Design of low-power high speed error-tolerant adder and its application in digital signal processing

Zhang, Weijia

2008

Zhang, W. (2008). Design of low-power high speed error-tolerant adder and its application in digital signal processing. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/15559>

<https://doi.org/10.32657/10356/15559>

DESIGN OF LOW-POWER HIGH-SPEED ERROR-TOLERANT ADDER AND ITS APPLICATION IN DIGITAL SIGNAL PROCESSING

SUBMITTED

BY

ZHANG WEIJIA

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF ENGINEERING

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

2008

Abstract

As technology advances, errors/defects in integrated circuits become unavoidable. At the same time, the pursuit of low-power and high-speed circuits is always restricted by the conventional circuit design technology. In this context, several new technologies that regard the accuracy of circuit as a new design parameter other than the conventional design metrics have been proposed. These technologies trade the accuracy of circuit for the improvements in power consumption and/or speed performance.

Stimulated by those emerging technologies, a novel and innovative type of adder, the Error-Tolerant Adder (ETA), is proposed. The detailed theoretical studies and circuit designs of two different realizations of this new type of adder are presented in this thesis. By incorporating special addition algorithms and circuit structures, and sacrificing certain degree of accuracy, the proposed ETA is able to achieve significant improvements in power consumption and speed performance as compared to the conventional adders.

To illustrate the practicality of the proposed ETA in real applications, the Fast Fourier Transform (FFT) function, which is a basic and important function in Digital Signal Processing (DSP), is taken as the platform to employ the proposed designs. This ETA-based FFT function is put in the context of digital image processing to demonstrate its functionality. Simulation results show that with a well-designed ETA, the ETA-based FFT function can be used in digital image processing to generate acceptable results.

Acknowledgement

Firstly, I would like to express my most sincere gratitude to my supervisors, Associate Professor Goh Wang Ling and Associate Professor Yeo Kiat Seng, for their countless help and continuous supports throughout the project. Their knowledgeable advices and guidance are indispensable for the completion of this project. The knowledge and thoughts I have gained from them through the numerous discussions with them will definitely benefit my future life.

I would also like to thank Mr. Loy Liang Yu and Mr. Zhu Ning, for their kind help in the course of the project. The discussions with them kindled my thought. They are also the co-authors of my two published/submitted papers, respectively.

In addition, I would like to give my thanks to my parents and my friend Zhang Bingzhi, for their supports and encouragements in the past two years.

At last, I would like to thank Nanyang Technological University for providing me the research scholarship to support me to complete the project.

Table of Contents

	Page
Abstract	i
Acknowledgement	ii
Chapter 1 Introduction	1
1.1 Background and Motivation	1
1.2 Objective	4
1.3 Organization of Thesis	4
Chapter 2 Literature Review	5
2.1 Probabilistic CMOS (PCMOS)	5
2.1.1 Concepts	5
2.1.2 Probabilistic Switch	5
2.1.3 Relationship between Probability and Energy Consumption	7
2.1.4 Applications of PCMOS Technology	8
2.2 Error-Tolerance	11
2.2.1 Concepts	11
2.2.2 Integrated Circuit Testing Methodology that Supports Error-Tolerance	12
2.2.3 A Case Study of Error-Tolerance	13
2.3 Conventional Designs of Digital Adder	15
2.3.1 Half Adder and Full Adder	15
2.3.2 Ripple-Carry Adder	20
2.3.3 Carry-Skip Adder	21
2.3.4 Carry-Select Adder	22
2.3.5 Carry-Lookahead Adder	24
2.3.6 Carry-Save Adder	28
2.3.7 Chinese Abacus Adder	28

2.4	Power Consumption of Adder	32
2.4.1	Dynamic Power Consumption	32
2.4.2	Short-Circuit Power Consumption	33
2.4.3	Static Power Consumption	34
Chapter 3	Error-Tolerant Adder	36
3.1	Introduction	36
3.2	ETA Type I	37
3.2.1	Proposed Addition Algorithm	38
3.2.2	Relationships between AP, MAA, Dividing Strategy, and Size of Adder	40
3.2.3	Hardware Implementation	44
3.2.4	Design of a 32-bit ETAI	45
3.2.5	Circuit Simulation	49
3.2.6	Optimization of the Proposed 32-bit ETAI	52
3.2.7	Comparison with Conventional Adders	54
3.2.8	Further Study of the Relationship between Accuracy Performance and Input Patterns	55
3.3	ETA Type II	57
3.3.1	Theoretical Analysis	57
3.3.2	Architecture of ETAIL	58
3.3.3	Dividing Strategy	59
3.3.4	Implementation of a 32-bit ETAIL	60
3.3.5	Relationship between Accuracy Performance and the Range of Input Patterns	61
3.3.6	Modified ETAIL	62
3.3.7	Comparison with Conventional Adders	64
3.4	Comparison between ETAI and ETAIL	64

Chapter 4 Application of ETA in Digital Signal Processing	66
4.1 Applications of ETA	66
4.2 Fast Fourier Transform and Digital Signal Processing	67
4.2.1 Discrete Fourier Transform (DFT)	67
4.2.2 Fast Fourier Transform (FFT)	68
4.2.3 Software Implementation of FFT	69
4.2.4 Application of FFT in DSP	71
4.2.5 Fixed-Point Number and Floating-Point Number	72
4.3 ETA-based FFT Function	74
4.4 Digital Image Processing	79
4.5 Application of ETA-based FFT in Digital Image Processing	82
Chapter 5 Conclusions and Suggestions for Future Work	86
5.1 Conclusions	86
5.2 Suggestions for Future Work	88
Publications	90
References	91
Appendices	95
Appendix A: Hspice netlist of ETAI	95
Appendix B: C code for testing the accuracy of ETAI	99
Appendix C: Hspice netlist of ETAI	105
Appendix D: C code for testing the accuracy of ETAI	109
Appendix E: Hspice netlist of ETAI	112

Chapter 1 Introduction

1.1 Background and Motivation

The famous Moore's Law provides us an important trend in the development of integrated circuit technology. According to Moore's Law, the number of transistors that can be inexpensively placed on an integrated circuit doubles every two years [1]. This trend has continued for about half a century and is not expected to stop in at least next decade. However, as the feature size of the complementary metal-oxide-semiconductor (CMOS) devices approaches the deep sub-micron "nano-scale", significant challenges to sustaining Moore's Law have emerged. Two of these challenges are the impact of noise [2, 3, 4] and achieving low-power consumption [5, 6]. The conventional view towards the unexpected noise is treating it as an impediment and trying the best to eliminate its impact. It is stated in the 2003 International Technology Roadmap for Semiconductors (ITRS) [7] that the increasing noise sensitivity has become an important issue in the design of devices, circuits, and systems due to a reduction in operating voltage by 20% per technology node. However, the requirement for increasing noise immunity contradicts with the traditional methodology to achieve low-power consumption, which is addressed by voltage scaling, as reducing the voltage level may greatly degrade the noise immunity of the circuits.

Under this circumstance, a new technology, Probabilistic CMOS (PC MOS) technology, was proposed [8, 9, 10]. In contrast with conventional point of view, the PC MOS technology regards the noise in a digital integrated circuit as a resource rather than an impediment. By introducing noise into a digital integrated circuit, errors are injected into the circuit and this results in a circuit that behaves probabilistically

rather than being deterministic. As such, the PCMOS circuit is also known as probabilistic circuit. There are two categories of applications that can make use of the PCMOS technology. One is the ultra-low power application and the other is the probabilistic application. On one hand, by allowing the existence of certain errors generated by noise, the PCMOS circuit relaxes the limitation of voltage scaling, allowing the circuit to operate with very low supply voltage, so that to be used in those ultra-low power computational systems. On the other hand, the probabilistic character of a PCMOS circuit makes it an excellent candidate for implementing probabilistic algorithms [11].

The PCMOS technology only considers the impact of noise that may generate errors in a digital integrated circuit. As the scale of integrated circuits become larger and larger, many factors other than noise, such as the process variations and the interconnect defects, are likely to cause very unpredictable circuit performance. It is actually difficult to make a defect-free chip [7, 12]. A similar but more general concept, the Error-Tolerance technique, which takes considerations of the possible errors generated by different kinds of factors, was proposed by Professor Breuer [13]. By avoiding making special effort to detect and eliminate all the errors in a system, the Error-Tolerance technique can be used to implement ultra-low power systems.

The common ground of the PCMOS technology and the Error-Tolerance technology is that they both allow the existence of certain amount of errors and trade the accuracy loss for the improvements in power consumption and/or other performance metrics. The major difference between these two technologies is that the PCMOS technology focuses more on the physical nature (noise) of a circuit so that the relevant researches and designs are at the transistor-level while the Error-Tolerance technology considers a more general range of error-generating factors and targets at the system- or application-level.

Since the original concept of Error-Tolerance proposed by Professor Breuer is derived

from the perspective of digital integrated circuit testing, it mainly concentrates on defect models such as the stuck-at, bridging, and delay faults [48]. The benefits of an error-tolerant circuit are also limited to the cost of manufacturing, verification, and testing. In this thesis, the concept of Error-Tolerance has been extended from the field of circuit testing to the field of circuit design. The error-generating factors have also been expanded from the defect models to more general ones, such as circuit structures and computation algorithms. When “imperfect” algorithms and circuit structures are employed, the substantial yields for an error-tolerant digital circuit, in terms of power consumption, speed performance, and transistor count, will be obtained.

Adopting the ideas and techniques in PCMOS and Error-Tolerance technologies in the design of digital adders, a novel and innovative type of adder—Error-Tolerant Adder (ETA) has been designed and this is the major contribution of the thesis. The incentive to design such a new type of adder using the emerging technologies is the fact that adder is the most critical arithmetic block in computational systems and is always the dominant factor in determining the overall performance of a system. For modern computational systems, the increasingly huge data set and the need for instant response require the adder to be large and fast. Meanwhile, as portable digital devices become more and more popular, the requirement on power consumption has also become rigorous. The conventional Ripple-Carry Adder consumes very low power, but its speed performance hinders it from being employed in high-speed systems. The Carry-Lookahead Adder has excellent speed performance due to its intrinsic advantage in eliminating the carry propagation. However, its characteristics of high power consumption and large circuit area render it not suitable for use in low power systems. As a matter of fact, one of the restrictions in conventional digital circuit design is the trade-off between power consumption and speed performance that always exists. Obtaining high speed usually means more power will be consumed and low power will normally degrade the speed of a circuit. So, to breakthrough this bottleneck in conventional technologies for designing a real low-power and high-speed digital circuit, a new metric besides power and speed should be brought

into the design process. In the proposed designs, the *accuracy* plays the role of such a new metric. By sacrificing some degree of accuracy, great improvements in both power consumption and speed performance can be achieved.

1.2 Objective

The first objective of this work is to introduce a new type of adder—ETA and its two realizations with different addition algorithms. The second objective is to provide a detailed description of the hardware implementations of the proposed ETA's. The simulation results of the ETA's will be compared with conventional adders to demonstrate the advantages of the proposed ETA's. The third objective is to discuss on the application of the proposed ETA's in digital signal processing systems and to illustrate the practicality of ETA in real applications.

1.3 Organization of Thesis

The thesis is organized in the following manner. A literature review of PCMOS technology, Error-Tolerance technique and conventional digital adder designs is provided in Chapter 2. Chapter 3 presents the ETA designs, including the mathematical analyses, hardware implementations, simulation results, and comparisons with conventional designs. Two different realizations of ETA are presented in this chapter. The application of the proposed ETA in DSP systems is discussed in Chapter 4. Finally in Chapter 5, the conclusions of this work and the suggestions for future work are given.

Chapter 2 Literature Review

2.1 Probabilistic CMOS (PCMOS)

2.1.1 Concepts

PCMOS technology was originated from Professor Krishna V. Palem's theory of probabilistic switching [8]. As mentioned in Section 1.1, the PCMOS technology regards the noise of a digital integrated circuit as resource rather than impediment, making the conventional deterministic circuits probabilistic. In a PCMOS circuit, the outputs are not always correct, rather, they can only be correct with certain probability. This *probability of correctness*, which is often simply named as *probability* when no confusion would occur, is taken as the most important parameter in PCMOS technology. The value of the probability of correctness ranges theoretically from 0 to 1. When the probability equals to 1, the PCMOS circuit becomes conventional CMOS circuit. Therefore, the conventional CMOS circuit can actually be viewed as an extreme situation of PCMOS circuit. As for the lower bound, when the probability is lower than 0.5, the circuit will most often generate errors instead of giving correct results. Hence, the meaningful value range of probability is from 0.5 to 1.

2.1.2 Probabilistic Switch

In PCMOS technology, the most basic and smallest cell is the probabilistic switch (p-switch). It is simply a CMOS switch with a noise source coupled at its input node [10]. The prototype of a p-switch is depicted in Figure 2.1. Just as the CMOS switch is the nucleus of conventional digital designs, the p-switch is the foundation of all PCMOS digital designs.

Figure 2.2 shows the realization of a p-switch in today's technology [10]. The resistor shown in the figure is taken as a source of thermal noise. Theoretically, the noise introduced to the circuit can be any kind of noise. The thermal noise is usually taken as the target for study, because, on one hand, it widely exists in all kinds of circuits, and on the other hand, it is a random variable following the Gaussian distribution whose statistical characteristics are meaningful and easy to control. The amplifier added after the noise source is to amplify the noise signal to a much higher level that is comparable to the supply voltage that can be obtained in today's technology. In fact, the PCMOS technology aims at the future technology where the operation voltage of a digital circuit can be reduced to a very low level that is comparable to the naturally generated noise signal without amplification. So, to some extent, the amplifier is only used for study purpose and may eventually be eliminated.

Figure 2.1 Prototype of p-switch

Figure 2.2 Realization of a p-switch

2.1.3 Relationship between Probability and Energy Consumption

According to the investigation that had been done, when the thermal noise source, which is a random variable following the Gaussian distribution, is coupled at the input node of a CMOS switch, the probability of correctness of this p-switch can be computed as in Equation (2.1) [10]:

$$p = \frac{1}{2} + \frac{1}{4} \operatorname{erf}\left(\frac{V_m}{\sqrt{2}\sigma}\right) - \frac{1}{4} \operatorname{erf}\left(\frac{V_m - V_{dd}}{\sqrt{2}\sigma}\right) \quad (2.1)$$

where p is probability of correctness, V_m is the threshold voltage of the switch, V_{dd} is the supply voltage, σ is the RMS value of noise, and erf is the well-known

error function [14], whose expression is $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\frac{t^2}{2}} dt$. This equation can be

derived from Figure 2.3. The probability of correctness is equal to $p = 1 - \frac{e_{10} + e_{01}}{2}$

[10], which leads to Equation (2.1).

Figure 2.3 Probability density of correctness of the p-switch [10]

Assume that $V_m = \frac{1}{2}V_{dd}$, Equation (2.1) can be simplified to:

$$p = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{V_{dd}}{2\sqrt{2}\sigma}\right) \quad (2.2)$$

Equation (2.2) can also be expressed as follow:

$$V_{dd} = 2\sqrt{2}\sigma \times \operatorname{erf}^{-1}(2p - 1) \quad (2.3)$$

It is also known that for one switching step, the energy consumption can be computed as follow:

$$E = \frac{1}{2} C V_{dd}^2 \quad (2.4)$$

where E is the energy consumption and C is the load capacitance of the switch.

Then, by substituting Equation (2.3) into Equation (2.4), the relationship between probability and energy consumption of a p-switch can be expressed as:

$$E = 4C\sigma^2[\text{erf}^{-1}(2p-1)]^2 \quad (2.5)$$

As shown in Equation (2.1), the probability of a p-switch depends on the supply voltage and the RMS value of noise. This conclusion leads to the following useful consequence: *To tune the probability of a p-switch, there are two ways: either by adjusting the supply voltage or by changing the amplitude of the noise signal.*

According to Equation (2.5), the other conclusion can be drawn that the energy consumption (E) of a p-switch is exponentially related to the probability (p) and quadratically to the RMS value of noise (σ). Then another consequence can be deduced: *A small amount of the probability of a p-switch can be traded for a great improvement in energy consumption whenever the magnitude of noise remains constant.*

Actually, the above two consequences can be extended to any other PCOMS digital circuits and thus form the theoretical foundation for the PCMOS technology.

