Formal Probabilistic Error Analysis of Approximate Adders

By

Amina Qureshi

NUST201362864MSEECS61213F

Supervisor

Dr. Osman Hasan

Department of Electrical Engineering

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Electrical Engineering (MS EE)

In

School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(May 2017)

Approval

It is certified that the contents and form of the thesis entitled "**Formal Probabilistic Error Analysis of Approximate Adders**" submitted by **Amina Qureshi** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Osman Hasan**

Signature: _____

Date: _____

Committee Member 1: **Dr. Hassan Aqeel Khan**

Signature: _____

Date: _____

Committee Member 2: **Dr. Khawar Khurshid**

Signature: _____

Date: _____

Committee Member 3: **Dr. Sajid Saleem**

Signature: _____

Date: _____

Abstract

Approximate computing is an emerging trend in hardware and software design that leverages upon the inherent tolerance for inaccuracy in applications to optimize their power consumption, latency and area. Due to the widespread usage of adders in digital hardware designs, numerous approximate adders, offering various error margins and area, latency and power constraints, have been proposed. Probabilistic error analysis of these adders holds a significant step in selecting an appropriate approximate adder for a given application. Traditionally, this analysis is conducted using Monte-Carlo simulations or analytical analysis, which do not ascertain a sound and complete analysis. In order to overcome these limitations, we propose to use interactive theorem proving for error analysis. For this purpose, we present a higher-order-logic formalization of probability distributions and error related events encountered in high-speed, low-latency approximate adders (LLAA) based on the probability theory formalization available in the HOL4 theorem prover. We also propose an algorithm for analyzing the probability of error as a metric for accurate comparison for all the adders that comprise of sub-adder units of uniformly distributed inputs. For illustration purposes, we present the formal error analysis of three of the most widely acclaimed

approximate adders, i.e., ETA-I, ACA-II and GeAr.

Dedication

To My Parents

Sisters and Brother

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Amina Qureshi**

Signature: _____

Acknowledgment

In the Name of Allah, the Most Merciful, the Most Compassionate, all praise be to Allah, the Lord of the worlds; and prayers to Muhammad (PBUH) and his progeny, His servants and messengers.

Foremost, I am deeply thankful to my supervisor, Dr. Osman Hasan, a torch bearing personality who has always been devoted and generous during all phases of the research for his guidance, support and encouragement which greatly boasted my confidence throughout my thesis. Besides this, I am also highly indebted to have had his very useful comments and remarks, which, truly speaking culminated into the completion of this work.

I would also like to acknowledge the help of my lab fellows especially Waqar Ahmad.

Finally, I wish to thanks my parents, sisters and brother for their prayers and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Approximate computing [33] is an emerging computing paradigm that calls for compromising the accuracy to optimize the performance, area and power consumption of error-resilient applications [8, 10]. It is widely being advocated these days for various domains of intensive data computations, including search engines, [28], synthesis [36], cognitive applications [35], machine learning [30], image processing [17], signal processing [15], scientific computing [24] and wireless communications [18]. For example, in multimedia processing for human sensing, either an image or video is the final output, occasional errors produced by dropping a particular frame or a minute image quality loss can rarely be noticed due to the restricted perceptual capability of humans when interpreting images/videos. Approximate computing has been successfully applied at all abstraction levels of computer system design, ranging from the circuit [31] to software [25] and application levels [11].

In this thesis, we mainly focus on the error analysis of approximate hardware at the circuit level due to the huge cost associated with an uncaught hardware bug at the design time. Several approximate adders have been recently proposed as they form the basic building blocks of all the basic arithmetic circuits used for addition, subtraction, multiplication and division. Most of the high-speed, low-latency approximate adders are designed by overlooking the carry chain and, therefore, permitting all their sub-adders to perform addition in parallel. In other words, they mainly predict the value of the carry bit based on a fewer input bits and thus make carry propagation chain significantly shorter. Some prominent examples of these high-performance adders include, error tolerant adders (ETAs) [37], almost correct adder (ACA-I) [32], variable latency speculative adder (VLSA) [32], accuracy configurable adder (ACA-II) [16], gracefully-degrading adder (GDA) [34] and generic accuracy configurable adder (GeAr) [27]. In order to select an appropriate approximate adder for a given application, appropriate metrics are taken into consideration for evaluating the probability of error as well as the efficiency of approximate adders in terms of area, power and latency.

Traditionally, the error evaluation and comparison for the above-mentioned approximate adders is done using computer simulation. For small sized adders, the analysis is done using exhaustive testing but as the adder size, in terms of bits, increases these simulations become computationally infeasible. This issue is generally catered for by performing Monte-Carlo simulations but, due to their inherent incompleteness, the analysis results cannot be completely trusted and all the conditions under which an error occurs are not explicitly available, which makes a fair comparison between various approx-

imate adders almost impossible. The error analysis of approximate adders have recently been done analytically [19]. This method tends to provide a fair comparison among a wide range of arbitrary bit approximate adders by providing useful insights about the relationship between circuit specifications and error statistics. However, this technique is inclined to human error and requires a lot of manual effort for large adders.

The above-mentioned limitations can be overcome by formally reasoning about the error relationships of approximate adders in a theorem prover. Formal methods, which give computerized mathematical proofs, solve previously described confinement by giving exact analysis and removing human error. Formal verification is the way toward demonstration of verifying or refuting the correctness of a framework with specific prerequisites or property by utilizing formal methods of mathematics. The principle thought is to create a mathematical model and then to formally verify that it fulfill the required specifications. Theorem proving and model checking and are the two most widely used formal methods. If the user require automatic verification of the required goals than Model checking is the right choice. Model checking is a formal analysis technique which is most widely used , is automatic and can be expressed as a finite-state machine. Various methodologies for probabilistic model checking have been set forward e.g., [12], [22], what's more, in light of these methodologies, various tools have been created, e.g., PRISM [5] and VESTA [26] but due to its restricted system expressiveness and that it may also suffer from state space explosion; its usage for probabilistic analysis is to some degree constrained. Then again, theorem proving is an interactive technique which is generally utilized formal analysis technique as it is not

restricted to the extent of the state space and is more powerful in terms of expressiveness. The most widely used theorem provers are HOL4 [29], HOL Light [2] and Isabelle [3].This thesis makes use of HOL4 theorem prover for formalization and verification tasks. HOL4 theorem prover [13] is a widely-used computer program that provides an interactive environment for the modeling and verification of mathematical proofs in HOL. HOL4 is built upon four basic axioms and eight primitive inference rules, and these inference rules are used to verify the theorems. As nobody has yet attempted Theorem proving for approximate computing, one of the principle purposes of this dissertation is that with increase in approximate computing demand this dissertation will be helpful for future utilize and it will also expand existing formalizations of probability theory in HOL4.

The foremost requirements for formal verification of error relationships of approximate adders are the formalizations of the probability distributions and error events, encountered in an approximate adder, and the error analysis method. This thesis utilizes the formalization of probability theory [20] and principle of inclusion-exclusion [7], available in the HOL4 theorem prover, to fulfill these requirements for a low-latency approximate adder (LLAA) model. Our focus is on analyzing the adders that comprise of sub-adder units of uniformly distributed inputs and the formalization of the corresponding error analysis algorithm [19]. The main motivation for selecting this class of adders for our analysis is their high-speed and low-latency characteristics and thus their widespread usage. Keeping in mind the end goal to show the utilization and effectiveness of the proposed error analysis approach, we use it to formally verify the probability of error in ETA-I, ACA-II and GeAr adders,

which are widely used in various applications of approximate computing.

1.2 Problem Description

Most of the high-speed, low-latency approximate adders are designed by truncating the carry chain which results in error at output of approximate adders. This error is used as a metric for comparison between low-latency approximate adders. Usually computer simulation is used for error evaluation, but it has limitations, as it requires exhaustive testing so as the size of adder increases more and more memory and time is required making them computationally infeasible. So the analysis results cannot be trusted to overcome this limitation the error analysis of approximate adders have recently been done analytically [19], however,this method is prone to human error and requires a lot of manual effort while analyzing large systems.

1.3 Proposed Solution

The above-mentioned problem can be solved by formally reasoning about the error relationships of approximate adders in a theorem prover. From all the research done on formal verification and approximate computing, to the best of our knowledge, no one has tried modeling of approximate adders or has formally verified probability of error in HOL. Hence, the main goal of our work was to formalize error analysis algorithm for low latency approximate adders but to do that we required formal modeling of approximate adders which according to best of our knowledge was not previously done by any

other person so due to lack of availability of model of approximate adders we also needed to provide formal modeling of approximate adders which could later be used for further analysis of different approximate adders.

