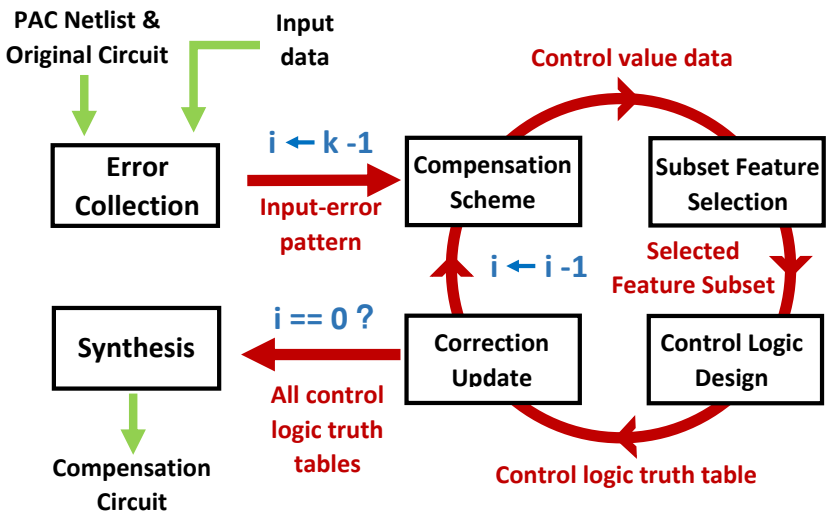


Figure 4.1. General design flow

PAC are required for initiating the proposed method. The overall design flow is shown in Fig. 4.1, whose details will be explained in the next section.

4.2 Design Flow

4.2.1 Compensation Scheme

According to the input-error patterns of the PAC, the proposed methodology first determines the bit-length of the compensation output by analyzing the error statistics. As opposed to the additive compensation circuit [45], we propose to correct k bits in a pairwise manner with each control signal (ct_i) indicating whether a correction to a PAC output bit (pac_i) is necessary and k is determined by the error dynamic range. An XOR gate is introduced for each individual bit that needs correction. A basic block for the compensation scheme is shown in Fig. 4.2, where ct_i and pac_i are the two inputs of the XOR gate and the corrected bit op_i is the output. According to the XOR logic, if pac_i needs a correction (i.e., from 1 to 0 or from 0 to 1), ct_i will be set to 1. In order to correct the output of each sample as close to the original output as possible, without loss of generality, we use the following procedure to construct the pac_i's compensation vector, $\mathbf{G_i} = (g_1, g_2, ..., g_n)_i$, where n is the total number of samples. Assume the bit-length of the PAC is w, then we denote the output of PAC as $\tilde{\mathbf{P}} = (\tilde{p}_1, \tilde{p}_2, ..., \tilde{p}_n)$ and the corresponding w-bit output of the

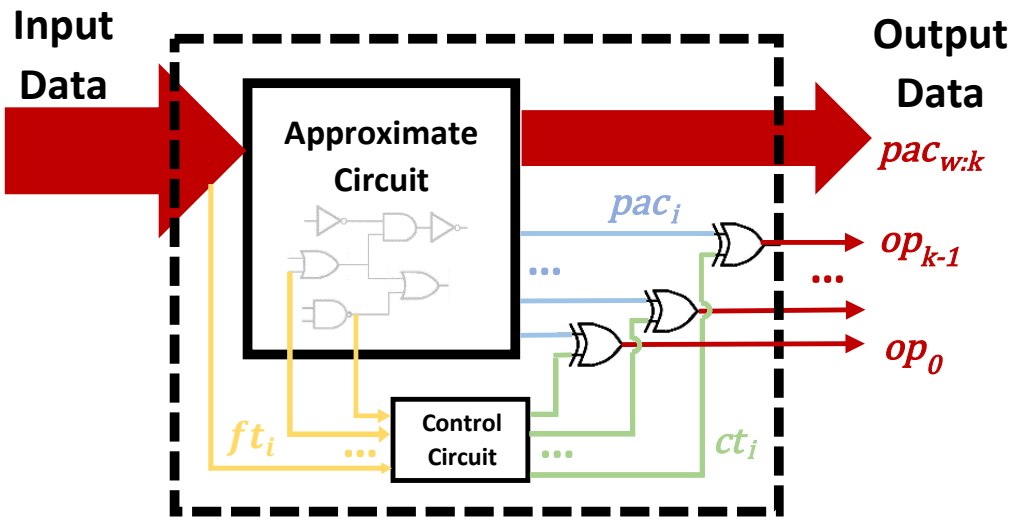


Figure 4.2. Compensation structure

original circuit as $\mathbf{P} = (p_1, p_2, ...p_n)$. Then $\forall\, j \in [1, ..., n]$:

$$g_j = \begin{cases} p_j[i] & p_j[w-1:i] = \tilde{p}_j[w-1:i] \\ 1 & p_j[w-1:i] > \tilde{p}_j[w-1:i] \\ 0 & p_j[w-1:i] < \tilde{p}_j[w-1:i] \end{cases} \tag{4.1}$$

where $p_j[w-1:i]$ represents the partial binary number from the MSB to the i-th bit of p_j.