2.1.4 Applications of PCMOS Technology

As mentioned in Section 1.1, there are two categories of applications that can make use of the PCMOS technology. An example of low-power application is presented in

[15].

By applying the *biased voltage scaling* (BIVOS) scheme and taking the impact of noise into consideration, the PCMOS adder was proposed in [15]. The BIVOS approach is based on the precondition that each one-bit adder contains noise in its circuit and thus has an associated probability of correctness. Its core idea is that the higher order bits of a binary sequence play a more significant role in representing a number so that should contain fewer errors than the lower order bits do. To achieve low-power computation while still maintaining a high accuracy, the one-bit adder cells used for computing the higher order bits should be assigned with higher supply voltages whereas the lower order bits can be assigned with lower supply voltages. According to Equation (2.2), higher supply voltage leads to higher probability while lower supply voltage has the inverse effect. The BIVOS scheme is depicted in Figure 2.4.

Figure 2.4 BIVOS scheme in PCMOS adder design

To illustrate the advantages of this BIVOS-based PCMOS adder in the application context, the experiment that embedding the PCMOS adder (software implementation) into the *synthetic aperture radar* (SAR) imaging [16] system has been performed. Although some errors have been injected into the system by the PCMOS adder, the output image is visually indistinguishable with the image after standard SAR processing. Meanwhile, the SAR system employing the PCMOS adder yields a great energy saving. If using the conventional uniform voltage scaling scheme, to achieve the same energy saving, the quality of the output image will be degraded to an

unacceptable level, provided that the noise of the same magnitude exist. The simulation results are presented in [15].

The other kind of application is the probabilistic system. A good example has been described in [17]. A Bayesian network is a probabilistic graphical model that represents a set of variables and their probability dependencies [25]. Because of the probabilistic character of the Bayesian network, the PCMOS technology can be made use of in the hardware implementation of a Bayesian network.

The critical part of a Bayesian network is the random number generator. In the proposed design of hardware implementation of Bayesian network in [17], the p-switches are used to generate the probabilistic bit sequences. Compared with the conventional hardware Pseudo-Random Number Generator (PRNG), the PCMOS-based random bit generator consumes less power, costs smaller area, has higher speed, and more importantly, generates outputs with higher quality of randomness. The output of a PCMOS circuit is highly randomized because the noise introduced into the circuit is a “natural” source rather than a “man-made” source.

The general structure of the PCMOS-based hardware implementation of a Bayesian network is shown in Figure 2.5. The whole system consists of two major parts: the probabilistic generating block and the logic network. The probabilistic generating block is made up of a number of probabilistic generating cells (PGC). Each PGC, whose structure is given in Figure 2.6, can generate a bit of “1” with certain probability. As shown in the figure, a PGC consists of three parts: a p-switch, a buffer, and a flip-flop. The p-switch is used to generate random bit sequence. The buffer is to strengthen the output signal of the switch and to restore the signals whose voltage levels hover around $\frac{V_{dd}}{2}$ to the logic “high” or “low”. The flip-flop added here is for synchronization purpose. The random bits generated by the probabilistic generating block are then input into the subsequent logic network to be further processed.

Figure 2.5 Architecture of the PCMOS-based hardware implementation of Bayesian network

Other applications of PCMOS technology include: random neural network [26], probabilistic cellular automata [27], hyper-encryption [28], and so on.

Figure 2.6 Probabilistic Generating Cell (PGC)

2.2 Error-Tolerance

2.2.1 Concepts

In conventional digital VLSI design, a usable circuit/system is usually assumed to be perfect and can always give us definite and accurate results. But such perfect things

can actually seldom be found in the real non-digital world. This world always accepts “analog computation”, which generates “good enough” results rather than totally accurate results [12]. In fact, for many digital systems, the data they process have already contained errors. In many applications, for example, a communication system, the analog signal coming from outside world is first sampled and quantized to digital data on the front end, then the digital data are processed and transmitted in a noisy channel, at last the digital data are converted back to analog signal on the back end. In this process, errors may occur everywhere. Since it is impossible or difficult to constantly maintain the correct data/results, it may be better for users to be more “generous” to accept certain amount of errors. This is the basic idea of Error-Tolerance.

According to the definition given in [18], a circuit is error-tolerant with respect to a specific application, if (1) it contains defects that cause internal and may cause external errors, and (2) the system that incorporates this circuit produces acceptable results. When incorporates the error-tolerant circuit, a digital system is no longer totally “correct”. Instead, certain errors may be generated in the output. This “imperfect” attribute seems to be not appealing. However, the need for the error-tolerant circuit was foretold in the 2003 International Technology Roadmap for Semiconductors (ITRS) [7]. It was quoted that: “Relaxing the requirement of 100% correctness in both transient and permanent failures of signals, logic values, devices, or interconnects may reduce the cost of manufacturing, verification and testing.”

2.2.2 Integrated Circuit Testing Methodology that Support Error-Tolerance

The original concept of Error-Tolerance is derived from the perspective of circuit testing, so several testing methodologies that support error-tolerance have been proposed and developed [20, 23, 24]. Although the testing methodology is not the concern of our work, the ideas, attributes, and analysis methods proposed in these

work help us build a better view of error-tolerant digital integrated circuits design, which is the main contribution of this thesis.

In conventional integrated circuit testing techniques, the targets of testing are all possible faults that may occur in the circuit. However, in the error-tolerance supported testing methodology, the targets of testing are reduced to only the **unacceptable** faults that are predetermined by designer/user.

An important attribute that has been proposed in the error-tolerance supported testing is the error-rate. It is defined as the fraction of incorrect results that a system produces [19]. Figure 2.7 shows an error-rate based testing methodology that supports error-tolerance [23]. In this methodology, each individual fault in the target circuit has a corresponding error-rate that quantitatively indicates the probability that the specific fault happens in the target circuit. For every error-tolerance supported system, there is a maximum acceptable system error-rate specified by the designer/user. Those faults whose error-rates are higher than the maximum acceptable system error-rate are considered as unacceptable faults while the rest faults are expected to be tolerated by the system. The idea and attribute described in the error-tolerance supported testing methodology are actually the prototype of the idea and attribute that will be employed in the ETA design.

Figure 2.7 Error-rate based testing methodology

2.2.3 A Case Study of Error-Tolerance

A framework for the analysis of the applicability of the Error-Tolerance technique is presented in [29]. The framework is illustrated with respect to a digital telephone-answering device (DTAD).

The target system of DTAD has two main components: the microcontroller and the flash memory, which is assumed to be defective. In the proposed framework, the relationships between the *defect density (error-rate)*, the *acceptable performance*, and the *effective yield* are investigated. The *defect density* is defined as the ratio between the number of faults and the size of the flash memory. The *acceptable performance* is referred to the performance (subjective or objective) that is acceptable to the user according to certain measurement standard. The *effective yield* represents the yield in manufacturing process due to the employment of Error-Tolerance technique.

A brief introduction of the working mode of the DTAD is given as follow. In the answering mode, the ADC device in the system samples and quantizes the speech signal, the codec encodes this quantized signal, and the output bit-stream is stored in the flash memory. When the user listens to the recorded speech, the microcontroller extracts the encoded data stored in the memory, and the codec decodes the data and finally recovers the speech.

Because the flash memory employed in the DTAD is defective, the quality of the output of this system is degraded. If the “imperfect” output is acceptable to the user according to certain measure standard, this system can be regarded as an error-tolerant system.

The fault model considered in [29] is the multiple stuck-at fault model. The erroneous bits in the memory are either stuck-at-1 or stuck-at-0. Faults are randomly allocated through the memory based on the uniform distribution. Then twenty different fault

densities between 0% and 1% are simulated. For each fault density, fifty different random distributions of faults are considered.

To measure the quality of the performance of the target DTAD, a kind of subjective test whose guidelines form a *mean opinion score* (MOS) [30] is conducted to the simulation results. The qualitative interpretations of the MOS are: 1 (bad), 2 (poor), 3 (fair), 4 (good), 5 (excellent). According to [29], if the acceptance threshold value T , which is the lowest acceptable MOS, is set to 3 (fair), the corresponding acceptable fault density for the DTAD is 0.20%. That means when 0.20% of all the bits in the flash memory are defective, the whole system still has acceptable performance. The resulting yields for this error-tolerant DTAD can reach to around 75%, which is a substantial improvement.

2.3 Conventional Designs of Digital Adder

Adder is the most basic and important cell in most computational systems. It is usually the dominant factor in determining the overall performance of the whole system. Before the ETA is discussed, a brief review of the conventional designs of adder is given first.

2.3.1 Half Adder and Full Adder

A half adder accepts two input bits (A and B) and generates two output bits, sum (S) and carry-out (C_o). Table 2.1 is the truth table for a half adder. The Boolean expressions are given in Equations (2.6) and (2.7):

$$S = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B} \quad (2.6)$$

$$C_o = A \cdot B \quad (2.7)$$

The logic structure of a half adder is shown in Figure 2.8.

Table 2.1 Truth table for half adder

A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 2.8 Logic structure of half adder

Table 2.2 Truth table for full adder

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A full adder takes 3 inputs, two addend bits (A and B) and a carry-in bit (C_i), and, like the half adder, generates 2 outputs, sum (S) and carry-out (C_o). The truth table for a full adder is given in Table 2.2.

According to the truth table, the Boolean expressions for the full adder can be derived as follows:

$$\begin{aligned} S &= A \oplus B \oplus C \\ &= \overline{A} \cdot \overline{B} \cdot C_i + \overline{A} \cdot B \cdot \overline{C_i} + A \cdot \overline{B} \cdot \overline{C_i} + A \cdot B \cdot C_i \end{aligned} \quad (2.8)$$

$$C_o = A \cdot B + A \cdot C_i + B \cdot C_i \quad (2.9)$$

For many implementation strategies, such as Carry-Lookahead Adder, the intermediate signals, G (generate), D (delete), and P (propagate) are needed in the design processes. These three intermediate signals are defined as follows:

$$G = A \cdot B \quad (2.10)$$

$$D = \overline{A} \cdot \overline{B} \quad (2.11)$$

$$P = A \oplus B \quad (2.12)$$

With the above, the expressions for S and C_o can be written in terms of P and G :

$$S = P \oplus C_i \quad (2.13)$$

$$C_o = G + P \cdot C_i \quad (2.14)$$

One possible logic structure of a full adder is shown in Figure 2.9. There is a variety of implementations of a full adder with different circuit structure, transistor count, and performance. Figure 2.10 provides the schematic diagrams of six different implementations of a full adder. Figure 2.10 (a) is the conventional 28-transistor full adder (28T) which is a complementary CMOS circuit derived directly from the logic equation [31]. The drawbacks of the 28T adder are that it consumes a large circuit

area and its speed is slow. Figures 2.10 (b) and (c) show the transmission gate adder (TGA) [32] and transmission function adder (TFA) [33] that are based on the transmission gate and transmission function theory, respectively. They have less transistor count than the 28T adder. The implementations with even lesser transistors have also been proposed [34, 35, 36]. Figures 2.10 (d), (e), and (f) present the static energy-recovery full adder (SERF) [34], 14-transistor full adder (14T) [35], and 10-transistor full adder (10T) [36], respectively. Full adders with only 10 transistors (e.g., SERF and 10T) have the least number of transistors in existing technology. These three types of full adder consume small circuit area and have good performance in power consumption. The downside is that they suffer from the threshold-loss (non-full swing) problem. Note that all these circuits can be implemented using minimum-sized transistors.

Figure 2.9 Logic structure of full adder

(a) 28-transistor full adder [31]

(b) Transmission gate full adder [32]

Figure 2.10 Different implementations of full adder

(c) Transmission function full adder [33]

(d) Static energy-recovery full adder [34]

(e) 14-transistor full adder [35]

Figure 2.10 (continued) Different implementations of full adder

(f) 10-transistor full adder [36]

Figure 2.10 (continued) Different implementations of full adder

2.3.2 Ripple-Carry Adder

Ripple-Carry Adder (RCA) [31] is the simplest architecture of adder. An N -bit RCA is just constructed by cascading N full adders in series. The carry-out signal of one full adder serves as the carry-in signal of the next full adder, i.e., $C_{o,k} = C_{i,k+1}$, where $0 \leq k \leq N - 2$. The structure diagram is demonstrated in Figure 2.11.

Because of the simple and regular structure, RCA consumes less power and occupies smaller area than any other conventional adders. However, the time delay of this architecture can be enormous. In the worst case, the carry signal will be propagated from the LSB all the way to the MSB. So the critical path in RCA is the entire carry propagation chain. The delay time is linearly proportional to the total number of full adders, N . Thus, RCA is regarded as the slowest adder among all conventional adders and cannot meet the rigorous requirement on circuit/system speed in today's technology.

Figure 2.11 Ripple-Carry Adder

To shorten the critical path of adder, many techniques have been developed. In the following subsections, several improved architectures of adder are presented.

2.3.3 Carry-Skip Adder

Carry-Skip Adder (CSK) [37] is also named as Carry-Bypass Adder. Its concept can be illustrated by Figure 2.12. For a 4-bit adder module, an additional connection between the carry-in signal $C_{i,0}$ and the carry-out signal $C_{o,3}$ is added to the normal carry propagation path via a multiplexer. When all the propagation signals P_k ($k=0, 1, 2, 3$) in such a module are high (i.e., $P_0P_1P_2P_3=1$), the carry-in signal $C_{i,0}$ is forwarded immediately to the next block as the carry-out signal $C_{o,3}$, by skipping the whole propagation path in this block. If this is not the case, the carry-out signal is obtained through the normal carry propagation path. The block diagram of a 16-bit CSK is given in Figure 2.13. The critical path of the adder is shaded in gray in the figure.

Figure 2.12 4-bit Carry-Skip Adder

Figure 2.13 16-bit Carry-Skip Adder

2.3.4 Carry-Select Adder

The major problem of Ripple-Carry Adder is that each full adder cell has to wait for the carry signal coming from the previous stage before a correct carry-out signal can be generated. The idea of Carry-Select Adder (CSL) [38] is to consider both possible values of the carry-in signal and generate the carry-out signals for both possibilities in advance. Once the “real” value of carry-in is known, the correct result will be selected with a simple multiplexer stage. Figure 2.14 demonstrates an implementation of the CSL. From the figure, it can be seen that the whole adder has been divided into a number of equal-length adder stages. For each stage, instead of waiting for the arrival

of the carry generated by the previous stage, both the “0” and “1” possibilities are evaluated. When the carry-in signal finally settles, either of the two possible results is selected and passed to the next stage. In this way, the critical path is greatly shortened compared with the RCA.

Figure 2.14 Linear Carry-Select Adder

The structure in Figure 2.14 can actually be further optimized. For each multiplexer, there are three inputs, two pre-calculated carry signals that serve as the candidates to be selected and the real carry signal coming from previous stage that plays the role as a control signal. It can be observed that there exists a mismatch between the arrival times of those signals. The outputs of the two parallel carry-generation blocks are stable long before the control signal arrives. To equalize these two propagation paths, the full adder stages can be built in a progressive-sized manner instead of the equal-sized manner. The modified structure is illustrated in Figure 2.15. In the original structure, each stage contains the same number of full adder cells. The delay time of this structure is linearly proportional to the size of the adder, N , so the adder with this structure is called Linear Carry-Select Adder (LCSA) [31]. On the other hand, in the modified structure shown in Figure 2.15, each stage contains different number of full adder cells and the number increases by one from one stage to the next. The delay time of the modified structure is proportional to $\sqrt{N}$ instead of N , so the adder with the modified structure is called Square-Root Carry-Select Adder (SRCSA) [31].