1.4 Proposed Methodology

The main reason behind this dissertation is the modeling of approximate adders and the formalization of error analysis of the approximate adders. In order to have a metrics for error performance evaluation among approximate adders, in this thesis, we propose to formally verify the probability of error in approximate adders which can be used as a metric for comparison among different approximate adders. We have proposed a methodology for the modeling of approximate adder and the formalization of error generated at output in approximate adders. In this first approximate adders are discussed and the need of formal verification of probability of error and then the conditions that lead to error in output of approximate adder are identified. These conditions help us in modeling of approximate adder. After the modeling of approximate adders probabilistic analysis is formalized in HOL which is carried out by using basic theorems of the probability theory. Low-latency, high speed approximate adders class are selected, architecture of these approximate adders is that they comprise of multiple sub-adders. The error induced in approximate adders output is due to wrong carry prediction and carry-chain truncation among consecutive sub-adders units. Probability of error can be calculated in these approximate adders which can help us to select approximate adder according to our requirements. In this thesis we

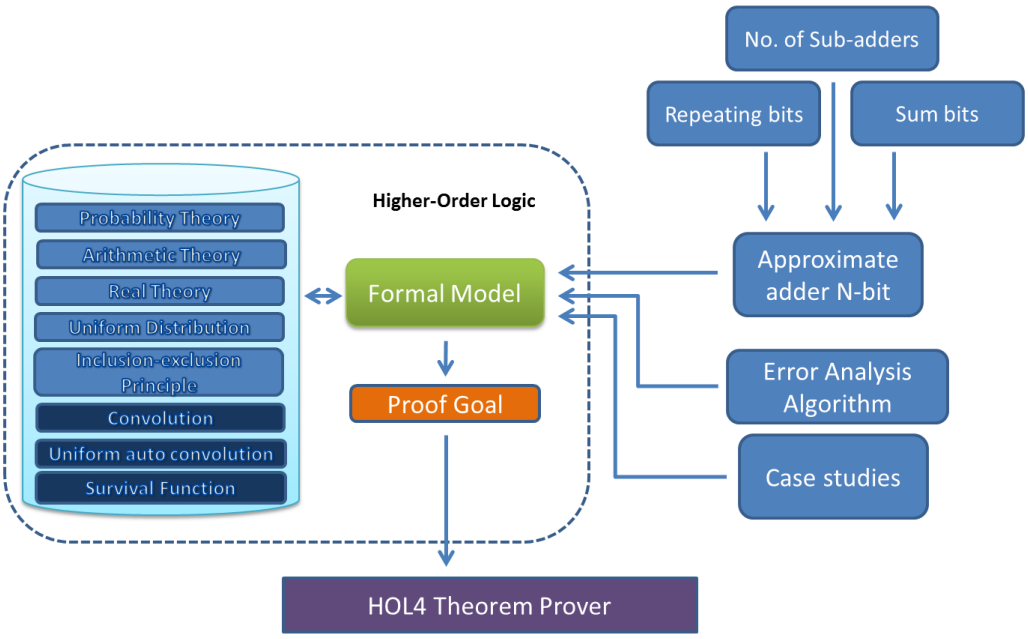

Figure 1.1: Proposed Methodology

have formalized the adders for N-bit approximate adders and we have formalized the functions needed to evaluate the probability of error, by using these we can evaluate probability of error in any low-latency approximate adders. One of the main advantages of this methodology is as the size of approximate adders increases simulations require more and more time to calculate the result whereas this thesis will allow us to calculate the probability of error for arbitrary bit width approximate adders and as a result we will save time required for simulations which is a used in all other performance metrics. A general overview of our formalization is illustrated in Fig. 1.1 .

1.5 Thesis Organization

The rest of the thesis is sorted out as follows: In Chapter 2, we give a short prologue to the HOL theorem prover and an overview of the Mhamdi's probability theory [20] and equip the reader with some notation and concepts that will be utilized in the rest of this thesis. Chapter 3 describes the approximate adder model then we identify the errors in the intermediary logical elements of the approximate adder that contribute to error in the output and then mathematically relate the occurrence of such errors with $Pr[Error]$ which will be expressed as the sum of probabilities of joint carry-in propagation and carry-out generation events for specified groups of input bits. Chapter 4 presents the HOL formalization of probability distribution of inputs and outputs of approximate adder and formalization of events that lead to error is output of approximate adder. In this process, we had to formally verify some foundational probability theory laws, such as convolution of probability distributions, auto-convolution of uniformly distributed probability distribution and survivor function of sum of two uniformly distributed random variables, which are also presented in this chapter. Chapter 5 provides the formalization of error analysis algorithm which includes the formalization of error in intermediary logical elements of the approximate adder, formalization of error analysis algorithm and formalization of total error occurring in output of approximate adder. In order to demonstrate the practical usefulness of the proposed methodology, in Chapter 6, we provide three case studies of different most widely acclaimed approximate adders, i.e., ETA-I, ACA-II and GeAr. Finally, Chapter 7 concludes the thesis and outlines some future

research work.

Chapter 2

Preliminaries

The overview of Theorem proving and Hol theorem prover are provided in this chapter along with the description of terminologies used in this thesis so that reader can get familiar with terminologies and can understand formalization better.

2.1 Theorem Proving

Theorem proving is a formal hardware verification method which is used to formalize a system and to verify its desired properties. Depending upon the requirements a system model is built upon mathematical logic such as propositional logic, first-order logic or higher-order logic. Inference rules are used to formalize theorems and to verify that a system model fulfill the desired properties. Theorems that are based of First-order logic (FOL) [13]are proved with ease as compared to higher-order logic (HOL) [9], as FOL can be significantly automated. Theorems that are comprised of HOL require more

effort when proving as HOL proof can be automated because HOL have undecidable nature. Now a days designs created are becoming more and more complex and when we consider the formalizations of these designs we prefer HOL as it provides more quantifiers and is more expressive as compared to others. Propositional logics or FOL cannot be used to formalize all complex designs whereas HOL can be used to formalize any complex design due to its expressive nature. When we talk about probabilistic analysis we formalize random variables as functions in HOL that map randomness of random variable to an appropriate set of numbers. We also formalize random variable properties such as probability mass function (PMF) by quantifying over random variable functions. We need to use HOL to conduct probabilistic analysis over FOL as FOL does not allow quantification over predicates and functional variables. Theorem proving can further be subdivided into two types i.e., automated theorem proving and interactive theorem proving. Automated theorem provers automatically verify the desired specifications of formal models, while interactive theorem provers require user interaction to construct formal model and to verify the desired specifications. The most popular theorem provers include HOL4 [29], Isabelle/HOL [3], HOL Light [2] and Coq [1], the user interface, automatic decision procedures and mathematical logic used in these systems makes them different from each other. We have selected HOL4 theorem prover for our work mainly because it provides high degree of expressiveness and we build on probability theory of Mhamdi [20].

2.2 HOL Theorem Prover

HOL provides an interactive environment for the formalization and verification of mathematical proofs and was developed at Cambridge University by Mike Gordon [14]. Several versions of HOL has been introduced, the latest version is HOL4 and the previous versions were HOL88 followed by HOL90 and HOL98. HOL4 is mainly based on HOL98 and is represented in standard meta language (SML) [21], which is a strongly-typed functional programming language. SML is implemented in two ways and either one of them can be used in HOL which are Moscow ML [6] and Poly ML. The base of HOL4 is built on five basic axioms and eight primitive inference rules [4], which are implemented as ML functions. Every new theorem or property formalized in HOL is proved by using these primitive inference rules and basic axioms or theorems which are proved using these rules, to ensure the soundness of theorem. HOL theorem prover has been widely used for the hardware and software verification.

2.2.1 Terms

In HOL theorem prover we have four types of terms which are as follows:

- **Variables**: are successions of digits or letters that begin with a letter.
- **Constants**: syntax similar to variables is used for constants with an exception that they cannot be bounded by quantifiers.
- **Function applications**: for computing any function for a certain argument, function applications are used in HOL.

- **lambda-abstractions**: for representing a function f that takes y as argument and returns $f(y)$ lambda-abstractions are used.

2.2.2 Types

All the variable or constant created in HOL environment have a type. Type can also be used to distinguish the variables, meaning that there can be two variables with same but with two different types. While working on our dissertation we mainly used three types which are boolean, number and real types denoted by bool, num and real. In HOL, type checking algorithm assigns a type whenever we introduce a new term in formalization but user can also inferred type explicitly if type checking algorithm is not able to deduce it automatically e.g $(y : num)$ or $(y : real)$.

2.2.3 Inference Rules

In the HOL environment when we are proving the formalized theorems or goals we use inference rules, represented as ML functions, which are applied until one reaches the conclusion. HOL provides with a set of eight primitive inference rules, which are then used to prove any new theorem. These inference rules are Reflexivity, Assumption introduction, Substitution, Abstraction, Beta-conversion, Type instantiation, Modus Ponens and Discharging an assumption [4]. All other rules are derived by using these inference rules and axioms.

2.2.4 Theory

A collection of axioms, definitions and proven theorems is called a theory in HOL. Each theory consists of verified theorems which are verified using already proven theorems or inference rules which assures the soundness of theorem prover. When we are proving a theorem we divide the theorem into many small theorems and verify the main theorem by verifying these smaller theorems one by one, to prove these smaller theorems we may use already proved theorems and defined definitions which are available in form of theories and can be loaded in HOL environment which saves user time and effort spend proving theorems already proved by someone else. To reuse already existing work, theory may acquire, from HOL theories, available types, definitions and theorems, i.e., hierarchical method is used to organize HOL theories. In fact, one of the main reason behind choosing the HOL theorem prover for our dissertation was to take advantage of already existing formalized theories. The theory on which all other theories are built is *min* theory in which type constant for booleans, binary type operator for functions and the type constant for individuals are defined. Bool theory is the most basic thoery of HOL and is building block of all other theories. It consist of five axioms of HOL. For developing all the standard mathematics all we need are these axioms together with the eight inference rules.