The next step is to calculate the control value vector of each bit, $\mathbf{C_i} = (c_1, c_2, ...c_n)_i$ with $\mathbf{G_i}$ and $\tilde{\mathbf{P}}$. $\forall\, j \in [1, ..., n]$:

$$c_j = g_j \oplus \tilde{p}_j. \tag{4.2}$$

4.2.2 Modified Forward Stepwise Selection

For the purpose of reducing the hardware complexity, we only select a small subset of nodes (features) to generate each control logic. χ^2 univariate feature selection technique is used in [45], which ranks each feature individually according to its correlation with the output. Thus, due to its greedy nature, it may not be able to capture the inter-correlation among the features in the subset. To address this problem, we propose a modified Forward Stepwise Selection (FSS) algorithm as described in Algorithm 3 for feature subset

selection. We denote the overall candidate feature pool as $\mathbf{F}$ and a single feature in $\mathbf{F}$ as ft. The conventional FSS begins with a null feature subset, and then adds features to the subset one by one, until meeting a stop criteria [82]. In our experiments, we set the max number of features as the stop criteria. Note that this value can also be considered as a user-defined parameter for trading off between error and hardware complexity. In order to evaluate the performance of a selected feature subset as a whole, we use a cost function, $\mathcal{C}(\cdot)$, as expressed by Equation 4.3:

$$\mathcal{C}(\hat{F}) = \|\mathbf{C_i} - \mathcal{TT}(\hat{F})\|_0 \tag{4.3}$$

where $\tilde{F}$ is the selected feature subset and $\mathcal{TT}$ is the generated truth table for $\tilde{F}$, which will be explained in Section 4.2.3.

Algorithm 3 Modified Forward Stepwise Selection

Input: Candidate feature pool $\mathbf{F}$, control value vector $\mathbf{C}_i$, number of subsets h, number of selected features m, cost function $\mathcal{C}\ (\cdot)$

Output: Selected feature subset $\hat{F}$

```
for x ← 1 to h do
    reset F to initial;
    let F̂_{x,0} denote the null subset;
    for y ← 0 to m − 1 do
        if y ≠ 0 and C (F̂_{x,y}) = C (F̂_{x,y−1}) then
            remove last added feature from both F̂_{x,y} and F;
            y ← y − 1;
        end
        else
            ft = arg min_F (F̂_{x,y} ∪ ft);
            F̂_{x,y+1} ← F̂_{x,y} ∪ ft;
        end
    end
end
Select F̂_{x,y} with lowest C as F̂;
return F̂
```

Based upon the conventional FSS, we modify the feature selection procedure specifically for our approximate logic design methodology based on the following two observations.

4.2.2.1 Observation 1

For a model whose features and response variables are both binary, conventional FSS sometimes settles at a local minimum. In particular, we have seen many such cases in our experiments that the feature

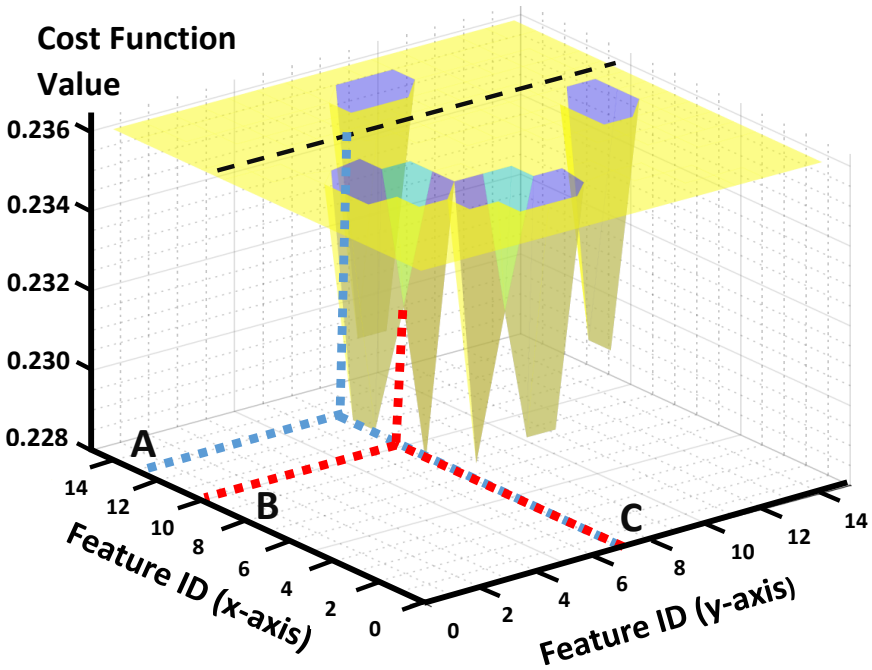


Figure 4.3. Observation 1 example: cost function distribution for pairwise coupling 15 features

with the highest score (i.e., cost function reduction) converge to a local minimum where no further reduction on the cost function is possible by adding any additional feature thus terminating the conventional FSS. In addition, due to the binary representations, many features yield the same scores. Thus, we argue that always selecting one feature with the highest score as in the χ^2 univariate feature selection might not be optimal for approximate logic design. Fig. 4.3 illustrates a case in our experiments of the cost function distribution from exhaustively pairing 15 feature candidates. Each of the features can individually reduce the cost function value to 0.236. In other words, any of these features may be selected at the first iteration of FSS. However, it can be seen that feature #13 (point A) is a local minimum, since pairing any additional features does not reduce the cost function, which is indicated by the black dash line. Neither feature #10 (point B) nor feature #7 (point C) is a local minimum, as the cost function can be further lowered by adding an additional feature into the feature subset. As shown in Algorithm 3, we propose a modification such that when a local minimum is found, the algorithm removes the previously added feature from both the feature subset ($\hat{F}$) and the candidate feature pool ($\mathbf{F}$) to escape the local minimum. Through this approach, bad local minimums are gradually avoided, which eventually results in a better feature subset.