Figure 2.15 Square-Root Carry-Select Adder

The major problem of the CSL is that an additional set of carry generation circuits is needed so that the whole circuit consumes more power and occupies more area.

2.3.5 Carry-Lookahead Adder

In the CSK and CSL described above, the carry-rippling effect still exists even though they have shortened the critical path in one way or another. To design even faster adders, this carry-rippling effect should be totally eliminated. According to Equations (2.13) and (2.14), the following relation holds for the k -th bit position in an N -bit adder.

$$C_{o,k} = G_k + P_k C_{i,k} = G_k + P_k C_{o,k-1} \quad (2.15)$$

By recursively applying Equation (2.15), the following fully expanded form can be obtained:

$$C_{o,k} = G_k + P_k (G_{k-1} + P_{k-1} (\dots + P_1 (G_0 + P_0 C_{i,0}))) \quad (2.16)$$

The sum on the k -th bit position can then be expressed as follow:

$$\begin{aligned}
S_k &= P_k \oplus C_{o,k-1} \\
&= P_k \oplus (G_{k-1} + P_{k-1}(G_{k-2} + P_{k-2}(\dots + P_1(G_0 + P_0 C_{i,0}))))
\end{aligned} \tag{2.17}$$

From Equations (2.16) and (2.17), it can be seen that the carry-out bit and sum bit on any bit position can be derived with just the input bits, without involving any internal carry signals. Thus, theoretically speaking, all the sum bits can be generated simultaneously, and almost immediately after receiving the inputs. In this way, the carry propagation path is totally eliminated. The adder derived from Equations (2.16) and (2.17) is named Carry-Lookahead Adder (CLA) [39]. The block diagram of a 4-bit CLA is depicted in Figure 2.16. One of many possible implementations of a 4-bit CLA is shown in Figure 2.17 [32].

While the CLA is superior in speed performance, its costs in power consumption and circuit area are tremendous. When the size of the adder, N , increases, the power consumption and circuit area of the adder will increase dramatically. So, the carry-lookahead structure shown in Figure 2.16 is only suitable for small adders (usually, $N \leq 4$).

To construct large adders, several techniques have been proposed. The simplest way is to use the carry-lookahead technique to construct a number of 4-bit adders and then cascading these 4-bit adders in the ripple-carry way to form the large adder (illustrated in Figure 2.18). Because this design strategy contains two techniques, carry-lookahead technique and ripple-carry technique, it can also be called *hybrid adder* (Note that the term *hybrid adder* can be referred to any design scheme that makes use of two or more design techniques.). This hybrid adder combines the characteristics of both CLA and RCA, so it achieves a balance between high speed performance and low power consumption.

Figure 2.16 Block diagram of 4-bit Carry-Lookahead Adder

Figure 2.17 Implementation of 4-bit Carry-Lookahead Adder [32]

Figure 2.18 N-bit Carry-Lookahead Adder constructed in the ripple-carry way

(a)

(b)

Figure 2.19 16-bit Carry-Lookahead Adder: (a) implementation of 4-bit carry-lookahead structure; (b) architecture of the whole adder [40]

Another methodology to construct large adder with the carry-lookahead technique is

to recursively make use of the carry-lookahead structure [40]. This methodology divides an adder into several levels, each of which is implemented using carry-lookahead technique. Figure 2.19 shows a 16-bit adder using this methodology. The number of levels, M , of such an adder, can be computed using the equation, $M = \lceil \log_4 N \rceil$, where $\lceil X \rceil$ means the smallest integer that is larger than X . This pure CLA structure is the fastest adder structure because it eliminates the whole carry propagation path. However, its power consumption and circuit area are considerable.

2.3.6 Carry-Save Adder

All the adders described above are dealing with the two operands addition. The multiple operands N -bit adder can be constructed by cascading a number of N -bit two operands adders. But this could be a very slow process. To complete the multiple operands addition concurrently, a new architecture of adder, Carry-Save Adder (CSA) [41], has been developed (shown in Figure 2.20). In this architecture, the carry signals are no longer propagated in an adder stage but saved for the next adder stage instead. Only at last stage, a RCA is used to compute the final sum outputs. The CSA is the basis of the Braun Multiplier (also called the Carry-Save Array Multiplier).

2.3.7 Chinese Abacus Adder

Besides the above conventional adders, many other new design techniques have also been proposed. The interesting and promising Chinese Abacus Adder [42] is one of them.

The Chinese abacus is a very popular technique used for centuries in China. It has

Figure 2.20 Carry-Save Adder

been proved to be an efficient technique for arithmetic computation. A Chinese abacus consists of a set of unity elements representing the various decades of decimal numbers. Each element has five beads that are with unity weight and two beads that are with the weight of five. So the value range of the decimal number that can be represented using one abacus element is from 0 to 15. The number representation used in the Chinese abacus refers to the digital numeric system, but what an electronic engineer is mostly interested in is the binary-based coding system. So, for convenience, a modified Chinese abacus technique was proposed and used in the electronic adder design [42]. In the modified abacus technique, a basic element is made up of four unity-weight beads and two beads having a weight of four units. Thus, one basic element of the abacus is able to represent a number ranging from 0 to 12.

The circuit implementation of an adder based on the Chinese abacus approach consists of four basic blocks: the binary-to-thermometric (B/T) conversion block, the shift-up (SU) block, the thermometric-to-abacus (T/A) coding block, and the abacus-to-binary (A/B) conversion block. The circuit implementations of these four basic blocks are depicted in Figures 2.21 to 2.24. An 8-bit adder can be constructed using the four basic blocks. Its architecture is illustrated in Figure 2.25.

Figure 2.21 The binary-to-thermometric (B/T) conversion block

Figure 2.22 The shift-up (SU) block

Figure 2.23 The thermometric-to-abacus (T/A) coding block

Figure 2.24 The abacus-to-binary (A/B) conversion block

Figure 2.25 8-bit adder based on Chinese abacus technique

2.4 Power Consumption of Adder

The power consumption of a digital circuit determines how much energy is consumed per operation, and how much heat the circuit dissipates. These factors affect a large number of critical design decisions, such as the battery lifetime, supply line sizing, packaging and cooling requirements. In the world of high-performance computing, power consumption limits, dictated by the chip package and the heat removal system, determine the number of circuits that can be integrated onto a single chip, and how fast they are allowed to switch. Low power consumption is one of the most desirable characteristics that IC designers are always pursuing.

There are three major sources of power dissipation, namely: (1) dynamic dissipation due to charging and discharging capacitances; (2) dissipation due to short-circuit current; (3) static power dissipation due to leakage current [49].

2.4.1 Dynamic Power Consumption

Dynamic power is usually the largest source of power dissipation. It is consumed through charging and discharging the capacitances that exist in an integrated circuit, and can be computed by the following formula [43]:

$$P_{dynamic} = A \cdot C_L \cdot V_{DD}^2 \cdot f_{clk} \quad (2.18)$$

where A is the fraction of gates actively switching, C_L is the total capacitance, V_{DD} is the supply voltage, and f_{clk} is the switching frequency of gates. From Equation (2.18), it can be seen that the dynamic power can be reduced by reducing the number of gates that are involved in the switching activity (In this way, the term of $A \cdot C_L$, which is also called *effective capacitance*, can be reduced.), the supply voltage, and the switching frequency. In modern digital IC technology, as more and more

transistors are integrated onto a single chip and the clock frequency also keeps increasing, the commonly used method to reduce dynamic power consumption is to reduce the supply voltage. Although reducing V_{DD} has a quadratic effect on $P_{dynamic}$ so that is a very effective way, the usage of it is always limited by many constraints, such as technology restrictions and speed requirements.

For an adder and many other digital CMOS circuits, a large portion of dynamic power is actually consumed by the spurious switching activities that are usually caused by the signal delay. Using the proposed ETA that will be described in next chapter, the spurious switching can be greatly reduced, resulting in achieving low dynamic power consumption.

2.4.2 Short-Circuit Power Consumption

Because in actual designs, the input waveform for a circuit has the non-zero rise and fall times, a direct current path may exist between V_{DD} and GND for a short period of time during switching, when both the pull-up and pull-down networks are conducting simultaneously. The direct-path current leads to the short-circuit power dissipation. This source of power dissipation is often classified to dynamic power consumption because it is also closely related to the switching activity. An accurate evaluation of the short-circuit power, P_{SC} , for short-channel devices has been presented in [44] and [45], and can be simplified to the following formula:

$$P_{SC} = \left[\frac{k_N V_{DD}^3 f_{clk} \tau}{3(1 + \delta_n)} \right] \cdot (1 - p - n)^3 + 2f_{clk} C_L V_{DD}^2 \left[c - \frac{k_N V_{DD} \tau}{6C_L (1 + \delta_n)} (1 - p - n)^3 - 1 \right] \quad (2.19)$$

where

$$c = 1 + \frac{x_2 - 1 + p}{1 + \delta_p} + \frac{k_N V_{DD} \tau}{6C_L (1 + \delta_n)} (x_2 - n)^3 - \frac{k_P V_{DD} \tau}{6C_L (1 + \delta_p)} (x_2 - 1 + p)^3 \quad (2.20)$$

where k_N and k_P are NMOS and PMOS transconductances, τ is the input rise time, δ_n and δ_p are the Taylor series expansion coefficients of the bulk charge, n and p are equal to $\frac{V_{TN}}{V_{DD}}$ and $\frac{V_{TP}}{V_{DD}}$ respectively, and x_2 is the normalized time value when PMOS enters the saturation region.

2.4.3 Static Power Consumption

The static power dissipation is caused by the leakage currents and can be expressed by the relation [31]:

$$P_{static} = I_{leak} \cdot V_{DD} \quad (2.21)$$

where I_{leak} is the leakage current that flows between supply rails in the absence of switching activity.

There are two sources of leakage current. One is the gate-oxide leakage current and the other is the subthreshold current. So the leakage current can be expressed as:

$$I_{leak} = I_{ox} + I_{sub} \quad (2.22)$$

where I_{ox} is the gate-oxide leakage current and I_{sub} is the subthreshold current.

The gate-oxide leakage current is caused by the tunneling of electrons (or holes) from the bulk silicon through the gate-oxide potential barrier into the gate. The equation for I_{ox} has been presented in [46]:

$$I_{ox} = K_2 W \left(\frac{V_{DD}}{T_{ox}} \right)^2 e^{\frac{-\sigma T_{ox}}{V_{DD}}} \quad (2.23)$$

where K_2 and σ are experimental parameters, W is the width of the gate, and T_{ox} is the oxide thickness.

The subthreshold current can be computed using the equation also given in [46]:

$$I_{sub} = K_1 W e^{\frac{-V_T}{nV_\theta}} (1 - e^{\frac{-V_{DD}}{V_\theta}}) \quad (2.24)$$

where K_1 and n are experimental parameters and V_θ is the thermal voltage.

A simplified equation to calculate the static power, P_{static} , is given in [47] and can be presented as below:

$$P_{static} = N \cdot k_{design} \cdot k_{tech} \cdot 10^{\frac{-V_T}{\beta}} \cdot V_{DD} \quad (2.25)$$

where N is the total number of transistors, k_{design} is a design dependent parameter, and k_{tech} and β are technology dependent parameters.

Chapter 3 Error-Tolerant Adder

3.1 Introduction

The Error-Tolerant Adder (ETA) is defined as a digital adder that does not always yield correct results but is still usable in some systems by generating “acceptable” results. In an ETA, errors may occur at the output of the adder due to some internal or external factors. According to the definition given above, the ETA is a broad category of adders. There can be numerous ways to implement an ETA. In this chapter, two methodologies that serve to provide an investigation in this emerging research area are presented. In the proposed designs, the errors are caused by special addition mechanisms and circuit structures.

Prior to discussing on the ETA, the exact definitions and explanations of some commonly used terminologies in this thesis are given as follows:

- *Overall error (OE)*. It is defined as the difference between the correct result and the obtained result. It can be computed by using the following equation:

$OE = |R_c - R_e|$, where R_e is the result obtained by the adder, and R_c denotes the correct result (both results are represented as decimal numbers).

- *Accuracy (ACC) of adder*. In the scenario of error-tolerant design, the accuracy of an adder is used to indicate how “correct” the output of an adder

is. It is defined as $ACC = (1 - \frac{OE}{R_c}) \times 100\%$. Its value ranges from 0% to

100%. According to the mathematical expression, it can be seen that the accuracy of an adder is depending on the output result so that is not a constant. Actually, the accuracy of an adder can be regarded as a variable with respect to the output/input pattern and its value is equal to the accuracy of a specific obtained output. In this thesis, for convenience, the term

“accuracy” is sometimes used to denote both the accuracy of an adder and the accuracy of its output.

- *Minimum acceptable accuracy (MAA)*. Although some errors are allowed to exist in the output of an ETA, the accuracy of an acceptable output should be “high enough” (higher than a threshold value) to meet the requirement of the whole system. Minimum acceptable accuracy is just that threshold value. The obtained results whose accuracy is higher than the minimum acceptable accuracy are called acceptable results. The value of the minimum acceptable accuracy is often preset by the customers/designers according to specific applications.
- *Acceptance probability (AP)*. Since the accuracy of an adder is dependent on the output/input pattern and the outputs/inputs of a digital system are often regarded as random signals, the accuracy of an adder can also be taken as a random variable. Acceptance probability is the probability that the accuracy of an adder is higher than the minimum acceptable accuracy. It can be expressed as $AP = P(ACC > MAA)$ and its value ranges from 0 to 1. This parameter is usually used as an important metric indicating the accuracy performance of an ETA.

3.2 ETA Type I

According to the definition given at the beginning of this chapter, the ETA can be a broad category of adders. In this section, one of the many ways to implement an ETA from the perspective of addition algorithm is proposed. For convenience, this implementation of ETA is named ETA Type I, or simply ETAI.

3.2.1 Proposed Addition Algorithm

In a conventional adder circuit, the delay is mainly attributed to the carry propagation chain along the critical path, from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). Moreover, a significant proportion of the power consumption of an adder is due to the glitches that are also caused by the carry propagation. Therefore, if the carry propagation can be eliminated or curtailed, a great improvement in both the speed performance and power consumption can be achieved. In this section, for the first time, an innovative and novel addition algorithm that can attain great saving in speed and power consumption is proposed. This new addition algorithm can be illustrated via an example shown in Figure 3.1.

Figure 3.1 Addition algorithm for ETAI

First the input operands are split into two parts: an accurate part that includes a number of higher order bits and an inaccurate part that is made up of the remaining lower order bits. The lengths of each part need not necessarily be equal. The addition process starts from the middle (joining point of the two parts) towards the two opposite directions simultaneously. In the example, the two 16-bit input operands, A = “1011001110011010” (45978) and B = “0110100100010011” (26899), are divided

into two equal-sized parts, each of which contains 8 bits.

For the higher order bits of the input operands that fall into the accurate part, the operation is performed from right to left (LSB to MSB) and normal addition method is applied. This segment is named the *accurate part* because it follows the conventional accurate addition algorithm. For the example shown in Figure 3.1, the partial sum generated in the accurate part is “100011100”, which is perfectly correct.

For the lower order bits of the input operands that fall into the inaccurate part, a special addition mechanism is applied. In this part, no carry signal will be generated or taken in at any bit position such that the carry propagation path no longer exists. To minimize the overall error caused by eliminating the carries, a special strategy is adopted. Its operational process is described as follow: check every bit position from left to right (MSB to LSB); and on a bit position, if either of the two input operand bits is “0”, normal one-bit addition is performed to derive the sum bit on that position and the operation proceeds to next bit position; if both of the input bits are “1”, the checking process is stopped and from this bit onwards, all the sum bits are set to “1”. In this way, the overall error generated due to the elimination of carry bits can be reduced to minimal. In the example, at the fifth bit position, the two input bits, A_4 and B_4 , are both equal to “1”, so all the sum bits on its right are set to “1”. The partial sum generated in the inaccurate part is therefore “10011111”, which contains error.