2.2.5 Proofs in HOL

Two approaches, forward and backward approach, can be followed while proving or verifying theorems in HOL. In the forward approach, user starts

from the already proved theorems and inference rules and using them to achieve the coveted goal. This approach is not the easiest solution since to use this approach the detailed knowledge of the theories the exact proof steps in advance. In the backward approach, which is a reverse of forward approach, where user starts from the goal and simplifies it by splitting main goal into smaller sub-goals and then prove these sub-goals by using already proved theorems and primitive inference rules. There are many automatic tactics provided by HOL which helps in breaking the goal into simple sub-goals while other proof steps can be verified through user interaction. Some of the sub-goals are proved by matching axioms or assumptions. In HOL there exist built-in decision procedures which are applied as tactic on the sub-goal to prove the sub-goal. To prove the desired theorem all the intermediate sub-goals are proved by repeatedly applying tactics mentioned above, until all the sub-goals are proved. To prove a goal user is provided with proof editor with contains already defined tactics, these tactics are applied by user interactively one by one to prove the desired theorem. One other feature available in HOL is that in HOl there exists few automatic proof procedures, which solve some proof steps automatically.

2.2.6 HOL Symbols

In our dissertation we have used HOL functions and symbols which have mathematical interpretation. We have provided the table so the reader can easily understand the symbols henceforth the formalization we have provided in the later chapters.

Table 2.1: HOL Symbols and Functions

Symbols in Hol	Standard Form	Explaination
$\wedge$	and	Logical and
<==>	=	Equality
==>	$\Rightarrow$	Implication
λ x.t	λ x.t	Function that maps x to $t(x)$
!x.t	$\forall$x.t	for all x : t
$\neg$	not	Logical negation
SUC k	(k+1)	Successor of natural number
++	append	Joins two lists together
::	cons	New element is added to a list
HD lt	head	Returns the first element of list lt
TL lt	tail	Returns the last element of list lt
[ ]	[ ]	empty List
exp y	e^y	Exponential function
LENGTH lt	length	Number of elements in list lt
MEM a lt	member	True if element a exists in list lt

2.3 Probability in HOL

Probability has been used in all domains for centuries and is defined as Mhamdi formalized probability theory in HOL4 using Kalmogorov work as the base and defined probability space as a triple (Ω,Σ,Pr), i.e, measure space. where sample space, set of all the possible outcomes, is represented by Ω, set of events by Σ and probability measure by Pr. Mhamdi build up the probability theory on three basic kolmogorov axioms:

- $\forall$A. $0 \leq \Pr(A)$
- $\Pr(\Omega) = 1$
- For all countable mutually exclusive events, B0, B1,...,
  $\Pr(\bigcap_{n\in\Omega} \Pr(B_n)) = \sum_{n\in\Omega} \Pr(B_n)s$

Probability theory provide us many basic probability axioms which have been formally verified, ready to be used. Random variable is the function defined on sample space, what it does is that it takes all the possible outcomes of sample space and maps them to mathematically convenient outcome labels, usually a real number.

Chapter 3

Low-Latency Approximate Adders

3.1 Approximate Adder Model

One of the foremost design principles of low-latency approximate adders is the fact that the frequency for long carry propagation, i.e., a carry-out = 1 for every adjacent single-bit adder, is usually very less. Low latency approximate adders (LLAA) exploit this principle by assuming carry chain truncation at various positions in the given adder and thus computing the sum of the sub-adders, separated by the truncated carry positions, with or without overlapping bits, in parallel. This parallel execution results in an enhanced performance. The corresponding probability of error in the approximate sum (SUM_{approx}) for inputs A and B can be defined as:

$$\Pr[Error] = \Pr[SUM_{approx} \neq A + B] \tag{3.1}$$

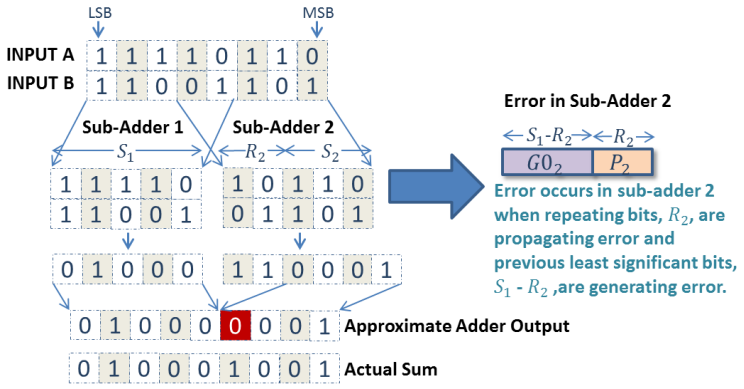

Figure 3.1: Low-Latency Approximate Adder

For illustration, consider the scenario depicted in Fig. 3.1. The main adder is divided into two sub-adders. A sub-adder may include overlapping bits, also known as *repeating* bits, which are utilized for the prediction of carry-in in the given sub-adder. For the considered set of inputs, shown in Fig. 3.1, the carry generated in Sub-adder 1 is truncated as the repeating bits of Sub-adder 2, i.e., R_2, are not able to predict the carry-out generated by the Sub-adder 1 correctly. Thus, for the considered input combination, the result of the approximate adder would be incorrect. Based on this example, it can be observed that an error occurs in a sub-adder if its repeating bits are in the propagation mode and the bits before the repeating bits in the previous sub-adder, i.e., $S_1 - R_2$ in the case of the considered example of Fig. 3.1, are in the carry generation mode.

3.2 Identifying Events Leading to Approximation Error

The N-bit approximate adder can be generalized as a composition of L sub-adders such that the i^{th} sub-adder is a $R_i + S_i$ bit adder, where R_i represent the repeating bits from the previous sub-adder, which are used to predict the carry for the i^{th} sub-adder, [19] and S_i represents the sum bits for the i^{th} sub-adder. Each sub-adder adds the respective R_i+ S_i bits of two operands to find the respective output. In the case of the first sub-adder, there are no repeating bits, i.e, $R_1 = 0$ and S_1 is equal to length of first sub-adder. Thus, there is no carry prediction required in the first sub-adder and the output of the first sub-adder is always precise. Various approximate adders can be developed, based on this generic model, by varying the size of sum and repeating bits and the number of sub-adders. For example, ACA-II [16], GeAr [27] and ETA-II [37] have the same number of sum and repeating bits, i.e, $S_i = R_i$, for every sub-adder i, whereas the modified ETA-II [37] and GDA [34] use different S_i and R_i bits for every sub-adder. Since, an error occurs in a sub-adder due to the co-occurrence of a carry-out generation event ($G0$) and a carry-in propagation (P) event so the probability of error in the i^{th} sub-adder can be determined by finding in sub-adder i, the prediction bits, R_i, are propagating carry (P_i) and the previous sub-adders bits are generating carry-out $G0_i$:

$$\Pr[E_i] = \Pr[G0_i \wedge P_i] = \Pr[G0_i] \Pr[P_i] \tag{3.2}$$

as P_i and $G0_i$ are independent events since they represent conditions on

disjoint groups of bits of the adder inputs. Now, the probability of error in the complete adder can be modeled and simplified based on probability theory laws as [19]:

$$\Pr[Error] = \Pr\left[E_2 \vee E_3 \vee ... \vee E_L\right] = \sum_{i=2}^{L} \Pr\left[E_i\right] - \sum_{2\leq i<j\leq L} \Pr\left[E_i \wedge E_j\right] + \sum_{2\leq i<j<k\leq L} \Pr[E_i \wedge E_j \wedge E_k] - ... + (-1)^L \Pr\left[\bigwedge_{i=2}^{L} E_i\right] \tag{3.3}$$

3.3 Evaluation of Joint Probabilities

We can observe from the above equation that the evaluation of probability of error of LLAA mainly depends on the evaluation of the probability terms involving co-occurring error events E_i of sub-adders. For illustrating the evaluation of these probability terms, consider the example given in Fig. 3.2(a)-(d), where an approximate adder is divided into three sub-adders, such that $R_2 = R_3 = 3$ and $S_2 = S_3 = 4$. Error events occurring in Sub-adder 2 and Sub-adder 3 are not mutually independent as these events are not based on disjoint set of input bits. These dependent events can however be partitioned into mutually independent events, involving the carry-in propagation (P), carry-out generation with no carry-in ($G0$) or carry-out generation with carry-in equal to one ($G1$) events [19]. According to the proposed methodology in [19], the events are transformed first by replacing all the propagating events by a single propagation event, such that the new event imposes the propagation condition on all the bits involved in individual propagation events of sub-adder error as shown in Fig. 3.2(b). Then,

all the carry generating events with carry-in = 1 are replaced with $G1$ and carry generating events with carry-in = 0 are replaced with the $G0$ event. Thus, in the example, depicted in Fig. 3.2, there is a carry-out at the 3^{rd} bit position due to $G0_2$ and there was no carry-in at the 0^{th} bit so the event for bit locations 0-3 can be represented as a $G0$ event. Next bits are propagating the carry so there will be a carry-out at the 6^{th} bit. This acts as the carry-in for the 7^{th} bit and there will be a carry-out at the 7^{th} bit position since it is a carry generating event. This event can be replaced by a $G1$ event as shown in Fig. 3.2(c). Thus, the joint event $(E_2 \wedge E_3 \wedge E_4)$ becomes $(P \wedge G0 \wedge G1)$, where $G0 = G0_2$, as shown in Fig. 3.2(d). Since these new events are now defined on a disjoint sets of input bits so all the events are mutually independent of each other so:

$$\Pr[E_2 \wedge E_3 \wedge E_4] = \Pr[P \wedge G0 \wedge G1] = \Pr[P] \Pr[G0] \Pr[G1] \tag{3.4}$$

Just like this example, other probability terms of Equation 3.3 can also be determined to asses the probability of error of the complete adder. In this thesis, interactive theorem proving is what for the evaluation of this error. Thus, we present a formalization of the adder model and the error analysis method in the next two sections, respectively.