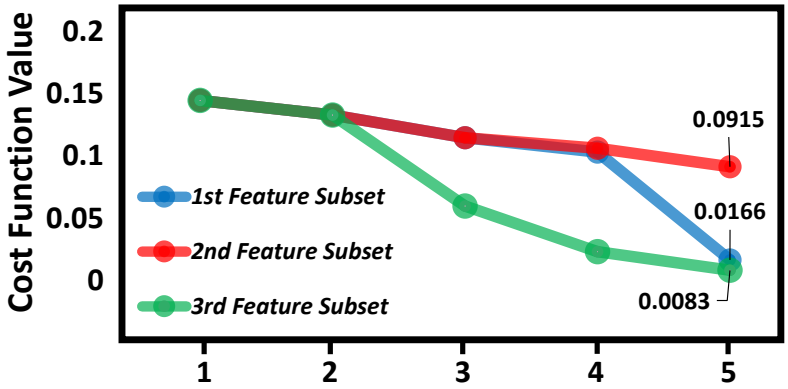


Figure 4.4. Observation 2 example: cost function value for different selected feature subsets

4.2.2.2 Observation 2

As we mentioned above, there are usually multiple candidate features that achieve a same cost function reduction under Boolean logic applications. Since we find it difficult to predict which feature could yield the best performance in the subsequent steps, we repeat the modified FSS for h times and hence build h different feature subsets at each step. In general, the larger h is, a better feature subset can be obtained; however, the time complexity of the algorithm will increase. In the example of Fig. 4.4, we generated 3 different feature subsets with 5 selected features in each subset. It can be seen that although the first 2 selected features for all the 3 feature subsets have the same cost function values, they progress differently in the subsequent steps. In our algorithm, we select the feature subset with the lowest cost function among the h subsets at the last step as the final $\hat{F}$ (i.e., the 3rd feature subset in this example).

4.2.3 Control Logic Design

We introduce the procedure of generating the reduced truth table for the control logic of each output bit, ct_i, which is used in selecting feature subset as well as in constructing the final control logic (ct_i) truth table.

After given the selected feature subset, $\hat{F}$, we construct the reduced control bit truth table, $\mathcal{TT}$, with m selected features. The control bit value for each feature pattern in this truth table is determined by the

Table 4.1. Reduced truth table generation

(a) Original $\mathbf{C_i}$ truth table

Feature $\mathbf{F}$				Response
ft_w	ft_x	ft_y	ft_z	$\mathbf{C_i}$
0	1	1	0	1
0	0	0	0	0
0	0	1	1	1
0	0	1	0	0

(b) Reduced truth table

Feature $\hat{F}$		Response
ft_w	ft_z	ct_i
0	0	0
0	1	1

larger appearance between 0's and 1's to minimize the overall error as expressed in Equation 4.4:

$$\mathcal{TT}(j) = \begin{cases} 0 & Pr(0) > Pr(1) \\ 1 & Pr(0) < Pr(1) \\ X & Pr(0) \approx Pr(1) \end{cases} . \tag{4.4}$$

where $TT(j)$ represents the response value for the feature pattern j, and $Pr(0)$, $Pr(1)$ are the appearance percentages of 0's and 1's, respectively. We assign don't cares when $Pr(0)$ and $Pr(1)$ are equal or close, which could be exploited in logic optimization for reducing the hardware cost. We use an example in Table 4.1 to illustrate the basic idea. In this table, we have a total number of 4 candidate features and 1 response, which is the control value vector of the i-th bit, $\mathbf{C_i}$. Suppose features ft_w and ft_z are selected to build a reduced truth table for ct_i. As it can be seen in Table 4.2 that when $\{ft_w, ft_z\} = 01$, ct_i is 1, regardless the values of ft_x and ft_y. When $\{ft_w, ft_z\} = 00$, $Pr(0) = \frac{2}{3}$, which is larger than $Pr(1) = \frac{1}{3}$. Therefore, when $\{ft_w, ft_z\} = 00$, we choose ct_i to be 0. In this example, there is one row out of four rows in the original true table does not match to the reduced truth table. Thus, $\mathcal{C}(\{ft_w, ft_z\}) = \frac{1}{4}$.

4.2.4 Correction Data Update

After the logic for the current control bit is determined, we need to update PAC's corresponding output bit, i.e., $\tilde{P}$, since the current bit may not be corrected completely thus may affect lower bits' compensation scheme. We update these compensation values as: $\forall\, j \in [1, ..., n]$:

$$\tilde{p}_j[i] = \tilde{p}_j[i] \oplus \mathcal{TT}(\hat{F}) \tag{4.5}$$

Then, the algorithm iterates on to the next lower bit. The overall algorithm is presented in Algorithm 4. After the logic for all the control bits is determined, the compensation circuit is synthesized and connected

Algorithm 4 Compensation Block Generation

INPUT: $\mathbf{P}$, $\tilde{\mathbf{P}}$ and $\mathbf{F}$;
OUTPUT: $\mathcal{TT}$ and $\hat{F}$ for all correction bit;

```
for i = k − 1 to 0 do
    for each sample do
        assign G_i according to Equation 4.1;
    end
    for each sample do
        assign C_i according to Equation 4.2;
    end
    Feature Selection: Modified FSS;
    Construct truth table TT for ct_j;
    if i = 0 then break
    ;
    for each sample do
        Update P̃ by Equation 4.5;
    end
end
```

to the PAC as shown in Fig. 4.2.

4.3 Case Study and Experimental Results

4.3.1 Experimental Setup

We implement the proposed algorithm on Python Jupyter Notebook. For comparison, we also use scikit-learn [95], a well-known machine learning library in Python, to perform the χ^2 feature selection. We use two real-world datasets, ALYA and GADGET [97], to demonstrate the advantage of the proposed data-driven compensation circuit design. We randomly select 10000 samples from ALYA and all of the 601 samples from GADGET to generate input-error patterns. All the circuits including the PAC and the final approximate circuits in our experiments are synthesized into netlists using a 32nm technology node. The features are obtained from the Value Change Dump (VCD) file, which is generated by Synopsys Verilog Compiler and Simulator (VCS). The VCD file is an ASCII-based format used to store circuit signal transition information. Fig. 4.5 is a screenshot of the signal transition information we extracted from the VCD file for the 4-tap FIR filter netlist. The rows indicates time steps while columns indicate the features (wires). As it can be seen, this table record 1142 features with a total of 5 samples (transitions). 1.0 and 0.0 indicates the

	n928	n1042	n932	x1[4]	n732	x3[8]	n631	h0[2]	n1043	h2[6]	...	n507	n506	n521	n520	n519	n518	n517	n516	n515	n514
0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0
40000	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
80000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1.0	NaN	0.0	1.0	1.0	NaN	1.0	NaN	1.0	1.0
120000	NaN	NaN	NaN	1.0	NaN	1.0	NaN	NaN	NaN	NaN	...	NaN	NaN	1.0	0.0	0.0	NaN	1.0	NaN	1.0	NaN
160000	NaN	NaN	NaN	0.0	0.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 1142 columns