The final result of the complete addition is therefore “10001110010011111” (72863). This is the result obtained using the proposed addition algorithm. On the other hand, the correct result of this addition, which can be derived using the normal addition algorithm, is “10001110010101101” (72877). So the overall error generated in this example is:

$$OE = 10001110010101101 (72877) - 10001110010011111 (72863) = 1110 (14).$$

The accuracy of the adder with respect to these two input operands is:

$$ACC = (1 - \frac{14}{72877}) \times 100\% = 99.98\% .$$

In this new addition method, the carry propagation only exists in the accurate part. The accurate part is constructed in the conventional way because the higher order bits of a result need to be made as accurate as possible, as they play a more important role (have higher weights) than the lower order bits do. This idea is similar with the BIVOS scheme in PCMOS technology that was mentioned in Section 2.1.4. By eliminating the carry propagation path in the inaccurate part and performing the addition in two separate parts simultaneously, the overall delay time is greatly reduced and so is the power consumption.

3.2.2 Relationships between AP, MAA, Dividing Strategy, and Size of Adder

As mentioned in Section 3.1, there is a minimum acceptable accuracy (MAA) associated with an ETA. If a result obtained by the adder has an accuracy that is higher than the MAA, this result is taken as the acceptable result. Upon further evaluation of the proposed addition algorithm, it can be seen that the accuracy of the ETAI is closely related to the input pattern. Assume that the inputs of an ETAI are random numbers, there exists a probability of obtaining an acceptable result (i.e., the AP). Dividing strategy, which is the main design strategy when designing an ETAI, is the strategy of deciding the sizes for both the accurate part and the inaccurate part. In this subsection, the relationships between the MAA, the AP, the dividing strategy and the size of adder are investigated.

First, the extreme situation where the users only accept the perfectly correct result is considered. The minimum acceptable accuracy in this “perfect” situation is 100%. According to the proposed addition algorithm, the correct results can be obtained only

when the two input bits on every position in the inaccurate part are not equal to “1” at the same time. The equation to calculate the AP associated with the proposed ETAI with different sizes and different dividing strategies can therefore be derived. This equation is given as follow:

$$P(ACC = 100\%) = \frac{4^{N_t - N_l} \times 3^{N_l} + 2^{N_t - N_l}}{4^{N_t} + 2^{N_l}} \quad (3.1)$$

where N_t is the total number of bits in the input operand (also regarded as the size of the adder) and N_l is the number of bits in the inaccurate part (which is indicating the dividing strategy).

Based on Equation (3.1), the probability of getting a correct result using ETAI with different sizes (assume the dividing point is always at the right middle of the whole adder, i.e., $N_l = \frac{N_t}{2}$) can be plotted in Figure 3.2. The figure illustrates that the chance of obtaining correct results is comparatively high for small adders. As the adder becomes larger, the probability of getting correct results decreases dramatically.

Figure 3.2 Probability of getting correct results with the proposed addition algorithm for ETAI

Next, situations where the requirement on accuracy is somewhat relaxed are investigated. A C program (similar with the program given in Appendix B but with different parameters) was engaged to simulate a 16-bit adder that had adopted the proposed addition algorithm. By checking the output results, the relationship between MAA and AP can be derived, as depicted in Figure 3.3. In this study, simulations of adders with different dividing strategies were performed. In Figure 3.3, the 4 curves represent 4 different dividing strategies, each of which has been assigned a name “N-M” where “N” denotes the size of the accurate part and “M” is for the size of the inaccurate part. For example, “6-10” means the size of the accurate part of the adder is 6-bit and that of the inaccurate part is 10-bit. For the input patterns, 10,000 inputs were randomly selected from all possible input patterns (i.e., 0--65535).

It can be deduced from Figure 3.3 that the lower the MAA set, the higher the AP for the adder. Figure 3.3 also illustrates that different dividing strategy leads to different accuracy performance. When the size of the accurate part is made larger, the AP of this adder will also increase.

Figure 3.3 Relationship between AP and MAA

As the modern VLSI technology advances, the size of adder has to increase to cater to the application need. So the trend of the accuracy performance of an ETA, when the size of the adder increases, needs to be investigated. Figure 3.4 shows such a trend. The 5 curves are associated with different MAA's, 95%, 96%, 97%, 98%, and 99%, respectively. Note that all adders follow the same dividing strategy that the size of the inaccurate part is three times larger than that of the accurate part. This figure presents a totally opposite trend of the acceptance probability when compared to Figure 3.2. It illustrates that if some degree of errors can be permitted, the chance of getting acceptable results will be very high and this chance is becoming higher when the size of the adder increases. It should be noted that those unacceptable results often occur when both of the input operands are small numbers. This is because small numbers will be calculated only in the inaccurate part of the adder. So the proposed ETAI is especially suitable for large input patterns.

Figure 3.4 Relationship between AP and size of adder

3.2.3 Hardware Implementation

The block diagram of the hardware implementation of ETAI is provided in Figure 3.5. This most straightforward structure consists of two parts: an accurate part and an inaccurate part. The accurate part, which contains $n-m$ bits, is constructed using a conventional adder such as the RCA, CSK, CSL or CLA. The carry-in of this adder is connected to ground. The accurate part is used to compute the higher order bits of the sum. The inaccurate part, whose size is m -bit, constitutes two blocks: a carry-free addition block and a control block. The carry-free addition block generates the sum bits on the lower order bit positions. The control block is used to generate the control signals to determine the working mode of the carry-free addition block. In the next subsection, the design of a 32-bit adder, taken as an example, is described to elaborate on the design process and detailed circuit implementation of an ETAI.

Figure 3.5 Block diagram of the hardware implementation of ETA I

3.2.4 Design of a 32-bit ETAI

I. Strategy of Dividing the Adder

The first step to design a proposed ETAI is to divide the adder into two parts in a specific manner. The dividing strategy depends on the requirements, in terms of accuracy, speed and power.

First of all, the accuracy performance of the adder should meet the requirements preset by the designer/customer. For example, for a specific application, one may require the minimum acceptable accuracy to be 98%, with an acceptance probability of 0.99. With such criteria, the proposed adder should be divided in such a way that 98% accuracy can be attained for at least 99% of all possible inputs.

Secondly, the delay of the proposed adder is defined as $T_d = \max(T_h, T_l)$, where T_h is the delay in the accurate part and T_l is the delay in the inaccurate part. With proper dividing strategy, a designer can make T_h approximately equal to T_l and hence achieve the optimal time delay.

Thirdly, due to the simplified circuit structure and the elimination of switching activities in the inaccurate part, putting more bits in this part yields more power saving.

Having considered the above, the proposed 32-bit ETAI is divided in such a way that 12 bits are assigned to the accurate part and 20 bits in the inaccurate part.

II. Design of the Accurate Part

As mentioned earlier, the accurate part can be constructed using any type of conventional adder. In our proposed design, the most common Ripple-Carry Adder is used. Because with the proposed design strategy, the overall delay time is determined

by the inaccurate part instead of the accurate part (this can be seen later in this section), the accurate part need not be a fast adder. In addition, the Ripple-Carry Adder is the most power-saving conventional adder.

III. Design of the Inaccurate Part

The inaccurate part is the most critical section in the proposed ETAI as it determines the characteristics of accuracy, speed performance and power consumption of the adder. As described in Section 3.2.3, the inaccurate part consists of two blocks: one is the carry-free addition block and the other is the control block.

The carry-free addition block is made up of twenty Sum Generating Cells (SGC), each of which is used to generate a sum bit. The block diagram of the carry-free addition block and the schematic implementation of the SGC are shown in Figure 3.6. In the circuit of SGC, three extra transistors, M1, M2, and M3, are added to a conventional XOR gate. “CTL” is the control signal coming from the control block and is used to determine the operation mode of the circuit. When $CTL = 0$, M1 and M2 are turned on, while M3 is turned off, leaving the circuit to operate in the normal half-addition mode. When $CTL = 1$, M1 and M2 are both turned off, while M3 is turned on, allowing the output node to be directly connected to VDD (this working mode is also named pull-up mode), setting the sum output to “1”.

The control block, depicted in Figure 3.7, consists of twenty Control Signal Generating Cells (CSGC). Each of these cells can generate a control signal for the SGC at the corresponding bit position in the carry-free addition block. The function of the control block is to detect the first bit position where two input bits are both “1”, and to set the control signal on this position as well as those on its right to high.

It can be seen that for the control signal on a specific position, if any of the control signals on its left is high, it should also be set to high. From this observation, the control block can be constructed as that shown in Figure 3.7. As can be seen in this

figure, all the CSGC's are cascaded by connecting the output of one cell to the input of the cell on its right. For the i -th CSGC, if its input control signal CTL_{i+1} is high, its output signal CTL_i is also set to high. In this way, if any of the control signals is set to high, this high signal will be propagated to all the bit positions on its right. But this cascading strategy renders a very long control signal propagation path in the control block. The worst case happens when $A_{19} = B_{19} = 1$ while $A_i \times B_i \neq 1$ where $i = 0, 1, 2, \dots, 18$. In this case, the high control signal will propagate from leftmost bit position all the way down to the rightmost bit position. The worst-case propagation path of this structure consists of twenty CSGC's.

Figure 3.6 Carry-free addition block: (a) overall architecture; (b) schematic diagram of an SGC.

Figure 3.7 Block diagram of the control block

To speed up the setup process of the control signals, the twenty cascaded CSGC's are divided into five equal-sized groups [see Figure 3.8 (a)] and extra connections are added between every two neighboring groups. Figure 3.8 (a) shows that the control signal generated by the leftmost cell of each group is fed into the input of the leftmost cell in next group. These extra connections allow the propagated high control signal to “jump” from one group to another instead of passing through all the twenty cells. In this way, the worst-case propagation path, which is shaded in gray in Figure 3.8 (a), consists of only ten cells.

In the proposed architecture, there are two different types of CSGC: the leftmost cells of each group [denoted as “II” in Figure 3.8 (a)] and the rest of the cells [denoted as “I” in Figure 3.8 (a)]. The schematic implementations of these two types of CSGC are provided in Figure 3.8 (b). When both of the input bits, A_i and B_i , are “1” or either of the incoming control signals CTL_i and CTL_{i+4} is high, the output of a CSGC will be set to high.

Figure 3.8 Control block: (a) overall architecture; (b) schematic implementations of CSGC.

3.2.5 Circuit Simulation

The transistor-level simulation of the proposed ETAI circuit is performed using HSpice. The simulation parameters are provided in Table 3.1.

Table 3.1 Simulation parameters

Process	Chartered Semiconductor Manufacturing Ltd's 0.18- μm CMOS process			
Minimum Transistor Size	NMOS (W/L)		PMOS (W/L)	
	0.3 μm /0.18 μm		0.6 μm /0.18 μm	
Input	Frequency	Number	Character	Range
	100 M	100 patterns	Random	$0 \sim 2^{32} - 1$

The simulation results of the proposed ETAI, including power, delay, power-delay product (PDP), and transistor count are shown in Table 3.2.

Table 3.2 Simulation results of the proposed ETAI

Power (W)	Delay (S)	PDP (J)	TC**
1.2888E-04	2.2896E-09*	2.9508E-13	1006

*: $T_h = 1.8889E-9$, $T_l = 2.2896E-9$, $T_d = \max(T_h, T_l) = 2.2896E-9$

** : Transistor Count

To demonstrate the functionality of the proposed ETAI, another five input patterns are randomly selected and simulated. These five input patterns and the corresponding results and accuracies are listed in Table 3.3 (note that all the numbers are written in the hex format for the purpose of convenience). The waveforms of five out of total thirty-two outputs ($S_5, S_{10}, S_{15}, S_{20}, S_{25}$) are shown in Figure 3.9.

Table 3.3 Five input patterns and their corresponding results and accuracies

A	B	Correct Result	Obtained Result	Accuracy
006868B4	00D611D1	013E7A85	013E79FF	99.99%
00000053	00000016	00000069	0000005F	90.48%
003C95CC	00525389	008EE955	008EDFFF	99.97%
00000B2E	00001669	00002197	00001FFF	95.26%
029A3674	0138D8AE	03D30F22	03CFFFFFFF	99.69%

(a) Output waveform at node S5

Figure 3.9 Waveforms at five selected output nodes

(b) Output waveform at node S10

(c) Output waveform at node S15

(d) Output waveform at node S20

Figure 3.9 (continued) Waveforms at five selected output nodes

(e) Output waveform at node S25

Figure 3.9 (continued) Waveforms at five selected output nodes

According to analysis, the bit sequences at the five selected output nodes should be “10111”, “00111”, “00101”, “10000”, and “00001”. These analytical results are exactly the same with the simulation waveforms shown in Figure 3.9.

3.2.6 Optimization of the Proposed 32-bit ETAI

The structure of the inaccurate part proposed in previous subsection can actually be further simplified.

First, the XOR gate in the carry-free addition block is actually not necessary. Because according to the proposed addition algorithm, when a SGC is working in the half-addition mode, its output bit (sum) will be equal to “0” only when both of the input bits are “0”. So the XOR logic in the SGC can be replaced with the OR logic, which is much simpler.

Secondly, the stack transistor M1 in Figure 3.6 (b) is redundant. The stack transistor M2 is able to cut off the discharge path from output node to ground, making the SGC working in the pull-up mode. Actually, the elimination of transistor M1 can make the

output capacitor being charged faster, because there will be an additional pull-up path through the PMOS transistors in OR logic gate. At the same time, when the transistor M1 is eliminated, the control signal engaged in the SGC becomes single-phased, so that the inverter, which is used to generate the complementary control signal, may also be eliminated. Figure 3.10 shows the simplified structure of a SGC.

Figure 3.10 Simplified structure of SGC

Thirdly, when this simplified structure of SGC is employed, the OR gate will be deactivated by low control signal, which is different with the previous design. So, in the optimized design, the CSGC's in the control block also need to be modified. The schematic diagrams of the CSGC's in the modified control block are depicted in Figure 3.11. Instead of AND gates and OR gates in original design, NAND gates and AND gates are used to implement the CSGC's in the modified control block. The control signal propagated in the control block becomes "0" instead of "1". Using this modified design, one inverter can be saved for each CSGC. This transistor reduction also leads to power saving as well performance improvement.

The circuit simulation is also performed using Hspice (the circuit netlist is provided in Appendix A) and under the conditions listed in Table 3.1. The simulation results provided in Table 3.4 demonstrates the significant improvements of the optimized structure of ETAI in power consumption, speed performance, and transistor count

when compared with its original counterpart.

Figure 3.11 Schematic diagrams of CSGC's in the modified control block

Table 3.4 Simulation results of original and optimized ETAI

Type of Adder	Power (W)	Delay (S)	PDP (J)	TC
Original ETAI	1.2888E-04	2.2896E-09	2.9508E-13	1006
Optimized ETAI	9.6718E-05	1.8882E-09	1.8262E-13	706

3.2.7 Comparison with Conventional Adders

To demonstrate the advantages of ETAI, four different conventional adders, RCA, CSK, CSL, and CLA, were also constructed and simulated. Note that for RCA, each adder cell is implemented using the 28T full adder structure shown in Figure 2.10 (a); for CSL, the LCSL structure, which is depicted in Figure 2.14, is adopted; and for CLA, it is constructed using the hybrid structure that is given in Figure 2.18.

The simulation tool, circuit technology, and input patterns are all the same with those introduced in last subsection. The simulation results along with the data shown in Table 3.4 are all tabulated in Table 3.5.

From the table, it is evident that the proposed ETAI outperforms the conventional

adders in all metrics, including power consumption, speed performance, and transistor count. The PDP of ETAI can be at least 79.51% better than the conventional adders. The tremendous improvements in these aspects make the proposed ETAI an excellent candidate for the ultra low-power and high-speed digital systems.