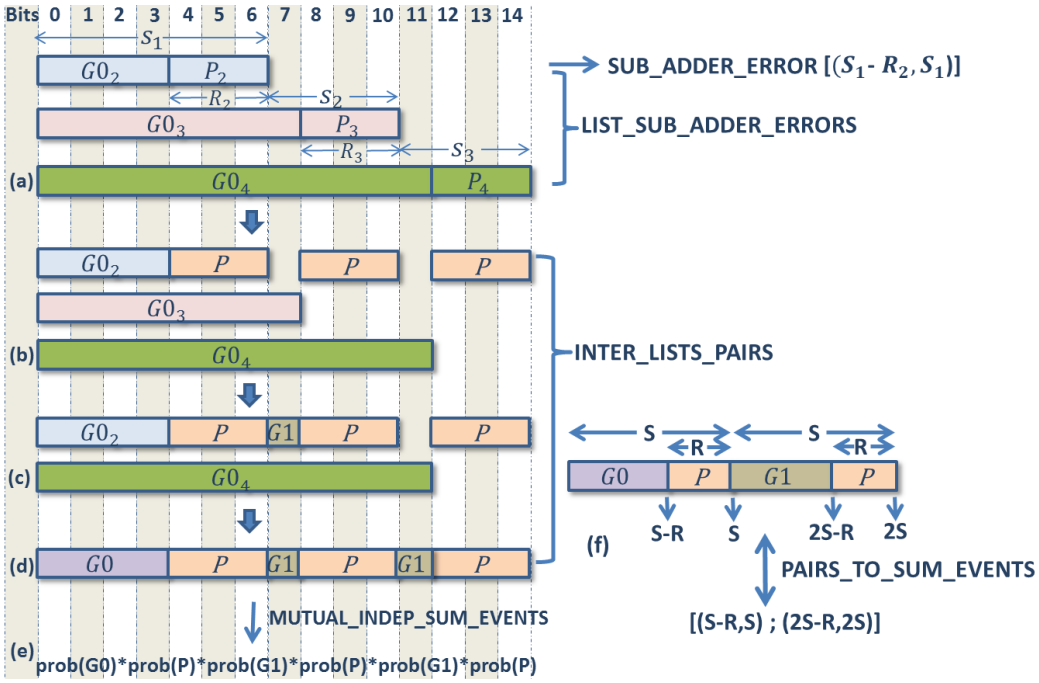

Figure 3.2: Transformation

Chapter 4

Formalization of Probability Distributions and Events

4.1 Probability Distributions for Inputs and Outputs

Just like regular adders, approximate adders are used to calculate the sum of two inputs. Since all the input combinations are equally likely to occur so it can be fairly assumed, the uniform distribution of the inputs of adders. We formalized the uniform distribution in HOL4 as follows:

Definition 1 (*Uniform Distribtion*)

$\vdash$ *Uniform_pmf p n X x =*

$\forall$n X x. pmf p X x = (if (x $\leq$ (n-1)) then (1)/(n) else 0)

The HOL4 function, *pmf*, used in above definition represents the *probability mass function* (PMF) and is defined as the probability of the event $\{X = x\}$.

$$f_X(x) = Pr(X = x) = Pr(\{s \in S : X(s) = x\}) \tag{4.1}$$

Where S is the sample space. Now, the PMF can be formalized in HOL4 as:

Definition 2 (*PMF*)

```
⊢ pmf p X x = distribution p X {y | y = x}
```

The function pmf takes a probability space p, a random variable $\mathrm{X} : \Omega \rightarrow \mathbb{N}$ and $\mathrm{x} : \mathbb{N}$. It gives back a real number that represents the probability distribution of the random variable X.

As discussed in Section 3, the inputs of a low-latency approximate adder are divided in L sub-adders. If the random variables X and Y represent the inputs of a sub-adder with n-bits then X and Y are both uniformly distributed between 0 and $2^n - 1$.

$$p_X(k; n) = p_Y(k; n) = \begin{cases} \dfrac{1}{2^n}, & \text{for } 0 \leq k \leq 2^n - 1 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

Theorem 1 (*Distribution of a Sub-adder*)

```
⊢ ∀ p X k x n .  prob_space p ∧ PREIMAGE X {k} IN events p ∧
   Uniform_pmf p m X x ⇒
( if ( k ≤ (2^n -1) ) then (prob p (PREIMAGE X {k}) = 1/ (2^n))
      else (prob p (PREIMAGE X {k}) = 0) )
```

Considering that the two sub-adder inputs X and Y are not dependent on each other, we can verify that the PMF of their sum, $Z = X + Y$, is the convolution of their corresponding PMFs. Moreover, since both random

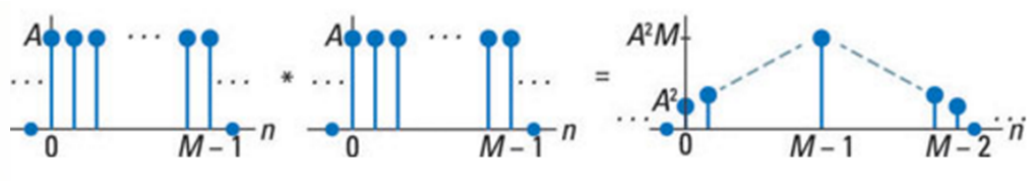

Figure 4.1: Uniform Auto convolution

variables are uniformly distributed between 0 and $2^n - 1$ so the PMF of their sum can be verified to be equal to the auto convolution of the uniform distribution, which in turn is equal to the triangular distribution as shown in Fig. 4.1:

$$p_Z(k;n) = p_X(k) * p_Y(k) = \begin{cases} \dfrac{k+1}{2^{2n}}, & 0 \le k \le 2^n - 1 \\ \dfrac{2^{n+1} - k - 1}{2^{2n}}, & 2^n - 1 < k \le 2^{n+1} - 2 \\ 0, & \text{otherwise} \end{cases} \quad (4.3)$$

Theorem 2 (*PMF of SUM of Two Uniformly distributed Random Variables*)

```
⊢ ∀ p X Y k m.  prob_space p ∧ ( m > 0) ∧
   (∀ l t.  indep p (PREIMAGE X {l})(PREIMAGE Y {t}) ∧
   (Uniform_pmf p (2^n) X l) ∧ (Uniform_pmf p (2^n)Y t) ) ⇒
    ( pmf p (λs .  X s + Y s) k) =
    ( if ( k ≤ (2^n-1) ) then ((k+1)/((2^n)^2))
      else if ( k ≤ (2 * 2^n) - 2) then
        ((2 * 2^n - k - 1)/( (2^n)^2 )) else 0)
```

The verification of Theorem 2 is basically based on the convolution of probability distributions and auto convolution of uniformly distributed random variables.

4.1.1 Convolution of Probability Distributions of Two Random Variables

Lemma 1 (*Convolution of Probability Distributions*)

```
⊢ ∀ p X Y. prob_space p ∧
(∀ k t.  indep p (PREIMAGE X {k}) (PREIMAGE Y {t})) ⇒
   ( ∀z .  ( pmf p ( λs .  X s + Y s ) (z) =
     ( sum (0,SUC z) (λx.  pmf p X x * pmf p Y (z - x) ))))
```

The first two assumptions ensure that the events generated by the random variables, X and Y, are independent in the probability space p. Lemma 1 is primarily verified based on the following lemma according to which the event corresponding to the sum of random variables, X and Y, can be expressed as the union of disjoint events:

```
⊢ ∀ X Y k.  {s | X s + Y s IN {k}} =
   BIGUNION (IMAGE (λi.  {x | x IN {x | X x IN {i}} ∧
      x IN {x | Y x IN {k - i}}}) { i | i ≤ k})
```

4.1.2 Auto-Convolution of Probability Distributions of Two uniformly distributed Random Variables

We verified that the auto convolution of the Uniform random variable exhibits the triangular distribution in HOL4 as follows, which was in turn used to verify Theorem 2.

Lemma 2 (*Auto Convolution of the Uniform Random Variable*)

```
⊢ ∀ p X Y k m.  prob_space p ∧ ( m > 0) ∧
   (∀ l t.  indep p (PREIMAGE X {l})(PREIMAGE Y {t}) ∧
   (Uniform_pmf p m X l) ∧ (Uniform_pmf p m Y t) ) ⇒
    ( pmf p (λs .  X s + Y s) k) =
    ( if ( k ≤ (m-1) ) then ((k+1)/(m²))
      else if ( k ≤ (2 * m) - 2) then
        ((2 * m - k - 1)/(m²)) else 0)
```

4.2 Formalization of Error Related Events

As discussed in Section 3, LLAA involves three possible events, i.e., carry propagation (P), carry generation with no input carry ($G0$) and carry generation with input carry ($G1$).