Figure 4.5. Signal transition information from VCD file

	n928	n1042	n932	x1[4]	n732	x3[8]	n631	h0[2]	n1043	h2[6]	...	n507	n506	n521	n520	n519	n518	n517	n516	n515	n514
0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0
40000	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	...	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
80000	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	...	1.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0
120000	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0
160000	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0

5 rows × 1142 columns

Figure 4.6. Extracted signal state information from VCD file

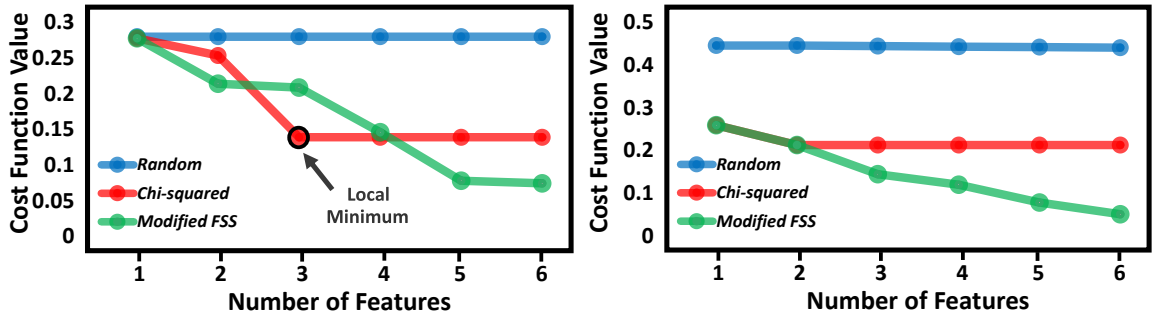
signal transit to 1.0 (vdd) and 0.0 (gnd), respectively. From this table, we can extract signal state information shown in Fig. 4.6. Mean absolute error (MAE) and error rate (ER) are used as the error metrics to evaluate the performance of the generated approximate logic circuits.

4.3.2 Comparison between the Modified FSS and χ^2 Feature Selection

We first compare the performance on the feature subset selection between the proposed modified FSS and χ^2 used in [45]. Fig. 4.7 shows the cost function trends for the 3rd and 2nd LSBs during the design of a 12-bit approximate multiplier. As we expected, both the proposed modified FSS and χ^2 feature selection achieve better performance than randomly selecting features. However, due to the greedy nature of the χ^2 feature selection, it might converged to a bad local minima, as the highlighted point in Fig. 4.7(b). In contrast, the proposed modified FSS could escape these local minima to achieve lower cost functions by gradually eliminating these features from the candidate pool.

4.3.3 Approximate Fixed-width Multiplier Design

To illustrate the effectiveness as well as for the comparison purpose, we apply the proposed methodology to compensate truncated fixed-width radix-4 booth multipliers under various settings based on 3 input data: ALYA, GADGET and UNIFORM (uniformly distributed inputs). UNIFORM is used to build reference



(a) (b)

Figure 4.7. Comparison between the modified FSS and χ^2 feature selection on a 12-bit truncated fixed-width multiplier: (a) the 3rd LSB, (b) the 2nd LSB.

circuits, which correspond to the data-independent compensation circuits. Since different input data result in different error distributions, the number of compensation bits is determined by the error dynamic range and then we follow the algorithm described in Section 4.2 to generate the final approximate logic circuits.

The performances on different input data are summarized in Table 4.2, where DD-ALYA represents the approximate multiplier generated from the ALYA dataset. The corresponding hardware costs are shown in Fig. 4.8. As it can be seen that by using the proposed method, the error can be significantly reduced from the PAC while only incurring very small hardware overheads. If we constrain the candidate feature pool as in designing DD-AYLA to the internal nodes that are close to the primary inputs, the delay of the compensated circuit would not increase. However, if we want to find the optimal feature subset by including all the internal nodes in the candidate feature pool, the delay might be increased slightly (e.g., DD-GADGET and DD-UNIFORM) when the selected features are close to the output of the circuit. In this case, the propagation delay of the compensation block added into the path from the primary inputs to the selected internal nodes might exceed the critical path of the PAC. Therefore, for applications that delay is the primary concern, we should limit the candidate feature pool to only internal nodes that close to the primary inputs.

All of the approximate logic circuits designed using the proposed data-driven methodology show superior performance in terms of accuracy when tested on their own data, which is indicated by the bold numbers in Table 4.2. It can also be observed that the approximate circuits designed without considering data patterns are sub-optimal in applications that have specific input distributions. For example, both ALYA and GARGET datasets yield larger errors on the 10-bit DD-UNIFORM approximate multiplier than the uniformly distributed data. Thus, it can be concluded from these results that it is important to consider the input data distribution in designing approximate logic circuits for data-driven tasks.