Table 3.5 Comparison between the proposed ETAI and conventional adders

Type	Power (W)	Delay (S)	PDP (J)	PDP Saving	TC
RCA	2.2076E-04	4.0365E-09	8.9110E-13	79.51%	896
CSK	4.7260E-04	3.0191E-09	14.2683E-13	87.20%	1760
CSL	6.0915E-04	3.0666E-09	18.6802E-13	90.22%	2176
CLA	5.1348E-04	2.3741E-09	12.1905E-13	85.02%	2208
ETAI	9.6718E-05	1.8882E-09	1.8262E-13	--	706

3.2.8 Further Study of the Relationship between Accuracy Performance and Input Patterns

It should be noted that the data that are collected and presented in Section 3.2.2 are based on a precondition that the input operands are randomly distributed over all the possible numbers and this distribution follows the uniform distribution, which means every input pattern occurs with the same probability. For example, for a 32-bit adder, its input operand can be any number from 0 to $2^{32} - 1$, and each number has the same possibility to occur. This precondition is actually an ideal assumption that seldom happens in real world. For some applications, most of the data being processed by the systems are small numbers while for other applications, large numbers play the major role. So, it is necessary for us to study the accuracy performance of the proposed ETAI when the input patterns fall in different number ranges.

Performing the similar simulations introduced in Section 3.2.2 (the C program for

doing these simulations are presented in Appendix B) to the proposed 32-bit ETAI, the values of AP associated with different MAA's and input ranges can be obtained. All the simulation results are tabulated in Table 3.6. The columns represent the AP values associated with the input operands in four different ranges, and the rows show the AP values based on different MAA's from 90% to 100%.

Table 3.6 AP of ETAI with different MAA's and input ranges

Input Range	$0 \sim 2^8 - 1$	$0 \sim 2^{16} - 1$	$0 \sim 2^{24} - 1$	$0 \sim 2^{32} - 1$
MAA=100%	0.0988	0.0096	0.0032	0.0025
MAA=99%	0.1364	0.1491	0.5650	1.0
MAA=98%	0.1897	0.2055	0.7324	1.0
MAA=97%	0.2239	0.2472	0.8375	1.0
MAA=96%	0.2656	0.2808	0.9014	1.0
MAA=95%	0.3002	0.3124	0.9385	1.0
MAA=94%	0.3344	0.3409	0.9570	1.0
MAA=93%	0.3568	0.3614	0.9679	1.0
MAA=92%	0.3858	0.3899	0.9754	1.0
MAA=91%	0.4040	0.4140	0.9812	1.0
MAA=90%	0.4266	0.4318	0.9856	1.0

From the table, it is clear that, for small input operands, the accuracy performance of the proposed ETAI is very poor. As the inputs become larger, the AP values also become larger. When the input operands are large enough, the AP values of the proposed ETAI are almost equal to "1". The simulation results prove that the proposed ETAI is only suitable for large input operands.

To improve on the imbalanced accuracy characteristic of ETAI for different ranges of input operands, another different type of ETA is proposed and described in next section.

3.3 ETA Type II

In this section, another implementation of ETA is introduced. Different from ETAI, this new implementation of ETA, which is named as “ETA Type II” (or ETAI for short), does not eliminate the entire or part of the carry propagation path. Instead, it splits the entire carry propagation path into a number of short paths and completes the carry propagations in these short paths concurrently. In this way, the speed performance of the adder can be significantly improved and almost without degrading its power consumption. However, just like ETAI, the accuracy of the adder may be affected due to the performance improvement.

3.3.1 Theoretical Analysis

Consider the binary number addition process, the carry signal at each bit position is determined by all the previous input bits. For the worst case, this carry signal is generated at the LSB and propagates along the carry chain to the current bit position. If this worst case happens, a lot of time and power will be consumed along the carry propagation path. But in fact, the worst case seldom happens. For most of the cases, this carry signal can be determined by just several input bits on the right of the current bit position. Assume that the input operands are perfectly random, the probabilities of getting correct carry signal on the i -th bit position when different number of bit positions are involved in determining this carry signal can be derived. Some of the results are tabulated in Table 3.7. From this table, it can be seen that if only the neighboring one bit position is considered, for 75% of the cases, one can get the correct carry signal and thus arrive at the correct sum result; if the number of bit positions taken into consideration increases from 1 bit to 4 bits, the chance of obtaining a wrong carry signal is only about 3% of all the possible cases; if the number continues to increase to 8 bits or even 12 bits, the probability of getting incorrect carry signal is reduced to about 0.2% and 0.01%, respectively.

Table 3.7 Probability of getting correct carry signal when different number of bit positions are taken into consideration

Number of bit positions considered	1 bit	2 bits	4 bits	8 bits	12 bits
Probability of getting correct carry signal	0.75	0.875	0.96875	0.99805	0.99988

Of course, everyone hopes to get a perfectly correct result in any case. The problem is how much (in terms of power and time) one has to sacrifice to remove the incorrect results that may occur only with very small probabilities and whether it is worthy to do so. Just as discussed in previous chapters, if some errors are allowed to exist, great achievements in power- and time-saving can be obtained.

3.3.2 Architecture of ETAIL

The architecture of ETAIL is depicted in Figure 3.12. For an N -bit adder, it is divided into M ($M \geq 2$) blocks, each of which contains N/M bits. In each block, there are two separate circuitries. One is to generate the carry-out signal based **ONLY** on the input bits in this block (i.e., it does not take in the carry signal from previous block), and the other takes the carry-in signal from previous block and generates all the sum bits in this block. In this manner, the carry propagation only exists between two neighboring blocks instead of lying along the entire adder structure. In other words, the longest carry propagation path of ETAIL is $2N/M$ -bit long. The worst-case delay path is shaded in gray in Figure 3.12. So, the worst-case delay of the ETAIL is only $2/M$ times of the conventional adder. Again, as the carry propagation path has been curtailed, the dynamic power consumption of the adder caused by the spurious switching activities is also reduced. However, this power reduction can be offset by the additional carry generators.

Figure 3.12 Architecture of ETAlI

3.3.3 Dividing Strategy

Similar with the design of ETAl, the dividing strategy may affect the overall performance of the adder. In design of ETAlI, the dividing strategy refers to the choice of the number of blocks the adder is divided into. If fewer blocks are engaged and thus more bits are contained in one block, the probability of getting correct results becomes higher while the delay path also becomes longer; on the contrary, if the adder is divided “finer” (i.e., more blocks are engaged), the speed performance can be increased while the possibility that incorrect results occur becomes higher. Take a 32-bit ETAlI for instance, the simulation results of the AP under different dividing strategies and MAA’s are provided in Table 3.8. These simulations are completed by a C program (see Appendix D). 10000 random input patterns in the range of $0 \sim 2^{32} - 1$ are tested.

Table 3.8 AP of ETAII under different dividing strategies and MAA's

	M = 1 bit	M = 2 bits	M = 4 bits	M = 8 bits	M = 16 bits
MAA=100%	0.0118	0.2204	0.8306	0.9961	1.0
MAA=99%	0.5238	0.7927	0.9725	0.9985	1.0
MAA=98%	0.5419	0.8075	0.9795	0.9985	1.0
MAA=97%	0.5534	0.8119	0.9815	0.9995	1.0
MAA=96%	0.5706	0.8236	0.9819	0.9995	1.0
MAA=95%	0.5864	0.8352	0.9833	1.0	1.0
MAA=94%	0.5894	0.8367	0.9833	1.0	1.0
MAA=93%	0.5973	0.8386	0.9838	1.0	1.0
MAA=92%	0.6062	0.8401	0.9838	1.0	1.0
MAA=91%	0.6155	0.8425	0.9843	1.0	1.0
MAA=90%	0.6213	0.8468	0.9843	1.0	1.0

3.3.4 Implementation of a 32-bit ETAII

The block diagram of the 32-bit ETAII implementation is depicted in Figure 3.13. Note that when compared with Figure 3.12, the carry generator in the highest (leftmost) block has been eliminated, because this circuit is actually a dummy in this structure. In the proposed design, the whole adder is divided into 8 blocks, each of which has 4 bits. The carry generator is implemented employing the carry-lookahead methodology. It adopts the circuit proposed in Figure 2.17, Section 2.3.5. The sum generators are realized as the most simple ripple-carry adders for the purpose of power saving. Each adder cell adopts the 28T full adder structure depicted in Figure 2.10 (a), Section 2.3.1.

The circuit simulation is performed using Hspice (the circuit netlist is provided in Appendix C), and the results are tabulated in Table 3.9. All the simulation parameters are the same with those described in Table 3.1.

Figure 3.13 Implementation of a 32-bit ETAII

Table 3.9 Simulation results of ETAII

Type	Power (W)	Delay (S)	PDP (J)	TC
ETAII	2.4303E-04	9.4981E-10	2.3083E-13	1372

3.3.5 Relationship between Accuracy Performance and the Range of Input Patterns

As concluded in Section 3.2.8, when the inputs are small numbers, the accuracy performance of the proposed ETAI is very poor. To test whether this situation also exists in ETAII, the same simulations are performed to ETAII and the results are provided in Table 3.10.

From Table 3.10, it is obvious that the accuracy performance of the adder for small input operands has been significantly improved using the new method, although the accuracy performance for large input operands is degraded (compare the last column of Table 3.6 and Table 3.10). It can be seen that there is no big difference of the AP values between different input ranges using the new method. In other words, the unacceptable results are distributed over the whole input range from 0 to $2^{32}-1$ instead of “squeezing” in the range of small numbers.

Table 3.10 AP of ETAII with different MAA's and input ranges

Input Range	$0 \sim 2^8 - 1$	$0 \sim 2^{16} - 1$	$0 \sim 2^{24} - 1$	$0 \sim 2^{32} - 1$
MAA=100%	0.9691	0.9090	0.8590	0.8389
MAA=99%	0.9691	0.9408	0.9393	0.9699
MAA=98%	0.9691	0.9414	0.9417	0.9780
MAA=97%	0.9691	0.9414	0.9422	0.9824
MAA=96%	0.9691	0.9439	0.9453	0.9829
MAA=95%	0.9691	0.9493	0.9517	0.9829
MAA=94%	0.9691	0.9545	0.9557	0.9829
MAA=93%	0.9691	0.9582	0.9598	0.9829
MAA=92%	0.9691	0.9612	0.9624	0.9834
MAA=91%	0.9691	0.9617	0.9636	0.9834
MAA=90%	0.9691	0.9638	0.9663	0.9839

3.3.6 Modified ETAII

The structure of ETAII introduced in Section 3.3.4 is designed in an unbiased manner, which has a good performance in both power and speed. However, the degraded accuracy performance of this structure for large input operands that has been indicated in Section 3.3.5 may restrict its usage in some applications. By adopting the idea of BIVOS that was introduced in Chapter 2, a biased structure can be implemented to further improve the accuracy performance of ETAII. According to the idea of BIVOS, the high order bits should be more accurate than the low order bits as they play a more important role in representing a number. Therefore, for the several higher order bit positions in an adder, more input bits should be taken into consideration when calculating the carry signals. Figure 3.14 depicts the modified structure of the 32-bit ETAII (for convenience, the modified ETAII is named ETAIIM). In this structure, the first three carry generators are cascaded together to generate the carry signals for the

highest 2 blocks. In this way, the carry signal for the highest block is generated by the following 12 bits and the one for the second block is generated by 8 bits. The rest of the circuit is still the same with that shown in Figure 3.13. The accuracy performance of this modified structure is given in Table 3.11. Comparison between these simulation results and those highlighted in Table 3.8 shows that the accuracy performance of the adder has been improved. The circuit implementation of ETAIIM has been simulated (see Appendix E) and the results are given in Table 3.12. Compared with ETAIL, the delay time has increased significantly. This degradation of speed performance can be regarded as the trade-off for improvement of accuracy.

Figure 3.14 Structure of 32-bit ETAIIM

Table 3.11 Accuracy performance of the ETAIIM

MAA	AP	MAA	AP
100%	0.8854	95%	1.0
99%	0.9990	94%	1.0
98%	0.9995	93%	1.0
97%	1.0	92%	1.0
96%	1.0	91%	1.0

Table 3.12 Simulation results of ETAIIM

Type	Power (W)	Delay (S)	PDP (J)	TC
ETAIIM	2.4310E-04	1.4654E-09	3.5624E-13	1372

3.3.7 Comparison with Conventional Adders

The comparison of circuit performance (i.e., the power consumption, delay time, PDP, and transistor count) between the proposed ETAIIM and conventional adders are shown via Table 3.13. From the table, it is illustrated that the proposed ETAIIM has the best speed performance when compared with the conventional adders and meanwhile still maintains a low power consumption (almost as low as the RCA). Its PDP is 60.02%, 75.03%, 80.93%, and 70.78% better than that of the RCA, CSK, CSL, and CLA, respectively. Although the transistor count of the ETAIIM is larger than the RCA, it is better than other conventional adders.

Table 3.13 Comparison between ETAIIM and conventional adders

Type	Power (W)	Delay (S)	PDP (J)	PDP Saving	TC
RCA	2.2076E-04	4.0365E-09	8.9110E-13	60.02%	896
CSK	4.7260E-04	3.0191E-09	14.2683E-13	75.03%	1760
CSL	6.0915E-04	3.0666E-09	18.6802E-13	80.93%	2176
CLA	5.1348E-04	2.3741E-09	12.1905E-13	70.78%	2208
ETAIIM	2.4310E-04	1.4654E-09	3.5624E-13	--	1372

3.4 Comparison between ETAI and ETAII

Two different kinds of ETA have been introduced in Sections 3.2 and 3.3. Each of these two kinds of ETA has advantages and disadvantages. The major differences between them are listed as below:

1. ETAI has introduced a novel mechanism to minimize the overall error generated by the special addition algorithm. So, in the worst case (in terms of accuracy), the accuracy of the adder is 50%. On the contrary, because ETAII does not have such

mechanism, its worst case accuracy is much worse.

2. ETAI has simplified the circuit structure of the adder by eliminating the carry signal generating circuits in the lower part, so its transistor count and power consumption are smaller than ETAI (as can be seen by comparing Table 3.4, Table 3.9, and Table 3.12).
3. ETAI has a very small probability of getting correct result, while ETAI shows a much larger probability of correctness (as can be seen by comparing the second row of Tables 3.6 and 3.10).
4. ETAI is strongly input-dependent. The accuracy performance of this kind of ETA is strongly dependent on the range of the input patterns. In contrast, ETAI is much better in this aspect.

Chapter 4 Application of ETA in Digital Signal Processing

4.1 Applications of ETA

After introducing the algorithms, mathematical properties, and schematic implementations, the question naturally comes to the application-level. The proposed ETA's have shown their outstanding performance in both power and speed, but what most people are more concerned about is the practicality of these ETA's. If nowhere can these adders be made use of, the proposed idea will become totally useless, no matter how good it looks.

Of course, not all digital systems can employ the error-tolerant circuit. In some digital systems, such as a control system, the correctness of the output signals is extremely important and hence eliminating the use of error-tolerant circuit. However, for many digital signal processing (DSP) systems that process the signals relating to human senses such as sound, images, smells and touch, for example, the image processing and speech processing systems, the error-tolerant circuits may be applicable. That is because human behaviors/feelings are always error-tolerant. One possible application of the error-tolerant DSP system is the medical electronic devices that need to be implanted into human bodies to help people who suffer from the loss of vision or hearing. In this case, replacing batteries may require invasive surgery and thus, it is of paramount importance to prolong battery life. The ultra-low power character of the proposed designs offers a great hope to realize this. Another application is the portable electronic devices, such as cell phone and laptop, where both low-power and high-speed are desired by users.

In the following sections, the application of ETA in DSP systems is discussed. The

proposed ETA's will be used to implement the Fast Fourier Transform (FFT) function and the ETA-based FFT function will be employed in digital image processing to demonstrate its functionality.