4.2.1 Propagation Event

Propagation event, P, is formalized by using the fact that the carry-in is propagated by an n-bit addition only when all n single bit adders of this adder are in the propagating mode. This happens when the bits in every single bit addition are unequal. Under these conditions, the carry-out of the sub-adder will be equal to its carry-in and the sum will be exactly equal to $2^n - 1$, which can be formalized in HOL4 as follows:

Definition 3 (*Propagation Event*)

```
⊢ P_EVENT X Y n = PREIMAGE ( \s.X s + Y s ) { y | y = 2^n - 1 }
```

The probability of the propagation event can be verified as:

$$\Pr[P] = p_Z(2^n - 1) = \frac{1}{2^n} \tag{4.4}$$

Theorem 3 (*Propagation Probability*)

⊢ ∀ *p X Y n.prob_space p* ∧

(∀ l t u.indep p (PREIMAGE X {l}) (PREIMAGE Y {t}) ∧

(Uniform_pmf p 2^n *X l)* ∧ *(Uniform_pmf p* 2^n *Y t))* ⇒

(prob p (P_EVENT X Y n)) = 1/2^n*)*

The verification of Theorem 3 is based on Theorem 1 and Lemmas 1 and 2.

4.2.2 G0 Event

A carry-out in an n-bit addition, with a carry-in = 0, is generated when the value of the sum cannot be accommodated ins n bits, which means that the value of the sum has to be greater than $2^n - 1$.

Definition 4 (*G0 Event*)

⊢ *G0_EVENT X Y n = PREIMAGE (\s.X s + Y s) { y | y >* 2^n *- 1 }*

The probability of $G0$ event is as follows:

$$\Pr[G0] = \Pr[Z > 2^n - 1] = \sum_{k=2^n}^{2^{n+1}-2} p_Z(k) = \sum_{k=2^n}^{2^{n+1}-2} \frac{2^{n+1} - k - 1}{2^{2n}} = \frac{2^n - 1}{2^{n+1}} \tag{4.5}$$

Theorem 4 (*Carry Generation Probability with Carry-in=0*)

⊢ ∀ *p X Y n.prob_space p* ∧

(∀ l t.indep p (PREIMAGE X {l}) (PREIMAGE Y {t}) ∧

(Uniform_pmf p 2^n *X l)*∧ *(Uniform_pmf p* 2^n *Y t)))* ⇒

(prob p (GO_EVENT X Y n)) = (2^n*-1)/(*2^{n+1}*)*

The proof of Theorem 4 primarily utilizes Theorem 1, Lemmas 1 and 2 and the following Lemma of Survival function of Sum of uniformly distributed random variables.

Survival Function of Sum of Uniformly Distributed Random Variables

The probability that the sum takes a value greater than $g-1$ can be equated as summation of PMFs of all the events for which sum is greater than $g-1$ called Survival function of sum.

$$\Pr[Z > g-1] = \sum_{k=g}^{2^{n+1}-2} p_Z(k) \qquad g < 2^{n+1}-2 \tag{4.6}$$

Lemma 3 (*Survival Function of Sum*)

```
⊢ ∀ p X Y g n.  prob_space p ∧ ( m > 0) ∧
   (∀ l t.  indep p (PREIMAGE X {l})(PREIMAGE Y {t}) ∧
   (Uniform_pmf p (2^n) X l) ∧ (Uniform_pmf p (2^n) Y t) ) ∧
   (g > 0) ∧ (g ≤ (2 * 2^n - 1) ⇒
(distribution p ( λs .  X s + Y s ) {y | (y >( g - 1) )}) =
sum ( g,((2 * 2^n - 2) + 1 - g))(λj.  prob p {x | X x + Y x IN
{j}})
```

We verified the above lemma by splitting the event $y > g-1$ into two sub-events, i.e., $g-1 < y \leq 2*2^n-2$ and $y > 2*2^n-2$ since the probability of the sum for $y > 2*2^n-2$ can be verified to be equal to 0.

4.2.3 G1 Event

Just like $G0$, the event $G1$ happens when the sum of two n-bit inputs, while considering carry-in=1, is greater than $2^n - 1$.

Definition 5 (*G1 Event*)

⊢ `G1_EVENT X Y n = PREIMAGE (\s. X s + Y s + 1) {y|y > 2^n-1}`

The probability of the $G1$ event can be found as follows:

$$\begin{aligned}\Pr[G1] &= \Pr[Z + 1 > 2^n - 1] = Pr[Z > 2^n - 2] \\ &= \sum_{k=2^n-1}^{2^{n+1}-2} p_Z(k) = \frac{1}{2^n} + \sum_{k=2^n}^{2^{n+1}-2} \frac{2^{n+1} - k - 1}{2^{2n}} = \frac{1}{2^n} + \frac{2^n - 1}{2^{n+1}}\end{aligned} \tag{4.7}$$

Theorem 5 (*Carry Generation Probability with Carry-in=1*)

⊢ ∀ `p X Y n.prob_space p ∧`

`(∀ l t.indep p (PREIMAGE X {l}) (PREIMAGE Y {t}) ∧`

`(Uniform_pmf p 2^n X l)∧ (Uniform_pmf p 2^n Y t)) ⇒`

`(prob p (G1_EVENT X Y n) ) = ( 1 / 2^n) + ( (2^n - 1) / (2^n+1))`

The above theorem is verified by using Theorem 1 and Lemmas 1, 2 and 3.

Chapter 5

Formalization of Error Analysis Algorithm

The result of an approximate adder becomes erroneous when an error occurs in any one of the sub-adders. Thus, the first step in the formal modeling of error in a LLAA is to model the error in one of its sub-adders.

5.1 Formalization of Error in Sub-adder

As explained in Section 3, an error occurs in sub-adder i when its generation and propagation events occur simultaneously, i.e., $(G0_i \wedge P_i)$. This error event in a single sub-adder can be formalized by having an intersection between the $G0$ and P events. We propose to capture the $G0$ and P events based on a pair of bit locations of the occurrence of these events. Considering the example shown in Fig. 3.2(a) and taking $S_2 = S_3 = s$, $R_2 = R_3 = r$ and $S1 = r + s$, we can observe that the end of $G0_2$ event for the 2^{nd} sub-adder

occurs at bit position s, the end of event $G0_3$ for the 3^{rd} sub-adder occur at bit position $s + s$ and the end of event $G0_4$ for the 4^{th} sub-adder occurs at bit position $s+s+s$. Thus, the position of the end of the $G0_i$ event for the Sub-adder i can be generalized as $(i-1)s$, representing the first element of pair. Similarly, it can also be seen that for every i^{th} sub-adder, the propagation event starts from the end of the generation event and ends at bit location $(i-1)s + r$, representing the second element of pair. Thus, we formalized the error event in the i^{th} sub-adder as follows:

Definition 6 (*Sub-Adder Error*)

```
⊢ sub_adder_error p X Y i s r =
big_inter p (pairs_to_sum_events X Y [((i-1)*s,(i-1)*s + r)])
```

Where p represent the probability space and X and Y are random variables that model the inputs to the given sub-adder. In order to make the function generic, instead of considering just one pair for $G0$ and P events of sub-adder, we considered a list of such pairs as errors can simultaneously occur in more than one sub-adder which may lead to more than one generation and propagation events, as discussed in Section 3.

5.2 Formalization of Conversion from List of Pairs to Error Events in Sub-adders

The function *pairs_to_sum_events* converts the list of pairs to the corresponding events, as shown in Fig. 3.2(f), and is formalized in HOL4 as follows:

Definition 7 (*Conversion from Pairs to Events*)

```
⊢ pairs_to_sum_events X Y (L:(num # num) list) =
  ( [(G0_EVENT X Y (FST (HD L)));
   (P_EVENT X Y ((SND (HD L)) - (FST (HD L))))] ++
    (pairs_to_sum_events_helper X Y (SND (HD L)) (TL L)))
⊢ (pairs_to_sum_events_helper X Y (lv:num) [] = [] ) ∧
(pairs_to_sum_events_helper X Y lv (h::(t:(num # num) list)) =
   ([ G1_EVENT X Y ((FST h) - lv) ;
    P_EVENT X Y ((SND h) - (FST h))] ++
    (pairs_to_sum_events_helper X Y (SND h)(t)) ))
```

Where the functions FST and SND return the first and second components of their argument pairs, respectively. The function *pairs_to_sum_events* accepts the list of pairs, L, and generates the corresponding P, $G0$ and $G1$ events by considering that the first element of every pair represents the bit location of end of a generation event and the second element represents the end of the propagation event. It utilizes our formalization of P, $G0$ and $G1$ events, given in Definitions 3-5, which require the number of bits which generate or propagate carry. The number of bits that generate a carry can be found by calculating the difference between the bit location of end of the generation event and the bit location of end of the previous propagation event, $(FSTh) - lv$, where the variable lv in the function *pairs_to_sum_events_helper* represents the bit location where the previous propagation event ends. Similarly, the number of bits that propagate carry can be found by computing the difference between the bit location of the

end of the propagation event and the end of the generation event, $(SND\ h) - (FST\ h)$.

