The Dissertation Committee for Jin Miao
certifies that this is the approved version of the following dissertation:

# Modeling and Synthesis of Approximate Digital Circuits

Committee:

Michael Orshansky, Supervisor

Andreas Gerstlauer, Co-Supervisor

Adnan Aziz

Robert van de Geijn

Earl Swartzlander

# Modeling and Synthesis of Approximate Digital Circuits

**by**

**Jin Miao, B.E., M.S.E.**

**DISSERTATION**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**DOCTOR OF PHILOSOPHY**

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2014

To my beloved family

# Acknowledgments

I would like to express my deepest gratitude to my advisors Professor Orshansky and Professor Gerstlauer for their continuous support and valuable guidance over the years. They spent a lot of time with me and helped me tremendously in my progress. I sincerely appreciate their great patience in correcting my mistakes. I also admire their ambitious and rigorous academic spirit which will continue benefiting me in every aspect of my research and life.

I would like to thank all other committee members for their helpful comments and suggestions on my research and this dissertation specifically. In particular, I would like to thank Professor Aziz for many valuable discussions of my research and career development. I would like to thank Professor Swartzlander and Professor Geijn for their kindness and useful suggestions on developing this dissertation.

It is my great luck to meet so many talented friends at UT-Austin over the years. I would like to express my sincere thanks to all of them, including Bei Yu, Ku He, Boyang Zhang, Song Zhang, Qiaoyang Ye, Ye Wang, Xiaoqing Xu, Meng Li, Zhuoran Zhao, Xinnian Zheng, Mukund Kalyanaraman, Jaeyoung Park, Jiwoo Pak, Kareem Ragab, Seogoo Lee, and many others. I sincerely thank them for all their help and support for my research and life.

The last but not the least, I would like to deeply thank my wife, Dorothy Hu, for her love, for her consistent encouragement. I also would love to give my heartfelt gratitude to my parents and my beloved grandmother, for their deepest love and steadiest support.

# Modeling and Synthesis of Approximate Digital Circuits

Jin Miao, Ph.D.
The University of Texas at Austin, 2014

Supervisors: Michael Orshansky
Andreas Gerstlauer

Energy minimization has become an ever more important concern in the design of very large scale integrated circuits (VLSI). In recent years, approximate computing, which is based on the idea of trading off computational accuracy for improved energy efficiency, has attracted significant attention. Applications that are both compute-intensive and error-tolerant are most suitable to adopt approximation strategies. This includes digital signal processing, data mining, machine learning or search algorithms. Such approximations can be achieved at several design levels, ranging from software, algorithm and architecture, down to logic or transistor levels. This dissertation investigates two research threads for the derivation of approximate digital circuits at the logic level: 1) modeling and synthesis of fundamental arithmetic building blocks; 2) automated techniques for synthesizing arbitrary approximate logic circuits under general error specifications.

The first thread investigates elementary arithmetic blocks, such as adders and multipliers, which are at the core of all data processing and often consume most of the energy in a circuit. An optimal strategy is developed to reduce energy consumption in timing-starved adders under voltage over-scaling. This allows a formal demonstration that, under quadratic error measures prevalent in signal processing applications, an adder design strategy that separates the most significant bits (MSBs) from the least significant bits (LSBs) is optimal. An optimal conditional bounding (CB) logic is further proposed for the LSBs, which selectively compensates for the occurrence of errors in the MSB part. There is a rich design space of optimal adders defined by different CB solutions.

The other thread considers the problem of approximate logic synthesis (ALS) in two-level form. ALS is concerned with formally synthesizing a minimum-cost approximate *Boolean function*, whose behavior deviates from a specified exact Boolean function in a well-constrained manner. It is established that the ALS problem un-constrained by the frequency of errors is isomorphic to a Boolean relation (BR) minimization problem, and hence can be efficiently solved by existing BR minimizers. An efficient heuristic is further developed which iteratively refines the magnitude-constrained solution to arrive at a two-level representation also satisfying error frequency constraints. To extend the two-level solution into an approach for multi-level approximate logic synthesis (MALS), Boolean network simplifications allowed by external don't cares (EXDCs) are used. The key contribution is in finding non-trivial EXDCs that

can maximally approach the external BR and, when applied to the Boolean network, solve the MALS problem constrained by magnitude only. The algorithm then ensures compliance to error frequency constraints by recovering the correct outputs on the sought number of error-producing inputs while aiming to minimize the network cost increase.

Experiments have demonstrated the effectiveness of the proposed techniques in deriving approximate circuits. The approximate adders can save up to 60% energy compared to exact adders for a reasonable accuracy. When used in larger systems implementing image-processing algorithms, energy savings of 40% are possible. The logic synthesis approaches generally can produce approximate Boolean functions or networks with complexity reductions ranging from 30% to 50% under small error constraints.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The explosive growth of portable and wearable computing devices, such as smart phones, tablets, smart watches, and many other personal multi-media or communication devices all demand a long battery life while support increasingly complex functionalities. At the same time, battery technology development falls far behind the growth of the power consumptions in VLSI systems. Minimizing power consumption has therefore become the major concern in the design of VLSI systems.

The power consumption of VLSI circuits can be split into static and dynamic components, where the former is primarily driven by leakage currents, which relates to the area and density of the chip, while the latter is a function of the circuit's switching activity. Importantly, the supply voltage level is a primary contributing factor for both static and dynamic power.

There have been many research efforts over the years to minimize VLSI power consumption, e.g., scaling supply voltages [1, 2], reducing unnecessary circuit activity through clock-gating and power-gating [3–5], minimizing chip area through synthesis optimizations [6], or scaling down transistor sizes and developing novel power-efficient transistors [7, 8].

All of these conventional low power approaches are based on guaranteeing the correct functionality of a chip. Importantly, however, in many applications 100% correct functionality is not necessary, and applications that naturally tolerate errors do not necessarily need high accuracy. For example, results of data mining and machine learning algorithms are intrinsically in probabilistic form and hence can tolerate a certain level of inaccuracies. Similarly, audio or video processing systems tolerate errors due to the insensitivities of human perception to small or high frequency variations. Furthermore, digital signal processing systems naturally have noise floors due to finite precision and necessary quantization in any computing system. This provides an additional design dimension to lower VLSI power consumption: trading off computation accuracy for improved power efficiency. The key challenge lies in how to best take advantage of small allowed errors in order to achieve significant power reductions.

## 1.1 Approximate Computing

Many recent approaches have studied the possibility of inaccurate or approximate computing at different levels ranging from softwares [9], algorithms [10, 11] and architectures [12–17] to the logic or transistor levels [18–20].

At the software level, a novel approximate programming model was proposed in [9], where programmers are granted the flexibilities to specify approximate type variables. Systems can automatically map these approximate

types to customized low-power approximate hardware components, e.g., low power approximate computation units or approximate storage elements that operate under ultra low supply voltage. The proposed model guarantees the isolation between the approximate computing and exact computing via static type checking mechanisms.

At the algorithm level, in [10], algorithms and systems are designed or transformed based on the idea of "incremental refinement," where less important computing steps or iterations can be discarded for energy improvement while maintaining an acceptable quality level. Similar techniques are also used in [11]. By taking advantage of DCT algorithm properties, some unnecessary or less contributing steps are skipped.

At the architectures level, in [12], a novel ISA was proposed to support approximate programming, which exposes hardware flexibilities for energy savings at the cost of accuracy losses to the programmer. A dual-voltage microarchitecture called Truffle is also proposed for demonstrating the effectiveness of the new ISA. In [13], a novel approximate vector processor ISA and microarchitecture named QUORA are proposed. This vector processor contains 289 processing elements, which are divided into three categories: approximate, mixed-accuracy, and completely accurate elements. Depending on the user-specified accuracy level, operations are decomposed and dispatched to appropriate processing units to meet an overall computing accuracy while reducing the energy consumption. In [14], a fuzzy instruction memoization technique is proposed in the context of multimedia floating-point computation. Instruction

memoization is a technique to memorize each historical input-output pair and hence saves future re-computations if the same input arises. Taking advantage of the error-tolerant characteristics of multimedia processing applications, the authors associate *similar* inputs to the same output to further reduce the energy consumption.

In the domain of custom architectures for signal processing applications, the work in [15] employs an error-correction scheme. The system consists of a main computing block running at a voltage lower than the critical level and a much simpler error-correcting block running at a higher voltage, where the latter block selectively corrects errors in order to guarantee an overall acceptable output quality. In [16], low energy consumption is achieved via an automatic adaptive precision control mechanism on the arithmetic units. Finally, in [17], the control and data flow graph of operations in an image processing application is re-ordered during hardware synthesis. With this, input patterns that lead to long carry propagations in adders are significantly reduced, while those patterns that still require long chains are postponed to the last computation stage in which a longer delay is budgeted via re-timing and re-scheduling techniques. The overall energy reduction comes from an over-scaled supply voltage driving the entire system.

At the logic level, a general scheme, which is by redesigning the logic to reduce complexity and allow for voltage scaling, has been employed for deriving approximate logic units [18, 19, 21–26]. Finally, at the transistor level, optimizations of approximate full adder cells are proposed in [20]. With

reduced area complexity, those approximate full adder cells are used to build low-power multi-bit approximate adder designs.

## 1.2 Dissertation Overview

This dissertation presents two research threads for deriving approximate digital circuits at the logic level. The first thread addresses the modeling and synthesis of elementary arithmetic blocks. The second thread discusses systematic approaches to automatically synthesizing approximate digital circuits compliant to properly specified error constraints. The following subsections give brief introductions on these two research threads, including comparisons to related work.

### 1.2.1 Approximate Arithmetic

Arithmetic units, i.e., adders and multipliers, are the most fundamental building blocks for all computing. Furthermore, such blocks often consume the majority of energy in a circuit. As such, improving energy efficiency for arithmetic blocks is crucial in low-power approximate computing. Adders thereby lie at the heart of all computer arithmetic. Almost all arithmetic blocks are built out of basic adders, e.g., sequential multipliers consist of an adder and a shifter, while modern tree multipliers require adders for both the partial product matrix reduction as well as the last stage two-operand summation. Therefore, this dissertation primarily investigates the modeling and synthesis of approximate adders [25], and will also demonstrate application

of such approximate adders to the design of approximate multipliers.

It is known that scaling supply voltages is one of the most effective approaches for lowering the energy consumption of VLSI circuits. This approach, however, increases gate delays and may cause timing violations or timing starvation of the circuits. The approximate adder research starts by studying conventional exact adders operating under timing violations, where the assumption is that a reduced timing budget is due to the over-scaling of supply voltages. In adders, the critical path is defined by the carry propagation chain. As such, errors in a timing-starved adder are due to the failure of carry propagations. In adders under starvation, some output sum-bits become timing-inaccessible for a primary carry-in. The typical carry chain length is, however, much smaller than the maximum one. In fact, the likelihood of a long carry-chain is quite small. For an $N$-bit adder, the expected worst-case carry length is close to $logN$ [27–29], and with extremely high probability is less than $logN + 12$ [22]. The lowest likelihood of errors is thus ensured even under large timing starvation if the sum-bits are each allowed to have their maximum possible carry-chain. This statistical feature has been employed in designing approximate adders. The work in [30] proposes approximate adders that are designed to have simplified logic structures, where the carry-chains have been significantly reduced. Though with a low probability of producing erroneous results, possible errors can have very large magnitude. A similar concept is also used in deriving fast but inaccurate speculative adders [22]. In addition to the speculative adders, the authors further construct a reliable ver-

sion of such adders that can detect and correct errors when they occur. Due to the low probability of error occurrence, overhead for error correction is small. There are other related efforts for approximate adder designs as well. In [24], an accuracy configurable pipelined approximate adder is proposed in which the accuracy of the result is configurable at runtime. The number of pipeline stages can thereby be configured to tradeoff accuracy versus performance or energy. When granted the full number of pipeline stages, the proposed approximate adder can produce error-free results. In [23], a carry-prediction mechanism is employed to improve the bit error rate of the proposed approximate adder, which is also equipped with dedicated logic blocks to reduce the magnitude of possible errors. In [18], various supply voltage levels are applied to an overclocked ripple carry adder, where overall errors are minimized by optimally assigning candidate voltage levels to the logic gates while satisfying an overall energy constraint. In [19, 21], approximate adders are designed with bounded error magnitude at the cost of area overhead. However, all of the above approximate adders were designed in an ad-hoc manner. Before the time of this dissertation, there was no formal answer to the question: given a fixed amount of timing starvation, what is the optimal design strategy for approximate adders, specifically, in the context of signal processing applications? This dissertation develops a formal analysis for designing such energy-optimal timing-starved approximate adders.

The following contributions are made in the modeling and synthesis of optimal approximate adders: (1) a formal model to identify a fundamental

error frequency-magnitude tradeoff and to prove that under a quadratic error measure in signal-processing applications, limiting error magnitude rather than error frequency is the energy-optimal addition strategy; (2) a timing-starvation model demonstrating that an optimal approximate adder reduces carry chains for a large fraction of sub-bits to a length significantly below what is allowed by the timing budget using an aligned, fixed internal-carry (AFIC) structure for higher significance bits; (3) a formal analysis concluding that a conditional bounding (CB) logic is the optimal structure for realization of lower significance bits in conjunction with an AFIC adder for higher significance ones; (4) a set of models and algorithms to efficiently find Pareto-optimal realizations of inexact CB logic; and (5) a dithering approximate adder that mixes under- and overestimating CB logic to produce a reduced-variance zero-centered error distribution. Finally, the application of such adders to various design examples and as building blocks for approximate multiplier designs are demonstrated.

### 1.2.2 Approximate Logic Synthesis

Approximate logic synthesis (ALS) seeks to find inexact realizations of Boolean functionality or Boolean networks result in logic implementations of reduced complexity, smaller area, delay, and energy. ALS is accomplished by modifying some of the outputs of a function's truth table or using don't cares (DC) to simplify a Boolean network, all while ensuring that the produced error is within given constraints. ALS is driven by the need to automatically derive

approximate circuits for different functionalities. Designing approximate circuits in an ad-hoc manner is not a viable option as there exists a large design space with trade-offs between acceptable accuracy and energy, where acceptable errors may vary from application to application and function to function. Importantly, depending on the application, error tolerance is primarily a function of either the frequency of errors, the magnitude of errors, or both. For example, in applications that can not directly accept erroneous results, the frequency of triggering correction mechanisms determines ultimate overhead. By contrast, in image and video processing applications, if the produced pixel values have small error magnitudes, a human will not be able to distinguish such subtle changes. For a high-quality Peak Signal-to-Noise Ratio (PSNR) value of 50dB at 8-bit pixel depth, this can, for example, be translated into a frequency constraint of 7% for error magnitudes smaller than 3, or up to 65% errors if pixel values are allowed to have error magnitudes of no more than 1. Thus, overall, there is a need for rigorous automation to perform approximate logic synthesis (ALS) under both types of constraints.

Existing ALS approaches thus far have focused on single error metrics only. A two-level approximate logic synthesis algorithm was introduced in [31]. In that work, the objective was to synthesize a minimized circuit under constrained error frequency. The algorithm did not consider constraints on error magnitude. Moreover, it suffers from high runtime complexity, especially, at large error frequencies. By contrast, in [25] and [32], absolute and relative error magnitude constraints were set without limiting error frequency. Both tech-

niques are built upon an unmodified conventional logic synthesis flow, which is not as efficient as integrating support for approximation into the logic synthesis engine directly. In [33, 34], the authors consider both error frequency and relative error metrics. However, distinct solutions are provided and the two constraints are never explored jointly. Furthermore, due to the nature of the proposed pattern-driven approach, the optimization space is restricted to only a small subset of inputs. In contrast, this research aims to address general ALS problems under arbitrary and simultaneous error magnitude and frequency constraints.

First, a two-level approximate logic synthesis problem is addressed, where the goal is to derive minimum-cost approximate Boolean functions in sum-of-product (SOP) form. An efficient algorithm is developed, which rigorously synthesizes a minimum literal-cost cover of a Boolean function that is allowed to deviate from an exact Boolean function in a constrained manner. This work adopts a two-phase approach. The first phase solves the problem that is constrained by magnitude of error only. In the second phase, this frequency unconstrained problem is iteratively refined to arrive at a solution that also satisfies the original error frequency constraint.

Two major contributions are made in solving the two-level ALS problem. The first contribution is the realization that the approximate synthesis problem un-constrained by the frequency of errors is isomorphic with the problem of minimizing a Boolean relation (BR), which is a generalization of a Boolean function. The error magnitude constraints can be formulated as

constraints on the possible values of Boolean function outputs and thus are equivalent to Boolean relations. This mapping allows to exploiting recently developed fast algorithms for BR problems to solve the error magnitude-only constrained ALS problem.

The second contribution is an efficient heuristic algorithm for iteratively refining the magnitude-constrained solution to arrive at a solution also satisfying the error frequency constraint. The algorithm (a) finds the optimal set of function minterms on which the exact outputs must be enforced, and (b) systematically corrects, in a greedy fashion, the erroneous outputs of the BR solution that lead to the smallest cost increase until the error frequency constraint is met.

Subsequently, the more comprehensive problem of multi-level approximate logic synthesis (MALS) is addressed. Although sharing a similar problem formulation as in two-level ALS, MALS assumes the existence of an optimized exact *Boolean network*, where the goal is to derive approximate networks of minimum gate cost whose logic function is within specified error deviations. Network simplifications allowed by external don't cares (EXDCs) are employed. The key contribution is in finding non-trivial EXDCs that can maximally approach the external BR. Error frequency-unconstrained MALS is therefore solved by applying the EXDCs onto the exact Boolean network. The algorithm then ensures compliance to error frequency constraints by recovering the correct outputs on the sought number of error-producing inputs while aiming to minimize the network cost increase.

## 1.3 Dissertation Outline

This rest of the dissertation is organized as follows: Chapter 2 discusses details of modeling and synthesis of approximate arithmetic blocks, Chapter 3 addresses the two-level ALS problem while Chapter 4 discusses the MALS problem. Chapter 5 concludes this dissertation.

# Chapter 2

# Modeling & Synthesis of Approximate Arithmetic Units

This chapter [1] addresses the modeling and synthesis of approximate arithmetic blocks. The primarily emphasis is on approximate adders, since many other arithmetic blocks, such as multipliers, are built on them. A discussion of how to apply such adders to the design of approximate multipliers is included at the end of the chapter.

One key question is how to balance error probability and error magnitude in the design of approximate adders. Maximally reducing error probability for each sum-bit minimizes overall error frequency. Yet, in many signal processing applications the relevant metric of approximation quality is a quadratic error measure, e.g., SNR/PSNR, that involves error magnitude as well as frequency. As is shown in Section 2.1.2, due to possible error patterns, there is a fundamental trade-off between error frequency and error magnitude in a timing-starved adder. Different solutions on a frequency-magnitude trade-off curve are generated by different arrangements of shortened carry segments,

[1] This Chapter is based on [25]. In the original publication, conceptual ideas were discussed with co-authors Michael Orshansky and Andreas Gerstlauer. Ku He provided the IDCT/filter simulation infrastructures.

where the PSNR-optimal choice also depends on the statistics of operands. This drives the first key contribution of this work: a formal proof that for signal processing applications with quadratic error measures and assuming a uniform distribution of operands, reducing bit-wise error frequency is sub-optimal and a quality-optimal approximate addition is achieved by limiting maximum error magnitude while accepting a larger error frequency. This is realized by reducing carry chains significantly below what is allowed by the timing budget for a large fraction of sum-bits, using an aligned, fixed internal-carry structure for higher significance bits. Crucially, such a structure also allows for maximal sharing of logic across all aligned carry segments, thus resulting in an area- and energy-optimal design. To enable formal analysis and derive these results, a model of timing-starved addition is introduced, which allows to analyzing the error patterns and their frequency. The model is general and applies to ripple-carry as well as prefix/tree-type adders.

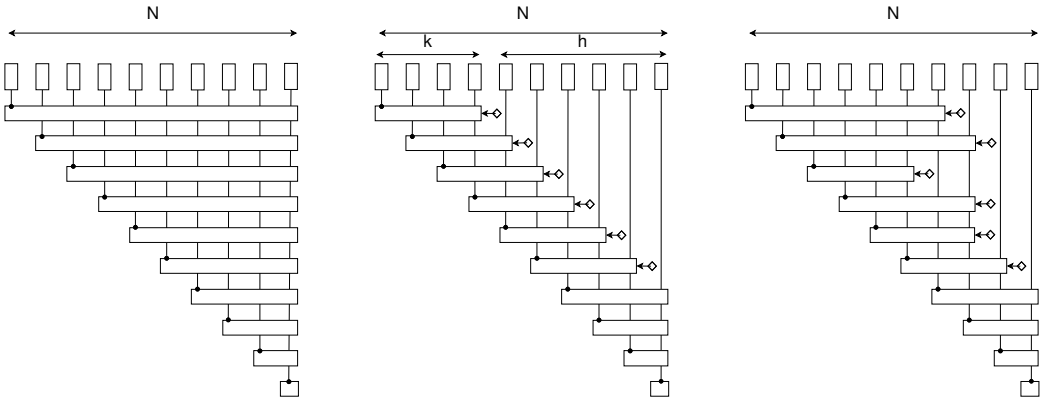
It is further shown that while maximum error is minimized by an aligned fixed internal carry adder for higher significance bits, it is crucial to further minimize average error. (In other words, just truncating the adder is a bad idea). This can be achieved by designing the least significant bits (LSBs) logic to produce an intentionally incorrect result that compensates for the error due to timing starvation. Logic is introduced that generates LSBs that saturate their output when an error is generated in the most significant bits (MSBs), i.e., conditionally. The key insight is that energy cost of such conditional bounding (CB) logic can be substantially reduced by realizing its

*logically inexact* version. The inexact CB logic synthesis problem is formalized and the existence of a rich space of Pareto-optimal alternatives with different area/energy-error behavior is demonstrated. Note that while other instances of bounding approximate addition have been reported, e.g., [21], [19], they are introduced heuristically without the proof of optimality or formal synthesis methods. Finally, it is demonstrated that both under- and overestimating approximate adders are possible. Further, several implementations of *dithering approximate adders* that mix under- and overestimating behavior to produce a zero-centered error distribution are introduced. The effectiveness of a dithering adder in reducing accumulation errors in consecutive additions by exploiting error averaging is demonstrated.

Synthesized approximate adders with energy up to 60% smaller than that of a conventional timing-starved adder are observed, where a 30% reduction is due to the superior synthesis of inexact CB logic. When used in a larger system implementing an image-processing algorithm, energy savings of 40% are possible.

## 2.1 Timing-Starved Addition: Properties & Optimality

This section develops a timing-starved adder model as a tool for analyzing the key features of approximate addition. It is used to demonstrate a fundamental trade-off between error frequency and error magnitude in a timing-starved adder. It is shown that for signal processing applications in which a quadratic error measure is used, reducing bit-wise error frequency is




(a) Full-budgeted adder (b) Timing-starved RCA (c) Timing-starved CLA

Figure 2.1: Timing-starved adder model (TSAM)

sub-optimal and limiting maximum error magnitude is paramount.