4.2 Fast Fourier Transform and Digital Signal Processing

4.2.1 Discrete Fourier Transform (DFT)

The discrete Fourier transform (DFT) is one of the specific forms of Fourier transform, which transforms a signal in the time/space domain to its frequency domain representation. It deals with the discrete-time signals. After the transform, the signals can be more conveniently processed according to specific need. DFT is commonly used in a wide range of digital signal processing (DSP) applications, such as image processing and speech recognition.

The DFT can be defined by the following formula [50]:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N-1 \quad (4.1)$$

where x_n is a sequence of input signal in the time domain; N is the number of discrete data in this input sequence; n is the index for each input point; X_k is the sequence of output signal, which also contains N points, in the frequency domain; k is the index for each output point; and $e^{\frac{2\pi i}{N}}$ is the primitive N 'th root of unity.

Meanwhile, the signal in frequency domain often needs to be converted back to the time/space domain representation because the signal in time/space domain is more direct and meaningful to our human beings. So the corresponding inverse DFT (IDFT) has also been studied and is defined by the following formula [50]:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \quad n = 0, \dots, N-1 \quad (4.2)$$

By observing Equation (4.1), it is noted that computing an N -point DFT would take about N^2 arithmetical operations. So its computation complexity is $O(N^2)$. This large amount of computations would seriously affect the working efficiency of a DSP system. So, to achieve high-speed computation, many fast Fourier transform (FFT) algorithms have been proposed [51]. They will be introduced in next subsection.

4.2.2 Fast Fourier Transform (FFT)

There are many different FFT algorithms, including the Cooley-Tukey algorithm [52], the Prime-factor algorithm [53, 54], Bruun's algorithm [55], Rader's algorithm [56], and Bluestein's algorithm [57]. The computation complexity of these FFT algorithms is only $O(N \log N)$.

Among these algorithms, the Cooley-Tukey algorithm is the most popular one. It is a divide and conquer algorithm that recursively breaks down a DFT into two pieces of size $N/2$ at each step. Therefore, using the Cooley-Tukey algorithm, an N -point DFT only requires $N \log_2 N$ arithmetical operations.

Since the FFT algorithms are so commonly employed in computing DFT, these two terms are often used interchangeably in colloquial settings. In the following sections, for the purpose of convenience, when no confusion would be caused, the term FFT is sometimes used for both the FFT algorithm and the DFT. However, people should understand that there is a clear distinction between these two concepts. DFT refers to a mathematical transformation, regardless of how it is computed, while FFT algorithm

is any one of the several efficient algorithms for the DFT.

4.2.3 Software Implementation of FFT

In Matlab, there is a set of functions that adopts the FFT algorithms to perform the DFT. The commands are *fft*(), *fft2*(), and *fftn*(). They are for conducting the one-dimensional, two-dimensional, and multi-dimensional DFT, respectively. Correspondingly, there is a set of inverse FFT commands, namely *ifft*(), *ifft2*(), and *ifftn*(). Using these commands, one can conveniently and quickly accomplish the DFT and inverse DFT in DSP applications. This thesis mainly focuses on the two-dimensional FFT because it is commonly used in digital image processing, which will be taken as an example of DSP applications to show the practicality of the proposed ETA's in later sections.

Although there are standard FFT functions in Matlab, it is not convenient to access their internal structures and make modifications to them. To demonstrate the practicality of the proposed ETA's, a self-defined FFT function has been written and is used as a platform to embed the proposed designs. The Matlab codes for the self-defined two-dimensional FFT function are provided and shown in Figures 4.1 and 4.2.

```
[m n] = size(A); % A is the original two-dimensional array
num = n;
for j=1:n
    B(:,j) = FFT2SELF(double(A(:,j)), num, num);
end
num = m;
for k=1:n
    C(k,:) = FFT2SELF(B(k,:), num, num);
end
disp(C); % C is the two-dimensional array after FFT
```

Figure 4.1 Matlab code for the main function of two-dimensional FFT

```

function [X] = FFT2SELF(x,k,N)
half = N/2;
G = zeros(k, 1);
H = zeros(k, 1);
W = zeros(k, 1);
if(N == 4)
    for m=1:k
        first = 0;
        second = 0;
        for n=0:half-1
            c = exp(-i.*(2.*pi./half).*(n.*(m-1)));
            mul1 = x((2.*n)+1).* c;
            first = first+mul1;
            mul2 = x((2.*n)+2).* c;
            second = second+mul2;
        end
        G(m,1) = first;
        H(m,1) = second;
        W(m,1) = exp(-i.*(2.*pi./N).*(1.*(m-1)));
    end
    X = G + W.*H;
    return
else
    for n=0:half-1
        x1(n+1) = x((2.*n)+1);
        x2(n+1) = x((2.*n)+2);
    end
    G = FFT2SELF(x1,k,half);
    H = FFT2SELF(x2,k,half);
    for m=1:k
        W(m,1) = exp(-i.*(2.*pi./N).*(1.*(m-1)));
    end
    X = G + W.*H;
    return;
end

```

Figure 4.2 Matlab code for the N-point FFT function

The function **FFT2SELF(x,k,N)** is a recursive function that adopts the Cooley-Tukey algorithm to calculate an N-point FFT. Its three input parameters, x, k, and N, are the input data, the number of points being used in the current recursion, and the total number of points in the FFT, respectively.

4.2.4 Application of FFT in DSP

Digital signal processing (DSP) and analog signal processing are two subfields of signal processing. DSP is the study of signals in a digital representation and the processing methods of these signals. It includes subfields like: audio and speech signal processing, sonar and radar signal processing, sensor array processing, spectral estimation, statistical signal processing, digital image processing, signal processing for communications, biomedical signal processing, seismic data processing, etc [58]. As can be seen from above, the applications of DSP have penetrated into many different areas. Because of its importance in today's electronic technology, how to enhance the performance of DSP systems (i.e., has higher speed and consumes fewer resources) has become a very important issue.

FFT is the basis and one of the most important functions of DSP. In DSP applications, there are usually five processing steps: firstly, the analog signals sampled from outside world are converted to digital signals; secondly, the digital signals are transformed from time/space domain (For the signals like speech signal, it is time domain; for the signals like image signal, it is space domain.) to the frequency domain using FFT function; thirdly, spectral analysis and many kinds of signal processing, such as filtration and signal convolution, are conducted to the signals in the frequency domain; fourthly, the processed signals are transformed back from frequency domain to time/space domain using inverse FFT function; lastly, the digital signals are converted back to analog signals. Because most kinds of signal analysis and processing are conducted in the frequency domain, the FFT is indispensable for a DSP system. As indicated in previous sections, the FFT involves a lot of additions and multiplications. If the performance of the adders used in the system can be improved, the overall performance of the DSP system can also be improved.

4.2.5 Fixed-Point Number and Floating-Point Number

In all electronic computational systems, there are two kinds of representations of the real numbers. One is the fixed-point number representation and the other is the floating-point number representation [59]. Each of these two representations has its own advantages and disadvantages.

Fixed-point number is a real data type for a number that has a fixed number of digits after (and sometimes also before) the radix point (e.g., the decimal point “.” in the decimal notations). Based on this definition, a fixed-point number can be written as $I.F$, where I represents the integer part of a number, F represents the fractional part, and “.” is the radix point and has a fixed position in this representation. For example, a 32-bit fixed point binary number may be represented in the form shown in Figure 4.3 (note that the number of bits in the integer part and the fractional part may be different in different systems):

Figure 4.3 Format of fixed-point number representation

For fixed-point binary numbers, each of the bits in the integer part represents a power of two, while each of the bits in the fractional part represents an inverse power of two. For example, the binary number $a_2a_1a_0.a_{-1}a_{-2}$ is equal to the decimal number of $a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0 + a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2}$.

Because of the simple structure of fixed-point number representation, specialized DSP hardware typically uses it for lower cost and complexity and greater speed. However, it suffers from the problems of comparatively small value range and low accuracy. So, when a very large range of numbers needs to be represented or very high accuracy is required, the other number representation, *floating-point number*, is preferred.

The term *floating-point* refers to the fact that the radix point in a real number can “float” instead of being fixed. Thus, the floating-point number representation has much wider range of values.

In the floating-point number representation, any real binary number N can be represented as:

$$N = 2^{E-B} \times M \quad (4.3)$$

where E is the exponent, B is the bias, and M is the mantissa. E is the stored exponent and can only represent the positive number. However, the actual exponent may be negative. To solve this problem, a bias B , which is 127 for IEEE single-precision floats and 1023 for double precision, is added to the stored exponent in order to derive the actual exponent. Thus, the actual exponent is equal to $E-B$. The mantissa is a binary number that contains a hidden radix point just before the MSB. For example, if the mantissa is “0110101”, it actually represents the number “0.0110101”. The mantissa is multiplied by the base raised to the power of the exponent, equivalent to shifting the radix point from its implied position by a number of places equal to the value of the exponent—to the right if the exponent is positive or to the left if the exponent is negative.

The format of the floating-point number representation is illustrated in Figure 4.4.

S	E	M
1 bit	m bits	n bits

Figure 4.4 Format of floating-point number representation

In this format, S , which is the MSB of the whole binary sequence, is the sign bit; m and n are the numbers of bits in the exponent and the mantissa, respectively. According to IEEE Standard 754 [60], for a 32-bit single precision floating-point number, m equals to 8 and n equals to 23; and for a 64-bit double precision floating-point number, m equals to 11 and n equals to 52.

The major disadvantage of floating-point number representation is its complexity, which will seriously degrade the power consumption and speed performance of a system. Therefore, most low-cost embedded microprocessors and microcontrollers do not have the floating point unit (FPU).

In this thesis, the fixed-point number representation is the main concern. The Matlab programs that have been written to simulate the ETA algorithms are all based on the fixed-point number representation. For convenience, in the rest of the thesis, it is assumed that all the data to be processed are stored as 32-bit fixed-point numbers with 16 bits of integer part and 16 bits of fractional part.

4.3 ETA-Based FFT Function

It has been shown that there are a lot of additions involved in the FFT function. Replacing all the additions with the proposed ETA algorithms, the normal FFT function will become the ETA-based FFT functions. Employing these functions in DSP applications and comparing the results with those derived from the standard functions, the credibility and plausible application of the proposed designs can be evaluated.

The modified (ETA-based) two-dimensional FFT function is realized by replacing the normal N-point FFT function (**FFT2SELF(x,k,N)**) with the new ETA-based N-point FFT function (**FFT2ETA(x,k,N)**), whose implementation is illustrated in Figure 4.5. The lines where modifications are made have been bolded and circled with rectangles.

```

function [X] = FFT2ETA(x,k,N)
.....
if(N == 4)
    for m=1:k
        .....
        for n=0:half-1
            c = exp(-i.*(2.*pi./half).*(n.*(m-1)));
            mul1 = x((2.*n)+1).* c;
            first = sum_complex2(first,mul1);
            mul2 = x((2.*n)+2).* c;
            second = sum_complex2(second,mul2);
        end
        .....
    end

    T = W.*H;
    [a b] = size(T);
    for i=1:a
        for j=1:b
            X(i,j) = sum_complex2(G(i,j),T(i,j));
        end
    end
    return
else
    for n=0:half-1
        .....
    end
    G = FFT2ETA(x1,k,half);
    H = FFT2ETA(x2,k,half);
    for m=1:k
        W(m,1) = exp(-i.*(2.*pi./N).*(1.*(m-1)));
    end

    T = W.*H;
    [a b] = size(T);
    for i=1:a
        for j=1:b
            X(i,j) = sum_complex2(G(i,j),T(i,j));
        end
    end
    return;
end
end

```

Figure 4.5 Matlab code for the modified N-point FFT function

In this function, there is a new sub-function, **sum_complex2()**, that has replaced the common addition operation. Because the data being processed in the FFT function are always complex numbers, the **sum_complex2()** is a complex number addition function. Its implementation is given in Figure 4.6.

```
function S=sum_complex2(a,b)
FACTOR=16; %number of bits in the fractional part of an FP number
flag_r=0;
flag_i=0;
ra=real(a); %separate the real part and image part of the
ia=imag(a); %two input operands, a and b.
rb=real(b);
ib=imag(b);
ra=floor(ra*(2^FACTOR)); %convert the data into 32-bit
ia=floor(ia*(2^FACTOR)); %fixed-point numbers with 16 bits in
rb=floor(rb*(2^FACTOR)); %the fractional part
ib=floor(ib*(2^FACTOR));
if(ra<0)
    ra=2^32+ra;
    flag_r=flag_r+1;
end
if(ia<0)
    ia=2^32+ia;
    flag_i=flag_i+1;
end
if(rb<0)
    rb=2^32+rb;
    flag_r=flag_r+1;
end
if(ib<0)
    ib=2^32+ib;
    flag_i=flag_i+1;
end
s_re=addition(ra,rb);
s_im=addition(ia,ib);
s_re=s_re-2^32*flag_r;
s_im=s_im-2^32*flag_i;
s_re=s_re/(2^FACTOR);
s_im=s_im/(2^FACTOR);
S=s_re+i*s_im;
```

Figure 4.6 Matlab code for the self-defined complex number addition function

The core part of this function, which is used to embed the proposed ETA algorithms, is the function of **addition**(). According to the different kinds of ETA described in Chapter 3, there are different implementations of the function **addition**(). The following two implementations (see Figures 4.7 and 4.8) have employed the algorithms of ETAI and ETAIM, respectively.

(1) ETAI:

```
function result=addition(x,y)
N1=2^32;
N2=2^20;
z=0,x1=0,y1=0,x2=0,y2=0,z1=0,z2=0,n=N2,flag=0;
x1=floor(x/N2);
y1=floor(y/N2);
z1=(x1+y1)*N2;
x2=x-x1*N2;
y2=y-y1*N2;
n=n/2;
while(flag==0 && n>0)
    xt=floor(x2/n);
    yt=floor(y2/n);
    if((xt+yt)==1)
        z2=z2+n;
    else
        if(xt==1 && yt==1)
            z2=z2+n*2-1;
            flag=1;
        end
    end
    x2=x2-xt*n;
    y2=y2-yt*n;
    n=floor(n/2);
end
z=z1+z2;
result=z;
```

Figure 4.7 Matlab code for ETAI

(2) ETAIIM:

```

function result=addition(a,b)
TOTAL_NUMBER=32;
result=0;
for i=1:5
    x(i)=a-floor(a/2^4)*2^4;
    y(i)=b-floor(b/2^4)*2^4;
    a=(a-x(i))/2^4;
    b=(b-y(i))/2^4;
end
cin(1)=0;
for i=1:5
    if((x(i)+y(i))>15)
        cin(i+1)=1;
    else
        cin(i+1)=0;
    end
end
for i=1:5
    s(i)=x(i)+y(i)+cin(i);
    if(s(i)>15)
        s(i)=s(i)-16;
    end
end
rest=(a+b+cin(6))*2^20;
for i=1:5
    result=result+s(i)*2^(4*(i-1));
end
result=result+rest;

```

Figure 4.7 Matlab code for ETAIIM

4.4 Digital Image Processing

As mentioned in Section in 4.1.4, there are a lot of subfields of DSP. In this thesis, the digital image processing is taken as the platform to illustrate the application of the ETA-based FFT, which leads to the application of the proposed ETA's.

Visual information plays an important role in almost every area of people's life. Today, much of this information is represented and processed digitally. Digital image processing is ubiquitous, with applications ranging from television to tomography, from photography to printing, from robotics to remote sensing. Generally speaking, digital image processing is the use of computer algorithms to perform image processing on digital images. Before the application of FFT in digital image processing is discussed, a brief introduction of some basic knowledge of the digital image will first be given.