5.3 Formalization of List of Error in Sub-adder

Now, we utilize the error function in a single adder to form a list of error events in all the sub-adders of the given LLAA:

Definition 8 (*List of Sub-Adder Errors*)

```
⊢ (List_sub_adder_errors p X Y 0 s r = [] ) ∧
   (List_sub_adder_errors p X Y (SUC l) s r =
   (if (l = 1 ) then [(sub_adder_error p X Y (SUC l) s r )]
    else List_sub_adder_errors p X Y (l) (s) (r) ++
    [(sub_adder_error p X Y (SUC (l)) s r )]))
```

In this function, l represents the number of sub-adders and $s + r$ represents the number of bits per sub-adder, where r are the bits used to predict carry, i.e, repeating bits, and s represents the rest of the bits of the sub-adder. The function, *List_sub_adder_errors* recursively provides a list of error events corresponding to all sub-adders in the given sub-adder except the first one, since the output of Sub-adder 1 is always accurate as there is no carry prediction involved.

5.4 Formalization of Intersection of Lists of Errors

The next step is to formalize the probability terms, given in the right hand side of Equation 3.3, that involve the intersection of error events, corresponding to the co-occurring errors in sub-adders.

Definition 9 (*Intersection of Lists of Errors*)

⊢ *inter_lists_pairs p X Y =*

∀ *(L1:(num # num) list) L2. ((SND (LAST L1)) ≤ (SND (HD L2))) ⇒*

(((big_inter p (pairs_to_sum_events X Y L1)) ∩

(big_inter p (pairs_to_sum_events X Y L2))) = (big_inter p (pairs_to_sum_events X Y (inter_lists_pairs_helper L1 L2)))

⊢ *(inter_lists_pairs_helper [] (h2::t2) = ([h2] ++ t2)) ∧*

(inter_lists_pairs_helper (h1::t1) (h2::t2) =

((if (FST h2) ≤ (SND h1) then

(if (FST h2) > (FST h1) then ([((FST h1),(SND h2))] ++ t2) else ([h2]++t2)) else (h1 :: inter_lists_pairs_helper t (h2::t2))))

The above functions are used to perform the intersection in a recursive manner by considering the intersection of two events at a time. As depicted in Fig. 3.2(a), for the intersection $E_2 \wedge E_3 \wedge E_4$, the bit location of the end of the propagation event of every i^{th} error event is greater than the previous

error event or the result of intersection of the previous error events We used this result to model the process depicted in Fig. 3.2(b-d) to form a generic method for modeling the intersection of events corresponding to sub-adders.

5.5 Formalization of Error in Overall Approximate Adder

As given in Equation 3.3, the error in the overall adder can be modeled by performing union between all the sub-adder error events.

Definition 10 (*Error in Overall Adder*)

⊢ *ERROR X Y l su r = union_list(List_sub_adder_errors p X Y l su r)*

Based on the above-mentioned formalization of error in a LLAA, we now formally verify Equation 3.3, which can be compactly written as:

$$\Pr(\bigcup_{i=1}^{n} E_i) = \sum_{t1\neq\{\},t\subseteq\{1,2,\ldots,n\}} (-1)^{|t|+1} \Pr\bigcap_{j\in t} E_j. \tag{5.1}$$

Theorem 6 (*Union of Sub-Adder Errors*)

⊢ ∀ *p X Y su r l.* (*p_space p* ∧
(∀ *x.MEM x (List_sub_adder_errors p X Y l su r)* ⇒ *x IN events p*))⇒ (*prob p(ERROR p X Y l su r)) =*
sum_set {*t | t SUBSET (set (List_sub_adder_errors p X Y l su r))* ∧
¬(*t* = {})} (λ*t.* (*- &1) pow (CARD t + 1) * (prob p(BIGINTER t)))*

The proof of this theorem is primarily based on probabilistic Inclusion-Exclusion principle [7]. Theorem 6 plays a vital role in facilitating the formal reasoning about probability of error relationships for LLAA as shown in the next section. Moreover, Theorem 6 also raises the confidence level of our formal modeling, described in this and the last section.

Chapter 6

Case Studies

In this section, we present the formal error analysis of three of the most widely acclaimed approximate adders, i.e., ETA-I, ACA-II and GeAr.

6.1 Probability of Error in ACA-II Adder

Almost-correct adders (ACA) [16] architecture trades speed against accuracy to overcome the theoretical limitations of precise adder designs, producing acceptable results for many applications, by using adders with limited carry-propagation lengths which results in a decreased critical path delay. The generic adder model can be configured to represent the ACA-II adder by setting $S_2 = S_3 = ... = S_L = S = k$, $R_2 = R_3 = ... = R_L = R = k$ and $S_1 = 2k$. For 3 sub-adders probability of error in ACA-II is as follows:

$$\Pr[Error; k] = \Pr[E_2] + \Pr[E_3] - \Pr[E_2 \wedge E_3] \tag{6.1}$$

where $\Pr[P]$ and $\Pr[G0]$ are found using Eqs. (4.4) and (4.5), respectively,

we get

$$\Pr[E_2 \wedge E_3] = \Pr[P_2 \wedge G0_2 \wedge P_3 \wedge G0_3] \tag{6.2}$$

In order to handle the joint probability term we transform the dependent events into independent events according to the proposed methodology, as presented in Section 5. Thus, the probability of error is given as follows:

$$\begin{aligned}
&\Pr[Error; k] = \Pr[P; k, 2k-1]\Pr[G0; 0, k-1] \\
&+ \Pr[P; 2k, 3k-1]\Pr[G0; 0, 2k-1] + \Pr[P; k, 3k-1]\Pr[G0; 0, k-1] \\
&= \frac{1}{2^k}\frac{2^k-1}{2^{k+1}} + \frac{1}{2^k}\frac{2^{2k}-1}{2^{2k+1}} + \frac{1}{2^k}\frac{2^{3k}-1}{2^{3k+1}} - \frac{1}{2^{2k}}\frac{2^k-1}{2^{k+1}} \\
&- \frac{1}{2^{2k}}\frac{2^{2k}-1}{2^{2k+1}} - \frac{1}{2^{2k}}\frac{2^k-1}{2^{k+1}}\left(\frac{2^k-1}{2^{k+1}} + \frac{1}{2^k}\right) + \frac{1}{2^{3k}}\frac{2^k-1}{2^{k+1}}
\end{aligned} \tag{6.3}$$

Which can be formally verified in HOL4 as:

Theorem 7 (*probability of Error in ACA-II with 3 sub-adders*)

⊢ ∀ *p X Y k.* *p_space p* ∧

(∀ l t m. *indep p (PREIMAGE X {l}) (PREIMAGE Y {t})* ∧
(Uniform_pmf p m X l) ∧ *(Uniform_pmf p m Y t))* ∧
mutual_indep_sum_events p X Y ∧ *inter_lists_pairs p X Y* ∧ *(k > 0)*
⇒ *(prob p (union_list (List_sub_adder_errors p X Y 3 k k)) =*
((1/ 2^k) ((2^k-1)/(2^{k+1})) + (1/ 2^k)* ((2^{2*k}-1)/(2^{2*k+1})) -*
(1/ 2^{2*k}) ((2^k-1)/(2^{k+1}))))*

6.2 Probability of Error for the GeAr Adder

Generic Accuracy Configurable (GeAr) [27] is a low latency adder that gives a large number of potential configurations contrasted with cutting edge approximate adders, in this manner empowering a high level of adaptability and exchange-off among performance and output quality. The error model can be used to represent the GeAr adder by considering $R_2 = R_3 = ... = R_L = R$, $S_2 = S_3 = ... = S_L = S$ and $S_1 = S + R$. In GeAr model all the sub-adders are of same size but R and S may or may not be equal. For three sub-adders, the general expression for probability of error can be given by:

$$\Pr[Error] = \Pr[E_2] + \Pr[E_3] - \Pr[E_2 \wedge E_3] \tag{6.4}$$

Since R and S may or may not be equal, 3 sub-adders are divided into two cases when $R \geq S$ and $R < S$. The probability of error for the case $R \geq S$ is given by the following relationship:

$$\begin{aligned} \Pr[Error; S, R] &= \Pr[P; S, S+R-1] \Pr[G0; 0, S-1] \\ &+ \Pr[P; 2S, 2S+R-1] \Pr[G0; 0, 2S-1] \\ &- \Pr[P; S, 2S+R-1] \Pr[G0; 0, S-1] \\ &= \frac{1}{2^R}\frac{2^S-1}{2^{S+1}} + \frac{1}{2^R}\frac{2^{2S}-1}{2^{2S+1}} - \frac{1}{2^{S+R}}\frac{2^S-1}{2^{S+1}} \end{aligned} \tag{6.5}$$

We verified this result in HOL4 as the following theorem:

Theorem 8 (*Probability of Error in GeAr model with 3 sub-adders for $R \geq S$*)

```
⊢ ∀ p X Y su r.  p_space p ∧
```