### 2.1.1 Timing-Starved Adder Model

In order to formally study the error frequency and magnitude patterns in approximate addition, a timing-starved adder model is introduced (TSAM) as defined in Fig. 2.1. The model can represent a variety of actual adder implementations, including ripple carry and tree adders. For ease of presentation, the ripple carry (RCA) adder is considered first. In the TSAM model, the top-level blocks represent sum bits, the horizontal blocks represent logic to compute each sum bit $S_i$, and the rightmost point of each such segment defines the location of the farthest accessible internal carry under a given timing budget. Under a full timing budget, (Fig. 2.1(a)), all the sum-bits have access to the correct carry-in ($= 0$) at bit 0. Under a reduced timing budget (Fig. 2.1(b)) equivalent to $k < N$ bits, some sum-bits do not have enough time

to be impacted by (do not "have access" to) the correct zero-bit carry-in. The actual accessible carry, given by the shifted rightmost point of each segment, depends on the value left on the carry node by the previous computation cycle and is treated as *unknown*. This unknownness of the carry in a timing-starved adder are represented by a diamond, see Fig. 2.1(b). Note that if Fig. 2.1 is used to model more complex adder structures, e.g., carry look ahead adders (CLAs) or prefix types, the segments will not be regular due to differences in paths for each bit. The carry will propagate to higher significance bits via carry-look-ahead bypass logic, whereas less significant bits may still need a regular propagation path. This shifts the adder critical path from the MSB to LSBs. Thus, when timing starvation occurs, the MSBs may, surprisingly, have an accessible internal carry that is further than even its right neighbor bits. This is illustrated in Fig. 2.1(c) for the example of a CLA.

The model allows studying the behavior of error frequency and magnitude with onset of timing starvation depending on the pattern of access of individual sum-bits to internal carries. Specifically, it is shown that depending on an arrangement of carry segments, a trade-off curve of maximum error magnitude and error frequency exists. The minimum error frequency solution is achieved by minimizing bit-wise error probabilities. Because of the low probabilistic likelihood of long carry chains, it is concluded that to lower the bit-wise error frequency, the longest possible propagation chain needs to be allocated for each bit position under the given timing budget. This is represented by an implementation that mimics the models in Fig. 2.1(b) and

Fig. 2.1(c). While such behavior can be achieved by timing starvation directly in a simple ripple-carry adder, concerns about metastability or timing-closure may require breaking up the carry chain into over-lapping independent carry blocks. Furthermore, in tree adders with non-uniform default segment lengths, an independent implementation of identical blocks allows for capturing the maximum possible carry length in all sum bits. Several such implementations have been reported in [22] and [30]. Intriguingly, it is shown below that this strategy is sub-optimal for many applications because of the nature of the trade-off between error frequency and maximum magnitude of error under a quadratic quality measure. Furthermore, implementations with independent carry blocks carry a larger area and hence energy overhead than what can be optimally achieved.

### 2.1.2 Error Frequency-Magnitude Trade-off

The trade-off between error frequency and maximum error magnitude is caused by patterns of possible errors. For a timing budget below $k$ bits (see Fig. 2.1), any bit up to the MSB bit can be false. Thus, the largest possible error is $2^{N-1}$. However, the maximum error is reduced if the false bit is followed by a *string* of false bits. Thus, surprisingly, forcing a set of bits to be false reduces the maximum error, and, if this is done for every pattern, the maximum *possible* error is reduced. Below, it is shown how to achieve this effect and that its flip-side is the increased frequency of errors.

Error patterns are represented by their $F$ and $T$ bit positions, which

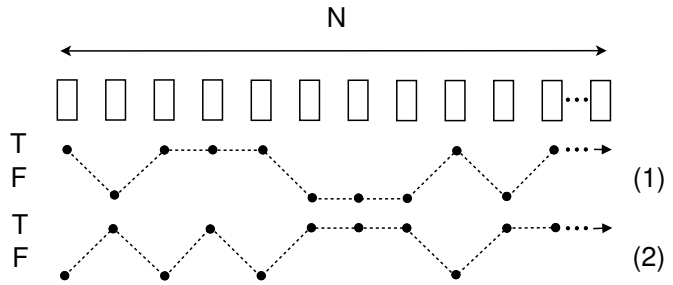
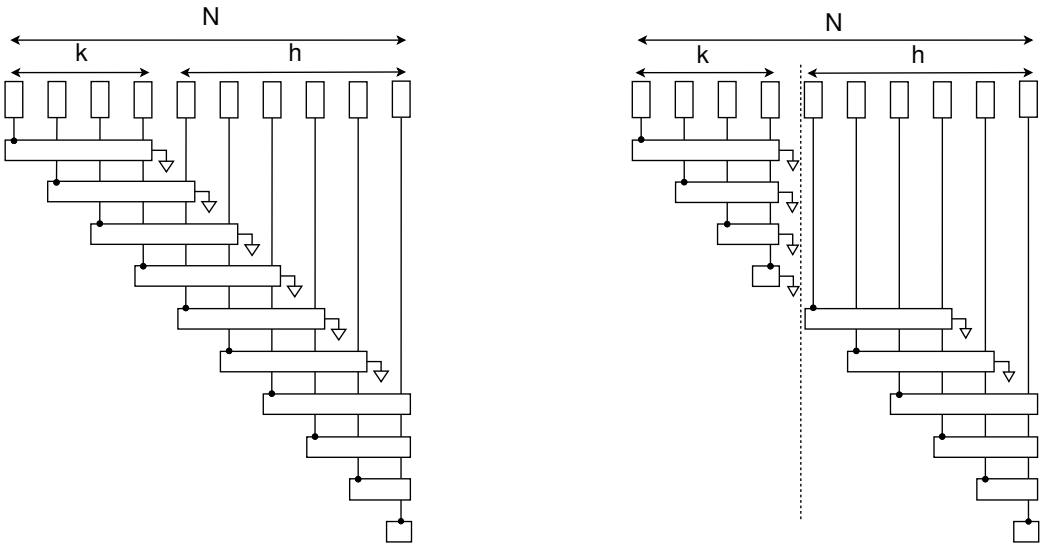



Figure 2.2: Error pattern waveforms.

indicate whether a bit is incorrect (false) or correct (true), respectively. Bit sequences are given in the form of regular expressions, where '$*$' indicates consecutive repetitions. Graphically, error patterns can be represented as arbitrary waveforms of correct and incorrect bits (see Fig. 2.2). It can be shown that a timing-starved adder can produce an $N$-bit output in which any pattern of $F^*$ and $T^*$ is possible. Importantly, the maximum error magnitude of an adder is defined by the location of the first left-most possible occurrence of an $F^*$ pattern; thus, the location of the first possible pair of $F^*T^*$ bounds the maximum error magnitude of an adder. This is called the *FT transition*. Furthermore, an $F^*$ pattern with a bitwidth of $m$, with a right-most bit in the pattern rooted at bit position $r$, can result in errors with only two magnitudes: $2^{m+r} - 1$ or $2^r$. In this case, whether the error pattern leads to a large or a small error depends both on the current adder inputs and the computational history for the internal carries. The key to the adder analysis is the realization that if logically all the internal carries are fixed, conditions under which $F^*$ would result in a large error (of magnitude $2^{m+r} - 1$) cannot occur, i.e., $F^*$ can only generate small errors with a magnitude of $2^r$ (See Section 2.6 for




(a) Hardwired to zero (FIC-TS) (b) Aligned carries (AFIC-TS)

Figure 2.3: Timing-starved adder with fixed internal carries.

details). Notice that internal carries can be fixed to either 0 or 1, leading to either lower- or upper-bounding of the result. (In what follows, it is assumed for the time being that the carries are fixed at 0). Such an adder is termed a fixed internal-carry timing-starved adder (FIC-TS), Fig. 2.3(a).

To reduce the maximum error magnitude, it is desirable to shift the $FT$ transition to a lower bit position. In a FIC-TS adder, the $FT$ transition can occur in the highest bit position and the maximum error is defined by the full length of the adder with a magnitude of $2^{N-1}$. Since errors cannot be avoided in general, the only way to shift the $FT$ transition within an $F^*T^*$ pattern is to convert as many $T$ bits as possible to $F$.

A bit $j$ is $F$ when the carry into its segment is incorrect, e.g., the correct carry is 1 while it is fixed to 0, and every downstream bit which is

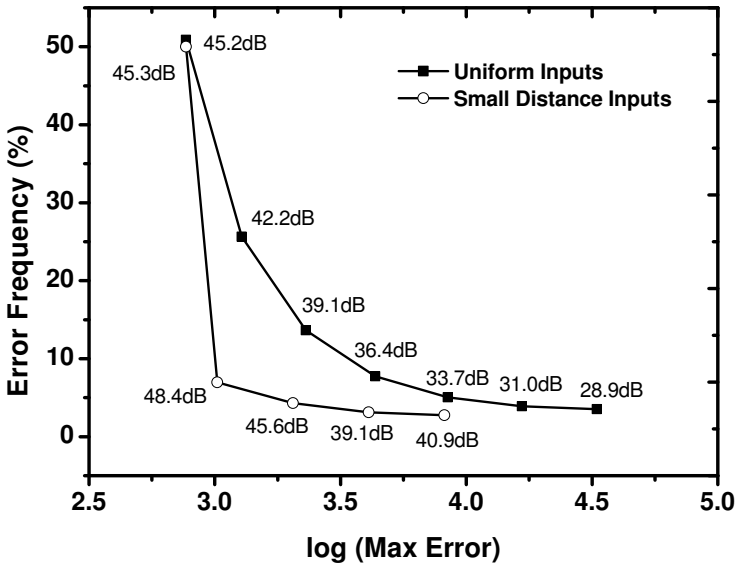



Figure 2.4: Error frequency vs. maximum error magnitude.

part of this segment has its propagate condition as true. In order to shift the $FT$ transition by one bit, it is necessary to ensure that if bit $j$ is $F$, bit $j-1$ also becomes $F$, which can be made true if the segment of bit $j-1$ also depends on the same incorrect carry-in. This can be achieved by aligning the right edges and hence inputs of the segments for bit $j$ and bit $j-1$. Now the correctness/incorrectness of bits $j$ and $j-1$ depends only on whether the accurate carry-in is zero (in which case both $j$ and $j-1$ are $T$) or one (in which case both $j$ and $j-1$ are $F$).

To shift the $FT$ transition as far right as possible, the above conversion is repeatedly applied starting at bit $j = N$. This results in aligning a set of segments of downstream bits to that of bit $N$. Clearly, segments of lower bits are shorter than $k$ (the length of the $N$th segment). Hence, it is impossible

to shift the $FT$ transition beyond $k$. The segment length $k$ is limited by the available timing budget, i.e., by the degree of timing starvation. However, the alignment of segments also means that the *effective carry chains* are reduced for the sum-bits below the MSB bit, which increases the probability of individual and thus overall error. Fig. 2.4 shows the trade-off curve between maximum error and frequency that results from this exploration for an increasing number of aligned segments up to $k = 7$ in a 16-bit RCA. The exact values of the Pareto-front depend on the statistics of adder operands, where results are shown both for an independent, uniform distribution as well as for input pairs that exhibit a small value distance across a uniformly distributed common magnitude range. Fig. 2.4 also shows the PSNR values that correspond to each configuration, which are discussed in the next section.

### 2.1.3 Optimal Approximate Addition under the PSNR Metric

Minimizing frequency of possible errors is justified in applications relying on error-correction. For other applications, such as in signal processing, it is the minimization of error magnitude that is more essential. In these applications, the quadratic error measure of adder error behavior, i.e., the quality of produced output, is most relevant. The specific metrics commonly used are the normalized mean squared error (MSE) and the related, peak signal-to-noise ratio (PSNR). PSNR depends on both magnitude and frequency of emerging errors. For adder operands that are uniformly distributed, i.e., where all the input values are equally likely, PSNR is much more heavily influenced

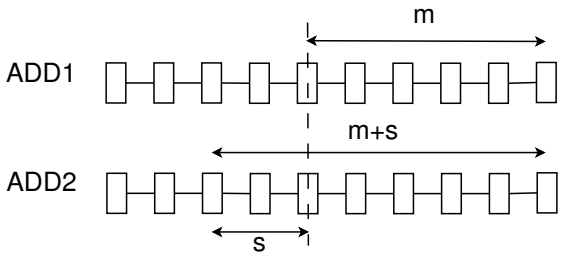



Figure 2.5: Adders $A_1$ and $A_2$ with locations $m$ and $m+s$ and magnitudes $2^m$ and $2^{m+s}$ of maximum errors.

by the magnitude of the maximum error rather than error frequency. Consider a trade-off between error magnitude and frequency at a fixed PSNR value. For example, two adders $A_1$ and $A_2$ are compared, shown in Fig. 2.5, that produce errors of maximum magnitude $\delta_{1_{max}}$ and $\delta_{2_{max}}$ with a frequency of $f_{1_{max}}$ and $f_{2_{max}}$, respectively. The quality loss in adder $i$ is measured as the sum of squared errors $SS_i = \sum_j^N \delta_{i_j}^2$ over $N$ additions, which is proportional to the inverse of PSNR. Given maximum error magnitudes and their frequencies, quality losses can be bounded from below and above as $Nf_{i_{max}}\delta_{i_{max}}^2 < SS_i < N\delta_{i_{max}}^2$ by assuming, in the best and worst case, that only maximum errors occur or that all additions lead to a maximum error, respectively. To understand when an adder $A_1$ has better quality than an adder $A_2$, it is necessary to establish the conditions under which $SS_1 < SS_2$. Using the upper and lower bounds above for $SS_1$ and $SS_2$, respectively, this is the case if $SS_1 < N\delta_{1_{max}}^2 < Nf_{2_{max}}\delta_{2_{max}}^2 < SS_2$, i.e., $f_{2_{max}} > (\delta_{1_{max}}/\delta_{2_{max}})^2$.

In an adder, the maximum error magnitude is determined by the position $m$ of the most significant bit in which an error can occur, and is equal to

$\delta_{max} = 2^m$. If two adders differ by $s$ bits in their maximum error location (see Fig. 2.5), the adder $A_1$ with smaller error magnitude will have better quality than the adder $A_2$ with larger error magnitude if the frequency of maximum errors in adder $A_2$ is at least $f_{2_{max}} > 1/4^s$. (Note that the inverse is not true, i.e., $f_{2_{max}}$ being below this bound does not necessarily imply that $A_2$ is better than $A_1$.) Thus, a larger error magnitude requires an exponential reduction in the frequency of such errors in order to remain below the quality budget set by an adder with lower error magnitude. This is confirmed by Fig. 2.4, which shows empirically collected PSNR values that correspond to each configuration. For the uniform input distribution, the peak PSNR is indeed achieved for a solution with the smallest maximum error magnitude. This is not the case, however, for all distributions: if an adder processes input pairs that have similar values (small distance) then the peak PSNR is achieved at a different point.

The rest of the section is focused on the analysis of the uniformly distributed (equally likely) adder inputs. Based on the analysis of the trade-off curve, the adder with the smallest maximum error is realized by aligning a set of segments of downstream bits to that of bit $N$. The resulting approximate adder structure is called an aligned fixed internal-carry timing-starved (AFIC-TS) adder. Fig. 2.3(b) shows the AFIC-TS adder, where the $FT$ transition is shifted to the dotted boundary at bit position $N - k$. This reduces the maximum error magnitude by a factor of $\frac{1}{2^{k-1}}$ to make the maximum possible error $2^{N-k}$. Note that the structure for the higher significant bits (left of the dotted

boundary in Fig. 2.3(b)) is logically equivalent to, and can be implemented as, a regular adder, e.g., a RCA or CLA, that spans the MSB segment length $k$ with a fixed carry in.

Using this analysis, the conditions under which an AFIC-TS adder has better error behavior than a FIC-TS one can be determined. The adders differ in their maximum error magnitudes by $k-1$ bit positions. Hence, an AFIC-TS adder will be better if the maximum error frequency of the FIC-TS adder is greater than $1/4^{k-1}$. The maximum error in an FIC-TS adder occurs if an incorrect carry propagates into its MSB while all other output bits are correct ($T^*$). This is the case if a carry is generated in the $k+1$st bit from the leftmost bit ($N-k-1$) and all higher significant bits propagate (but not generate) while all lower significant bits are correct. For uniform inputs, the probability of a bit to propagate or generate is $1/2$ and $1/4$, respectively. Furthermore, the probability for the lower significant bits to be correct is at least $1/2$ [30]. Thus, the maximum error frequency of the FIC-TS adder is at least $1/2^{k+2}$. In order to guarantee that the AFIC-TS adder is better, it is necessary to ensure that $1/2^{k+2} \geq 1/4^{k-1}$, i.e., $k \geq 4$. Since $k$ is a function of the available timing, this condition holds in almost all practical cases where budgets of at least 4 bit delays are allowed. Overall, the proof establishes that a AFIC-TS or equivalent adder (such as ETA [21]) is guaranteed to be better than a FIC-TS or equivalent adder (such as the approximation adder in [30]), regardless of the logic for lower significance bits (on the right side of the dotted boundary).

The discussion thus far has focused on proofs of quality optimality.

Depending on the implementability of various adder structures, there may be differences in logic complexity, area and hence energy. As such, a subset of non-quality-optimal adders can have a better energy than the quality-optimal structure and, thus, also may be Pareto-optimal in the quality-energy space. However, importantly, since aligning of segments allows for sharing of their logic, a maximally-aligned AFIC structure is not only optimal from a quality perspective, but also minimizes logic complexity.

When designing such optimal AFIC adders, there are remaining choices in regards to the logic of the lower significant bits and the value to which the fixed carry-in into the higher significance segment. Fixing the carry-in to zero or one will result in errors being always negative or positive, respectively. Depending on the desired behavior, adders therefore can be synthesized that over- or underestimate the result. This also opens the possibility of creating structures that dither to produce a zero-centered and reduced-variance errors. This choice also dictates the synthesis of the desired upper or lower bounding logic in the lower significance bits, as will be discussed further in the following sections.

## 2.2 Synthesis of Conditional Bounding Logic

After the maximum error is minimized with the aligned fixed internal carry adder for higher significance bits, it is crucial to reduce *average* error. This is achieved by using LSB logic to produce an intentionally incorrect result to compensate the error due to timing starvation. Logic is introduced that

generates LSBs that bound its correct output when an error is generated in the MSBs, i.e., conditionally. The energy cost of such conditional bounding (CB) logic can be shown to be substantially reduced by realizing its logically inexact version without substantial extra quality loss. In fact, there exists a range of Pareto-optimal adder implementations in the quality-energy design space. In the following, this design space is formulated and a heuristic is developed to synthesize adder implementations for different application requirements and target technologies.

The discussion is initially focused on the case when the timing budget (set by the MSB segment length $k$) is sufficient for the correct timing evaluation of the LSBs, i.e., $h = N - k \leq k$. When $h = N - k > k$, the synthesis approach will be able to trade off optimality for meeting a given timing budget. A hierarchical strategy is supported that partitions the entire LSB logic into several smaller segments that each individually meet their timing constraints. This requires the segments, however, to be isolated from each other with no carry propagation between them. As a result, this solution may come at a cost of further degradation in achievable PSNR value.

### 2.2.1 Conditional Bounding Logic Formalization

As discussed, depending on the value of the fixed (controlled) carry into the MSBs, an AFIC adder will always over- or underestimate the true result. An important observation is that a quality-optimal adder implementation can be achieved by designing matching, conditionally bounding LSB logic that

further minimizes remaining errors. Without loss of generality, first assume the design of an underestimating adder with internal carries fixed to zero for the following discussion. Let $C$ be the carry out of the LSB and carry into the MSB logic that is discarded. If $C = 0$, both MSB and LSB logic are correct. If $C = 1$, the MSB logic is incorrect, but an unmodified LSB logic still produces a correct result. This will always lead to the largest possible, negative error of $-2^h$. With these observations in mind, the optimal LSB logic should have the following properties: (a) produce a correct result when $C = 0$, and (b) produce the largest possible value (i.e., $11\ldots1$) when $C = 1$ to compensate for the large negative error in the MSBs as much as possible. This behavior is equivalent to the following Boolean equation for the desired LSB logic:

$$S_i' = S_i \vee C, \tag{2.1}$$

where $S_i$ is the true sum value for output $i$, $C$ is the carry-out of the entire LSB block, and $S_i'$ is the desired sum value for bit $i$.

As previously discussed, an alternative overall adder design possibility is to fix the carry into the MSB logic to one. In this case, the LSB logic should be reversed. It should produce a correct output when $C = 1$ and the smallest possible value (i.e., $00\ldots0$) when $C = 0$, which is logically described as:

$$S_i' = S_i \wedge C \tag{2.2}$$

This allows designing adders that are either over- or underestimating while minimizing the overall quality loss.

A general concern is that in either of these cases, consistent over- or under-estimation can result in errors that accumulate and grow when chaining several successive additions, as is the case, for example, in many applications that use accumulations. For applications that are sensitive to error accumulation, a structure is introduced that *alternates* between both types of logic in a dithering-style scheme in which statistical averaging reduces error variance in accumulation. This solution may come at an increased area cost, but due to the ability to synthesize reduced-area approximate bounding logic with the opportunity to share logic between both types, the area penalty is typically small. Furthermore, since at any given time only one block will be actively switching, there is very little energy overhead. A dithering adder is realized by a logic expression:

$$S_i^{'} = (D \wedge S_i \wedge C) \vee (\overline{D} \wedge (S_i \vee C)), \tag{2.3}$$

where $D$ is an external control signal. This external signal allows dithering to be controlled by the application, e.g., to exploit knowledge about input data statistics or required error behavior. Furthermore, simple, general control schemes can be designed that achieve averaging by driving the signal from a regularly alternating clock or through a history register that records whether a mismatch between hardwired and actual carry occurred and, if so, triggers the opposite bounding logic in the next addition in order to compensate.

As an alternative to external dither control, implementations can be considered in which the choice between over- and under-estimating, and hence

between upper or lower bounding LSB logic, is generated internally based on other, regular adder inputs. Crucially, it can be observed that an approximate AFIC adder will always produce a correct result iff the hardwired carry into the MSB matches the carry-out that would be produced by a regular LSB logic. Hence, if the LSB carry can be easily predicted from other inputs, and if the choice between different MSB carries and corresponding LSB bounding logic can be adapted accordingly, error frequency can be further minimized.