A digital image is stored as an array in computers. The value of each element represents the color of the corresponding pixel. For a grayscale image, the image is represented by a 2-dimensional (m-by-n) array. If the array is *double* (i.e., the data type of the elements in this array is *double*), the pixel value of 0.0 represents the color of black, the pixel value of 1.0 represents white, and the pixel values in between represent the different levels of gray; if the array is *uint8* (unsigned 8-bit integer), the white is represented by the pixel value of 255; if the array is *uint16* (unsigned 16-bit integer), the pixel value of 65535 represents the color of white. For convenience, in the rest of this thesis, if no notification is given, the image arrays being discussed are all assumed to be *uint8* arrays. For a color image, the image is represented by a 3-dimensional (m-by-n-3) array. The value for each pixel consists of three numbers giving the decomposition of the color in the three primary colors Red (R), Green (G), and Blue (B). The decomposition of a color in the three primary colors is quantified by a number between 0 to 255. For example, white is coded as $R = 255$, $G = 255$, and

$B = 255$; black will be known as $(R, G, B) = (0, 0, 0)$; and say, bright pink will be $(255, 0, 255)$. That is why the truecolor image is also called RGB image. A grayscale image can be converted to truecolor format by concatenating three copies of the original matrix along the third dimension. This thesis focuses mainly on the grayscale image.

Because each digital grayscale image is represented by an m -by- n matrix, the Fourier transform towards an image is actually performed on its representing matrix. After the transform, the *uint8* matrix will become a complex matrix whose elements are complex numbers. If inverse Fourier transform is performed on this complex matrix, the original matrix will be retrieved.

Figure 4.8 shows a very simple digital image, which is only a white spot in a black background, and its corresponding 16-by-16 matrix **A**. To demonstrate the functions of FFT and inverse FFT that have been introduced in previous sections, the standard FFT is first performed on matrix **A**, after which follows the inverse FFT. The simulation results are shown in Figure 4.9. From the figure, it can be seen that after FFT, the original matrix **A** has been transformed to a complex matrix, which can be regarded as the frequency domain representation of matrix **A**. The inverse FFT that is performed thereafter on the obtained complex matrix transforms it back to the original matrix **A**.

(a) The image of a white dot in a black background

Figure 4.8 A simple digital image and its matrix representation:

(a) the image; (b) the representing matrix

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 255 & 255 & 255 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 & 255 & 255 & 255 & 255 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 & 255 & 255 & 255 & 255 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 255 & 255 & 255 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) The 16-by-16 matrix representing the image

Figure 4.8 (continued) A simple digital image and its matrix representation

(a) the image; (b) the representing matrix

$$\begin{array}{c} A \\ \downarrow \text{FFT} \\ 1000 \times \begin{bmatrix} 4.0800 & -2.5639-2.5639i & \cdot & \cdot & \cdot & -2.5639+2.5639i \\ -2.0963-3.1373i & -0.6515-3.2753i & \cdot & \cdot & \cdot & 3.2753+0.6515i \\ \vdots & \vdots & & & & \vdots \\ -2.0963+3.1373i & 3.2753-0.6515i & \cdot & \cdot & \cdot & -0.6515-3.2753i \end{bmatrix} \\ \downarrow \text{Inverse FFT} \\ A \end{array}$$

Figure 4.9 Matrix A after FFT and inverse FFT

4.5 Application of ETA-based FFT in Digital Image Processing

As mentioned in previous sections, if the ETA is used to compute all the additions involved in an FFT process, the normal FFT will become the ETA-based FFT. To illustrate the functionality of an ETA-based FFT, it is performed on a matrix, for example, the matrix $\mathbf{A}$ shown in Figure 4.8 (b) and then use the normal inverse FFT to retrieve the “original matrix” (noted as $\mathbf{A}_E$). Comparing $\mathbf{A}$ and $\mathbf{A}_E$, one can see the difference between these two matrices and thereby the accuracy performance of the proposed ETA’s. It should be noted that after the inaccurate FFT, the complex matrix may not be able to be transformed back to a real integer matrix. Instead, the retrieved matrix may also be a complex matrix. As only the real integer numbers ranging from 0 to 255 are useful in representing colors, only the real integer part of those complex numbers are remained in $\mathbf{A}_E$ while other parts are discarded. The computational results using different ETA algorithms are shown in Figure 4.10.

From Figure 4.10, it can be seen that the matrices after being processed by ETAI-based and ETAIIM-based FFT are almost the same with the original one, except that some small errors (which can also be regarded as “noises”) have been introduced into the matrices. This result shows that the proposed two types of ETA have the potential to be made use of in the digital image processing.

To better demonstrate the practicality of the proposed ETA’s and show the results visually, the same simulation method described above is applied again to a more complex image. The original image along with the images processed by the ETA-based FFT’s, are displayed in Figures 4.11 to 4.13 for the quality comparison. The original image is shown in Figure 4.11 and the images processed using ETAI-based FFT and ETAIIM-based FFT are presented in Figures 4.12 and 4.13, respectively.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	254	254	255	0	0	0	0	0	0	0	0
0	0	0	0	255	252	251	252	255	0	0	0	0	0	0	0
0	0	0	0	254	253	253	253	254	0	0	0	0	0	0	0
0	0	0	0	0	252	252	252	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	0	0	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a) Matrix A_E using ETAI

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	255	255	255	0	0	0	0	0	0	0	0
0	0	0	0	255	255	255	255	255	0	0	0	0	0	0	0
0	0	0	0	255	255	255	255	255	0	0	0	0	0	0	0
0	0	0	0	1	255	255	255	1	0	0	0	0	0	0	0
0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

(b) Matrix A_E using ETAHM

Figure 4.10 Matrix A_E using different ETA algorithms

Figure 4.11 Original image

Figure 4.12 Image using ETAI-based FFT: (a) 12 bits in the accurate part and 20 bits in the inaccurate part; (b) 8 bits in the accurate part and 24 bits in the inaccurate part.

Figure 4.13 Image using ETAIIM-based FFT: (a) first three carry generators cascaded; (b) first two carry generators cascaded.

Comparisons between the images in Figures 4.11, 4.12, and 4.13 have shown that the difference between Figure 4.12 (a), Figure 4.13 (a), and Figure 4.11 is hardly noticed by human eyes and can be totally neglected. This demonstrates that using the ETAI-based FFT [see Figure 4.12 (a)] and the ETAIIM-based FFT [see Figure 4.13 (a)] proposed in Chapter 3, the quality loss to the image is very small and can be accepted by most users. Figure 4.12 (b) and Figure 4.13 (b) are provided here for the comparison purpose. Comparing Figures 4.12 (a) and (b), it is shown that when more bits are put into the inaccurate part and fewer bits in the accurate part, the quality of the digital image would be seriously degraded though the power consumption and speed performance would be significantly improved. Similarly, the comparison between Figures 4.13 (a) and (b) demonstrates that making the higher order bits more accurate can achieve great improvement in the overall accuracy of the output results, which is displayed through the quality of the image, although some power consumption and speed performance will be correspondingly sacrificed.

The simulation results shown in this section have illustrated that the proposed ETA's are useable in real applications. The FFT function embedded with ETAI or ETAIIM can be used in digital image processing and still perform acceptably.

Chapter 5 Conclusions and Suggestions for Future Work

5.1 Conclusions

This project has adopted an emerging concept in digital integrated circuit design and testing, the *Error Tolerance*, in the design of digital parallel adder to propose a new type of adder, named the *Error-Tolerant Adder* (ETA). Both the mathematical properties and electronic performance of this new type of adder have been thoroughly studied and investigated.

Two different implementations of the ETA have been proposed, namely the ETA Type I (ETAI) and ETA Type II (ETAI). The basic idea of ETAI is to divide an adder into two separate parts, an accurate part and an inaccurate part, and perform the operations simultaneously in these two parts. This will allow the addition process to be speed up and be more energy-efficient. Simulation results show that the ETAI is able to attain 79.51%, 87.20%, 90.22%, and 85.02% improvement in power-delay product (PDP) when compared with the conventional Ripple-Carry Adder (RCA), Carry-Skip Adder (CSK), Carry-Select Adder (CSL), and Carry-Lookahead Adder (CLA), respectively. The ETAII, on the other hand, is based on the idea that in most cases the carry signal for a bit position is determined by several neighboring bits instead of all the bits on its right. Hence, the critical path of the whole circuit can be greatly curtailed by dividing the whole adder into a number of blocks and conducting the addition operations in each block concurrently. With this method, improvements in power consumption and speed performance can be achieved. Simulation results show that the PDP of ETAIIM (a modified structure of the original ETAII) is 60.02%, 75.03%, 80.93%, and 70.78% better than that of the RCA, CSK, CSL, and CLA, respectively. As a trade-off, for both the ETAI and ETAII, to obtain the improvements in power and speed, they may suffer from certain accuracy loss.

The ultra-low power and high speed characteristics of the ETA provide an innovative way to break through the bottleneck of conventional digital integrated circuit design technology that always hinders the circuit designers from achieving both the low-power and high-speed simultaneously. When the accuracy of a circuit is brought into the design process, the two dimensional trade-off between power and speed becomes three dimensional, i.e., power-speed-accuracy. Adding the new design parameter provides the circuit designers with more flexibility when designing a circuit. If low-power and high-speed are desired, the accuracy can be used as a trade-off, and vice versa.

The ETA, which is actually an “inaccurate” adder, has its potential applications in the systems where 100% accuracy is not necessary or ultra-low power and high speed are crucial. One example is digital image processing. The FFT function, which is a basic function in digital image processing as well as other DSP applications, is taken as the platform to embed the proposed ETA’s. By making use of the ETA algorithms to implement the FFT function, the ETA-based FFT can be realized. The ETA-based FFT function is first engaged to perform the DFT of a target image, and then the normal inverse FFT function is used to retrieve the image. This process yields an “inaccurate” image which is to some extent blurred from the original one. The loss of image quality is however insignificant. The simulation results showed that the slightly blurred images are completely acceptable to human eyes and therefore the proposed ETA’s are proven to be usable in the image processing systems.

As described above, the transistor-level circuit simulations and the application (algorithm)-level Matlab simulations have shown the advantages and practicality of the proposed ETA’s. These conclusions demonstrate a bright future of the error-tolerant circuit. In the near future, the error-tolerant circuit/system may become an important part in many electronic devices, such as the biomedical electronic devices and personal portable electronic devices.

5.2 Suggestions for Future Work

This project has started a new direction of digital integrated circuit and system design. On the road of exploring in this new direction, there are a lot more work that can and need to be done.

Firstly, other implementations of ETA other than the proposed two presented in this thesis can be investigated. The aim of designing an ETA is to find a structure that can gain significant improvements in power consumption and speed performance while still maintaining a high accuracy (as high as possible). An efficient ETA structure can trade only a little loss in accuracy for great improvements in other metrics.

Secondly, the error-tolerant design can be extended from adder to other digital circuits and eventually the whole digital system. The concept of ETA has been proven to be feasible by employing it in digital image processing. It is therefore logical to extend this concept to other digital circuits such as the multiplier, and to a whole DSP system.

Thirdly, the applications of the error-tolerant circuit/system in the real world should be further studied and investigated. In this thesis, only the ETA-based FFT function in a single precision (32-bit) fixed-point DSP system is tested. More complex signal processing tasks other than FFT need to be involved to embed the error-tolerant circuits and be tested. Besides, the error-tolerant arithmetic operations in floating-point system need also be tested. This is because the procedure of addition and multiplication of floating-point numbers are totally different with the fixed-point numbers. For the same input operands and same error-tolerant algorithm, the accuracies of the results derived in different number representation systems are also different.

Fourthly, the complete design processes of the proposed ETA's need to be done. This project focuses on the application (algorithm)-level simulation and transistor-level pre-layout circuit simulation. As a complete design process, many other works, such as layout, post-layout simulation, fabrication, and testing, should be done.

Lastly, the error-tolerant design can be extended from the Application Specific Integrated Circuit (ASIC) implementation to the Field Programmable Gate Array (FPGA) implementation. FPGA is becoming more and more popular in digital system design because of its many advantages over the ASIC. Hence, employing the concept of Error-Tolerance in FPGA implementation may also be a valuable work.

Although there are still a lot of work needed to be done in the future, this project has already provided a good start in this completely new and very promising area.

Publications

1. Zhang Weijia, Goh Wang Ling, and Yeo Kiat Seng, "PCMOS-based Hardware Implementation of Bayesian Network," *Proc. of IEEE International Conference on Electron Devices and Solid-State Circuits*, pp. 337-340, Taiwan, Dec. 2007.
2. Loy Liang Yu, Zhang Weijia, Kong Zhi Hui, Goh Wang Ling, and Yeo Kiat Seng, "Body-Bootstrapped-Buffer Circuit for CMOS Static Power Reduction," *IEEE Asia Pacific Conference for Circuit and Systems (APCCAS) 2008*, to be published.
3. Zhang Weijia, Goh Wang Ling, Yeo Kiat Seng, and Zhu Ning, "Design of Low-Power High-Speed Error-Tolerant Adder and Its Application in Digital Signal Processing," *IEEE Trans. on VLSI*, submitted for publication.

References

- [1] Gordon E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, 1965.
- [2] K. L. Shepard, "Design methodologies for noise in digital integrated circuits," in *Proc. Design Automation Conference*, pages 94--99, San Francisco, California, June 1998.
- [3] L. B. Kish, "End of Moore's law: thermal (noise) death of integration in micro and nano electronics," *Physics Letters A*, 305: 144--149, Dec. 2002.
- [4] K. Natori and N. Sano, "Scaling limit of digital circuits due to thermal noise," *Journal of Applied Physics*, 83: 5019--5024, May 1998.
- [5] R. Hegde and N. R. Shanbhag, "Toward achieving energy efficiency in presence of deep submicron noise," *IEEE Trans. VLSI Syst.*, 8: 379--391, August 2000.
- [6] J. D. Meindl, "Low power microelectronics: retrospect and prospect," *Proc. IEEE*, 83: 619--635, April 1995.
- [7] <http://public.itrs.net/Files/2003ITRS/Home2003.htm>
- [8] K. V. Palem, "Energy aware computing through probabilistic switching: a study of limits," *IEEE Trans. on Computers*, Vol. 54, No. 9, September 2005.
- [9] S. Cheemalavagu, P. Korkmaz, and K. V. Palem, "Ultra low energy computing via probabilistic algorithms and devices: CMOS device primitives and the energy-probability relationship," In *Proc. of the 2004 International Conference on Solid State Devices and Materials*, pages 402--403, Tokyo, Japan, September 2004.
- [10] P. Korkmaz, B. E. S. Akgul, K. V. Palem and L. N. Chakrapani, "Advocating noise as an agent for ultra-low energy computing: probabilistic complementary metal-oxide-semiconductor devices and their characteristics," *Japanese Journal of Applied Physics*, Vol. 45, No. 4B, 2006.
- [11] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.
- [12] N. R. Shanbhag, "Reliable and efficient system-on-chip design," *IEEE Computer*, 37(3): 42--50, May 2004.
- [13] Melvin A. Breuer, "Error-Tolerance," in *Proc. of the IEEE International Conference on Computer Design (ICCD)*, 2004.
- [14] A. J. Strecok, "On the calculation of the inverse of the error function, mathematics of computation," *Mathematics of Computation*, 22:144--158, Jan. 1968.
- [15] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem, "Probabilistic Arithmetic and Energy Efficient Embedded Signal Processing," *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Seoul, Korea, Oct. 2006.
- [16] C. Oliver and S. Quegan, *Understanding Synthetic Radar Images*. Raleigh, NC: SciTch Publishing, 2004.