(∀ l t m. indep p (PREIMAGE X {l}) (PREIMAGE Y {t}) ∧
(Uniform_pmf p m X l) ∧ (Uniform_pmf p m Y t)) ∧
mutual_indep_sum_events p X Y ∧ inter_lists_pairs p X Y ∧
(r ≥ su) ∧ (su > 0) ⇒
(prob p (union_list (List_sub_adder_errors p X Y 3 su r)) =
((1/2^r) ((2^su-1)/(2^(su+1))) + (1/2^r)* ((2^(2*su)-1)/(2^(2*su+1))) -*
(1/2^(su+r)) ((2^su-1)/(2^(su+1)))))*

The proof steps involve Definitions 8, 3, 4, 5 and some basic arithmetic reasoning. Similarly for the case when $R < S$:

$$\begin{aligned}\Pr[Error; S, R] &= \Pr[P; S, S+R-1]\Pr[G0; 0, S-1] \\ &+ \Pr[P; 2S, 2S+R-1]\Pr[G0; 0, 2S-1] \\ &- \Pr[P; S, S+R-1]\Pr[G0; 0, S-1] \\ &\times \Pr[P; 2S, 2S+R-1]\Pr[G1; S+R, 2S-1]\end{aligned} \tag{6.6}$$

For uniformly distributed inputs, $\Pr[Error]$ is given as follows:

$$\begin{aligned}\Pr[Error; S, R] = &\frac{1}{2^R}\frac{2^S-1}{2^{S+1}} + \frac{1}{2^R}\frac{2^{2S}-1}{2^{2S+1}} \\ &- \frac{1}{2^{2R}}\frac{2^S-1}{2^{S+1}}\left(\frac{2^{S-R}-1}{2^{S-R+1}} + \frac{1}{2^{S-R}}\right)\end{aligned} \tag{6.7}$$

Which can be formally verified in HOL4 as:

Theorem 9 (*Probability of Error GeAr adder with 3 sub-adders for $R < S$*)

⊢ ∀ p X Y su r. p_space p ∧
(∀ l t m. indep p (PREIMAGE X {l}) (PREIMAGE Y {t}) ∧
(Uniform_pmf p m X l) ∧ (Uniform_pmf p m Y t) ∧

mutual_indep_sum_events p X Y ∧ inter_lists_pairs p X Y ∧
(r < su) ∧ (0 < r)) ⇒
(prob p (union_list (List_sub_adder_errors p X Y 3 su r)) =
((1/ 2^r) (($2^s u$-1)/(2^{su+1})) + (1/ 2^r)* ((2^{2*su}-1)/(2^{2*su+1})) -*
(1/ 2^{2*r}) (($2^s u$-1)/(2^{su+1})) * (((2^{su-r}-1)/(2^{su-r+1})) + (1/*
2^{su-r}))))

6.3 Probability of Error in ETA-I Adder

The Error Tolerant Adder-I (ETA-I) [37] provides a significant improvement in the power and speed by compromising accuracy. The LLAA model can be configured to represent the ETA-I adder by setting $S_2 = s$, $R_2 = r$ and $S_1 = s + r$. The probability of error in ETA-I can be obtained using the following expression:

$$\begin{aligned} \Pr[Error; k] &= \Pr[E_2] \\ &= \Pr[P_2 \wedge G0_2] \\ &= \Pr[P; s, s+r] \Pr[G0; 0, s] \end{aligned} \tag{6.8}$$

$$\Pr[Error; k] = \frac{1}{2^r} \frac{2^{su} - 1}{2^{su+1}} \tag{6.9}$$

Which can be formally verified in HOL4 as:

Theorem 10 (*Probability of Error ETA-I adder*)

⊢ ∀ *p X Y su r. p_space p ∧*
(∀ l t m. indep p (PREIMAGE X {l}) (PREIMAGE Y {t}) ∧
(Uniform_pmf p m X l) ∧ (Uniform_pmf p m Y t) ∧

```
mutual_indep_sum_events p X Y ∧ inter_lists_pairs p X Y ∧
( r > 0 ) ∧ (su > 0) ) ⇒
   ( prob p (union_list (List_sub_adder_errors p X Y 2 su r)) =
   ((1/ 2^r)* ((2^su -1)/(2^(su+1))) ))
```

We used the proposed formalization, presented in the last two section, to formally verify the expressions, given in Table 6.1, for the probability of error for all the adders. The theorems, presented in Table 6.1, are verified based on the assumptions that the uniform random variables X and Y, which represent the inputs for the sub-adders, are independent and also fulfil the conditions given in the *inter_list_pairs* function, given in Definition 9. We formalized the mutual independence in this context as follows:

Definition 11 (*Mutually Independent Sub-adder Events*)

```
⊢ mutual_indep_sum_events p X Y =
  ( ∀ (L:(num # num) list).  (sorted_list_pairs L ) ⇒
(prob p (big_inter p (pairs_to_sum_events X Y L)) =
    (list_prod ( list_prob p (pairs_to_sum_events X Y L)))) )
```

According to the above function two random variables are mutual independent for all list of pairs for which the pairs appear in the sorted form using the function *sorted_list_pairs*. This function, given below, recursively ensures that every second element of a pair (bit location of end of propagation event), in the given list, is greater than its first element (bit location of end of generation event) and every first element of a pair is greater than the second element of the last pair in the given list:

Definition 12 (*Sorted List of Pairs*)

```
⊢ (sorted_list_pairs [] = F ) ∧
  ( sorted_list_pairs (h::t) =
(((FST h) < (SND h) ) ∧ (sorted_list_pairs_helper t (SND h))))
⊢ (sorted_list_pairs_helper [] c = T ) ∧
   (sorted_list_pairs_helper (h::t) c =
    ((c < (FST h)) ∧ ((FST h) < (SND h)) ∧
      (sorted_list_pairs_helper t (SND h))))
```

To facilitate the reasoning process about the theorems, given in Table 6.1, we also verify the property that if both the lists of pairs are not empty then the intersection of two sorted lists of pairs will also return a sorted list of pairs.

Theorem 11 (*Algorithm for Intersection Pairs returns sorted pairs*)

```
⊢ ∀ (L1:(num # num) list) L2.  ( ¬NULL(L1) ∧ ¬NULL(L2) ) ⇒
  ( ((sorted_list_pairs L1) ∧ (sorted_list_pairs L2)) ⇒
    sorted_list_pairs(algo_inter_pairs L1 L2) )
```

Similarly, another commonly used result for the verification of theorems of Table 6.1 is the fact that if the event $X + Y$ is in event space p then all the sub-adder error events are also in event space p.

Theorem 12 (*Sub-Adder Errors in events p*)

```
⊢ ∀ p X Y n su r.  p_space p ∧
   (∀ u.(PREIMAGE (λs.  X s + Y s) {u} IN events p) ) ⇒
   (sub_adder_error p X Y n su r) IN events p
```

The verification for all the theorems, given in Table 6.1, is based on Theorems 6-12 and some basic arithmetic reasoning.

The distinguishing features of the theorems, presented in Table 6.1, compared to traditional analysis methods, include their generic nature, in terms of universally quantified variables, and the added guarantee of their correctness, due to the involvement of a sound sound theorem proving tool for their verification. All the theorems use universally quantifiers for variables, k, su or r and thus each sub-adder can have any number of bits. Thus, the formally verified expressions hold for any values of these variables, which is clearly a distinguishing feature compared to simulations, which are quite limited for analyzing large adders. The reasoning about the above theorems was very straightforward and was composed of simple rewriting and arithmetic simplification steps. The proof scripts were merely about 250 lines long. These benefits have been attained based on the foundational formalization of probability distributions and events in LLAA and its error analysis algorithm reported in Sections 4 and 5 of the thesis, respectively. This foundational formalization took about 2000 lines of HOL4 code and 600 man-hours and is available for download at [23]. The formalization of the error analysis algorithm was one of the most challenging task of this development. To the best of our knowledge, such a scalable and formal approach for the error evaluation of an approximate adder is not reported in the literature and the error analysis of individual adders for a very small number of bits in sub-adder stages was reported, using paper-and-pencil based analytical analysis, in [19]. Moreover, we had to manually develop the reasoning for verifying the theorems, reported in this dissertation, as almost no support for their verifi-

cation could have been found from literature. Particularly, the verification of the generic error analysis theorem, i.e., Theorem 6, involved the principle of inclusion exclusion and was not very straightforward. The probability theory formalization in HOL4, was found to be very useful for our development.