A low-overhead carry prediction can be performed based on adder inputs $A_{h-1}$ or $B_{h-1}$ at the partition boundary bit position $h-1$. If both of these inputs are zero or one, the carry-out of the LSBs will also be zero or one, respectively, independent of any LSB-internal carry propagation. Hence, dithering can be controlled via the exclusive-or of those two inputs. In all unpredictable cases, the aim is to randomly alternate for statistical averaging. For that, both cases can be combined and simply control the choice of MSB carry and LSB logic based on the value of one of the two inputs. Thus, the LSB logic expression for a $h-1$-dithering adder can be written as:

$$S_i^{'} = (A_{h-1} \wedge S_i \wedge C) \vee (\overline{A}_{h-1} \wedge (S_i \vee C)), \tag{2.4}$$

where $A_{h-1}$ is the $h-1$ bit of input $A$.

The logic defined in Eq. (2.1), Eq. (2.2) and Eqs. (2.3)/(2.4), is referred to as Conditional Upper Bounding (CUB), Conditional Lower Bounding (CLB) and Conditional Dithered Bounding (CDB) logic, respectively. In

general, CB logic, where every sum output depends on the carry out of the complete LSB block, is more complex than that of a correct adder. An important part of the synthesis strategy is the idea that it is possible to implement a logical approximation of $S_i^{'}$, given that ultimately the entire adder will still produce errors even if the CB logic implements $S_i^{'}$ exactly. By implementing a logical (Boolean) approximation to $S_i^{'}$, significant area and energy reduction with only slightly worse error behavior can be achieved. There will be a wide range of possible approximations of $S_i^{'}$ with different energy and quality values, from which a Pareto-optimal set can be found.

### 2.2.2 Bounding Logic Synthesis

The ultimate good is a Pareto-optimal set of solutions in terms of MSE/PSNR and energy. However, a direct search seeking optimal points in this space appears intractable at the moment. Instead, a heuristic approach is proposed that adopts a principle fundamental to logic synthesis: the number of literals in the logical expression is a proxy for the complexity, and thus area and energy (ignoring differences in switching activity), of the realization of a logic function. Formally, the Pareto-optimal set is generated by the solution to the following approximate logic synthesis problem:

$$min\ L(f^{'}) \quad s.t.\ \triangle\,(f^{'}, f) \leq \triangle_{target} \tag{2.5}$$

where $L(f^{'})$ is the number of literals in function $f^{'}$ and $\triangle(f^{'}, f)$ is the distance between the two functions $f^{'}$ and $f$. Notice that setting the distance to zero,

Table 2.1: Row-based function changes and distances for 2-bit CUB logic.

| Input | | | | CUB | $D_1$ | $D_2$ |
|---|---|---|---|---|---|---|
| $A_1$ | $B_1$ | $A_0$ | $B_0$ | $\langle S_1' S_0' \rangle$ | $\langle S_1'' S_0'' \rangle$ | $\langle S_1'' S_0'' \rangle$ |
| 0 | 0 | 1 | 0 | 01 | 00,10 | 11 |
| 0 | 0 | 1 | 1 | 10 | 01,11 | 00 |
| 0 | 1 | 1 | 0 | 11 | 10 | 01 |
| 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 1 | 0 | 0 | 0 | 10 | 01,11 | 00 |
| 1 | 0 | 0 | 1 | 11 | 10 | 01 |
| 1 | 1 | 0 | 0 | 11 | 10 | 01 |
| 0 | 0 | 0 | 0 | 00 | 01 | 10 |

$\triangle(f', f) = 0$, would make the problem equivalent to the traditional (exact) logic minimization problem.

The problem above introduces a proxy distance metric in lieu of PSNR allowing a more efficient implementation of the optimization problem. The distance definition is closely coupled with the heuristic optimization to be implemented. The algorithm acts directly on a specification of the Boolean function in terms of the list of its ON-set/OFF-set minterms, i.e., on its truth table.

Without loss of generality, the following discussion is based on CUB logic synthesis using Eq. (2.1) as function $f$ to approximate. Synthesis of CLB and CDB logic is analogous and an identical algorithm can be applied. Consider the truth table for the CUB logic $S_i'$ of a 2-bit adder in terms of input operands $A_i$ and $B_i$ (Table 2.1). The distance measure needs to capture the difference between the desired exact function and its approximation in a way that captures the characteristics of the PSNR error metric. Assuming uniform

**Algorithm 1**: Inexact CUB synthesis

```
for #row_flips r = 1 ... r_max do
    for each row in each subset of rows of size r do
        for D_j = D_min ... D_max do
            for each output ⟨S'_i⟩ and all ⟨S''_i⟩ = ⟨S'_i⟩ ± D_j do
                Replace ⟨S'_i⟩ by ⟨S''_i⟩;
                Run two-level Boolean minimization;
                Record min (literal, TD) pairs;
```

input distributions, each row in the truth table is equally likely. Therefore, the number of rows in which a change in function output is considered captures the frequency of errors. For a given row, the *decimal distance (D)* is measured as the decimal difference in the binary output values between the desired and inexact CUB logic. By flipping output bits within rows, inexact outputs may be produced with different decimal distances. Due to the quadratic nature of PSNR, the total distance (*TD*) is the sum of squared decimal distances over all rows in which bits have been changed ($r$):

$$TD = \sum_{j}^{r} D_j^2, \tag{2.6}$$

where $D_j$ is the decimal distance due to a change in row $j$. This procedure is illustrated in Table 2.1, which shows a partial truth table with decimal distances $D_1 = 1$ and $D_2 = 2$ for each row.

Using the *TD* metric defined above, the optimization heuristic shown in Algorithm 1 was implemented to find Pareto-optimal solutions in the literal-

*TD* space. It was empirically verified, as will be shown in the next section, that the proposed proxy metrics provide good fidelity while allowing tractable optimization. Solutions were synthesized using Design Compiler to find true area and used behavioral simulation to extract PSNR values, creating a mapping between the literal-*TD* domain and the area-PSNR domain for a range of functions.

It is possible to make the above algorithm more efficient by avoiding the consideration of all possible $D_j$ in each row. The improvement is based on the conjecture that the Pareto set of total distance ($TD$) vs. number of literals ($L$) solutions of the inexact functions is to be found among solutions produced by only considering $D_{min}$ combinations for a given number of row changes (flips) $r$. The conjecture can be justified by the following argument: (1) For any given $TD$ value at a fixed $r$, the minimum achievable $L$ is a function of the number of possible solutions to explore; (2) Due to the smaller number of possible flips of a row for larger $D_j$, the number of solutions $Num(D_j)$ decreases with an increase in individual distances being considered, i.e., $Num(D_i) > Num(D_j)$ for any $i < j$; (3) It follows that $TD$ vs. $L$ for a fixed $r$ is monotonic and rising with increasing distance $D_j$. Hence, the Pareto set is among modifications formed by $D_{min} \times D_{min} \ldots D_{min}$ combinations and only these solutions need to be explored. This conjecture is verified empirically, where experiments for LSB adder logic of size 2 and 3 both confirmed the trend in (3).

Even with the described simplification, it is not feasible to use the algorithm for more than about $r_{max} = 8$ row flips. Whether this is sufficient

depends on the number of rows in the truth table, which is exponential in the width $h$ of the LSB block. With $r_{max} = 8$, a sufficient range of Pareto points for $h \leq 3$ can be explored. To enable synthesis of larger adders, a hierarchical optimization strategy is adopted that partitions the LSB block into smaller segments that are synthesized and optimized independently and separately (Fig. 2.6). Segments are isolated from each other and there is no carry propagation between them, leading to sub-optimal approximations of the desired logic. However, by recursively applying the same CB synthesis approach to each segment with discarded output carry, the accrued errors are kept bounded, and the results are considered to be acceptable. Also, as alluded to earlier, the described hierarchical partitioning of the LSB logic can be used to meet reduced LSB timing budgets. The hierarchical approach reduces the runtime from $O(N \cdot 4^{2N})$ to $O(N)$ at the cost of reduced solution density. The actual accuracy loss is limited: the gap is found to be no larger than 3dB in the worst case.

Hierarchical exploration proceeds by first constructing the $L$ vs. $TD$ Pareto fronts for LSB logic of bitwidth $h = 1 \ldots 3$. To construct Pareto so-

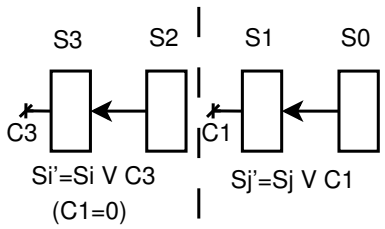
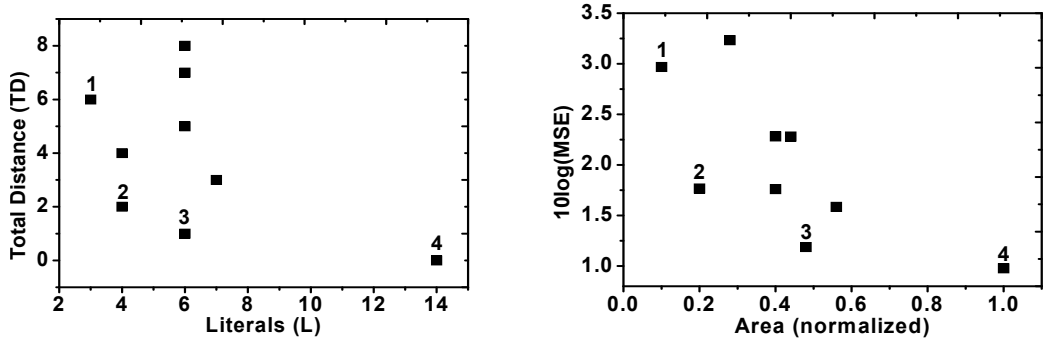



Figure 2.6: Hierarchical approach for partitioning of LSB logic and recursively applying CB synthesis to each segment.

lutions for larger bitwidths, all possible concatenations of smaller adders are explored and their different design points to find the best overall $L$ vs. $TD$ solutions.

## 2.3 Extension to Approximate Multiplication

The previous approximate adders can be extended to approximate multipliers. In general, multipliers can be classified into two types: sequential and combinational. Sequential multipliers are based on a sequence of shift-add operations to achieve the multiplication, which is essentially an accumulation structure. Therefore, the proposed dithering type adder can be immediately employed. For modern tree type multipliers, e.g., Wallace or Dadda multipliers, the reductions of partial product matrices are implemented by chaining full adders in each reduction stage. As such, reductions are formed by ripple carry adders, and, importantly, the last-stage summation is a conventional two-operand addition. Since the number of partial product reduction stages logarithmic in the multiplicand bitwidth and typically small, the majority of the critical path still lies in the last stage addition, where the last-stage adder can be replaced with the proposed approximate types. Note that for most signal processing applications, multipliers are intrinsically designed with a "truncation" feature due to the bitwidth augmentation after every multiplication. Therefore, most multipliers are already "approximated" by performing such truncations. Applying the proposed approximate adders on top of those truncated multipliers can produce further energy reductions while maintaining




(a) *TD* vs. $L$ (b) Area vs. quality

Figure 2.7: Synthesized inexact solutions for $h = 2$ CUB block.

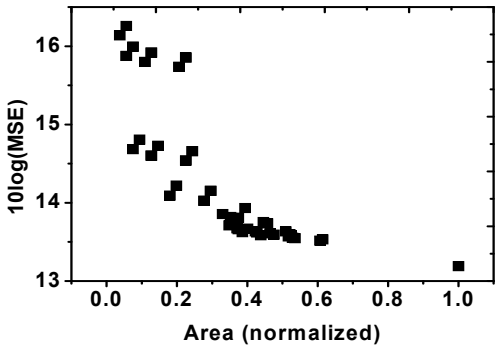



Figure 2.8: Inexact CUB synthesized hierarchically for 5-bit CUB block ($h = 5$).

a similar quality level. The relevant experiments are discussed in Section 2.4.

## 2.4 Experimental Results

First, the results of approximate LSBs block synthesis using the algorithm are demonstrated with Espresso [35] as the internal two-level Boolean minimization engine. Fig. 2.7(a) shows both the Pareto-optimal solutions and selected other design points explored by proposed algorithm for a 2-bit LSB

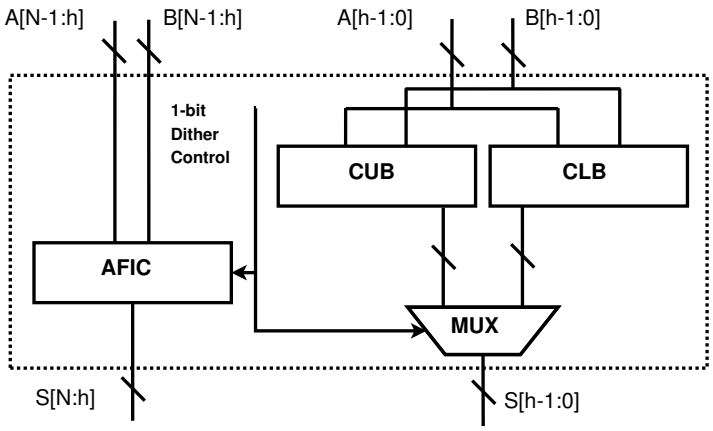



Figure 2.9: Dithering approximate adder.

block in the *TD* vs. $L$ space. The Boolean expressions for each of those solutions were synthesized with Synopsys Design Compiler [36] using the 45nm FreePDK [37]. Quality was estimated via simulation of the LSB block for 10,000 random, uniformly distributed input samples. The final area and quality values are shown in Fig. 2.7(b). Overall, good fidelity was observed: the points that are on the Pareto front in the *TD* vs. $L$ space are also Pareto-optimal in the quality-area space. Points at the extreme high and low ends of the $L$/area range thereby correspond to exact and minimum-area realizations of the desired CUB logic, respectively. Furthermore, Fig. 2.8 shows the set of solutions for the approximate realization of a 5-bit LSB block produced by the hierarchical synthesis approach. There is a wide range of trade-offs, with some solutions having 1/5th of the area of the exact CUB logic at a moderate quality loss.

As discussed previously, adders can be realized that combine over- and

underestimating behavior. Fig. 2.9 shows the conceptual design of the proposed dithering approximate adder. Its bounding behavior is controlled by an additional input signal, which determines both the carry into the MSB as well as the matching choice between CLB and CUB logic for the LSBs. In reality, the dithering LSBs can be synthesized as a combined CUB/CLB block with logic sharing. Overall, the overhead for a dithering-capable structure is low and its complexity remains well below that of a conventional adder. Note again that the dithering selection can be externally or internally controlled, either using more complex, adaptive schemes driven by the application or, simply, by a purely random signal, an alternating clock, based on carry-history or as a function of other inputs. After logic synthesis, a 24-bit RCA-based clock-dithering adder with $h = 10$ has a 34% area overhead compared to a standalone, minimum-area AFIC-CUB design. For an internally controlled $h - 1$-dithering adder, the area overhead compared to a plain CUB RCA is around 30%. With increasing base complexity, this relative overhead reduces to 11% and 7.8% for CLA and Kogge-Stone based designs, respectively.

Fig. 2.10 shows the achievable quality-energy tradeoffs of various 16-bit approximate CLAs using an AFIC structure with LSB lengths $h = 9$ & $h = 11$ under varying minimum-area, optimal-tradeoff, exact CUB and $h - 1$-dithering realizations of the LSB block. They are compared against a conventional timing-starved CLA design. Energy reductions through $V_{DD}$ scaling are assumed to be proportional to $CV^2$. For delay scaling, a curve-fitted model of HSPICE-simulated gate delays at different $V_{DD}$ values is utilized.

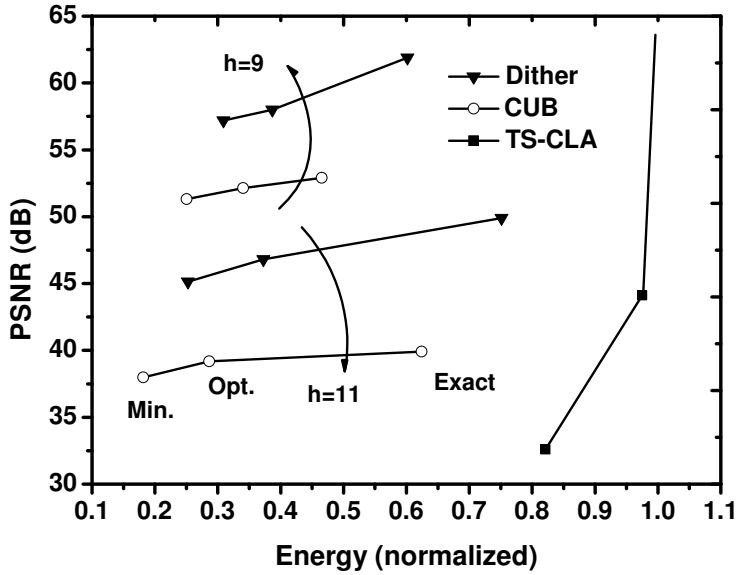



Figure 2.10: Quality-energy tradeoffs for different 16-bit CLAs.

Energy of AFIC adders was estimated assuming a timing budget and $V_{DD}$ value set by each nominal adder delay. Energy results are normalized to the base energy of the unscaled, original full-width CLA. Quality was measured by simulating adder results under scaled $V_{DD}$ for 10,000 random inputs.

Results show that the conventional timing-starved adder experiences a sharp drop in quality once their timing budget is exceeded. For AFIC adders, the base quality level as well as the timing budget is set by the LSB and MSB widths $h$ and $N - h$. Due to its ability for preemptively predicting the correct error compensation behavior purely from current adder inputs, a $h-1$-dithering scheme can in all cases significantly improve quality compared to its non-dithered counterparts. However, the added complexity comes at the cost of increased area and hence energy.

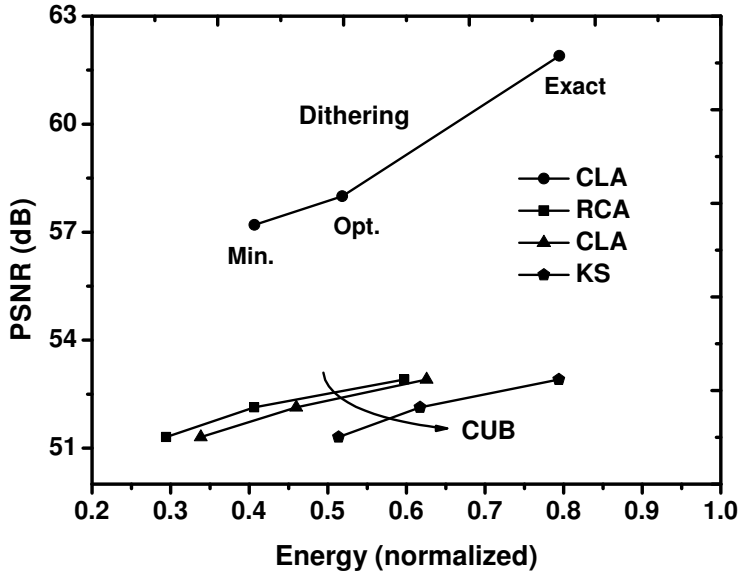



Figure 2.11: Quality-energy of 16-bit AFIC adders with $h = 9$.

Fig. 2.11 compares quality and energy of RCA, CLA and Kogge-Stone (KS) based designs of AFIC structures with $h = 9$, where energy is normalized against the base energy of an unscaled, regular RCA. Overall, even more so than the base adder structure, the partition boundary $h$ or the timing budget, the choice of logic in the LSBs has a large effect on the area of the design and hence on the maximal achievable energy savings. Savings vary by 30% to 40% depending on the LSB logic style. This confirms the significance of exploring the CUB/CLB design space when designing families of approximate adders.

Table 2.2 summarizes results for a 16-bit AFIC CLA with $h = 9$ and different non-dithering and dithering LSB realizations. For comparison, a truncating adder with an empty LSB block is included. Results show that a significant difference in achievable quality between different synthesized CUB

Table 2.2: 16-bit CLA with $h = 9$ and varying LSB logic.

| LSB Type | PSNR ($dB$) | Rel. Err. Full | Rel. Err. Small | Area ($\mu m^2$) | $V_{DD}$ ($V$) | Energy |
|---|---|---|---|---|---|---|
| Original | $\infty$ | 0% | 0% | 413.9 | 1.00 | 100% |
| Truncated | 38.9 | 1.1% | 100% | 128.1 | 0.83 | 21.3% |
| $CUB_{min}$ | 49.6 | 0.4% | 23.5% | 150.6 | 0.83 | 25.1% |
| $CUB_{opt}$ | 51.2 | 0.4% | 18.2% | 204.6 | 0.83 | 34.1% |
| $CUB_{exact}$ | 52.9 | 0.3% | 0% | 227.6 | 0.92 | 46.5% |
| $h-1_{min}$ | 57.2 | 0.2% | 23.4% | 185.8 | 0.83 | 30.8% |
| $h-1_{opt}$ | 58.6 | 0.2% | 17.8% | 232.4 | 0.83 | 38.7% |
| $h-1_{exact}$ | 61.9 | 0.2% | 0% | 264.7 | 0.97 | 60.2% |

designs. Specifically, dithering adders improve PSNR considerably. Some CUB designs show very poor relative error, which can be an important metric for realistic DSP systems. Relative errors are shown for two different uniform distributions of input with a full and a reduced range of magnitudes. For inputs that are smaller than the partition boundary, the design of the LSB logic has a large influence. In contrast to other realizations, an exact CUB realization will be error-free for such inputs. Overall, depending on application requirements, there exists a non-trivial tradeoff in finding a good compromise between quality and energy, as realized by the optimal $CUB_{opt}$ instance for the uniform input case.

(a) IDCT w/ AFIC-CUB ($h = 8$)
PSNR=34.57dB, Energy=0.67

(b) Filter w/ AFIC-CUB ($h = 12$)
PSNR=23.7dB, Energy=0.60

Figure 2.12: Approximate adders in image processing applications.

To demonstrate feasibility for practical scenarios, adder concepts were applied to an IDCT image decompression and an image sharpening design, where the latter realizes a high-pass filter as a 2D convolution operation in the pixel domain. Fig. 2.12 shows the images and quality-energy tradeoffs under scaled $V_{DD}$ when replacing a conventional 24-bit RCA in both designs with the minimal-area AFIC-CUB structure. Results are compared to the original IDCT and sharpening designs with a normalized energy of 1.0 and a PSNR of 44.6dB and 23.9dB, respectively.