Table 4.2. Comparison of the approximate multipliers on different input data: MAE (ER%)

(a) ALYA

	Truncate Circuits	**DD-ALYA Circuits**	DD-GADGET Circuits	DD-UNIFORM Circuits
10-bit	0.9114 (84.94%)	**0.4613 (39.93%)**	0.5452 (48.32%)	1.4636 (70.55%)
12-bit	0.9242 (85.46%)	**0.4641 (39.63%)**	1.0924 (94.05%)	0.548 (48.04%)
16-bit	0.9242 (86.27%)	**0.4623 (40.08%)**	1.3041 (93.91%)	1.4492 (70.43%)

(b) GADGET

	Truncate Circuits	DD-ALYA Circuits	**DD-GADGET Circuits**	DD-UNIFORM Circuits
10-bit	1.4792 (96.38%)	1.444 (94.84%)	**0.6772 (49.75%)**	1.0183 (70.38%)
12-bit	1.6855 (96.33%)	1.7154 (96.83%)	**0.8136 (56.57%)**	0.9617 (76.04%)
16-bit	2.7704 (99.8%)	2.7221 (98.66%)	**0.9417 (62.56%)**	1.0881 (72.21%)

(c) UNIFORM

	Truncate Circuits	DD-ALYA Circuits	DD-GADGET Circuits	**DD-UNIFORM Circuits**
10-bit	1.8653 (97.53%)	1.8586 (93.72%)	2.4257 (84.21%)	**1.0525 (65.38%)**
12-bit	2.2464 (99.03%)	2.2474 (98.05%)	1.8789 (83.77%)	**1.0907 (68.72%)**
16-bit	2.9999 (99.8%)	2.9836 (99.36%)	1.6411 (77.66%)	**1.1342 (70.84%)**

Table 4.3. Performance of further compensating the approximate multiplier design in [26]

	Original Circuit	Truncate Circuit	[26] Circuit	**DD-GADGET Circuit**
Area	1221.01	612.28	792.68	**881.40**
Power	193.49	88.74	120.67	**135.13**
Delay	2.62	1.95	2.26	**2.39**
MAE (ER%)	0 (0%)	1.4792 (96.38%)	0.2379 (23.79%)	**0.1697 (16.97%)**

4.3.3.1 Further Compensation on the Existing Approximate Multipliers

Since only the input-error patterns and the netlist are required, the proposed methodology is also capable of further compensating the existing approximate logic circuit to improve the error. In this experiment, we employ the proposed method on a manually designed 10-bit approximate multiplier [26] which has achieved a low error rate already. We use GADGET as the input data. The performance is presented in Table 4.3. It can be seen that based on the design in [26]. As it can be seen in this table, the manually compensated [26] circuit already achieves a good performance on reducing error. However, with the proposed method, we can further compensate [26] circuit with low hardware overhead and reduce both MAE and ER by 29%.

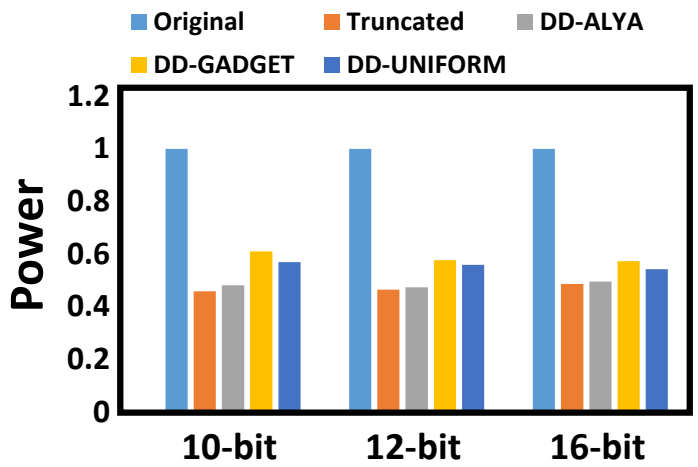
Table 4.4. Compensation performance on an approximate 4-tap FIR filter

	Original FIR	[26] Multiplier	[26] FIR	DD-GADGET FIR
Area	5074.94	792.68	3640.77	**3906.90**
Power	902.075	120.67	682.69	**737.89**
MAE (ER%)	0 (0%)	0.2379 (23.79%)	0.4732 (39.96%)	**0.2558 (20.90%)**

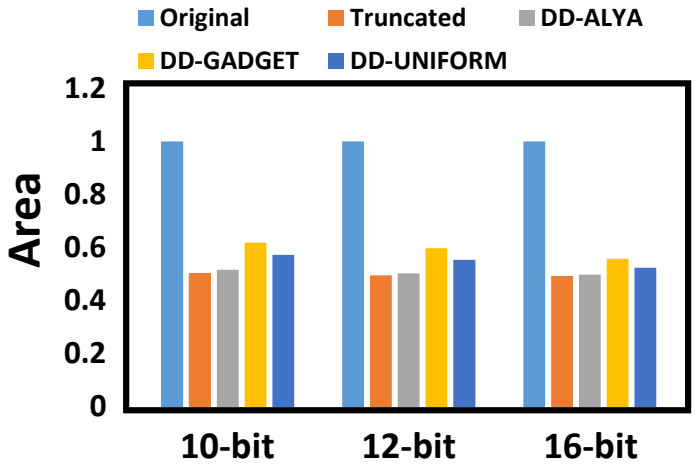
4.3.4 Approximate FIR Filter

In this experiment, we demonstrate the effectiveness of the proposed method on designing large and complex approximate circuits, which is one advantage of the proposed data-driven methodology compared to manual analysis. Alternatively, if we construct a large circuit with approximate arithmetic elements, the final output may accumulate to an intolerable error magnitude. As an example for demonstration, we build an approximate 10-bit 4-tap finite impulse response (FIR) filter circuit whose multipliers are replaced by the manually designed approximate multiplier in [26]. For a single 10-bit approximate multiplier, the MAE and ER are 0.2379 and 23.79%, respectively, as shown in Table 4.4. However, the MAE and ER of the corresponding approximate 4-tap FIR filter are aggravated to 0.4732 and 39.96%, respectively, which indeed proves that the straightforward implementation of building large approximate circuits with approximate arithmetic elements may not achieve a satisfied performance.