Table 6.1: Probability of Error in ETA-I, ACA-II and GeAr Adders

Adder	**Error modeling**	**Probability of Error**
ETA-I	2-Sub-adders $R_2 = r$, $S_2 = su$, $S_1 = su + r$	$\forall$ *p X Y su r. p_space p* $\wedge$ $(\forall$ *l t m. indep p (PREIMAGE X {l}) (PREIMAGE Y {t})* $\wedge$ *(Uniform_pmf p m X l)* $\wedge$ *(Uniform_pmf p m Y t)* $\wedge$ *mutual_indep_sum_events p X Y* $\wedge$ *inter_lists_pairs p X Y* $\wedge$ *(r > 0)* $\wedge$ *(su > 0))* $\Rightarrow$ *(prob p (union_list (List_sub_adder_errors p X Y 2 su r)) =* *((1/ 2^r)* ((2^su -1)/(2^su+1))))*
ACA-II	3-Sub-adders $R_2 = R_3 = k$, $S_2 = S_3 = k$, $S_1 = su + r$	$\forall$ *p X Y k. p_space p* $\wedge$ $(\forall$ *l t m. indep p (PREIMAGE X {l}) (PREIMAGE Y {t})* $\wedge$ *(Uniform_pmf p m X l)* $\wedge$ *(Uniform_pmf p m Y t)* $\wedge$ *mutual_indep_sum_events p X Y* $\wedge$ *inter_lists_pairs p X Y* $\wedge$ *(k > 0))* $\Rightarrow$ *(prob p (union_list (List_sub_adder_errors p X Y 3 k k)) =* *((1/ 2^k)* ((2^k -1)/(2^k+1)) + (1/ 2^k)* ((2^2*k -1)/(2^2*k+1)) -* *(1/ 2^2*k)* ((2^k -1)/(2^k+1))))*
GeAr	3-Sub-adders $R_2 = R_3 = r$, $S_2 = su$, $S_1 = su + r$ $R < S$	$\forall$ *p X Y su r. p_space p* $\wedge$ $(\forall$ *l t m. indep p (PREIMAGE X {l}) (PREIMAGE Y {t})* $\wedge$ *(Uniform_pmf p m X l)* $\wedge$ *(Uniform_pmf p m Y t)* $\wedge$ *mutual_indep_sum_events p X Y* $\wedge$ *inter_lists_pairs p X Y* $\wedge$ *(r < su)* $\wedge$ *(0 < r))* $\Rightarrow$ *(prob p (union_list (List_sub_adder_errors p X Y 3 su r)) =* *((1/ 2^r)* ((2^su -1)/(2^su+1)) + (1/ 2^r)* ((2^2*su -1)/(2^2*su+1)) -* *(1/ 2^2*r)* ((2^su -1)/(2^su+1)) * (((2^su−r -1)/(2^su−r+1)) + (1/ 2^su−r))))*
	3-Sub-adders $R_2 = R_3 = r$, $S_2 = S_3 = su$, $S_1 = su + r$ $R \geq S$	$\forall$ *p X Y su r. p_space p* $\wedge$ $(\forall$ *l t m. indep p (PREIMAGE X {l}) (PREIMAGE Y {t})* $\wedge$ *(Uniform_pmf p m X l)* $\wedge$ *(Uniform_pmf p m Y t)* $\wedge$ *mutual_indep_sum_events p X Y* $\wedge$ *inter_lists_pairs p X Y* $\wedge$ *(r ≥ su)* $\wedge$ *(su > 0)* $\Rightarrow$ *(prob p (union_list (List_sub_adder_errors p X Y 3 su r)) =* *((1/2^r)* ((2^su -1)/(2^su+1)) + (1/2^r)* ((2^2*su -1)/(2^2*su+1)) -* *(1/2^su+r)* ((2^su -1)/(2^su+1))))*

Chapter 7

Conclusion and Future Work

7.1 Conclusions

These days, a wide range of applications are exploiting the concept of approximate computing to relax the requirements of precise computing to gain on performance and reduce power consumption. Approximate adders are the most frequently used hardware components in approximate hardware design. In this dissertation, we presented a HOL formalization of a generic methodology, based on the probability theory formalization in HOL4, for analyzing the probability of error for approximate adders. The methodology can be applied to calculate exact probability of error in low latency approximate adders (LLAA). For illustration, we formally verified three commonly used approximate adders, i.e., ETA-I, ACA-II and GeAr, with 3 sub-adders with an arbitrary number of bits. The proposed method can be easily used to verify the expressions for large number of sub-adders as well but the results were not presented in the thesis as the error expressions become large and

difficult to comprehend. In future, we plan to extend the formal reasoning support for low-power approximate adders and approximate multipliers.

7.2 Future Work

The proposed work, the modeling of approximate adders and formal verification of error analysis of approximate adders opens many doors for future work in approximate computing. Approximate adders are the building block of many circuits and with the passage of time and ever-increasing data many systems do not require outcome to be fully precise as a result in future approximate computing demand will increase and this thesis will be helpful for others, working in formal verification of different directions of approximate adders as it provides modeling of approximate adder. Many other low-latency approximate adders have been proposed of which probability of error is yet to be calculated.

Bibliography

[1] Cog http://coq.inria.fr/, 2017.

[2] Hol light. http://www.cl.cam.ac.uk/ jrh13/hol-light/.

[3] Isabelle/hol. https://isabelle.in.tum.de/ 2017.

[4] M. norrish and k. slind. the hol system description. technical report, http://hol.sourceforge.net/documentation.html, 2017.

[5] PRISM http://www.prismmodelchecker.org, 2017.

[6] S. romanenko, c. russo, and p. sestoft. moscow ml language overview. http://mosml.org/mosmlref.pdf, 2000.

[7] Waqar Ahmed and Osman Hasan. Formalization of fault trees in higher-order logic: A deep embedding approach. In *Dependable Software Engineering: Theories, Tools, and Applications*, volume 9984, pages 264–279. Springer, LNCS, 2016.

[8] Melvin Breuer. Hardware that produces bounded rather than exact results. In *Design Automation Conference*, pages 871–876. ACM/IEEE, 2010.

[9] Chad E Brown. *Automated Reasoning in Higher-order Logic: Set Comprehension and Extensionality in Church's Type Theory*. College Publications, 2007.

[10] Srimat T Chakradhar and Anand Raghunathan. Best-effort computing: re-thinking parallel software and hardware. In *Design Automation Conference ACM/IEEE*, pages 865–870. IEEE, 2010.

[11] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Design Automation Conference*, page 113. ACM/IEEE, 2013.

[12] Luca De Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Citeseer, 1997.

[13] Roderic A Girle and Melvin Fitting. First-order logic and automated theorem proving, 1998.

[14] Mike Gordon. Introduction to the hol system. In *HOL Theorem Proving System and Its Applications, 1991., International Workshop on the*, pages 2–3. IEEE, 1991.

[15] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[16] Andrew B Kahng and Seokhyeong Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conference*, pages 820–825. ACM/IEEE, 2012.

[17] Daya S Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. Rumba: An online quality management system for approximate computing. In *Computer Architecture*, pages 554–566. ACM/IEEE, 2015.

[18] L Krishnamachari, Deborah Estrin, and Stephen Wicker. The impact of data aggregation in wireless sensor networks. In *Distributed Computing Systems*, pages 575–578. IEEE, 2002.

[19] Sana Mazahir, Osman Hasan, Rehan Hafiz, Muhammad Shafique, and Jorg Henkel. Probabilistic error modeling for approximate adders. *IEEE Transactions on Computers*, 66(3):515 – 530, 2017.

[20] Tarek Mhamdi, Osman Hasan, and Sofiène Tahar. On the formalization of the lebesgue integration theory in hol. In *International Conference on Interactive Theorem Proving*, number 6172, pages 387–402. Springer, LNCS, 2010.

[21] Robin Milner. *The definition of standard ML: revised*. MIT press, 1997.

[22] David Anthony Parker. *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, University of Birmingham, 2002.

[23] Amina Qureshi. Formal error analysis of low latency approximate adders in hol4, http://save.seecs.nust.edu.pk/projects/FEALLAA/, 2017.

[24] Michael Ringenburg, Adrian Sampson, Isaac Ackerman, Luis Ceze, and Dan Grossman. Monitoring and debugging the quality of results in approximate programs. In *ACM SIGPLAN Notices*, volume 50, pages 399–411. ACM, 2015.

[25] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, volume 46, pages 164–174. ACM, 2011.

[26] Koushik Sen, Mahesh Viswanathan, and Gul A Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *QEST*, volume 5, pages 251–252, 2005.

[27] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jorg Henkel. A low latency generic accuracy configurable adder. In *Design Automation Conference*, pages 1–6. ACM/IEEE, 2015.

[28] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Foundations of software engineering*, pages 124–134. ACM, 2011.

[29] Konrad Slind and Michael Norrish. A brief overview of hol4. In *International Conference on Theorem Proving in Higher Order Logics*, pages 28–32. Springer, 2008.

[30] Renée St Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger.

General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News*, 42(3):505–516, 2014.

[31] Rangharajan Venkatesan, Amit Agarwal, Kaushik Roy, and Anand Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *Computer-Aided Design*, pages 667–673. IEEE Press, 2011.

[32] Ajay K Verma, Philip Brisk, and Paolo Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Design, automation and test in Europe*, pages 1250–1255. ACM, 2008.

[33] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. Approximate computing: A survey. *IEEE Design & Test*, 33(1):8–22, 2016.

[34] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. On reconfiguration-oriented approximate adder design and its application. In *Computer-Aided Design*, pages 48–54. IEEE Press, 2013.

[35] Lofti A Zadeh. Fuzzy logic, neural networks, and soft computing. *Communications of the ACM*, 37(3):77–85, 1994.

[36] Qian Zhang, Feng Yuan, Rong Ye, and Qiang Xu. Approxit: An approximate computing framework for iterative methods. In *Design Automation Conference*, pages 1–6. ACM/IEEE, 2014.

[37] Ning Zhu, Wang Ling Goh, and Kiat Seng Yeo. An enhanced low-power high-speed adder for error-tolerant application. In *Integrated Circuits*, pages 69–72. IEEE, 2009.