(a) IDCT w/ Truncated Adder ($h = 10$)
PSNR=16.9dB, Energy=0.61

(b) Clock-dithering adder ($h = 10$)
PSNR=33.15dB, Energy=0.62

(c) History-dithering adder ($h = 10$)
PSNR=35.52dB, Energy=0.63

(d) $h - 1$-dithering adder ($h = 10$)
PSNR=36.92dB, Energy=0.62

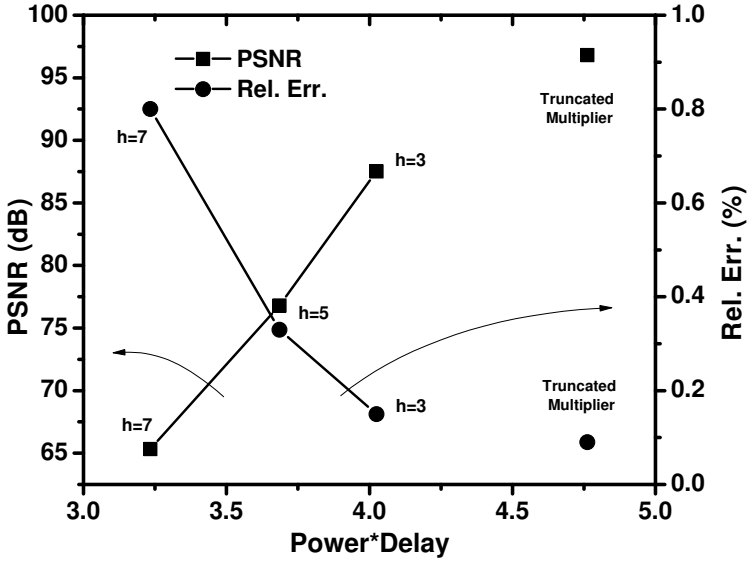
Figure 2.13: IDCT quality and energy of a truncated adder (a), and different dithering schemes (b-d).

While significant energy reductions can be achieved for a commonly accepted image quality above 30dB in the IDCT, error accumulations in the

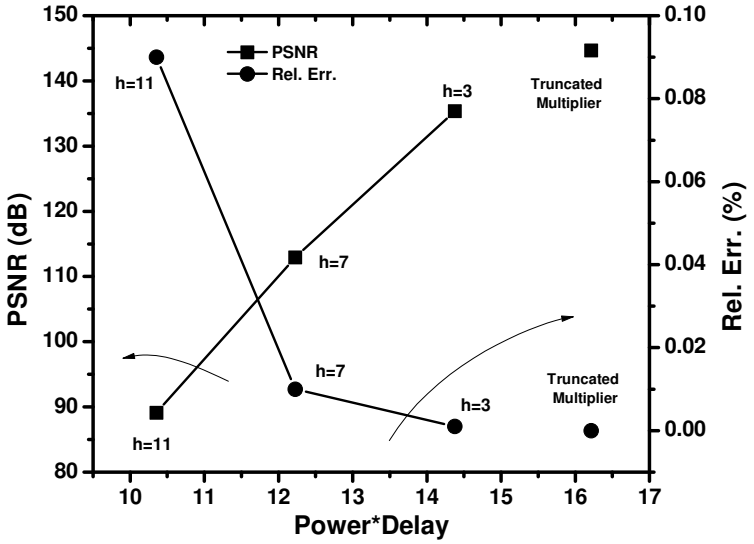
AFIC-CUB design lead to visual artifacts in the form of horizontal stripe patterns. By contrast, application of various dithering schemes provides both a much better PSNR as well as perceived quality. As shown in Fig. 2.13, by increasing the partition boundary $h$ and hence decreasing the timing budget, this quality gain can be traded off for further energy savings. The designs are compared against a traditional approach that works with reduced precision (i.e., truncation) to achieve similar energy savings. Both from a PSNR and subjective image quality standpoint, the $h-1$-dithering scheme is superior to truncation and any randomized external control. Dithering also leads to a reduction in the variance of observed errors. For the IDCT, the error distributions were measured. For an AFIC-CUB adder it has a mean of -0.95 and a variance of 5.16. By contrast, the clock-dithering adder produces errors with a mean of -0.1 and a variance of 0.94. Distribution of errors is not a concern in the sharpening filter. Here, a simpler AFIC-CUB adder with $h = 12$ already achieves similar results (Fig. 2.12(b)). In both cases, around 40% energy savings can be achieved while maintaining good image quality.

Two different approximate multipliers were examined with AFIC-CB adders as their last-stage two-operand addition in Figure 2.14(a) and Figure 2.14(b). The CB logic is applied on top of a conventional truncated multiplier, which is introduced in both figures. Results are shown for both PSNR and relative error metrics. Points on these curves are acquired for varying the $h$ values ($h = 3, 5, 7$ for 16-bit case; $h = 3, 7, 11$ for 24-bit case), where the left-most point corresponds to the largest $h$ value. Power*Delay was used as the

proxy to energy. From the figures, it is observe that the AFIC-CB structure for the last-stage addition can save about 10% energy at a similar quality level compared to a conventional truncated multiplier. When relaxing the quality level further, an up to 30% energy saving is possible.




(a) 16-bit multiplier with $h = 3, 5, 7$




(b) 24-bit multiplier with $h = 3, 7, 11$

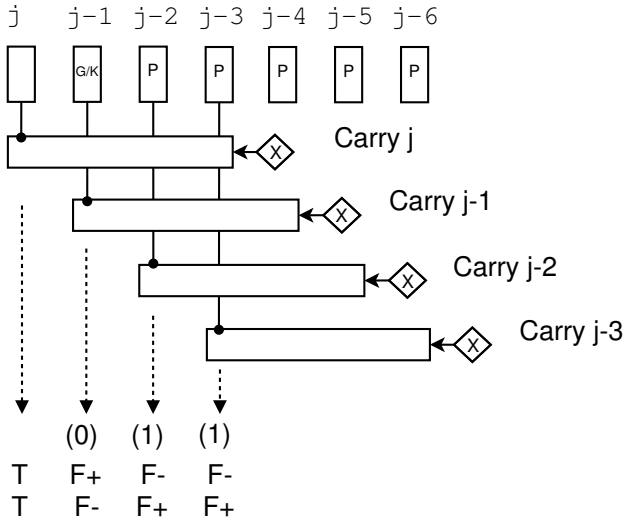
Figure 2.14: Approximate Dadda multipliers with AFIC-CB adder as the last two-operand addition.

## 2.5 Summary

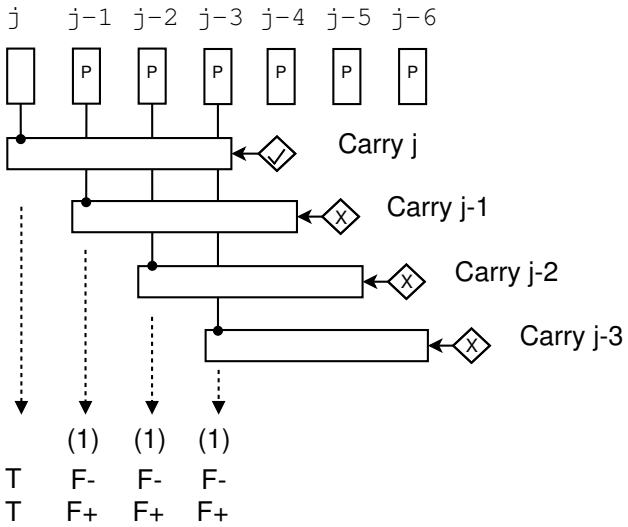
This chapter presents a theoretical approach for analysis and synthesis of approximate adders. The approach is general and the existence of optimal AFIC adder structures is formally proved in which higher significance bits are implemented using regular, aligned carry additions. Within the space of AFIC adders, it is further demonstrated that a rich set of design alternatives at varying quality-energy tradeoffs can be synthesized. This includes variants with overestimating, underestimating or dithering approximation behavior for use within different classes of application requirements. The results show that energy savings of up to 60% are possible at the individual adder level. Integrating the developed approximate adders into realistic image processing designs allows more than 40% total energy savings while maintaining excellent image quality.

## 2.6 Appendix: Proof of Fixed Internal Carry

The following presents an argument for the claim that if the internal carries are logically fixed, conditions under which an $F^*$ pattern would result in a large error $(2^{m+r}-1)$ can not occur, i.e., $F^*$ can only generate small errors with a magnitude of $2^r$:




(a) Case 1




(b) Case 2

Figure 2.15: Analysis of possible error patterns using TSAM.

1. Based on the sign of the error in each bit, two subcategories $F_+$ and $F_-$ of $F$ can be distinguished depending on whether the incorrect value is 1

when the correct value is 0 or vice versa.

2. The $F_+^*$ and $F_-^*$ sequences both produce the largest error magnitude $(2^{m+r}-1)$. By contrast, the $F_-F_+^*$ and $F_+F_-^*$ sequences produce the smallest error magnitude $(2^r)$. Those can be easily seen from their weighted decimal expressions.

3. The conditions under which a timing starved adder produces a $T$ or $F$ in a bit position can be clarified as follows. The location of the first $T$ to the left of a $F^*$ sequence is the focus. Using the $TFFF$ sequence as an example (see Figure 2.15):

   - There are only two ways to produce a $T$ in bit $j$: (1) there is at least one bit in the segment of bit $j$ (except for the bit $j$ itself) that generates $(G)$ or kills $(K)$ a carry propagation out of or into the bit (such as bit $j-1$ in Fig. 2.15(a)), or (2) if all the bits within the segment of bit $j$ are set to propagate $(P)$ their carries, the carry into the whole segment must be correct (indicated by the tick mark in the diamond of Fig. 2.15(b)).

   - On the other hand, a $F$ in bit $j$ is only triggered when all bits in its segment (except for bit $j$ itself) propagate and the carry into the segment is incorrect (such as bits $j-1$, bit $j-2$ and bit $j-3$ in Fig. 2.15(b)).

   - Crucially, if an $F$ in more than one bit is produced, then all the carries into the corresponding segments (such as $Carry_{j-1}$, $Carry_{j-2}$,

$Carry_{j-3}$) are guaranteed to have the same value.

4. The relationship between the $F$ bits in an $F^*$ sequence produced under one of the two conditions outlined above is the focus. In case (1) (Fig. 2.15(a)), the leftmost $F$ is produced in the $G/K$ bit. All bits to the immediate left of this bit position have to be $T$. In case (2) (Fig. 2.15(b)) the leftmost $F$ is produced by a $P$ bit. In both cases, all lower significant $F$ bits are produced by bits with a $P$ condition. Importantly, ignoring carries, all input patterns leading to a $P$ condition (patterns $0+1$ and $1+0$) result in a 0 sum whereas both $K$ and $G$ conditions ($0+0$ and $1+1$) produce an output of 1. Therefore, if the carry into any such bit position is incorrect, $P$ and $G/K$ bits will always produce errors of opposite sign. It follows that in case (1), the leftmost $F$ will have a different error sign than all the other $F$ bits whereas in case (2), all $F$ bits have the same error sign.

5. If all internal carries are fixed to an identical value, case (2) can no longer occur. Hence, only case (1) can produce a $TF$ transition, and the first $F$ of this $F^*$ block must have an opposite sign than the rest of the $F$ bits in the sequence.

6. Thus, if all the internal carries are set to a fixed value (0 or 1), conditions under which $F^*$ would result in a large error ($2^{m+r}-1$) can not occur, i.e., $F^*$ could occur only in the form of a $F_-F_+^*$ or $F_+F_-^*$ pattern and can only generate small errors with a magnitude of $2^r$.

# Chapter 3

# Two-level Approximate Logic Synthesis

The previous chapter covered some preliminary logic synthesis techniques when deriving the approximate CB logic. Those techniques, though effective for CB logic synthesis, are suboptimal in general. In fact, the interest is in a much broader question: how to automatically synthesize approximate circuits once a certain error-tolerant specification is given? This chapter [1] addresses the problem of approximate logic synthesis (ALS) under arbitrary error magnitude and error frequency constraints. A two-level logic minimization algorithm is developed which rigorously synthesizes a minimum-cost cover of a Boolean function that is allowed to deviate from an exact Boolean function in a constrained manner. A two-phase approach is adopted to solve the minimization. The first phase solves the problem that is constrained only by the magnitude of error. In the second phase, this frequency unconstrained problem is iteratively refined to arrive at a solution that also satisfies the original error frequency constraint. Experiments on adders and multipliers demonstrate literal count reductions of up to 60% under tight magnitude and frequency constraints.

[1] This Chapter is based on the previous publication [38]. Conceptual ideas were discussed with the co-authors Michael Orshansky and Andreas Gerstlauer.

## 3.1 ALS Constrained by Error Magnitude Only

This section discusses the approximate logic synthesis problem when constraining the magnitude of allowed error only. Only the patterns of allowed errors that the function may produce are considered, but not how often the errors occur.

*The first contribution is the realization that the approximate synthesis problem un-constrained by the frequency of errors is isomorphic with the Boolean relations problem.*

### 3.1.1 Isomorphism between Frequency-Unconstrained ALS and Boolean Relations

The most immediate domain of application of ALS is in synthesizing approximate arithmetic blocks for error-tolerant computing algorithms and applications. In applications that involve approximate arithmetic functions, such as in the signal processing domain, it is typically important to satisfy constraints on the magnitude of the possible error as well as the frequency of such errors. Here, frequency is defined as the number of minterms on which an error occurs as a fraction of the total number of minterms.

Constraining the magnitude of error is the most natural approach to limiting the outputs of arithmetic circuits, since a clear notion of distance is available for these functions. Consider a multi-output Boolean function $F : B^n \rightarrow B^k$ that defines a combinational network of an arithmetic circuit, e.g., an adder. Constraining the magnitude of possible errors is first considered.

The output of $F$ is the result of binary arithmetic computation. The aim is to synthesize its magnitude-constrained approximate version $F_m$, such that the only constraint is that $|F - F_m| \leq M$. Here, $|\cdot|$ is the absolute value operator, and thus the range of possible output values of the approximate function is constrained to be no greater than $M$. Note an important implicit aspect of the definition. The frequency-unconstrained function $F_m$ will have an *arbitrary* error frequency. Specifically, there is no implication that it has an error on every input.

To explicitly account for the error frequency (rate) of an approximate function, a modified notation is introduced and $F_{m,r}$ is denoted as an approximate version of $F$ with exactly $r$ minterms in error and with the constraint on the magnitude of error (no greater than $M$). Let the error frequency constraint be $R$ indicating that no more than $R$ minterms are allowed to be in error. With that, the full approximate logic synthesis problem is:

$$\begin{aligned} \min \quad & L(F_{m,r}) \\ \text{s.t.} \quad & r \leq R, \\ & |F(x) - F_{m,r}(x)| \leq M \quad \forall x \in B^n \end{aligned} \tag{3.1}$$

where $L(F)$ is the number of literals in a sum-of-products representation of function $F$.

One possible strategy for solving the above problem is to start with an exact function $F$ and gradually introduce errors while controlling *both* the frequency and magnitude of allowed errors.

However, the strategy to pursue in this chapter is based on a *two-phase* solution. In the first phase, the frequency unconstrained problem is solved. In the second phase, the unconstrained solution is iteratively refined to arrive at the solution that satisfies the original error frequency constraint. The frequency un-constrained problem is given by:

$$\begin{aligned} &\min \quad L(F_m) \\ &\text{s.t.} \quad |F(x) - F_m(x)| \leq M \quad \forall x \in B^n \end{aligned} \tag{3.2}$$

where $F$ and $F_m$ are the exact and approximate functions, respectively.

*The key observation is that the above ALS problem constrained only by error magnitude is isomorphic with minimization of Boolean relations,* which is a known and extensively-studied problem in traditional synthesis. A Boolean relation can be formally defined as follows [39]:

**Definition 3.1.1. Boolean relation.** A *Boolean relation* is a one-to-many, multi-output Boolean mapping, $R : B^n \rightarrow B^k$. A set of multi-output Boolean functions, $f_i$, each compatible with $R$, is associated with a relation. A Boolean relation is specified by defining for each input $x \in B^n$ a set of equivalent outputs, $I_x \subseteq B^k$.

Thus, Boolean relations are a generalization of Boolean functions, where each input corresponds to more than one output. An incompletely specified logic function with *don't care* is a special case of a single-output Boolean relation.

To establish the equivalence of ALS with the Boolean relation problem, it is observed that the constraint $|F - F_m| \leq M$ can be re-written minterm-wise: for each minterm $x_i$ of function $F$, allow the value of $F_m(x_i)$ to take values in the set $F \cup E_i$, where $E_i$ is the specified output error set for $x_i$. Thus, $E_i$ represents the additional values that the function can take while satisfying the error magnitude constraint. Now, each input corresponds to more than one output. The new formulation is given by:

$$\begin{aligned} &\min \quad L(F_m) \\ &\text{s.t.} \quad F_m(x_i) \in F(x_i) \cup E_i(x_i) \quad \forall x_i \in B^n \end{aligned} \tag{3.3}$$

**Example 3.1.1.** A simple example of an adder is used to illustrate the concepts being introduced. For a 1-bit half adder, the equivalence is illustrated via a tabular representation for $M = 1$:

| $F$ | |
|---|---|
| $a, b$ | $c, s$ |
| 00 | {00} |
| 01 | {01} |
| 10 | {01} |
| 11 | {10} |

| $F_m$ | |
|---|---|
| $a, b$ | $c, s$ |
| 00 | {00, 01} |
| 01 | {01, 00, 10} |
| 10 | {01, 00, 10} |
| 11 | {10, 01, 11} |

It is clear that the above tabular form sets up a Boolean relation (BR) representation, according to Def.3.1.1, where each input corresponds to more than one output.

### 3.1.2 Boolean Relation Solvers

To this point, the equivalence between the error frequency-unconstrained approximate logic synthesis problem and the Boolean relation minimization

problem has been established. This is advantageous as there exist several exact and heuristic approaches for solving the BR problem. Here gives a brief overview of the available BR minimization techniques. The exact method reported in [39] employs an approach similar to the Quine-McCluskey procedure [6]. The minimization is formulated as a binate covering problem and solved by integer linear programming. Other exact methods are [40] and [41]. As is common, the exact approaches are limited to solving small and medium-size BR instances due to the algorithm complexity. Heuristic solutions trade result optimality for computational tractability. Herb [42] is based on the two-level minimization algorithm of ESPRESSO [35] and test pattern generation techniques. Gyocro [43] also relies on ESPRESSO. While it improves on some of the weakness in Herb, it still remains slow.

A recently developed heuristic algorithm BREL [44] is adopted. BREL is a recursive algorithm that uses a branch-and-bound solution strategy. It first over-approximates (using the maximum flexibility provided by the relation) the BR into a multi-output Boolean function where each output is minimized independently using standard techniques for function minimization. If the minimized Boolean function is compatible with the original Boolean relation, then it is accepted as the solution. Otherwise, the algorithm splits the original Boolean relation $R$ into two sub-BRs $R_1$ and $R_2$. This is done by selecting one conflict minterm such that each sub-BR operates on one output component of this minterm. Sub-BRs are then solved independently following the same procedure recursively. BREL substantially outperforms the earlier tools in

terms of runtime and result quality.

## 3.2 Frequency-Constrained ALS Algorithm

This section describes the second major contribution: the development of an effective heuristic logic optimizer that accepts the solution of the frequency-unconstrained ALS and carries out further optimizations to guarantee the solution feasibility with respect to the frequency of errors.

Because the result of solving the Boolean relation minimization for $F_m$ does not constrain the number of minterms in error, the solution may not satisfy the constraints on error frequency. Let the result of solving the Boolean relation be the function $F_{M,k}$, where $k$ refers to the resulting actual error frequency. If the error frequency constraint $R$ is smaller than $k$, then the number of minterms need to be reduced on which the function is different from the exact one.

### 3.2.1 Mapping to Min-Cost Increase Problem

An important property of the solution of the Boolean relations problem is first clarified. As a solution to the problem of Equation (3.3), the function $F_{M,k}$ has the minimal cover (in terms of literals) among all functions that satisfy $F_{M,k} \in F \cup E_i$. Therefore, the following holds:

**Theorem 3.2.1.** *For any function* $F_{M,r}$

$$L(F_{M,r}) \geq L(F_{M,k}), \quad \textit{for any } r < k.$$

*Proof.* If there is an $r$ such that $L(F_{M,r}) < L(F_{M,k})$, then the BR solver reports $F_{M,r}$ as the BR solution since $F_{M,r}$ also satisfies the specified Boolean relation and has fewer literals. □

The problem to be solved in the second phase is now reformulated. To solve the problem in Equation (3.1), it is necessary to find the function $F_{M,R}$ that minimizes the literal increase $L(F_{M,R}) - L(F_M)$:

$$\begin{aligned} \min \quad & L(F_{M,R}) - L(F_M) \\ \text{s.t.} \quad & |F_{M,R} - F| \leq M \end{aligned} \tag{3.4}$$

An iterative and greedy algorithm is proposed that searches for $F_{M,R}$ by repeatedly identifying the minterms on which the correctness of the function should be enforced. The algorithm proceeds by making localized changes to the function by accepting steps that minimize literal increase while reducing the maximum number of error-minterms and guaranteeing that the magnitude constraint remains satisfied.

### 3.2.2 Formalization of the Frequency-Constrained ALS Algorithm

The algorithm works with a set of minterms on which the function $F_M$, produced by the frequency-unconstrained minimizer, is in error. First, all such minterms are formally defined and the types of exhibited errors are distinguished.

**Definition 3.2.1. DIFF minterm and DIFF set.** A minterm $x$ on which $F(x) \neq F_M(x)$ is called a *difference* minterm and is referred to as the *DIFF*

minterm. The difference set for a function, which is called the *DIFF* set, contains all *DIFF* minterms regardless of the type of error.

**Definition 3.2.2. Error types.** The error type is designated by $ET$. If for a given minterm and for a single output bit, $F(x) = 0$ and $F_M(x) = 1$, the error is of the $0 \rightarrow 1$ type. It is encoded as a two-bit value $ET = 01$. If for a given minterm and for a single output bit, $F(x) = 1$ and $F_M(x) = 0$, the error is of the $1 \rightarrow 0$ type. It is encoded as as two-bit value $ET = 10$. If there is no error, $ET = 00$. Let $ET_{i,j}$ be the two-bit encoding of the error type for an output bit $i$ on the minterm $j$. Let $CET_j$ be the concatenation of $ET_{i,j}$ for $i = 1$ to $k$, where $k$ is the number of outputs. $CET_j$ encodes the entire error pattern for function $F$ on the minterm $j$.

**Example 3.2.1.** An example is used to illustrate the definitions. The shaded minterms $x_1'x_0'$, $x_1x_0'$ in Table 3.1 form the $DIFF$ set for the 2-input, 2-output Boolean function. Minterm $x_1'x_0'$ has an error on the output bit $y_0$, which is an $ET = 01$; while there is no error for $y_1$ on this minterm. Minterm $x_1x_0'$ has errors on both $y_1$ and $y_0$ output bits, where $y_1$ has the $ET = 01$ error and $y_0$ has the $ET = 10$ error.