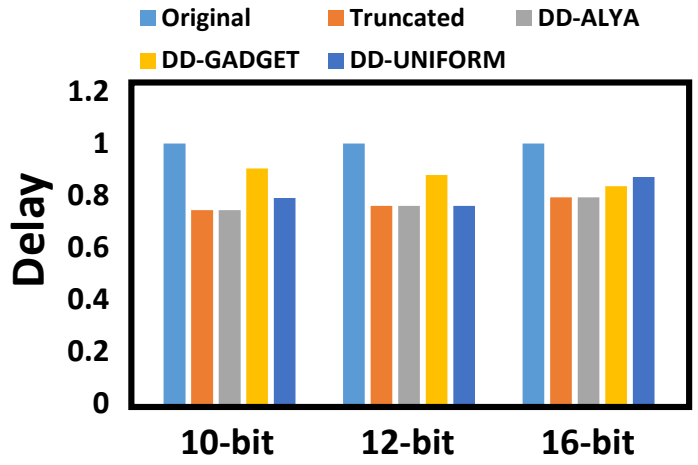
A significantly better performance can be achieved by using the proposed data-driven method to design the approximate FIR filter. As presented in Table 4.4, our method leads to a design with about a 50% error reduction and only a 8% hardware overhead. In addition, the design workload is also greatly reduced, compared to manual design. The synthesized correction circuit is shown in Fig. 4.9. As it can be seen, such complicated circuit will be very hard to generated via manual design strategies.



(a)



(b)



(c)

Figure 4.8. Hardware consumption comparison between different compensated fixed-width multipliers (normalized to the original circuits)

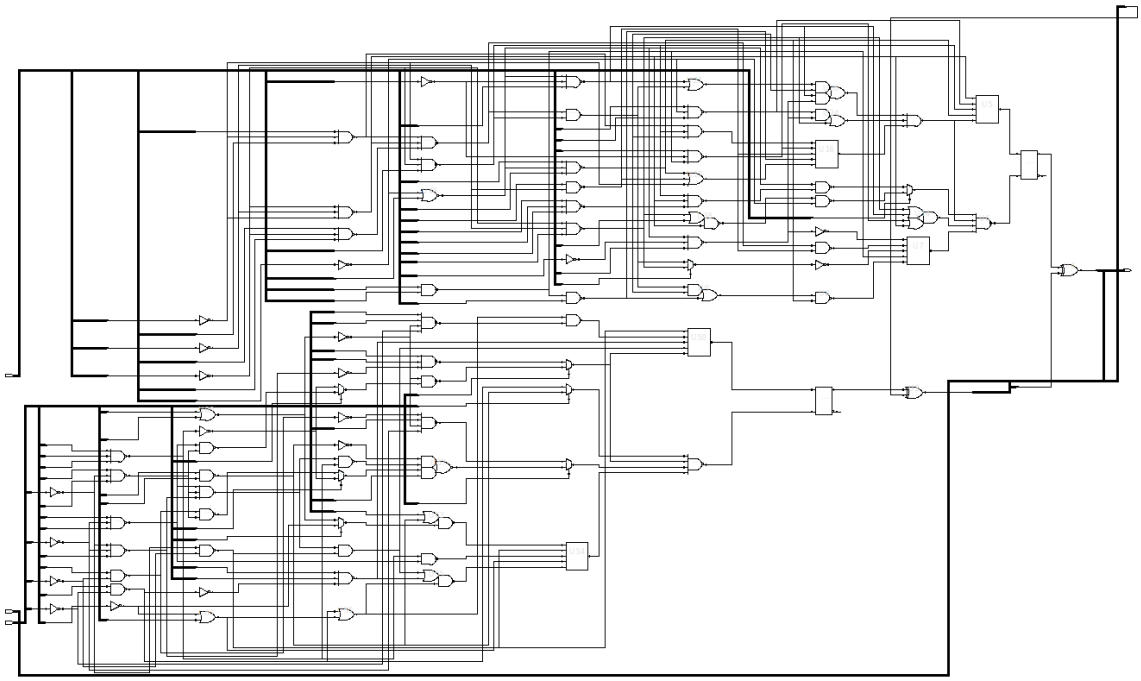


Figure 4.9. Synthesized correction circuit for approximating the FIR in [26]

Chapter 5

Conclusion and Future Work

This dissertation is motivated by the challenge of the recent significantly increased demand in computational intensive applications and their requirement on low power consumption. Our work focuses on a promising design paradigm, approximate computing to solve this obstacle. As apposed to most prior works that focus on designing approximate versions of fundamental computing elements, such as multipliers and adders, using manual design strategies, our work aims to introducing systematic and salable methodologies to design system-level approximate architectures using machine learning approaches.

In this dissertation, we proposed two novel approaches based on feature selection techniques to design approximate circuits in a systematic and data-driven manner. We demonstrated that our proposed approaches are extremely feasible and suitable for efficiently designing large scale approximate circuits. In particular, our approaches are able to mitigate approximate circuit's error for a wide range of circuits without including much design complexity. In Chapter 3, we presented a detailed framework of using input-error pattern to design compensation circuits. Our case study on designing fixed-width multiplier showed comparative results with other designs in the literature. In Chapter 4, we improved the previously proposed framework by incorporating data-driven design ideas, reducing compensation timing overhead and introducing a more accurate feature selection technique. We demonstrated that this approach can be implemented on a broad types of circuits including manually designed approximate multipliers and large approximate circuits.

Future research will focus on three tasks. The first task is to refine the details of the proposed approaches. For example, we can incorporate error and hardware overhead constraints into the design flow so that the target circuits can derive towards approximation in a constrained manner. The second task involves developing new schemes to simplify original circuits into approximate versions while providing the best

compensation opportunity based on our current work. The third task is to link and optimize our proposed methods to VOS and wire pruning techniques. In other words, our current work can be adjusted to fit better to these two techniques or other emerging approximation logic design techniques.

Bibliography

[1] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[2] K. Kumawat. 7nm vs 10nm vs 14nm: Fabrication technology explained, Jan 2019. [Online]. Available: https://www.techcenturion.com/7nm-10nm-14nm-fabrication.

[3] D. Bohn. This is the samsung galaxy s8, coming april 21st, Mar 2017. [Online]. Available: https://www.theverge.com/2017/3/29/15087530/samsung-galaxy-s8-announced-features-release-date-video-specifications.