The algorithm to be constructed seeks to find $F_{M,R}$ by enforcing correctness on some of the minterms of $F$ that have been modified by the solution to the Boolean relations problem. The key part of the algorithm is therefore the notion of correcting the function on a given minterm. To correct an $ET = 01$, the minterm needs to be moved from the ON-set of the function

Table 3.1: Example of DIFF minterms (shaded).

| $F$ | | $F_M$ | | $CET$ |
|---|---|---|---|---|
| $x_1x_0$ | $y_1y_0$ | $x_1x_0$ | $y_1y_0$ | $y_1y_0$ |
| 00 | {00} | 00 | {01} | $\{ET = 00, ET = 01\}$ |
| 01 | {01} | 01 | {01} | $\{ET = 00, ET = 00\}$ |
| 10 | {01} | 10 | {10} | $\{ET = 01, ET = 10\}$ |
| 11 | {10} | 11 | {10} | $\{ET = 00, ET = 00\}$ |

for this output bit back to the OFF-set. This is called a *correct-to-0* change. To correct an $ET = 10$, the minterm needs to be moved from the OFF-set of the function for this output bit to the ON-set, which is called a *correct-to-1* change. The result of minterm correction is a change in the literal count in the cover of the function $F_{M,r}$. It should be noted that both types of corrections may result in a literal count increase. Also, note that at an equal literal count increase, the algorithm will accept both types of corrections equally as long as the magnitude of error is not increased.

The proposed algorithm is greedy and gradually identifies the best minterms to correct. One possible approach is to correct one DIFF minterm at a time by selecting a minterm that causes the least literal cost increase. However, this is sub-optimal. Instead, the proposed algorithm is based on the principle that *at each step the largest number of DIFF minterms should be corrected for the minimum available literal increase.*

The central challenge of the algorithm is in identifying the optimal *changes* to the ON/OFF-sets of the function such that the cover is minimized. (Note the difference between the conventional logic minimization (LM) and

the above problem. Conventional LM is to find the minimum cover for given ON/OFF-sets. The problem is its dual and seeks to find the optimal *change* to the ON/OFF-sets for a minimum cover increase.)

First, consider a single-output function $F$. The set of possible correction decisions, which is represented by the DIFF set, can be represented separately by a pair of correction functions: one for *correct-to-1* and one for *correct-to-0*, where a correction function is 1 iff the given minterm is a member of the corresponding DIFF set and thus a candidate for correction. The *correct-to-1* function $CT1$ is defined by the set of its minterms, which are the DIFF set minterms with $ET = 10$. Correspondingly, the *correct-to-0* function $CT0$ is defined by the set of its minterms, which are the DIFF set minterms with $ET = 10$.

The key aspect of the algorithm is the idea that the identification of minterms to correct should proceed by first constructing a minimial cover for the two correction functions ($CT0$, $CT1$) and by using the prime implicants (PIs) of the covers to seek optimal changes to the ON/OFF-sets of the function. The prime implicants of the minimum cover of a correction function is called the **DIFF primes**.

The following notion of *cost* is used to compare the effectiveness of correcting a specific DIFF prime $j$ of a function:

$$cost_j = \frac{\text{literal increase due to correction of DIFF prime } j}{\text{number of minterms covered by DIFF prime } j} \tag{3.5}$$

The greedy decision-making is driven by selecting at every iteration the

best decision understood as the decision with the least cost, as defined above. This principle is formalized in the following theorem, which is proven later in the derivation after the function update strategy is fully explained:

**Theorem 3.2.2.** *For a single-output function $F$, the optimal set of minterms to add to the ON/OFF-set at the minimum literal increase in the cover of function $F_{M,r}$ lies among the prime implicants of the minimum cover of correction functions $CT0$ and $CT1$.*

The above results can be extended to multi-output functions and their corresponding correction functions. An important aspect of the allowed corrections for multi-output functions is that the magnitude of error cannot be increased. This can be guaranteed only if either the entire function is corrected on a given minterm or the entire output is not modified at all. This constraint, combined with the result of Theorem 3.2.2 that directs to seek optimal decisions among the minimum covers, leads to define the correction function for a multi-output case *not by the individual DIFF minterms but by sub-sets of DIFF minterms.* The sub-set, referred to as the DIFF group, is defined as:

**Definition 3.2.3. DIFF group** A DIFF group is a set of all DIFF minterms with identical $CET$.

**Example 3.2.2.** Table 3.2 shows an example of grouping the DIFF minterms, where four DIFF minterms are grouped into three DIFF groups.

The correction function for a multi-output case is defined in the same way as before, i.e., by its constituents DIFF minterms. In this case, the

Table 3.2: Example of DIFF groups.

| DIFF min. | $F$ | $F_M$ | $CET$ | Group |
|---|---|---|---|---|
| $x_1x_0$ | $y_1y_0$ | $y_1y_0$ | $y_1y_0$ | # |
| 00 | {01} | {10} | $\{ET = 01, ET = 10\}$ | 1 |
| 01 | {00} | {11} | $\{ET = 01, ET = 01\}$ | 2 |
| 10 | {10} | {00} | $\{ET = 10, ET = 00\}$ | 3 |
| 11 | {11} | {01} | $\{ET = 10, ET = 00\}$ | 3 |

minterms of a correction function belong to the same DIFF group. Each group contains minterms with identical error behavior on all outputs and thus logic minimization of each correction function individually allows finding the least cost ways of carrying out the same change to function $F_{M,r}$. The result of the above definition is that for a multi-output function with $k$ outputs, there may be up to $3^n$ distinct correction functions.

Each correction function is minimized using two-level Boolean minimization. The standard Boolean minimization tool ESPRESSO is used to generate a minimum cover of all DIFF minterms within each correction function. Algorithm 2 summarizes the procedure of getting the DIFF groups and DIFF primes.

**Example 3.2.3.** See Table 3.3, where the shaded DIFF prime covers the two shaded DIFF minterms in Table 3.2.

### 3.2.3 Function Updates and Cost Calculation

The algorithm repeatedly eliminates the best candidate DIFF primes in the current DIFF set and modifies the function $F_{M,r}$. The following se-

Table 3.3: Example of DIFF primes.

| DIFF prime | $CET$ | Group |
|---|---|---|
| $x_1x_0$ | $y_1y_0$ | # |
| 00 | $\{ET = 01, ET = 10\}$ | 1 |
| 01 | $\{ET = 01, ET = 01\}$ | 2 |
| 1- | $\{ET = 10, ET = 00\}$ | 3 |

**Algorithm 2**: Correction Function Minimization.

**Input**: Frequency unconstrained approximate function $F_M$
**Output**: DIFF primes for every correction function

```
// identify DIFF minterms and their error structure
foreach minterm in F do
    foreach output bit j do
        compare F and F_M;
        record error type ET at bit j: 00, 01, 10;
// determine DIFF groups and correction functions they
   define
group all DIFF minterms with identical error behavior;
// determine the DIFF primes
foreach DIFF group do
    call ESPRESSO to minimize the correction function for this
    DIFF group and return DIFF primes;
```

quence is thus executed repeatedly: (1) the best correction is identified, (2) the ON/OFF-sets of function $F_{M,r}$ are updated, and (3) all correction functions impacted by the current change are updated. The multi-output $F_{M,r}$ is algorithmically treated as a union of single-output functions whose ON/OFF-sets are defined individually. However, the procedure outlined in the previous section means that when the best DIFF prime is selected for a correction, the function $F_{M,r}$ needs to be updated *on all of its outputs*.

In the following, a related issue of efficient cost computation is discussed along with the update strategy. The restriction of the search space to the primes of correction functions reduces the number of possible solutions. Despite that, it is still necessary to evaluate candidates based on the specific increase in the literal count they produce.

The denominator of Equation (3.5) refers to how many DIFF minterms are simultaneously corrected by correcting the single DIFF prime $j$. It is easily computed as $2^{n-s}$, where $n$ is the number of input variables in function $F$ and $s$ is the number of literals in the DIFF prime $j$.

Unfortunately, evaluating the numerator is difficult. Only after the Boolean minimization on the updated $F_{M,r}$ is completed can one know the literal changes exactly, where a function update is the update of the ON/OFF-sets of $F_{M,r}$ for all outputs that are prescribed by the correction function currently being evaluated. However, since the cost computation needs to be done often, running a two-level minimizer for each evaluation is too expensive in terms of computation time. To address this issue, a proxy metric is proposed for estimating literal changes. One approximation that the proxy metric adopts is that the literal cost increase is the sum of literal cost increases for each output individually. In other words, an $n$-output function is treated as a collection of $n$ single-output functions. This is a conservative assumption that ignores the sharing of terms in the covers of multi-output functions.

Before the details of the function update strategy is described, it is necessary to introduce a basic encoding scheme for performing operations on

prime implicants.

**Definition 3.2.4. Positional-Cube Notation.** The *positional-cube* notation is a binary encoding of implicants. The symbols used in the input part are {0,1,-}. The positional-cube notation encodes each symbol by 2-bit *fields* as follows:

| | |
|---|---|
| Ø | 00 |
| 0 | 10 |
| 1 | 01 |
| - | 11 |

where the symbol Ø means none of the allowed symbols, i.e., the presence of Ø means this implicant is void and should be removed.

The proxy computation to estimate the corresponding literal changes depends on whether the candidate update is a *correct-to-0* or a *correct-to-1* update. The *correct-to-0* update strategy is first discussed. Consider estimating the literal changes for a candidate DIFF prime $p_i^{dif}$. First, the subset of primes in the current cover of $F_{M,r}$ is identified that has a non-zero intersection (where the **intersect** operation is defined in Def. 3.2.5) with $p_i^{dif}$. Let this subset be $P^f$ and denote each specific prime in this subset as $p_j^f$, where the uppercase $P$ indicates a set, and lowercase $p$ indicates a single prime. Let $p_j^{intr} = p_i^{dif} \cap p_j^f$ be the result of each intersection.

**Definition 3.2.5. Intersection of two implicants.** The *intersection* of two implicants is the largest cube contained in both. It is computed by the bitwise product using a positional-cube encoding. If the result contains Ø, i.e., a void implicant, the two implicants do not intersect.

To perform the update, all the primes in $P^f$ need to be modified, since they are modified after the removal of the candidate DIFF prime. For a *correct-to-0* update, it is required to keep all and only those minterms covered by the $p_j^f$ and not $p_j^{intr}$. This can be done by performing a **sharp** operation (defined in Def. 3.2.6) on $p_j^f$ and $p_j^{intr}$. The resulting prime(s) replace $p_j^f$ as the new prime(s). If the resulting prime is void, i.e., if $p_j^f = p_j^{intr}$, then $p_j^f$ is removed entirely.

**Definition 3.2.6. Sharp Operation.** The *sharp* operation, when applied to two implicants, returns a set of implicants covering all and only those minterms covered by the first one and not by the second one. The sharp operator is denoted by #. Let $\alpha = a_1 a_2 \dots a_n$, and $\beta = b_1 b_2 \dots b_n$, where $a_i, b_i, i = 1, 2, \dots, n$, represents their fields. The sharp operation can be defined as:

$$\alpha \# \beta = \begin{cases} a_1 b_1' & a_2 & \dots & a_n \\ a_1 & a_2 b_2' & \dots & a_n \\ \dots & \dots & \dots & \dots \\ a_1 & a_2 & \dots & a_n b_n' \end{cases} \tag{3.6}$$

Given the replacement of primes $p_j^f$ with the results of the sharp operation, let $N$ be the number of inputs in function $F$, $d_j$ be the cardinality of $p_j^f$ *XOR* $p_j^{intr}$, and $M_j$ be the number of variables for $p_j^f$. Then, the literal change $\delta L_j^{01}$ to correct an error of type $ET = 01$ on a single output is:

$$\delta L_j^{01} = (d_j - 1) \times M_j + d_j \tag{3.7}$$

*Proof.* Let the set of literals corresponding to $p_j^f$ be $X$. Then, $p_j^{intr}$ must have the form $Xa_1a_2 \ldots a_d$. (Obviously, $p_j^{intr} \subset p_j^f$, i.e., there are more variables in $p_j^{intr}$ than in $p_j^f$). Then the Boolean subtraction of $X - Xa_1a_2 \ldots a_d$ reduces to $Xa_1' + Xa_2' + \ldots + Xa_d'$. Equation (3.7) is acquired by counting the literal changes before and after the Boolean subtraction. □

To estimate the total changes in the literal count due to elimination of the DIFF prime $p_i^{dif}$, individual costs for every output that has an error of type $ET = 01$ are summed up. Let $h$ be the number of outputs with error of type $ET = 01$ and let the cardinality of $P^f$ be $l$:

$$\Delta L_i^{01} = \sum^{h} \sum^{l} \delta L_j^{01} \tag{3.8}$$

The following discusses the update strategy for a *correct-to-1* update and describes a way to efficiently estimate the literal changes after adding a new prime to $F_{M,r}$. It starts by finding a subset of primes $P^f$ of $F_{M,r}$ that are **adjacent** to the DIFF prime $p_i^{dif}$. Adjacency is an important criterion since it indicates that the selected primes can be merged to a larger prime (i.e., a prime with fewer literals). The adjacency information is acquired by using an **intersect** operator on $p_i^{dif}$ and $p_j^f$. If the result contains only one empty field ($\emptyset$), then the two implicants are adjacent. If $P^f$ is empty, i.e., there are no primes that are adjacent to the DIFF prime $p_i^{dif}$, then the literal change $\delta L_j^{10}$ is equal to the number of literals in $p_i^{dif}$ itself. However, the when $P^f$ is not empty two possibilities exist:

- $p_i^{dif}$ and $p_j^f$ together form a new single prime, which reduces the current literal counts;
- $p_i^{dif}$ becomes larger (has fewer literals) due to the adjacency with $p_j^f$;

Therefore, it is required to count the literal changes by selecting *one* $p_j^f$ that causes the minimum literal increase out of all primes in this $P^f$. To compute the literal increase for each pair of $p_j^f$ and $P^f$, the literals of the **consensus** of the two primes are evaluated, which are denoted by $p_j^{consen}$. Because the two primes intersect, there is only a single implicant for the consensus operation, which is defined in Def. 3.2.7. Then, the literal increase is computed as:

$$\delta L_j^{10} = L(p_j^{consen}) \tag{3.9}$$

**Definition 3.2.7. Consensus Operation.** The *consensus* operation is defined as follows. The consensus returns void when the two implicants have a distance larger than or equal to 2. The consensus returns a single implicant when the two implicants are adjacent. The consensus returns more than or equal to 2 implicants, when the two implicants are intersecting. The consensus operator is denoted by $\nabla$.

$$\alpha \ \nabla \ \beta = \begin{cases} a_1 + b_1 & a_2 b_2 & \dots & a_n b_n \\ a_1 b_1 & a_2 + b_2 & \dots & a_n b_n \\ \dots & \dots & \dots & \dots \\ a_1 b_1 & a_2 b_2 & \dots & a_n + b_n \end{cases} \tag{3.10}$$

The overall literal change due to a *correct-to-1* update is the sum of costs for all output bits with error type $ET = 10$. Notice that for each of the output bits, the minimum $\delta L_j^{10}$ out of all primes in $P^f$ is picked. For $h$ output bits with error type $ET = 10$, the overall literal increase as a result of an update due to $p_i^{dif}$ is:

$$\Delta L_i^{10} = \sum^{h} \min \{\delta L_j^{10}\} \tag{3.11}$$

Once the $p_j^{consen}$ with the minimum literal increase is identified, the resulting consensus prime is added to the erroneous output bit function of $F_{M,r}$. Importantly, if $p_j^f$ is covered by $p_j^{consen}$, it is removed. This happens under the first scenario considered above for the case of $P^f$ not being empty.

The overall literal change for a candidate DIFF prime $p_i^{dif}$ is the sum of $\Delta L_i^{01}$ and $\Delta L_i^{10}$:

$$\Delta L_i = \Delta L_i^{01} + \Delta L_i^{10} \tag{3.12}$$

So far the proposed proxy metrics has been discussed to estimate the literal changes due to the DIFF prime corrections. After eliminating a candidate DIFF prime, the corresponding primes in $F_{M,r}$ need to be simultaneously modified. This change affects the *cost* values of the remaining primes. Thus, it is necessary to update the *cost* values of all remaining DIFF primes that are *impacted* by the just-modified prime. To achieve this, dependency information is stored for each prime of $F_{M,r}$ that records all DIFF primes that use this prime to compute their cost values. Once a prime of $F_{M,r}$ is modified,

a list of the associated DIFF primes is immediately available for which cost updates are needed.

It is now a proper time to prove the Theorem 3.2.2, which is repeated here for the ease of reading.

**Theorem.** *For a single-output function $F$, the optimal set of minterms to add to ON/OFF-set at the minimum literal increase in the cover of function $F_{M,r}$ lies among the prime implicants of the minimum cover of correction functions $CT0$ and $CT1$.*

*Proof.* Consider a correction function $CT1$ and its minimum cover. Let $p_j$ be a prime in that cover. Let $M_j = \{m_1, m_2, \ldots, m_h\}$ be the set of minterms covered by $p_j$. The theorem is true if the literal increment of correcting any subset of $M_j$ is larger than correcting the entire $M_j$, i.e., the $p_j$. Letting $lit(\cdot)$ be the literal number in a cube, it is clear that $lit(p_j) < lit(M_{j,i})$ for any $i$. For $p_j$ that has error type $ET = 10$, the literal increment is smaller when adding $p_j$ to ON-set rather than any subset of $M_j$.

Now consider a correction function $CT0$ and its minimum cover. Let $p_j$ be a prime in that cover. To correct an error of type $ET = 01$, the minterms covered by $p_j$ are to be removed from the cover of $F_{M,r}$. For the prime implicants of $F_{M,r}$ that have non-zero intersection with $p_j$, the following holds: the larger is the intersection between $p_j$ and primes of $F_{M,r}$, the fewer non-overlapping variables there are between $p_j$ and a prime of $F_{M,r}$. This results in smaller $d_j$ in Equation (3.7) and hence a smaller literal increment. □

A complete description of GALS algorithm is in Algorithm 3.

**Algorithm 3**: Approximate logic synthesis algorithm.

**Input**: frequency-unconstrained approximate Boolean function
**Output**: minimized boolean function with constrained error magnitude and error frequency

```
// get the DIFF primes
call "Correction Function Minimization" subroutine;
// initialize the current solution, k is the initial
   error frequency by BR solver
F_{M,r} = F_{M,k};
// get the initial error count for F_{M,r}
ErrorCount = k;
if ErrorCount ≤ Error Frequency Constraint then
    return F_{M,r};
// initialize the Cost-List
foreach DIFF prime p_i do
    compute the cost and push p_i to Cost-List;
    foreach prime in F_{M,r} that are associated with p_i do
        push p_i to association list of this prime;
sort Cost-List by cost values with ascending order;
// main loop
while ErrorCount > Error Frequency Constraint do
    pop the DIFF prime with least cost value in Cost-List;
    modify the F_{M,r} after eliminating this DIFF prime;
    update all associated DIFF primes due to modifying the F_{M,r};
    Sort Cost-List by cost values with ascending order;
    update ErrorCount;
return F_{M,r};
```

## 3.3 Experimental Results

The GALS is implemented in a C++ environment using BREL [44] as the embedded BR solver engine for the first phase of the algorithm. To evaluate the capability of GALS for significant literal reductions under general magnitude and frequency constraints, GALS has been used to generate a range of approximate solutions of adders and multipliers. All experiments were performed on an Intel 3.4GHz Core i7 workstation.

$$F_1 = \begin{cases} C = a_1 b_1; \\ S_1 = a_1 b_1' + a_1' b_1; \\ S_0 = 1; \end{cases}$$

$$F_{1,\frac{2}{16}} = \begin{cases} C = a_1 b_1; \\ S_1 = a_1 b_1' + a_1' b_1 + a_0 b_0; \\ S_0 = a_1 b_1' b_0 + a_1' b_1 b_0 + a_0 b_0' + a_0' b_0; \end{cases}$$

$$F_{1,\frac{1}{16}} = \begin{cases} C = a_0 b_1 b_0 + a_1 b_1; \\ S_1 = a_1' b_1 b_0' + a_1' a_0' b_1 + a_0 b_1' b_0 \\ \qquad + a_1 a_0 b_0 + a_1 b_1'; \\ S_0 = a_1 b_1' b_0 + a_0 b_0' + a_0' b_0; \end{cases}$$

$$F = \begin{cases} C = a_0 b_1 b_0 + a_1 a_0 b_0 + a_1 b_1; \\ S_1 = a_1' a_0 b_1' b_0 + a_1 a_0 b_1 b_0 + a_1 b_1' b_0' \\ \qquad + a_1' b_1 b_0' + a_1 a_0' b_1' + a_1' a_0' b_1; \\ S_0 = a_0 b_0' + a_0' b_0; \end{cases}$$

Figure 3.1: Synthesized 2-bit adder variants.

The basic operation of the algorithm is first demonstrated on a simple 2-bit adder example with 4 inputs ($a_1, a_0$ and $b_1, b_0$) and 3 outputs (sum bits

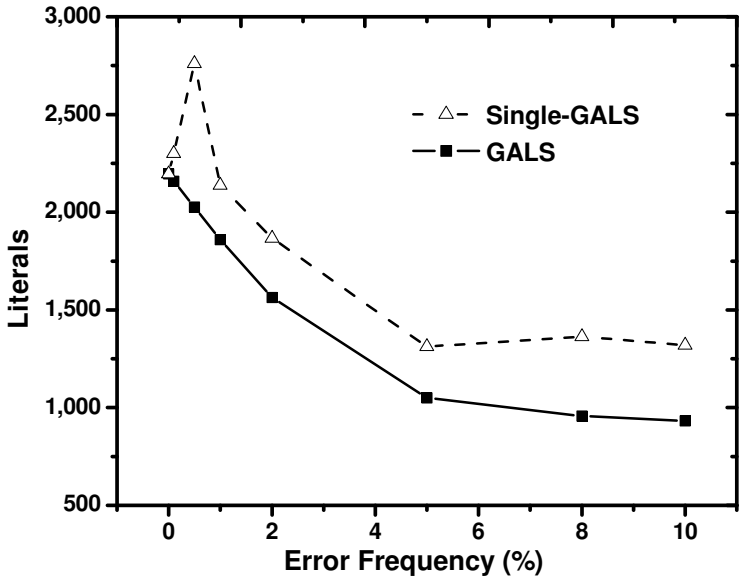



Figure 3.2: Effectiveness of GALS for 6-bit adders with a magnitude constraint of $M = 1$.

$S_0$ and $S_1$ and the carry $C$). Figure 3.1 shows the resulting logic equations for the exact adder ($F$), the frequency unconstrained solution ($F_M$), and both frequency and magnitude-constrained ($F_{M,R}$) approximate adder variants. In all cases, a magnitude constraint of $M = 1$ is applied. Error frequency constraints of one or two erroneous outputs out of the $2^4 = 16$ total minterms are also applied, i.e., $R = 1/16 = 6.25\%$ or $R = 2/16 = 12.5\%$. As expected, the frequency unconstrained solution ($F_1$) has the smallest literal count. The expression complexity increases with a decreasing frequency constraint. It is interesting to point out that the evolution of the logic does not follow an obvious pattern.

To further evaluate the effectiveness of the second-phase of GALS based

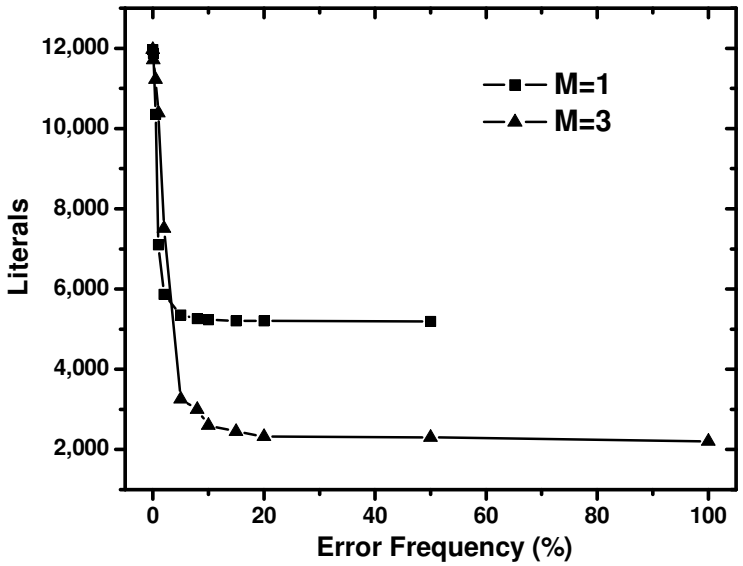



Figure 3.3: Synthesis results for 8-bit adders by GALS.

on Theorem 3.2.2, which operates with the primes of the correction functions, the performance of GALS is compared against an alternative implementation that greedily corrects only the single best minterm in each iteration. This alternative implementation is referred to as Single-GALS. Figure 3.2 plots the literal reductions achieved by both algorithms when applied to a 6-bit adder with a magnitude constraint of $M = 1$ and varying frequency constraints. Results validate the effectiveness of the proposed strategy: the GALS algorithm substantially outperforms the naive approach. On average, it produces 20% fewer literals while also being 37x faster.

Next, GALS is used to synthesize 8-bit and 10-bit approximate adders under magnitude constraints of $M = 1$ and $M = 3$ (Figures 3.3 and 3.4). Independent of the adder size, the frequency unconstrained solution at the

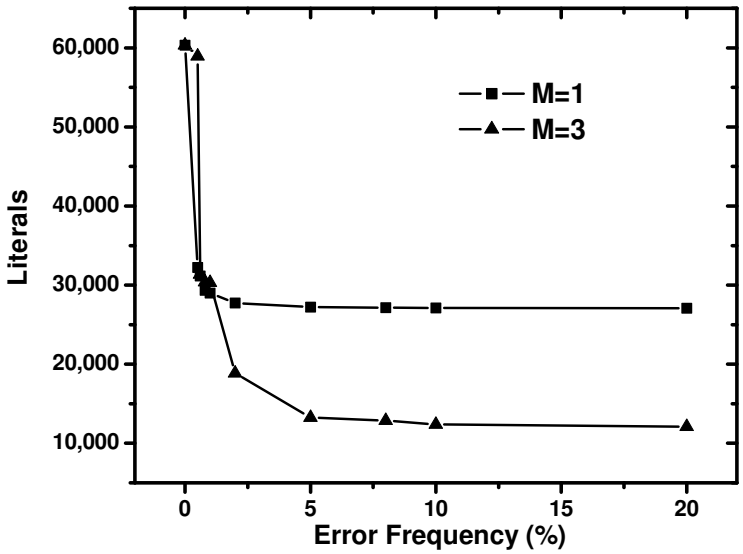
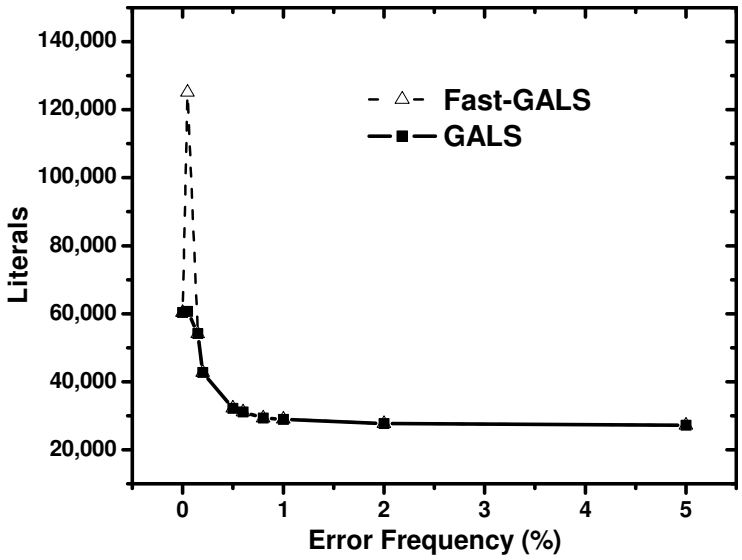
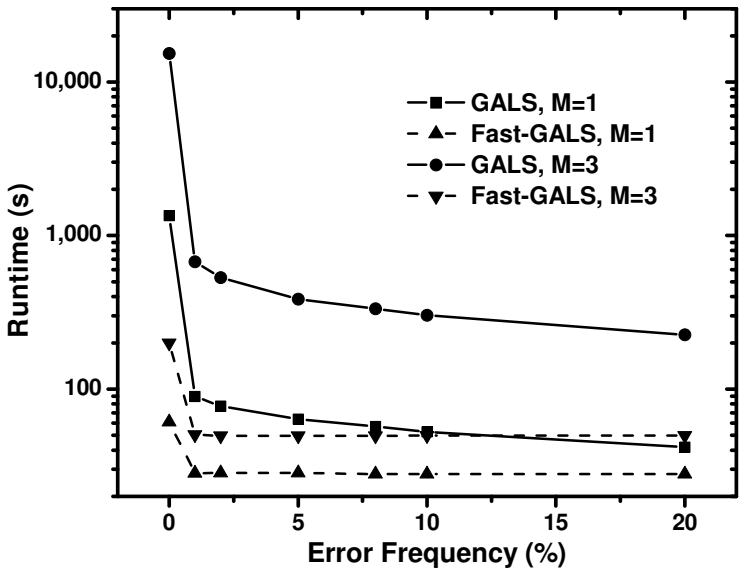



Figure 3.4: Synthesis results for 10-bit adders by GALS.

output of the first phase BR solver results in an error frequency of 50% and 100% at literal reductions of around 55% and 80% for $M = 1$ and $M = 3$, respectively. Results after further constraining error frequencies using GALS show that similar literal reductions can be maintained all the way down to error rates as low as 1-2%. Note that at extremely tight frequency constraints, literal counts of synthesized solutions for $M = 3$ grow faster than those for $M = 1$. The conjecture is that this is caused by the use of proxy cost metric in the second phase of GALS and the resulting sub-optimality of the greedy decision-making. Note that this effect is limited to only very small frequencies (below 0.6% for the 10-bit adder).




(a) Literals for 10-bit adders with $M = 1$.




(b) Runtime for 10-bit adders.

Figure 3.5: Comparison of GALS and Fast-GALS algorithms.

Runtimes for the first-phase BR solver range between 1s and 5s for 8-bits and between 50s and 2.5m for the 10-bit adder. Runtimes of the second-phase of GALS range between 2s and less than 5m for 8-bit adders, and between 30s and more than 3h for 10-bit designs. To further reduce runtime, a speed-up technique is investigated. One of the computationally expensive steps in GALS is the cost-updating routine that is repeatedly executed in the main loop of the algorithm (lines 14 - 15 in Algorithm 3). It is observed that using the Cost-List that is initialized once but is not updated on every iteration leads, in most cases, to a relatively small loss of optimality in the choice of a DIFF prime to be removed. Yet the runtime of the second-phase of the algorithm can be reduced significantly. Such a Fast-GALS algorithm is developed. Results of the comparison between GALS and Fast-GALS for the 10-bit adder are shown in Figure 3.5. It is observe that in most cases, resulting literal counts are very close while the runtime of Fast-GALS is one order of magnitude lower than GALS (Figure 3.5b). At tight frequency constraints, however, the cost updating mechanism plays a vital role. As a result, at very low frequencies, Fast-GALS produces solutions substantially worse than those of GALS, and in some cases even worse than the exact solution.

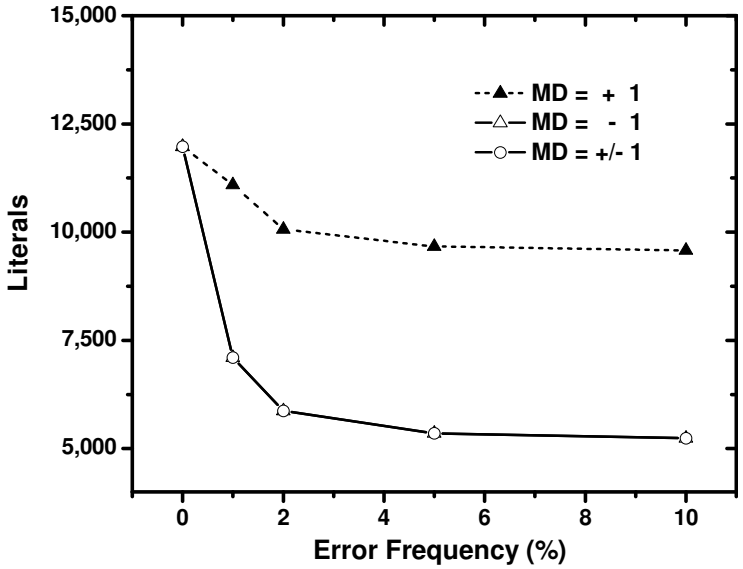



Figure 3.6: 8-bit adders under different error directions by GALS.

Depending on the application, not only error magnitude but also the error direction can be of importance. GALS supports setting different output relations during the first BR solving phase. Experiments were constructed on 8-bit adders in which the direction of error is further constrained to be only positive or negative. Results are shown in Figure 3.6, where *MD* is the value of the allowed adder error. It can be observed that for addition logic, allowing negative errors (exclusively or combined in both directions) results in circuits that are synthesized to have smaller literals.

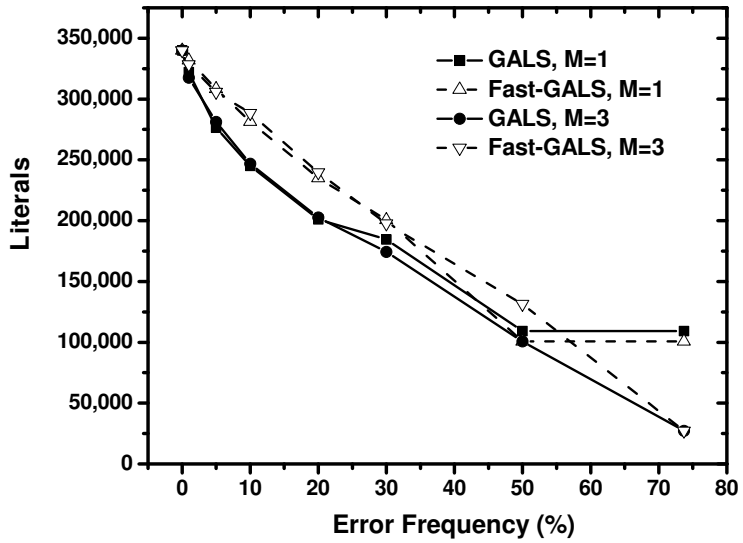



Figure 3.7: Synthesis results for 8-bit truncated multipliers.

Finally, GALS and Fast-GALS algorithms were applied to the larger test-case: an 8-bit truncated multiplier (Figure 3.7). Runtimes for the first-phase BR solver range between 4m and 5m. Runtimes for the two algorithms range between 20m and 3.3h for GALS and between 5m and 13m for Fast-GALS. Fast-GALS produces solutions that can be up to 20% worse in terms of literal count to the ones obtained with the slower GALS algorithm. Note that in the case of the multiplier, there is a significant dependence of literal count on error frequency over nearly entire range of frequencies. This further motivates the need for an application-specific synthesis solution.

## 3.4 Summary

This chapter presents a heuristic approach for solving a general two-level approximate logic synthesis problem. The error magnitude-only constrained problem is first addressed by casting it to a Boolean relation minimization, which is solved using recently proposed fast algorithms. The frequency-constrained problem is further solved by a novel greedy algorithm that finds the optimal set of function minterms on which the exact outputs must be enforced, and systematically corrects erroneous outputs until a given error frequency constraint is met. The proposed algorithm is capable of synthesizing approximate circuits for arbitrarily specified error deviations, and is most immediately applicable to arithmetic blocks, for which experiments demonstrate the effectiveness in achieving significantly reduced literal counts across a wide range of flexible error frequency and magnitude constraints.

# Chapter 4

# Multi-level Approximate Logic Synthesis

Chapter 3 demonstrated a strategy for two-level synthesis of approximate logic circuits under both magnitude and frequency types of constraints. However, an optimal two-level solution will not necessarily lead to an optimal multi-level Boolean implementation. This is especially important for complex logic blocks, such as arithmetic units, that are typically realized through carefully tuned macro libraries instead of being synthesized from their original Boolean function.

This chapter [1] addresses the problem of multi-level approximate logic synthesis (MALS) under arbitrary error magnitude and error frequency constraints. A heuristic is developed which effectively synthesizes approximate Boolean networks with reduced gate count whose errors deviate from an exact network in a constrained way. The magnitude constrained MALS is formulated using Boolean relations to capture the allowed error behavior. This formulation is more general than relying on incompletely specified functions and leads to better solutions. The proposed strategy uses network simplifications allowed by EXDCs. The core contribution is an algorithm that identifies

[1] This Chapter is based on the previous publication [45]. Conceptual ideas were discussed with the co-authors Michael Orshansky and Andreas Gerstlauer.

an EXDC set that maximally approaches the Boolean relation. It starts with an EXDC set that is overly relaxed and iteratively, and in a greedy fashion, identifies an optimal EXDC set by solving a series of conventional EXDC-based network optimizations. The algorithm then ensures compliance to error frequency constraints by recovering the correct outputs on the sought number of error-producing inputs while aiming to minimize the network cost increase. A novel network cost minimization principle is introduced for dealing with the multi-output case. It is based on the observation that in many networks there is a variation in the degree to which network outputs are dependent on the rest of the network. It is reasonable to expect that enforcing correctness on outputs with lower *embeddedness* leads to a lower network cost increase as it requires modifications to a smaller region of the network.

The algorithm is applied to several well-known adder and multiplier designs of varying bit-width. Even for small error magnitudes, the algorithm produces networks with gate count reduced by 30-50%, when the error frequency constraint is loose. This is up to 20% fewer gates than a naive EXDC-based approach.

## 4.1 MALS Formulation

Consider an $n$-input, $k$-output combinational logic network $G$ realizing a Boolean function $F$: $B^n \rightarrow \{0, 1, -\}^k$, where $-$ refers to a don't care. A multi-level approximate logic synthesis problem is concerned with formally synthesizing a minimum-cost (gate count) network whose behavior deviates

in a controlled manner from the specified exact Boolean function $F$. The deviations can be specified in terms of error magnitude and error frequency. The error magnitude constraint specifies the outputs that the approximate circuit is allowed to produce for each input $x_i \in B^n$. The total number of inputs that produce approximate outputs is described by the *error frequency* constraint. $G_{m,r}$ is denoted to be an approximate version of $G$ with $r$ inputs in error and with the largest magnitude of error being $m$. Let $R$ be the maximum number of inputs allowed to be in error and let $M$ be the maximum allowed deviation for a given output. It is assumed that a circuit produces a multi-bit output for which an arithmetic distance metric can be used to establish the degree of difference between the outputs. The notation $|G_{m,r}(x_i) - G(x_i)|$ is used to represent the absolute value of the arithmetic difference of the outputs produced by an exact and an approximate network. With this, the full multi-level approximate logic synthesis problem is:

$$\begin{aligned} \min \quad & C(G_{m,r}) \\ \text{s.t.} \quad & |G_{m,r}(x_i) - G(x_i)| \leq M, \quad \forall x_i \in B^n \\ & r \leq R \end{aligned} \tag{4.1}$$

where $C(G_{m,r})$ is the cost function, taken to be the gate count of $G_{m,r}$.

The problem defined in (4.1) captures the general constraints of both error types. The primary goal in problem (4.1) is to utilize the alternative outputs to simplify the circuit complexity when the outputs are constrained to take on *specific* patterns.

## 4.2 MALS under Error Magnitude and Frequency Constraints

Solving the problem of (4.1) when the error frequency constraint is not imposed will be first discussed. As shown in [38], the error magnitude constraint in (4.1) can be viewed as implicitly defining a *Boolean relation* (BR), where each input can be mapped to multiple outputs. A Boolean relation is a generalization of an incompletely specified Boolean function (ISF). It allows capturing a wider class of constraints on the outputs and thus allows for better solutions. Boolean relation minimization has been previously studied in the context of two-level optimization with the goal of identifying the minimum-cost two-level realization of a Boolean function compatible with a given Boolean relation [39, 43, 46]. This work focuses on simplifying an initial multi-level Boolean network by exploiting the flexibility captured by the Boolean relation. Such optimizations on an existing Boolean network are especially important for arithmetic blocks, which are often available directly in canonical form (e.g., various prefix adders).

There are currently no effective techniques to *directly synthesize* a Boolean network that satisfies a given Boolean relation. Therefore, the well-known principle of using external don't cares to simplify Boolean networks is adopted. In the context of approximate synthesis, such an approach has been proposed in [32]. In [32], the external don't care (EXDC) sets are based on conventional single output don't care extraction. However, a single-output approach does not exploit the full flexibility that may be permitted by the

error specification. For instance, a function that allows two outputs {11, 00} on some input can never be captured via single-output don't cares.

The contribution is in demonstrating that it is possible to find *non-trivial better* sets of external don't cares to drive multi-level optimization. The essence of the algorithm is that *it identifies an EXDC set that maximally approaches the Boolean relation.* Specifically, the algorithm starts with an EXDC set that is overly relaxed and iteratively, and in a greedy fashion, identifies an optimal EXDC set by solving a *series* of conventional EXDC-based network optimizations. The original Boolean network is ultimately minimized by using the optimal EXDC. To drive the algorithm, the multi-level synthesis tool is used as a black box. It is relied on the tool's ability to exploit the flexibility offered by EXDC. (As detailed later, SIS [47] is used for network optimization.)

The central challenge is to identify an EXDC set that maximally approaches the Boolean relation. Such a set has the following properties. First, it maximally shares the flexibility for network simplification described by the Boolean relation. Because the sought EXDC set is based on single-output don't cares, the difference between a *relaxation of a BR into a multi-output ISF* and a candidate EXDC as a measure of such flexibility is used. Second, it defines a function compatible with BR. This EXDC set is called a Relation-Aware EXDC (RA-EXDC) set.

An outline of the algorithm is as follows. The algorithm to find RA-EXDC is based on a relax-and-recover strategy: the error constraints of the

original problem are first relaxed, captured via a BR, to produce an EXDC set that permits more than the original deviations specified by BR. This EXDC is an upper bound for RA-EXDC. Because the network simplified by this EXDC violates the original error constraints, an iterative recovery procedure is employed to minimally refine EXDC until violations are removed. Along with the upper bound, it is shown that a lower bound on the sought EXDC is available and can also be extracted from BR. Both bounds are used during the recovery phase. Next, the strategy to compute both bounds will be discussed.

### 4.2.1 Extracting Lower and Upper Bounds

The interest lies in finding the least upper bound and the greatest lower bound. The least upper bound EXDC is the minimum superset of BR that can be expressed using ISFs. As the lower bound for the extraction of the optimum EXDC, the maximum subset of BR that can be expressed as an ISF and that contains the original function is utilized. In doing this, it is assumed that such an ISF provides the largest flexibility for network simplification *among all possible ISFs*. Note that the stated principle for finding the "best" ISF was used earlier in [39]. The choice of this ISF as a *useful* bound is justified via an assumption of a monotonic relation between the size of the EXDC and the potential complexity reduction of a network under this EXDC.

Some preliminary definitions are first provided. Hereafter, it is indistinctly to use the terms of incompletely specified function (ISF) and the corresponding don't care set described by this ISF.

A Boolean relation can also be specified by a characteristic function $R : B^n \times B^m \rightarrow B$, such that $(x, y) \in R \iff R(x, y) = 1$. Here, the $B^n$ and $B^m$ are the input and output sets of $R$, respectively. The characteristic function $R$ is used for the following discussion.

**Definition 4.2.1. MISF.** A multi-output ISF (MISF) is a function $f : B^n \rightarrow (B \cup \{-\})^m$, which is a vector of ISFs $f = (f_1, f_2, ..., f_m)$.

**Definition 4.2.2. Natural join [46].** The natural join over the input set $X$ between two relations $R$ and $S$ is defined as

$$R(X, Y) \bowtie_X S(X, Z) = \{(x, y, z) | (x, y) \in R \wedge (x, z) \in S\} \tag{4.2}$$

where $X = (x_1, x_2, \ldots, x_n)$ is used to denote the set of inputs and $Y = (y_1, y_2, \ldots, y_m)$ and $Z = (z_1, z_2, \ldots, z_m)$ for outputs of two relations, respectively.

**Definition 4.2.3. Projection of a Boolean relation [46].** The projection of a relation $R(X, Y)$ onto an output $y_i$ is another function $(R \downarrow y_i)$ such that

$$\begin{aligned}(R \downarrow y_i) = &\{(X, z) | \exists y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m \\ &\text{such that } (X, y_1, \ldots, y_{i-1}, z, y_{i+1}, \ldots, y_m) \in R\}.\end{aligned} \tag{4.3}$$

The projection of a relation $R$ onto an output $y_i$ defines an ISF for that output.

**Definition 4.2.4. MISF covering a Boolean Relation [46].** For a given Boolean relation $R$, an MISF covering R can be obtained as follows:

$$MISF_R(X, Y) = \bowtie_{X \atop i \in \{1,\ldots,m\}} (R \downarrow y_i) \tag{4.4}$$

It can be proved [46] that the $\text{MISF}_R$ is the minimum superset of Boolean relation $R$ that can be expressed by ISFs, i.e., an MISF. Hence, the least upper bound EXDC is $\text{MISF}_R$. $\text{MISF}_R$ is referred to as an over-approximated EXDC (O-EXDC).

A simple example is given to illustrate the extraction of O-EXDC from the Boolean relation table. Consider a 2-input, 3-output multi-level Boolean network with error magnitudes as specified in Table 4.1. The first output in each row refers to the correct output of the network.

Table 4.1: Boolean relation

| Inputs | Outputs |
|---|---|
| $x_1, x_0$ | $y_2, y_1, y_0$ |
| 00 | {000, 001} |
| 01 | {010, 011, 100} |
| 10 | {100, 101, 010} |
| 11 | {110, 100, 101} |

It takes two steps to derive O-EXDC.