[4] S. Stephen. Apple's a12 bionic cpu for the new iphone xs is ahead of the industry moving to 7nm chip manufacturing tech, Sep 2018. [Online]. Available: https://www.cnet.com/news/iphone-xs-a12-bionic-chip-is-industry-first-7nm-cpu/.

[5] R. Hof. Deep learning. [Online]. Available: https://www.technologyreview.com/s/513696/deep-learning/.

[6] A. Beam. Deep learning 101 - part 1: History and background, Feb 2017. [Online]. Available: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html.

[7] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, et al. Axilog: Language support for approximate hardware design. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 812–817. EDA Consortium, 2015.

[8] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *2011 24th Internatioal Conference on VLSI Design*, pages 346–351, Jan 2011.

[9] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi. Approximate xor/xnor-based adders for inexact computing. In *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*, pages 690–693, Aug 2013.

[10] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[11] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, April 2010.

[12] H. Jiang, J. Han, and F. Lombardi. A comparative review and evaluation of approximate adders. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 343–348. ACM, 2015.

[13] 0. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram. Rap-cla: A reconfigurable approximate carry look-ahead adder. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(8):1089–1093, 2018.

[14] J. Hu and W. Qian. A new approximate adder with low relative error and correct sign calculation. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1449–1454. EDA Consortium, 2015.

[15] A. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th Annual Design Automation Conference*, pages 820–825. ACM, 2012.

[16] C. Liu, J. Han, and F. Lombardi. An analytical framework for evaluating the error characteristics of approximate adders. *IEEE Transactions on Computers*, 64(5):1268–1281, 2015.

[17] V. Camus, J. Schlachter, and C. Enz. A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. Ieee, 2016.

[18] Z. Yang, J. Han, and F. Lombardi. Transmission gate-based approximate adders for inexact computing. In *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH´15)*, pages 145–150. IEEE, 2015.

[19] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A low latency generic accuracy configurable adder. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[20] M. Hanif, R. Hafiz, O. Hasan, and M. Shafique. Quad: Design and analysis of quality-area optimal low-latency approximate adders. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 42. ACM, 2017.

[21] M. Ayub, O. Hasan, and M. Shafique. Statistical error analysis for low power approximate adders. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 75. ACM, 2017.

[22] A. Becher, J. Echavarria, D. Ziener, S. Wildermann, and J. Teich. A lut-based approximate adder. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 27–27. IEEE, 2016.

[23] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel. Probabilistic error modeling for approximate adders. *IEEE Transactions on Computers*, 66(3):515–530, 2017.

[24] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi. Design of approximate radix-4 booth multipliers for error-tolerant computing. *IEEE Transactions on Computers*, 66(8):1435–1441, Aug 2017.

[25] K. Cho, K.l Lee, J. Chung, and K. K. Parhi. Design of low-error fixed-width modified booth multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(5):522–531, May 2004.

[26] Z. Zhang and Y. He. A low-error energy-efficient fixed-width booth multiplier with sign-digit-based conditional probability estimation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(2):236–240, Feb 2018.

[27] W. He, Y. Chen, and S. Jou. High-accuracy fixed-width booth multipliers based on probability and simulation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(8):2052–2061, 2015.

[28] Y. H. Chen and T. Y. Chang. A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(3):594–603, March 2012.

[29] E. J. King and E. E. Swartzlander. Data-dependent truncation scheme for parallel multipliers. In *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No.97CB36136)*, volume 2, pages 1178–1182 vol.2, Nov 1997.

[30] M. Song, L. Van, and S. Kuon. Adaptive low-error fixed-width booth multipliers. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 90(6):1180–1187, 2007.

[31] G. Zervakis, S. Xydis, K. Tsoumanis, D. Soudris, and K. Pekmestzi. Hybrid approximate multiplier architectures for improved power-accuracy trade-offs. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 79–84, July 2015.

[32] W. He, Y. Chen, and S. Jou. High-accuracy fixed-width booth multipliers based on probability and simulation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(8):2052–2061, Aug 2015.

[33] S. S. Kidambi, F. El-Guibaly, and A. Antoniou. Area-efficient multipliers for digital signal processing applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(2):90–95, Feb 1996.

[34] S. Hashemi, R 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*, pages 418–425. IEEE Press, 2015.

[35] Cong Liu, Jie Han, and Fabrizio Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 95:1–95:4, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association.

[36] S. Narayanamoorthy, and Z. Liu H. Moghaddam, T. Park, and N. Kim. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(6):1180–1184, 2015.

[37] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel. Architectural-space exploration of approximate multipliers. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 80. ACM, 2016.

[38] Z.Yang, J. Han, and F. Lombardi. Approximate compressors for error-resilient multiplier design. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 183–186. IEEE, 2015.

[39] G. Zervakis, K. Tsoumanis, S. Xydis, N. Axelos, and K. Pekmestzi. Approximate multiplier architectures through partial product perforation: Power-area tradeoffs analysis. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 229–232. ACM, 2015.

[40] D Duttweiler. Adaptive filter performance with nonlinearities in the correlation multiplier. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30(4):578–586, 1982.

[41] B. Shao and P. Li. Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(4):1081–1090, 2015.

[42] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power in a multiplier architecture. *Journal of Low Power Electronics*, 7(4):490–501, 2011.

[43] T. Yang, T. Ukezono, and T. Sato. Low-power and high-speed approximate multiplier design with a tree compressor. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 89–96. IEEE, 2017.

[44] B. Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[45] L. Qiu and Y. Lao. A systematic method for approximate circuit design using feature selection. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2018.

[46] Y. Wang, S. Wenger, J. Wen, and A. Katsaggelos. Error resilient video coding techniques. *IEEE Signal Processing Magazine*, 17(4):61–82, 2000.

[47] R. Talluri. Error-resilient video coding in the iso mpeg-4 standard. *IEEE Communications Magazine*, 36(6):112–119, 1998.