1) Project the Boolean relation onto each output $y_i, i = 0, 1, 2$. (see Table 4.2).

Table 4.2: Projections of BR on $y_i$

| $x_1, x_0$ | $R \downarrow y_2$ | $R \downarrow y_1$ | $R \downarrow y_0$ |
|---|---|---|---|
| 00 | {0} | {0} | {0,1} |
| 01 | {0,1} | {0,1} | {0,1} |
| 10 | {0,1} | {0,1} | {0,1} |
| 11 | {1} | {0,1} | {0,1} |

2) Perform a natural join on each projection of $(R \downarrow y_i)$ to form $\text{MISF}_R$. The {-} implies a don't care condition. The don't care condition is encoded by a "1" and a care condition by a "0" to form O-EXDC, as shown in Table 4.3. For example, O-EXDC for $y_2$ is $\{x_1^{'}x_0, x_1x_0^{'}\}$, O-EXDC for $y_1$ is $\{x_1x_0, x_1^{'}x_0, x_1x_0^{'}\}$, and O-EXDC for $y_0$ is the full set of inputs.

Table 4.3: $\text{MISF}_R$ and corresponding O-EXDC

| Inputs | $\text{MISF}_R$ | | | O-EXDC | | |
|---|---|---|---|---|---|---|
| $x_1, x_0$ | $R \downarrow y_2$ | $R \downarrow y_1$ | $R \downarrow y_0$ | $y_2$ | $y_1$ | $y_0$ |
| 00 | 0 | 0 | - | 0 | 0 | 1 |
| 01 | - | - | - | 1 | 1 | 1 |
| 10 | - | - | - | 1 | 1 | 1 |
| 11 | 1 | - | - | 0 | 1 | 1 |

Note that O-EXDC allows greater flexibility than BR. In Table 4.3, since all output bits are insensitive to input 01, the output of, for instance, 111 is also allowed by O-EXDC but not by Boolean relation in Table 4.1.

The greatest lower-bound EXDC that can be extracted from BR is discussed as follows. As stated above, the maximum subset of BR that can be expressed by an ISF and that contains the original exact function is used. It can be acquired by the following procedure:

1) For an input $x$, there is an output set $Y : (x, Y) \in R$. Identify the maximum prime implicant $p_x$ of set $Y$ that contains the original output. For instance, consider an input $x$ and $Y$={001, 011, 111}: $p_x$={0-1}. The prime $p_x$ provides the maximum number of output bits insensitive to $x$.

2) Repeat the above step for all $x \in B^n$. Hence, a set of independent ISFs is acquired, each for one output bit. These ISFs together form an MISF $f$.

**Theorem 4.2.1.** *$f$ is a maximum subset of BR that can be expressed as an MISF.*

*Proof.* First, $f$ is contained by BR needs to be proved. Since $p_x \subseteq Y, \forall x \in B^n$, then $R(x, p_x) = 1, \forall x \in B^n$. By definition, the MISF $f : x \to p_x, \forall x \in B^n$, therefore $R(X, f) = 1$.

Next, $f$ is the maximum subset of BR is proved by contradiction. Assume there is another MISF $h$ with a larger cardinality than $f$, i.e., $|f| < |h|$. Here, the cardinality refers to the number of don't cares contained by the MISF $h$. Denote $q_x$ as the prime implicant of set $Y$ that contains the original output for $h$. By definition, $|p_x| \geq |q_x|, \forall x \in B^n$. Hence, $\prod_{x \in B^n} |p_x| \geq \prod_{x \in B^n} |q_x|$. Consider $|f| = \prod_{x \in B^n} |p_x|$ and $|h| = \prod_{x \in B^n} |q_x|$, there is a contradiction against the assumption. Therefore, $f$ is the maximum subset of BR. □

The DC-set of $f$ is denoted as a conservative EXDC (C-EXDC). The relation between C-EXDC, RA-EXDC, and O-EXDC is summarized as follows:

$$\text{C-EXDC} \subseteq \text{RA-EXDC} \subseteq \text{O-EXDC} \tag{4.5}$$

Note that there is no obvious containment relation between RA-EXDC and the original Boolean relation. Table 4.4 illustrates the relation between C-EXDC, RA-EXDC, and O-EXDC through an example.

Table 4.4: Boolean relation, C-EXDC, and O-EXDC

| Inputs | Boolean Relation | C-EXDC | O-EXDC |
|---|---|---|---|
| $x_1, x_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ |
| 00 | {000, 001} | 001 | 001 |
| 01 | {010, 011, 100} | 001 | 111 |
| 10 | {100, 101, 010} | 001 | 111 |
| 11 | {110, 100, 101} | 010 | 011 |

### 4.2.2 Recovering Magnitude Conflicts

Optimizing the initial network with O-EXDC results in a minimal complexity network for the magnitude-only constrained MALS problem. However, the resulting network may produce outputs not allowed by the original Boolean relation. Inputs for which the error constraint is violated are referred to as *conflict inputs*. Notice that the network simplified by C-EXDC is guaranteed to be free of conflicts. This indicates that all conflicts fall in the *complement of C-EXDC in O-EXDC*, i.e., *O-EXDC*$\setminus$*C-EXDC*. A notion of *candidate inputs* is introduced to denote those inputs that can potentially cause conflicts.

**Definition 4.2.5. Candidate inputs.** A candidate input $x$ is defined as $\{x | x \in$ *O-EXDC* $\setminus$ *C-EXDC*$\}$.

Each candidate input $x$ may correspond to more than one output with possible conflicts. These outputs together form the *candidate outputs* for input $x$. For example, in Table 4.4, $y_1, y_2$ are candidate outputs for input 10.

It is important to see that not every candidate input is a conflict input. In other words, a network simplified with an EXDC set that contains a candidate input will not necessarily produce a conflict on this input. That happens

for two reasons. First, not every allowed flexibility is actually utilized for simplifying the network. Whether a flexibility is eventually utilized depends on the network structure. Second, even if the changes are made, the simplified network may produce outputs compatible with the Boolean relation and thus remain conflict-free. For example, in Table 4.4, candidate input 10 may be used to simplify the network and result in output 010 which is compatible with the original Boolean relation.

**Algorithm 4**: MALS under general error constraints

**Input**: $NL$: Original Boolean network, $BR$: Error magnitude constraint, $R$: Error frequency constraint, $N$: input bit #

**Output**: Minimized Boolean network under constrained errors

$EXDC_n = EXDC_o = \text{MISF}_\text{R}(BR)$; $EXDC_c = \text{ISF}(BR)$;
$NL_n = \text{Optimize}(NL, EXDC_o)$;
$Conf = \{x_i | NL_n(x_i) \notin BR(x_i), x_i \in B^n\}$;
$r = |\{x_i | NL_n(x_i) \neq NL(x_i), x_i \in B^n\}|/2^n$; $j = 0$;
**while** *(*$Conf \neq \Phi$*) or (*$r > R$*)* **do**
  **while** $Conf \neq \Phi$ **do**
    $y_{approx} = NL_n(x_i)$;
    **foreach** $x_i : NL_n(x_i) \notin BR(x_i)$ **do**
      $a = BR(x_i)$;
      **do**
        $y_{allow} = \text{argmin}_a \text{Hamming}(y_{approx}, a)$;
        $y_{remov} = (y_{allow} \oplus y_{approx}) \wedge \text{Candidate}(EXDC_o, EXDC_c)$;
        $a = a - y_{allow}$;
      **while** $y_{remov} \neq 0$ ;
      **foreach** non-zero bit $b$ of $y_{remov}$ **do**
        $EXDC_n \leftarrow$ Remove input $x_i$ from output $b$;
    $NL_n = \text{Optimize}(NL, EXDC_n)$;
    $Conf = \{x_i | NL_n(x_i) \notin BR(x_i)\}$;
  $r = |\{x_i | NL_n(x_i) \neq NL(x_i)\}|/2^n$;
  $k =$ Difference inputs # for output bit $j$;
  **while** $r > R$ **do**
    $c = 0$; Flag = True;
    **foreach** $x_i : (NL_n(x_i)[j] \neq NL(x_i)[j])$ **do**
      $EXDC_n \leftarrow$ Remove input $x_i$ from the $j^{th}$ bit;
      $c = c + 1$;
      **if** $c > \alpha k$ **then**
        Flag = Flase; Break;
    **if** *Flag* **then**
      $j = j + 1$;
    $NL_n = \text{Optimize}(NL, EXDC_n)$;
    $Conf = \{x_i | NL_n(x_i) \notin BR(x_i)\}$;
    **if** $Conf \neq \Phi$ **then**
      Break;
    $r = |\{x_i | NL_n(x_i) \neq NL(x_i)\}|/2^n$;
**return** $NL_n$;

Table 4.5: Error magnitude recovery example

| Inputs | Outputs | C-EXDC | O-EXDC | Approx. Outputs | Candidate Outputs | Updated EXDC |
|---|---|---|---|---|---|---|
| $x_1, x_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ |
| 00 | {000, 001} | 001 | 001 | 001 | 000 | 001 |
| 01 | {010, 011, 100} | 001 | 111 | 101 | 110 | 001 |
| 10 | {110, 101, 010} | 100 | 111 | 101 | 011 | 111 |
| 11 | {011, 111, 110} | 100 | 111 | 101 | 011 | 101 |

The following discusses how O-EXDC is modified to remove conflicts. First consider the example in Table 4.5. Suppose that after simplifying the original Boolean network using O-EXDC, the shaded two inputs produce outputs beyond the error magnitude constraint. In order to remove these conflicts, it is necessary to remove the flexibility currently given to the candidate outputs of the shaded inputs; for brevity, this operation is denoted as removing these candidate inputs from their candidate outputs. For instance, input 11 produces output 101, which conflicts with the error specification. It is observed that O-EXDC and C-EXDC differ for $y_1$ and $y_0$. Therefore, outputs $y_1$ and $y_0$ (but not $y_2$) are the candidate outputs for this input. Input 11 is, in turn, considered for removal from the EXDC of each of these outputs ($y_1$ and $y_0$).

However, it may not be necessary to remove 11 from don't care sets of both $y_1$ and $y_0$. It is assumed that the gate count of the simplified network increases monotonically with a decrease in EXDC. The goal of the algorithm is to ensure that some allowed output is produced while removing the least flexibility from EXDC, e.g., by reducing EXDC minimally in each iteration. That means it is required to change as few don't cares in EXDC for a candi-

date input as possible. This is achieved by aiming to match an allowed output that is least distinct from the output produced by the current network. The example of Table 4.5 illustrates this by showing that the number of conflicting outputs for a given input, say input 11, depends on the allowed output being considered. The current approximate output 101 has a difference in only one output when compared with 111 but it has two erroneous outputs when compared with 011 or 110.

Hamming distance provides the appropriate metric. Only the don't cares from the candidate input of those erroneous outputs that *intersect* with the candidate outputs are disallowed (removed). This guarantees the search space is restricted to lie within the given lower and upper bound, i.e., C-EXDC and O-EXDC. Since C-EXDC contains the original network outputs, there is always at least one allowed output, i.e., the correct output, that has a non-empty intersection with candidate outputs (in this case, the intersection contains all candidate outputs).

The strategy for selecting the candidates for removal from EXDC is summarized. For each input with conflicting outputs:

1) Find the Hamming distance between the approximate output and each allowed output.

2) Identify as target output the output with the minimum Hamming distance for which at least one output bit is contained in the output candidate list

(this is the output whose bitwise XOR difference from the approximate output also intersects with the candidate output set.)

3) Change all the output bits that are in the intersection from 2) to be sensitive to the input, i.e., remove the current input from the EXDC of each such output. Repeat this process for all inputs that produce conflicts.

Note that there can be multiple allowed outputs satisfying the strategy described in 2) and 3). The selected output is the one that is first found among the allowed outputs which are listed in an arithmetic ascending order. In Table 4.5, by comparing the approximate output 101 for input 01 with all allowed outputs $\{010, 011, 100\}$, it can be found that output 100 has the minimum Hamming distance with a difference only on $y_0$. However, $y_0$ is not a candidate output for this input. Therefore, 011 is selected as the targeted allowed output: it has the next minimal Hamming distance but it has outputs that are candidate outputs. In this case, the allowed output 011 differs from the current approximated output 101 in $y_1$ and $y_2$ both of which are in the candidate list. Therefore, $y_1$ and $y_2$ are modified to be sensitive to input 01, i.e., input 01 is removed from O-EXDC of $y_1$ and $y_2$. Similarly, input 11 is removed from O-EXDC of $y_1$. The non-shaded elements of EXDC remain the same as there are no other conflicts in this iteration.

Applying the updated EXDC set to the original Boolean network produces another version of the approximate Boolean network. However, while the conflict outputs found in the previous iteration have been removed, new

conflict outputs may be produced. Those conflicts are iteratively corrected by following the above steps. The algorithm clearly implements a greedy approach: in every iteration, *all conflicts manifest at this iteration* are corrected by making the minimum possible changes to EXDC. This heuristic solution strategy is observed to behave reasonably well. On the benchmarks that are utilized, it is observed that the number of iterations is typically small (3 to 4). Besides, the behavior is monotonic: with each iteration, as the number of conflict inputs is reduced, the network gate count monotonically increases. The algorithm stops when no conflict output exists. In the theoretical worst case, it stops when there are no candidate outputs left, i.e., O-EXDC is reduced to C-EXDC. On the benchmarks, it is observed that the algorithm stops earlier and produces a substantial improvement over using C-EXDC. Section4.3 provides more details of experiments and results. The above algorithm is summarized in Algorithm 4 up to Line 6.

An approximate network produced by the above algorithm is compatible with the original Boolean relation and satisfies the error magnitude constraint. Next subsection shows how to extend the above algorithm to solve the general MALS problem (4.1) that jointly considers error magnitude and frequency constraints.

### 4.2.3 Resolving Frequency Violations

The approximate Boolean network resulting from the above algorithm has an arbitrary error frequency (often close to 100%), and thus, typically,

violates the constraint. A recovery procedure is developed to produce a feasible solution with respect to error frequency.

Consider an input $x$ for which the simplified Boolean network produces an output different from the exact output.Such an input is called a *difference input* and the corresponding output a *difference output*. The error frequency reduction is based on recovering the correct outputs on some or all of the difference inputs while aiming to minimize the network cost increase. Corrections are achieved by updating EXDC to enforce the correct outputs on selected inputs.

The strategy to deal with a single-output case is discussed as follows. Suppose that the number of difference inputs is $N$ and the target error frequency is $R$. The goal is to identify at least $k = N - R$ difference inputs to correct out of the set of $N$ possible inputs. The fact that $N$ is typically very large and $R$ is small makes identifying a *good* set of $k$ inputs to correct very difficult. It is infeasible to try all possible sub-sets. A heuristic strategy is adopted that minimizes the dependence on the choice of a sub-set by picking an arbitrary smaller sub-set of $\alpha k$ inputs, where $\alpha$ is, typically, 0.2 to 0.5. Notice that aiming to correct $k$ inputs does not mean that the resulting network has exactly $R-k$ difference outputs since new errors may appear. Thus, depending on the error frequency constraint $R$, the procedure requires several, typically 6 to 10 iterations to remove all difference inputs.

The network cost minimization principle is further extended to handle multiple outputs. This is crucial, since an input remains a difference input as

long as one or more outputs are in error. In dealing with the multi-output case, it is observed that *in some cases*, forcing correctness on different outputs may predictably lead to different network cost increases. This is based on the fact that in many networks, there is a variation in the degree to which network outputs are dependent on the rest of the network. In other words, there is variation in the degree of *embeddedness* of an output in the network. This can be quantified by finding the ratio of the gates that are in the fan-in cone of an output over the overall network gate count. A higher ratio indicates a higher value of embeddedness. It is reasonable to expect that enforcing correctness on outputs with *lower* embeddedness leads to a lower network cost increase as it requires modifications to a smaller region of the network. This work is primarily concerned with arithmetic circuits where the degree of embeddedness is easily ascertained from basic arguments and therefore does not require explicit extraction. In arithmetic circuits, the outputs corresponding to the LSBs naturally have a lower degree of embeddedness. Therefore, the correctness is prioritized enforcing on network outputs corresponding to LSBs, which should lead to the smallest gate increase. The algorithm iteratively corrects difference inputs on outputs moving from LSB to MSB until the entire network satisfies the error frequency constraint. The error frequency recovery strategy is summarized:

1) For an output $i$, identify all of its $k_i$ difference inputs;

2) Remove $\alpha k_i$ fraction of difference inputs from the EXDC;

3) Repeat 2) until error frequency is satisfied or $k_i = 0$;

4) Repeat for output $i = i + 1$ from LSB to MSB until error frequency constraint is met.

See the example in Table 4.6 for an illustration of a single iteration of the algorithm. It is observed that altogether there are two difference inputs (00 and 10). Suppose the error frequency constraint is 25%. Therefore, in this iteration the aim would be to correct one input. Because the difference output for 00 is the LSB, this input is chosen to be corrected and $y_0$ is modified to be sensitive to input 00.

The complete approach is summarized in Algorithm 4.

Table 4.6: Error frequency recovery example

| Inputs | Exact Output | Old EXDC | Approx. Outputs | Updated EXDC |
|---|---|---|---|---|
| $x_1, x_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ | $y_2, y_1, y_0$ |
| 00 | {000} | 001 | 001 | 000 |
| 01 | {010} | 100 | 010 | 100 |
| 10 | {110} | 110 | 010 | 010 |
| 11 | {011} | 101 | 011 | 101 |

## 4.3 Experimental Results

The MALS algorithm is implemented in a C++ environment using ABC [48], SIS [47] and Design Compiler as the synthesis tools. To evaluate the capability of the proposed algorithm for significant gate count reduction under general error magnitude and frequency constraints, MALS is used to

generate a range of approximate solutions of different types of adders and multipliers with different bitwidth. All experiments were performed on an Intel 3.4 GHz workstation. Table 4.7 shows the circuit-specific information for the adders and multipliers that are used.

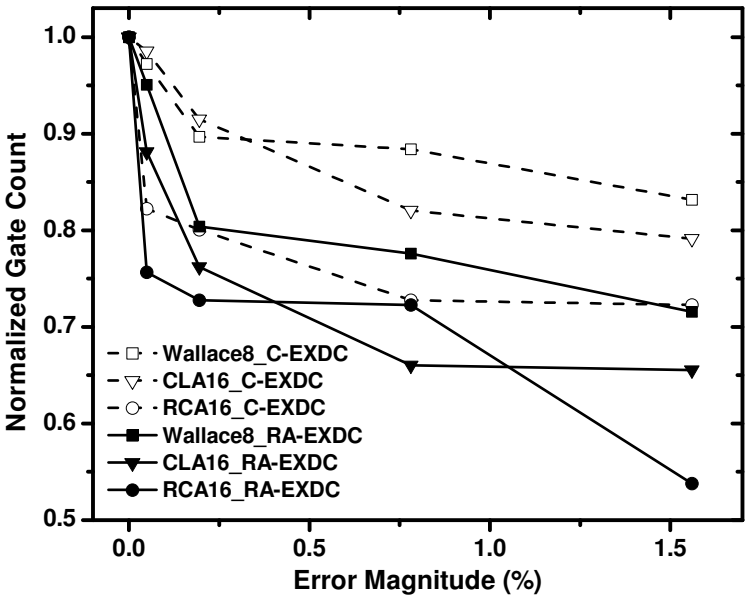
Table 4.7: Circuits used for MALS algorithm

| Name | Function | I/O | Gates |
|---|---|---|---|
| RCA8 | 8-bit Ripple Carry Adder | 16/9 | 323 |
| RCA16 | 16-bit Ripple Carry Adder | 32/17 | 411 |
| CLA16 | 16-bit Carry Lookahead Adder | 32/17 | 412 |
| KS16 | 16-bit Kogge Stone Adder | 32/17 | 465 |
| RCA32 | 32-bit Ripple Carry Adder | 64/33 | 834 |
| Wallace8 | 8-bit Wallace Multiplier | 16/16 | 1259 |
| Dadda8 | 8-bit Dadda Multiplier | 16/16 | 1128 |

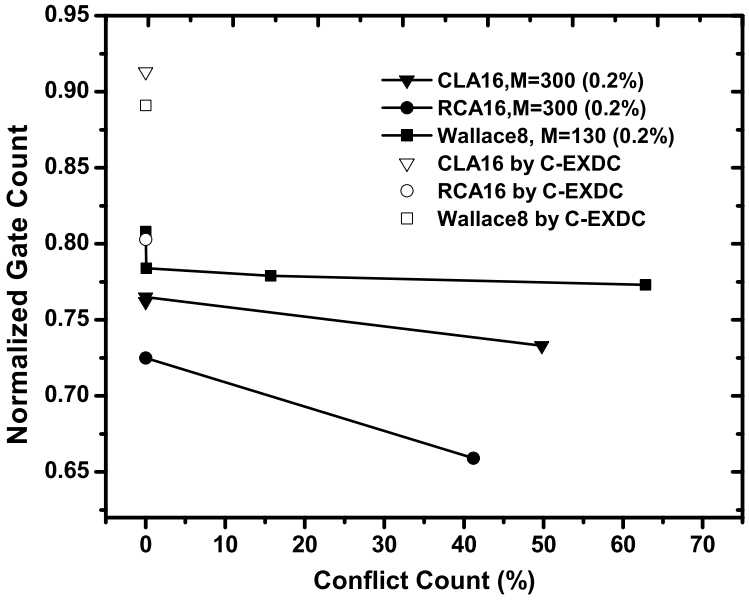
The MALS algorithm is applied to several types of adders and multipliers with different bitwidths (Table 4.7). For the 32-bit RCA, the algorithm is applied to the lower 18 bits. The runtime varies from a few seconds for a small adder to more than 20 hours for large adders and multipliers.

First, the effectiveness of the proposed MALS algorithm is demonstrated by comparing to networks synthesized using lower bound C-EXDC when only the magnitude of error is constrained. Figure 4.1(a) shows two types of 16-bit adders and an 8-bit Wallace multiplier synthesized by both the C-EXDC and RA-EXDC identified by the algorithm for several magnitudes of allowed error. The error magnitude is shown as the percentage of the maximum output value (since circuits have different bitwidths, errors of the

same absolute magnitude indicate varying relative significance). It is observed that the network simplified by the proposed RA-EXDC outperforms the approach using C-EXDC by up to 20% in terms of achieved gate count reduction. Figure 4.1(b) shows the gate count and conflict inputs changes over each iterations. Each curve refers to the iteration history of one network, where each point refers to one iteration. All iterations start from the O-EXDC (the point with the largest conflicts number) and monotonically reduce to zero conflicts. For the given benchmark circuits, the algorithm converged within 4 iterations in all cases. The solution identified by C-EXDC is also marked in this figure.




(a) Error magnitude sweep




(b) Gate count and conflicts over iterations

Figure 4.1: Networks simplified by C-EXDC and RA-EXDC

In order to give an intuition of how the network evolves over iterations, the successive optimizations performed on the fan-in cone of the sum bit 4 of a simple 8-bit RCA adder is shown in Figure 4.2. The process converges in iteration 3. Compared to an C-EXDC based approach, 4 gates are saved by using RA-EXDC.

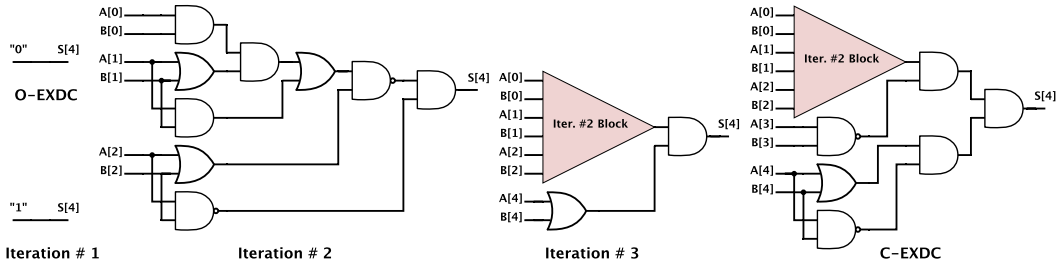



Figure 4.2: Logic cone of sum bit 4 in an 8-bit RCA as it changes over algorithm iterations

Figure 4.3, 4.4, and 4.5 show the results of synthesizing approximate networks jointly under both types of constraints. First each network is synthesized with error magnitude constraints equal to 300 and 1000 (corresponding to different relative error magnitudes). An error frequency sweep is further performed by running the algorithm and recording every possible frequency achieved during the error frequency recovery phase. Results show that depending on the error magnitude and circuit, gate count reductions ranging from 5% to 50% can be achieved if frequency is unconstrained. Achievable gate count reductions decrease with stricter error frequency constraints. The results indicate that in some cases, the space of solutions is sparse in terms of achievable error frequencies. Note that this sparsity reduces with increased

flexibility offered by larger error magnitude constraints.

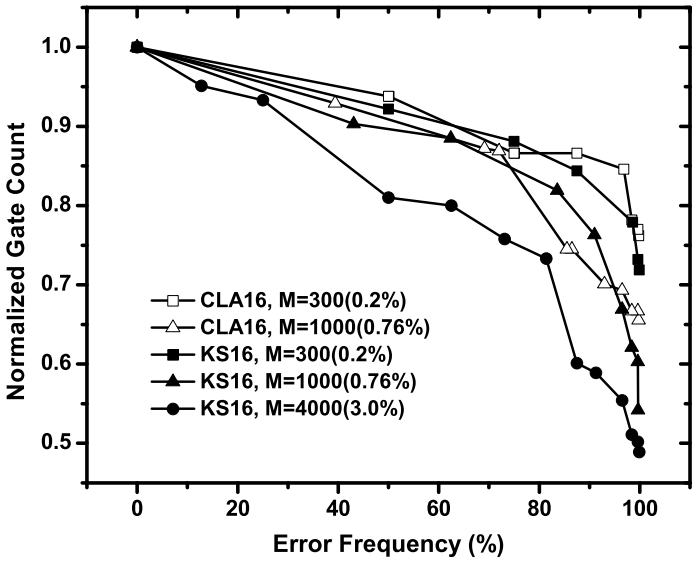



Figure 4.3: Error magnitude and frequency constrained solutions for 16-bit adders

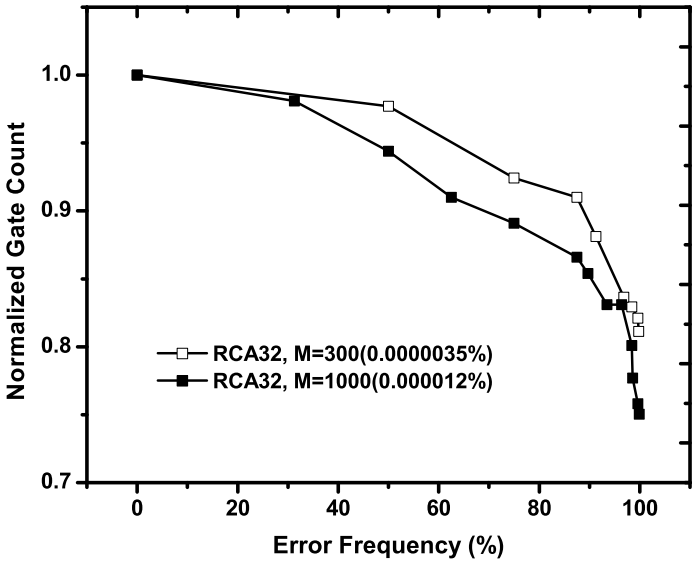



Figure 4.4: Error magnitude and frequency constrained solutions for 32-bit ripple carry adder

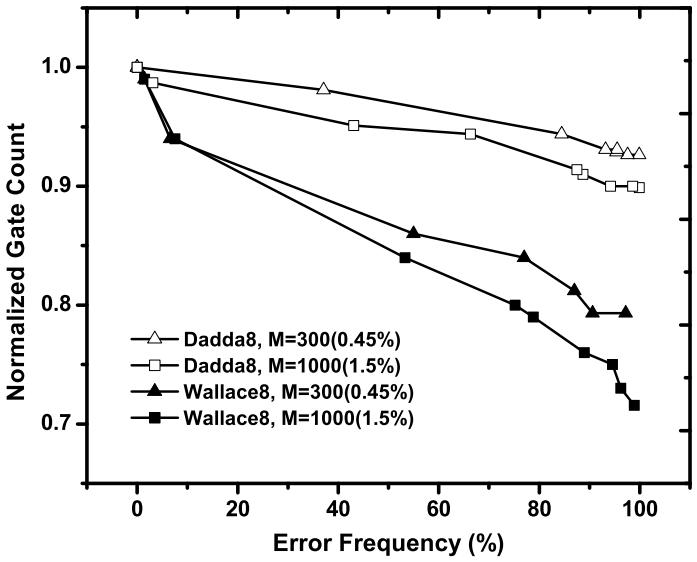



Figure 4.5: Error magnitude and frequency constrained solutions for 8-bit multipliers

## 4.4 Summary

This chapter addresses the multi-level approximate logic synthesis (MALS) problem under general error constraints. The error magnitude constrained MALS is formulated using Boolean relations to capture the allowed error behaviors. This formulation is more general, grants better flexibilities on error constraints and hence leads to better solutions than approaches based on incompletely specified functions. An algorithm is further presented to solve the MALS problem under general error magnitude and frequency constraints. The algorithm starts with a solution that is overly relaxed. In an iterative and greedy fashion, it then first identifies a solution satisfying the magnitude constraint by successively applying a series of less and less relaxed conventional

multi-level network optimizations. The algorithm further ensures compliance to the error frequency constraint by recovering the correct outputs on error-producing inputs to minimize the network cost increase until the frequency constraint is met. Experiments on a range of arithmetic circuit blocks demonstrated the effectiveness in achieving large gate count reductions across flexible error magnitude and frequency constraints.

# Chapter 5

# Conclusions

This dissertation presents the results of research on several distinct topics in approximate digital circuit design. The investigated topics include a formal analysis of timing-starved addition and novel techniques for inexact logic synthesis in two-level and multi-level forms. The dissertation has demonstrated the existence of a rich trade-off space for design of approximate circuits across different axis of quality metrics (error magnitude and error frequency) and conventional optimization objectives (e.g., energy or logic complexity).

## 5.1 Limitations of the Current Work

Although results of the dissertation show that trading off accuracy grants new opportunities in design optimizations, several limitations remain. For one, the synthesis of inexact CB logic in exploring the LSBs for approximate adders (Chapter 2) is based on a brute force search and is sub-optimal in both the synthesis quality and runtime. This also motivates the more general approximate logic synthesis research presented in Chapter 3 and Chapter 4.

With some major improvements, the proposed approximate logic synthesis (ALS) algorithms, however, still have some sub-optimality and scala-

bility limitations. One source of sub-optimality lies in the adoption of the two-phase strategy, in which the error frequency constraint is resolved on top of the error magnitude constraint only solution. Since, an optimal solution in the first phase does not necessarily lead to optimality in the error frequency recovery phase, no guarantees on the overall optimality can be made. Besides, the proposed algorithms are intrinsically heuristic. Nevertheless, the algorithms still demonstrate to be effective with large logic complexity reductions at reasonable error levels.

The scalability of the proposed ALS algorithms is limited by the use of Boolean relations. Runtime and space complexities are both exponential in the network bitwidth due to the necessity of exhaustive examinations for the relation construction. Several potential solutions exist for applying the ALS algorithms onto large bitwidth networks. When error magnitude constraints are small, the ALS algorithms can be applied only to the logic cones of the LSB network at the cost of some optimality loss. Otherwise, when error magnitudes are large, the bitwidth and error magnitude can be first proportionally scaled down to a smaller range before applying the algorithm.

## 5.2 Future Work

In this dissertation, hardware-level approximations have been studied in the context of design-time optimizations for application-specific systems. This relied on the assumption that target systems are intrinsically able to tolerate errors, and crucially, the error-tolerance level is fixed in the design

phase. As such, the proposed approaches are investigated under worst-case constraints and representative inputs (typically assumed to be uniformly distributed throughout this dissertation). In reality, requirements or inputs may vary dynamically at run-time, often significantly deviating from worst- or average-case assumptions made at design time. This can lead to overdesign, leaving optimization potentials untapped, as well as narrowing the application scope. Therefore, future work should investigate approximate computing techniques capable of run-time error-level adjustment that depends on the accuracy and energy requirements of a varying workload.

General-purpose processors are natural candidates for such systems. Run-time accuracy-adaptive computing blocks has been investigated and employed for design of approximate datapaths. Alternatively, the processor can be designed to switch and dispatch instructions among different fixed-accuracy computing blocks, where the blocks delivering different accuracy levels can be generated through logic-level approximate design/synthesis techniques discussed in this dissertation. For control logic, on the other hand, it is often believed that approximations can not be applied. However, this is not necessarily true. Broadly speaking, there have been studies on the approximation of control operations, e.g., techniques for branch predictors [49, 50] and approximate replacement policies for cache designs [51]. These techniques enable approximation of control logic. However, as a common feature, they are all equipped with some sort of error correction mechanism (e.g., pipeline rollback) to guarantee ultimate *error-free* operation. This is distinct from datapath or

other intrinsically error-tolerance approximations. For example, while the focus of error-tolerant applications is typically on minimizing error magnitude, in the presence of correction mechanisms, error frequency becomes most important. This further motivates the need for flexible synthesis approaches under general error constraints as presented in this dissertation. It is also worth mentioning, however, that the objectives in such control approximations often aim at high performance (through reductions of critical paths or cycle numbers) rather than low energy. As such, there is a need for approximate logic design and synthesis techniques under a wider array of simultaneous and competing objectives.

Ultimately, future research should focus on investigating a complete systems approach with support for approximations spanning from the higher levels (software and algorithms) down to hardware-, architecture- and logic-level approximations. In such a setup, the programmer should be able to declare some application outputs as approximate and allow the compiler to use the appropriate approximate instructions or computing blocks. This system-wide approach to approximate computing should be aimed at overall energy/accuracy optimality during the runtime of a program. As is in other power-optimization contexts, software- and algorithm-level techniques may thereby offer additional, and often more significant opportunities for approximation.

# Bibliography

[1] R. Gonzalez, B. Gordon, and M. Horowitz, "Supply and threshold voltage scaling for low power CMOS," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1210–1216, 1997.

[2] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Operating Systems Review*, vol. 35, pp. 89–102, 2001.

[3] G. Téllez, A. Farrahi, and M. Sarrafzadeh, "Activity-driven clock design for low power circuits," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, pp. 62–65, November 1995.

[4] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, pp. 415–420, 2000.

[5] D.-S. Chiou, S.-H. Chen, S.-C. Chang, and C. Yeh, "Timing driven power gating," in *Proceedings of the Design Automation Conference (DAC)*, San Francisco, CA, pp. 121–124, July 2006.

[6] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Higher Education, New York: McGraw-Hill Higher Education, 1994.

[7] D. Hisamoto, W. Lee, J. Kedzierski, H. Takeuchi, K. Asano, C. Kuo, E. Anderson, T. King, J. Bokor, and C. Hu, "FinFET: a self-aligned double-gate MOSFET scalable to 20 nm," *IEEE Transactions on Electron Devices*, vol. 47, pp. 2320–2325, 2000.

[8] J. Kao, A. Chandrakasan, and D. Antoniadis, "Transistor sizing issues and tool for multi-threshold CMOS technology," in *Proceedings of the Design Automation Conference (DAC)*, Anaheim, CA, pp. 409–414, June 1997.

[9] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, pp. 164–174, 2011.

[10] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig, "Approximate signal processing," *Journal of VLSI Signal Processing*, vol. 15, pp. 177–200, 1997.

[11] T. Xanthopoulos and A. Chandrakasan, "A low-power DCT core using adaptive bitwidth and arithmetic activity exploiting signal correlations and quantization," *IEEE Journal on Solid-State Circuits*, vol. 35, pp. 740–750, 2000.

[12] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *ACM SIGPLAN Notices*, vol. 47, pp. 301–312, 2012.

[13] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Davis, CA, pp. 1–12, December 2013.

[14] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE Transactions on Computers*, vol. 54, pp. 922–927, 2005.

[15] R. Hegde and N. Shanbhag, "Soft digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, pp. 813–823, 2001.

[16] A. Sinha and A. P. Chandrakasan, "Energy efficient filtering using adaptive precision and variable voltage," in *Proceedings of the International ASIC/SOC Conference*, Washington, DC, pp. 327–331, September 1999.

[17] K. He, A. Gerstlauer, and M. Orshansky, "Controlled timing-error acceptance for low energy IDCT design," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Grenoble, France, pp. 1–6, March 2011.

[18] Z. Kedem, V. Mooney, K. Muntimadugu, and K. Palem, "An approach to energy-error tradeoffs in approximate ripple carry adders," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Fukuoka, Japan, pp. 211–216, August 2011.

[19] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, "Enhanced low-power high-speed adder for error-tolerant application," in *Proceedings of the International SoC Design Conference (ISOCC)*, Incheon, Korea, pp. 323–327, November 2010.

[20] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: imprecise adders for low-power approximate computing," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Fukuoka, Japan, pp. 409–414, August 2011.

[21] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 1225–1229, 2010.

[22] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: a new paradigm for arithmetic circuit design," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, pp. 1250–1255, March 2008.

[23] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder

with carry skip for error resilient neuromorphic VLSI systems," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, pp. 130–137, November 2013.

[24] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proceedings of the Design Automation Conference (DAC)*, San Francisco, CA, pp. 820–825, June 2012.

[25] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, pp. 728–735, November 2012.

[26] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, pp. 48–54, November 2013.

[27] A. Burks, H. Goldstine, and J. Von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, Princeton, NJ: Inst. for Advanced Study, 1946.

[28] B. Gilchrist, J. H. Pomerene, and S. Y. Wong, "Fast carry logic for digital computers," *IRE Transactions on Electronic Computers*, vol. EC-4, pp. 133–136, 1955.

[29] D. R. Kelly and B. J. Phillips, "Arithmetic data value speculation," in *Proceedings of the Asia-Pacific conference on Advances in Computer Systems Architecture (ACSAC)*, Singapore, pp. 353–366, October 2005.

[30] S.-L. Lu, "Speeding up processing with approximation circuits," *IEEE Transactions on Computers*, vol. 37, pp. 67–73, 2004.

[31] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Proceedings of the Design, Automation Test in Europe Conference (DATE)*, Dresden, Germany, pp. 957–960, March 2010.

[32] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: systematic logic synthesis of approximate circuits," in *Proceedings of the Design Automation Conference (DAC)*, San Francisco, CA, pp. 796–801, June 2012.

[33] L. Chakrapani and K. Palem, "A probabilistic Boolean logic for energy efficient circuit and system design," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Taipei, Taiwan, pp. 628–635, January 2010.

[34] A. Lingamneni, C. Enz, J. L. Nagel, K. Palem, and C. Piguet, "Energy parsimonious circuit design through probabilistic pruning," in *Proceedings of the Design, Automation Test in Europe Conference (DATE)*, Grenoble, France, pp. 1–6, March 2011.

[35] P. McGeer, J. Sanghavi, R. Brayton, and A. Vincentelli, "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, pp. 432–440, 1993.

[36] Design Compiler, `www.synopsys.com`.

[37] FreePDK, `http://vlsiarch.ecen.okstate.edu/flows/FreePDK_SRC/`.

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

[39] R. Brayton and F. Somenzi, "An exact minimizer for Boolean relations," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, Santa Clara, CA, pp. 316–319, November 1989.

[40] B. Lin and F. Somenzi, "Minimization of symbolic relations," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, Santa Clara, CA, pp. 88–91, November 1990.

[41] S.-W. Jeong and F. Somenzi, "A new algorithm for the binate covering problem and its application to the minimization of Boolean relations," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, Santa Clara, CA, pp. 417–420, November 1992.

[42] A. Ghosh, S. Devadas, and A. Newton, "Heuristic minimization of Boolean relations using testing techniques," in *Proceedings of the International Conference on Computer Design (ICCD)*, Cambridge, MA, pp. 277–281, September 1990.

[43] Y. Watanabe and R. Brayton, "Heuristic minimization of multiple-valued relations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, pp. 1458–1472, 1993.

[44] D. Baneres, J. Cortadella, and M. Kishinevsky, "A recursive paradigm to solve Boolean relations," in *Proceedings of the Design Automation Conference (DAC)*, San Diego, CA, pp. 416–421, June 2004.

[45] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, pp. 504–510, November 2014.

[46] D. Baneres, J. Cortadella, and M. Kishinevsky, "A recursive paradigm to solve Boolean relations," *IEEE Transactions on Computers*, vol. 58, pp. 512–527, 2009.

[47] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, *SIS: A System for Sequential Circuit Analysis*, Tech. Report No. UCB/ERL M92, 1992.

[48] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proceedings of the International Conference on Computer Aided Verification (CAV)*, Edinburgh, UK, pp. 24–40, July 2010.

[49] J. E. Smith, "A study of branch prediction strategies," in *Proceedings of the Annual Symposium on Computer Architecture*, Los Alamitos, CA, pp. 135–148, 1981.

[50] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proceedings of the Annual International Symposium on Microarchitecture*, New York, NY, pp. 51–61, 1991.

[51] J. Handy, *The Cache Memory Book*, San Diego: Morgan Kaufmann, 1998.

[52] A. A. Del Barrio, R. Hermida, and S. O. Memik, "Exploring the energy efficiency of multispeculative adders," in *Proceedings of the International Conference on Computer Design (ICCD)*, Asheville, NC, pp. 309–315, October 2013.

[53] D. Shin and S. Gupta, "A new circuit simplification method for error tolerant applications," in *Proceedings of the Design, Automation Test in Europe Conference (DATE)*, Grenoble, France, pp. 1–6, March 2011.

[54] P. Albicocco, G. C. Cardarilli, A. Nannarelli, M. Petricca, and M. Re, "Imprecise arithmetic for low power image processing," in *Proceedings of the Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Pacific Grove, CA, pp. 983–987, November 2012.

[55] I. S. Chong and A. Ortega, "Hardware testing for error tolerant multimedia compression based on linear transforms," in *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, Monterey, CA, pp. 523–531, October 2005.

[56] M. R. Choudhury and K. Mohanram, "Approximate logic circuits for low overhead, non-intrusive concurrent error detection," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, pp. 903–908, March 2008.

[57] A. Burg, F. Girkaynak, H. Kaeslin, and W. Fichtner, "Variable delay ripple carry adder with carry chain interrupt detection," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, Bangkok, Thailand, pp. 113–116, May 2003.

[58] J. Huang and J. Lach, "Exploring the fidelity-efficiency design space using imprecise arithmetic," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Yokohama, Japan, pp. 579–584, January 2011.

[59] C. Liu, J. Han, and F. 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)*, Dresden, Germany, pp. 1–4, March 2014.

[60] J. T. Ludwig, S. H. Nawab, and A. P. Chandrakasan, "Low-power digital filtering using approximate processing," *IEEE Journal of Solid-State*

*Circuits*, vol. 31, pp. 395–400, 1996.

[61] J. Park, S. Kwon, and K. Roy, "Low power reconfigurable DCT design based on sharing multiplication," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Orlando, FL, pp. 3116–3119, May 2002.

[62] G. Karakonstantis, D. Mohapatra, and K. Roy, "System level DSP synthesis using voltage overscaling, unequal error protection and adaptive quality tuning," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS)*, Tampere, Finland, pp. 133–138, October 2009.

[63] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, "Opportunities and challenges for better than worst-case design," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Shanghai, China, pp. 2–7, January 2005.

[64] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, San Diego, CA, pp. 7–18, December 2003.

[65] S. Uramoto, Y. Inoue, A. Takabatake, J. Takeda, and Y. Yamashita, "A 100-MHz 2-D discrete cosine transform core processor," *IEICE Transactions on Electronics*, vol. 75, pp. 390–397, 1992.

[66] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Proceedings of the Design Automation Conference (DAC)*, Anaheim, CA, pp. 555–560, July 2010.

[67] V. Chippa, A. Raghunathan, K. Roy, and S. Chakradhar, "Dynamic effort scaling: Managing the quality-efficiency tradeoff," in *Proceedings of the Design Automation Conference (DAC)*, San Diego, CA, pp. 603–608, June 2011.

[68] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, pp. 600–612, 2004.

[69] F. J. Kurdahi, A. Eltawil, K. Yi, S. Cheng, and A. Khajeh, "Low-power multimedia system design by aggressive voltage scaling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 852–856, 2010.

[70] W. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transforms," *IEEE Transactions on Communications*, vol. 25, pp. 1004–1009, 1977.

[71] V. Dimitrov and K. Wahid, "Multiplierless DCT algorithm for image compression applications," *Information Theories and Applications*, vol. 11, pp. 162–169, 2004.

[72] L. Wang and N. R. Shanbhag, "Low-power filtering via adaptive error-cancellation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 51, pp. 575–583, 2003.

[73] E. L. Lawler, "An approach to multilevel boolean minimization," *Journal of the ACM*, vol. 11, pp. 283–295, 1964.

# Vita

Jin Miao was born in Luoyang, one of the Four Great Ancient Capitals of China. He received his Bachelor degree in Engineering from Zhejiang University, China, in 2010. In the same year, he joined in The University of Texas at Austin (UT-Austin), and was enrolled in the Ph.D. program in Electrical and Computer Engineering Department. In 2012, he received the Master of Science degree in Engineering from UT-Austin, and continued his Ph.D. study afterwards.

Address: jinmiao@utexas.edu

This dissertation was typed by the author.