[48] C. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[49] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[50] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[51] X. Chu, I. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2201–2206. ACM, 2016.

[52] I. Ilyas, X. Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

[53] E. Rahm and H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[54] T. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[55] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212, 2003.

[56] Q. Xu, T. Mytkowicz, and N. Kim. Approximate computing: A survey. *IEEE Design & Test*, 33(1):8–22, 2016.

[57] S. Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):62, 2016.

[58] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 92–101. ACM, 2014.

[59] R. Hegde and N. R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No.99TH8477)*, pages 30–35, Aug 1999.

[60] M. Alioto. Ultra-low power vlsi circuit design demystified and explained: A tutorial. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(1):3–29, 2012.

[61] B. Shim and N. Shanbhag. Energy-efficient soft error-tolerant digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(4):336–348, 2006.

[62] Y. Liu, T. Zhang, and Parhi K.K. Computation error analysis in digital signal processing systems with overscaled supply voltage. *IEEE transactions on very large scale integration (VLSI) systems*, 18(4):517–526, 2010.

[63] and S. Das P. Whatmough, D. Bull, and I. Darwazeh. Circuit-level timing error tolerance for low-power dsp filters and transforms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(6):989–999, 2013.

[64] Y. Liu, T. Zhang, and J. Hu. Low power trellis decoder with overscaled supply voltage. In *2006 IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 205–208. IEEE, 2006.

[65] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *2011 Design, Automation Test in Europe*, pages 1–6, March 2011.

[66] K. He, A. Gerstlauer, and M. Orshansky. Circuit-level timing-error acceptance for design of energy-efficient dct/idct-based systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(6):961–974, June 2013.

[67] J. Lin, Y. Hwang, M. Sheu, and C. Ho. A novel high-speed and energy efficient 10-transistor full adder design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(5):1050–1059, May 2007.

[68] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 667–673. IEEE, 2011.

[69] J. Miao, A. Gerstlauer, and M. Orshansky. Multi-level approximate logic synthesis under general error constraints. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 504–510, Nov 2014.

[70] D. Shin and S. Gupta. Approximate logic synthesis for error tolerant applications. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 957–960, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[71] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: systematic logic synthesis of approximate circuits. In *Proceedings of the 49th Annual Design Automation Conference*, pages 796–801. ACM, 2012.

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

[73] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan. Aslan: Synthesis of approximate sequential circuits. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 364. European Design and Automation Association, 2014.

[74] V. Camus, J. Schlachter, and C. Enz. Energy-efficient digital design through inexact and approximate arithmetic circuits. In *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pages 1–4. IEEE, 2015.

[75] J. Schlachter, V. Camus, C. Enz, and K. Palem. Automatic generation of inexact digital circuits by gate-level pruning. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 173–176. IEEE, 2015.

[76] J. Schlachter, V. Camus, C. Enz, and K. V. Palem. Automatic generation of inexact digital circuits by gate-level pruning. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 173–176, May 2015.

[77] Z. Zhang, Y. He, J. He, X. Yi, Q. Li, and B. Zhang. Optimal slope ranking: An approximate computing approach for circuit pruning. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2018.

[78] A. Lingamneni, C. Enz, K. Palem, and C. Piguet. Synthesizing parsimonious inexact circuits through probabilistic design techniques. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2s):93, 2013.

[79] J. Schlachter, V. Camus, and C. Enz. Design of energy-efficient discrete cosine transform using pruned arithmetic circuits. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 341–344. IEEE, 2016.

[80] D. Koller and M. Sahami. Toward optimal feature selection. Technical report, Stanford InfoLab, 1996.

[81] A. Jain and D. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE transactions on pattern analysis and machine intelligence*, 19(2):153–158, 1997.

[82] G. James, D. Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[83] Z.Hira and D. Gillies. A review of feature selection and feature extraction methods applied on microarray data. *Advances in bioinformatics*, 2015, 2015.

[84] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863, 2003.

[85] H. Liu, R. Setiono, et al. A probabilistic approach to feature selection-a filter solution. In *ICML*, volume 96, pages 319–327. Citeseer, 1996.

[86] M. Hall and L. Smith. Feature selection for machine learning: comparing a correlation-based filter approach to the wrapper. In *FLAIRS conference*, volume 1999, pages 235–239, 1999.

[87] Z. Zhu, Y. Ong, and M. Dash. Wrapper–filter feature selection algorithm using a memetic framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):70–76, 2007.

[88] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8):1226–1238, 2005.

[89] M. Kabir, M. Islam, and K. Murase. A new wrapper feature selection approach using neural network. *Neurocomputing*, 73(16-18):3273–3283, 2010.

[90] S. Wang, J. Tang, and H. Liu. Embedded unsupervised feature selection. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[91] Y. Saeys, I. Inza, and P. Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[92] M. McHugh. The chi-square test of independence. *Biochemia medica: Biochemia medica*, 23(2):143–149, 2013.

[93] H. Liu, R. Setiono, et al. A probabilistic approach to feature selection-a filter solution. In *ICML*, volume 96, pages 319–327, 1996.

[94] M. McShane, B. Cameron, G. Coté, M. Motamedi, and C. Spiegelman. A novel peak-hopping stepwise feature selection method with application to raman spectroscopy. *Analytica chimica acta*, 388(3):251–264, 1999.

[95] B. Lars, L. Gilles, B. Mathieu, P. Fabian, M. Andreas, G. Olivier, N. Vlad, P. Peter, G. Alexandre, G. Jaques, L. Robert, V. Jake, J. Arnaud, H. Brian, and V. Gaël. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[96] J. P. Wang, S. R. Kuang, and S. C. Liang. High-accuracy fixed-width modified booth multipliers for lossy applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(1):52–60, Jan 2011.

[97] UEABS. Unified european applications benmark suite, 2013. `http://www.prace-ri.eu/ueabs/`.