- [17] Zhang Weijia, Goh Wang Ling, and Yeo Kiat Seng, "PC MOS-based Hardware Implementation of Bayesian Network," *Proc. of IEEE International Conference on Electron Devices and Solid-State Circuits*, pp. 337-340, Taiwan, Dec. 2007.
- [18] Melvin A. Breuer, "Let's think analog," *Proc. of the IEEE Computer Society Annual Symposium on VLSI*, 2005.
- [19] M. A. Breuer, S. K. Gupta, and T. M. Mak, "Defect and error-tolerance in the presence of massive numbers of defects," *IEEE Design and Test of Computers* 21(3): 216--227, 2004.
- [20] M. A. Breuer, "Intelligible test techniques to support error-tolerance," *Proc. Asian Test Symp.*, pp. 386--393, 2004.
- [21] H. Chung and A. Ortega, "Analysis and testing for error tolerant motion estimation," *Proc. International Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 514--522, 2005.
- [22] I. Chong and A. Ortega, "Hardware testing for error tolerant multimedia compression based on linear transforms," *Proc. International Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 523--531, 2005.
- [23] Kuen-Jong Lee, Tong-Yu Hsieh and M. A. Breuer, "A novel test methodology based on error-rate to support error-tolerance," *Proc. International Test Conf.*, pp. 1136--1144, 2005.
- [24] Tong-Yu Hsieh, Kuen-Jong Lee, and Melvin A. Breuer, "Reduction of detected acceptable faults for yield improvement via error-tolerance," *Design, Automation and Test in Europe Conference and Exhibition*, pp. 1--6, April 2007.
- [25] Kevin B. Korb, Ann E. Nicholson, *Bayesian Artificial Intelligence*. CHAPMAN & HALL/CRC, 2004.
- [26] E. Gelenbe and F. Batty, "Minimum graph covering with the random neural network model," In *Neural Networks: Advances and Applications*, vol. 2, 1992.
- [27] H. Fuks, "Non-deterministic density classification with diffusive probabilistic cellular automata," *Physical review E, Statistical, Nonlinear, and Soft Matter Physics*, 66(066106), 2002.
- [28] Y. Z. Ding and M. O. Rabin, "Hyper-encryption and everlasting security," *Lecture Notes In Computer Science, Proceeding of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, 2285:1—26, 2002.
- [29] Melvin A. Breuer and Haiyang Zhu, "Error-tolerance and multi-media," in *Proc. of the 2006 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2006.
- [30] *Recommendation ITU-T P.830, Subjective performance assessment of telephone band and wideband digital codecs*, Int'l Telecommunications Union, 1996.
- [31] Jan M. Rabaey, Anantha Chandrakasan, Borivoje Nikolic, *Digital Integrated Circuits*, Prentice Hall, 2003.
- [32] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A System Perspective*. Reading, MA: Addison-Wesley, 1993.
- [33] N. Zhuang and H. Hu, "A new design of the CMOS full adder," *IEEE J. of Solid-State Circuits*, vol.21, no.5, pp.840--844, May 1992.

- [34] R. Shalem, E. John, and L. K. John, "A novel low power energy recovery full adder cell," in *Proc. IEEE Great Lakes VLSI Symp.* pp. 380--383, Feb. 1999.
- [35] E. Abu-Sharna and M. Bayoumi, "A new cell for low power adders," in *Proc. Inf. Midwest Symp. Circuits and Systems*, pp.1014--1017, 1995.
- [36] AA. Fayed, M. A. Bayoumi, "A low power 10-transistor full adder cell for embedded architectures," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol.4, pp.226--229, Sydney, Australia, May 2001.
- [37] M. Lehman and N. Burla, "Skip techniques for high-speed carry propagation in binary arithmetic units," *IRE Transactions on Electronic Computers*, vol.EC-10, pp.691--698, December 1962.
- [38] O. Bedrij, "Carry Select Adder," *IRE Trans. on Electronic Computers*, vol.EC-11, pp.340--346, 1962.
- [39] O. MacSorley, "High speed arithmetic in binary computers," *IRE Proceedings*, vol.49, pp.67--91, 1961.
- [40] N. TAgagi, H. Tago, C. R. Baugh, and S. MUroga, *Logic Design*, CRC Press, 2003.
- [41] I. Koren, *Computer Arithmetic Algorithms*, Englewood Cliffs, New Jersey: Prentice Hall, 1993.
- [42] Franco Maloberti and Chen Gang, "Performing arithmetic functions with the Chinese abacus approach," *IEEE Transaction on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 46, no. 12, pp. 1512 – 1515, Dec. 1999.
- [43] L. Jozwiak, D. Gawlowski, A. Slusarczyk, and A. Chojnacki, "Static power reduction in nano CMOS circuits through an adequate circuit synthesis," in *Proceedings of 14th International Conference on Mixed Design of Integrated Circuits and Systems*, pp. 21-23, Jun 2007.
- [44] L. Bisdounis, S. Nikolaidis, O. Koufolavlou, and C.E. Goutis, "Modeling the CMOS short-circuit power dissipation," *IEEE International Symposium on Circuits and Systems*, Vol. 4, May 1996.
- [45] L. Bisdounis, O. Koufolavlou, and S. Nikolaidis, "Accurate evaluation of CMOS short-circuit power dissipation for short-channel devices," *IEEE International Symposium on Low Power Electronics and Design*, Aug 1996.
- [46] A. Chandrakasan, W. Bowhill, and F. Fox, "Design of high-performance microprocessor circuits," *IEEE Press*, 2001.
- [47] J. A. Butts, and G. S. Sohi, "A static power model for architects," *IEEE International Symposium on Microarchitecture*, Dec 2000.
- [48] Paul H. Bardell, *Built-in Test for VLSI*, Wiley, 1987.
- [49] Kiat-Seng Yeo, Kaushik Roy, *Low-Voltage, Low-Power VLSI Subsystems*, McGraw-Hill, 2005.
- [50] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, 1999.
- [51] E. O. Brigham, *The fast Fourier transform and its applications*, Prentice Hall, 1988.
- [52] James W. Cooley, and John W. Tukey, "An algorithm for the machine

- calculation of complex Fourier series," *Math. Comput.* 19: 297–301, 1965
- [53] I. J. Good, "The interaction algorithm and practical Fourier analysis," *J. R. Statist. Soc. B* 20 (2), 361-372, 1958.
 - [54] L. H. Thomas, "Using a computer to solve problems in physics," in *Applications of Digital Computers*, 1963.
 - [55] Georg Bruun, "z-Transform DFT filters and FFTs," *IEEE Trans. on Acoustics, Speech and Signal Processing* (ASSP) 26 (1), 56-63, 1978.
 - [56] C. M. Rader, "Discrete Fourier transforms when the number of data samples is prime," *Proc. IEEE* 56, 1107–1108, 1968.
 - [57] Leo I. Bluestein, "A linear filtering approach to the computation of the discrete Fourier transform," *Northeast Electronics Research and Engineering Meeting Record* 10, 218-219, 1968.
 - [58] John G. Proakis, and Dimitris Manolakis, *Digital Signal Processing - Principles, Algorithms and Applications*, Pearson.
 - [59] William Stalling, *Computer organization and architecture: designing for performance*, Prentice Hall, 2000.
 - [60] *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Std 754-1985, IEEE Computer Society (1985).
 - [61] Rafael C. Gonzales, *Digital Image Processing*, Prentice Hall, 2002.
 - [62] Gerard Blanchet, *Digital Signal and Image Processing Using Matlab*, ISTE Ltd., 2006.

Appendices

Appendix A: Hspice netlist of ETAI

*****INVERTER*****

```
.SUBCKT INVERTER IN OUT VN VP
M1 OUT IN VP VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT IN VN 0 nmos_1p8 L=0.18U W=0.3U
.ENDS
```

*****2-INPUT AND GATE*****

```
.SUBCKT AND A B OUT
M1 OUT_B A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT_B B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 OUT_B A TEMP 0 nmos_1p8 L=0.18U W=0.6U
M4 TEMP B 0 0 nmos_1p8 L=0.18U W=0.6U
X1 OUT_B OUT 0 VDD INVERTER
.ENDS
```

*****3-INPUT AND GATE*****

```
.SUBCKT AND3 A B C OUT
M1 OUT_B A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT_B B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 OUT_B C VDD VDD pmos_1p8 L=0.18U W=0.6U
M4 OUT_B A TEMP1 0 nmos_1p8 L=0.18U W=0.9U
M5 TEMP1 B TEMP2 0 nmos_1p8 L=0.18U W=0.9U
M6 TEMP2 C 0 0 nmos_1p8 L=0.18U W=0.9U
X1 OUT_B OUT 0 VDD INVERTER
.ENDS
```

*****NAND GATE*****

```
.SUBCKT NAND A B OUT
M1 OUT A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 OUT A TEMP 0 nmos_1p8 L=0.18U W=0.6U
M4 TEMP B 0 0 nmos_1p8 L=0.18U W=0.6U
.ENDS
```

*****SPECIAL OR GATE*****

```
.SUBCKT ORS A B OUT VN VP
M1 TEMP A VP VDD pmos_1p8 L=0.18U W=1.2U
M2 OUT_B B TEMP VDD pmos_1p8 L=0.18U W=1.2U
```

```
M3 OUT_B A VN 0 nmos_1p8 L=0.18U W=0.3U
M4 OUT_B B VN 0 nmos_1p8 L=0.18U W=0.3U
X1 OUT_B OUT VN VP INVERTER
.ENDS
```

*****NEW HALF ADDER*****

```
.SUBCKT HA_NEW A B OUT CTLB
M2 VN CTLB 0 0 nmos_1p8 L=0.18U W=0.9U
X1 A B OUT VN VDD ORS
M3 OUT CTLB VDD VDD pmos_1p8 L=0.18U W=0.6U
.ENDS
```

*****28-TRANSISTORS FULL ADDER*****

```
.SUBCKT FA A B CIN SUM COUT
M1 T1 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 T1 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 T2 B T1 VDD pmos_1p8 L=0.18U W=0.6U
M4 X A T2 VDD pmos_1p8 L=0.18U W=0.6U
M5 X CIN T1 VDD pmos_1p8 L=0.18U W=0.6U
M6 X CIN T3 0 nmos_1p8 L=0.18U W=0.3U
M7 T3 A 0 0 nmos_1p8 L=0.18U W=0.3U
M8 T3 B 0 0 nmos_1p8 L=0.18U W=0.3U
M9 X A T4 0 nmos_1p8 L=0.18U W=0.3U
M10 T4 B 0 0 nmos_1p8 L=0.18U W=0.3U
M11 T5 CIN VDD VDD pmos_1p8 L=0.18U W=0.6U
M12 T5 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M13 T5 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M14 T6 A T5 VDD pmos_1p8 L=0.18U W=0.6U
M15 T7 B T6 VDD pmos_1p8 L=0.18U W=0.6U
M16 T8 CIN T7 VDD pmos_1p8 L=0.18U W=0.6U
M17 T8 X T5 VDD pmos_1p8 L=0.18U W=0.6U
M18 T8 X T9 0 nmos_1p8 L=0.18U W=0.3U
M19 T9 A 0 0 nmos_1p8 L=0.18U W=0.3U
M20 T9 B 0 0 nmos_1p8 L=0.18U W=0.3U
M21 T9 CIN 0 0 nmos_1p8 L=0.18U W=0.3U
M22 T8 CIN T10 0 nmos_1p8 L=0.18U W=0.3U
M23 T10 A T11 0 nmos_1p8 L=0.18U W=0.3U
M24 T11 B 0 0 nmos_1p8 L=0.18U W=0.3U
M25 SUM T8 VDD VDD pmos_1p8 L=0.18U W=0.6U
M26 SUM T8 0 0 nmos_1p8 L=0.18U W=0.3U
M27 COUT X VDD VDD pmos_1p8 L=0.18U W=0.6U
M28 COUT X 0 0 nmos_1p8 L=0.18U W=0.3U
.ENDS
```


*****FOUR BITS INACCURATE PART*****

```
.SUBCKT LOWER_PART_4 A3 A2 A1 A0 B3 B2 B1 B0 CTL0 CTL1 CTL2 CTL3 S0 S1 S2 S3
XADDER0 A0 B0 S0 CTL0 HA_NEW
XADDER1 A1 B1 S1 CTL1 HA_NEW
XADDER2 A2 B2 S2 CTL2 HA_NEW
XADDER3 A3 B3 S3 CTL3 HA_NEW
.ENDS
```

*****FOUR BITS ACCURATE PART*****

```
.SUBCKT HIGHER_PART_4 A31 A30 A29 A28 B31 B30 B29 B28 CIN24 S28 S29 S30 S31
COUT31
X5 A28 B28 CIN24 S28 COUT28 FA
X6 A29 B29 COUT28 S29 COUT29 FA
X7 A30 B30 COUT29 S30 COUT30 FA
X8 A31 B31 COUT30 S31 COUT31 FA
.ENDS
```

*****FOUR BITS CONTROL BLOCK*****

```
.SUBCKT CONTROL_4 CTLIN1 CTLIN2 A3 A2 A1 A0 B3 B2 B1 B0 CTL0 CTL1 CTL2 CTL3
XC55 A3 B3 TEMP3 NAND
XC56 CTLIN1 CTLIN2 TEMP3 CTL3 AND3
XC57 A2 B2 TEMP2 NAND
XC58 CTL3 TEMP2 CTL2 AND
XC59 A1 B1 TEMP1 NAND
XC60 CTL2 TEMP1 CTL1 AND
XC61 A0 B0 TEMP0 NAND
XC62 CTL1 TEMP0 CTL0 AND
.ENDS
```

*****MAIN CIRCUIT*****

```
X1 A3 A2 A1 A0 B3 B2 B1 B0 CTL0 CTL1 CTL2 CTL3 S0 S1 S2 S3 LOWER_PART_4
X2 A7 A6 A5 A4 B7 B6 B5 B4 CTL4 CTL5 CTL6 CTL7 S4 S5 S6 S7 LOWER_PART_4
X3 A11 A10 A9 A8 B11 B10 B9 B8 CTL8 CTL9 CTL10 CTL11 S8 S9 S10 S11
LOWER_PART_4
X4 A15 A14 A13 A12 B15 B14 B13 B12 CTL12 CTL13 CTL14 CTL15 S12 S13 S14 S15
LOWER_PART_4
X5 A19 A18 A17 A16 B19 B18 B17 B16 CTL16 CTL17 CTL18 CTL19 S16 S17 S18 S19
LOWER_PART_4

X6 A23 A22 A21 A20 B23 B22 B21 B20 0 S20 S21 S22 S23 COUT23 HIGHER_PART_4
X7 A27 A26 A25 A24 B27 B26 B25 B24 COUT23 S24 S25 S26 S27 COUT27
HIGHER_PART_4
X8 A31 A30 A29 A28 B31 B30 B29 B28 COUT27 S28 S29 S30 S31 COUT31
HIGHER_PART_4
```

X9 VDD VDD A19 A18 A17 A16 B19 B18 B17 B16 CTL16 CTL17 CTL18 CTL19
CONTROL_4

X10 CTL19 CTL16 A15 A14 A13 A12 B15 B14 B13 B12 CTL12 CTL13 CTL14 CTL15
CONTROL_4

X11 CTL15 CTL12 A11 A10 A9 A8 B11 B10 B9 B8 CTL8 CTL9 CTL10 CTL11 CONTROL_4

X12 CTL11 CTL8 A7 A6 A5 A4 B7 B6 B5 B4 CTL4 CTL5 CTL6 CTL7 CONTROL_4

X13 CTL7 CTL4 A3 A2 A1 A0 B3 B2 B1 B0 CTL0 CTL1 CTL2 CTL3 CONTROL_4

Appendix B: C code for testing the accuracy of ETAI

```

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define BIT_NUMBER 32
#define BIT_ERROR 20
#define BIT_INPUT 32
#define SAMPLES 10000
#define th0 1.0
#define th1 0.99
#define th2 0.98
#define th3 0.97
#define th4 0.96
#define th5 0.95
#define th6 0.94
#define th7 0.93
#define th8 0.92
#define th9 0.91
#define th10 0.90
char input_a[BIT_NUMBER]={0},
char input_b[BIT_NUMBER]={0},
char result_correct[BIT_NUMBER+1]={0},
char result_new[BIT_NUMBER+1]={0}, difference[BIT_NUMBER+1]={0};
void random_input_generator(char input_a[], char input_b[])
{
    for(int i=0;i<BIT_INPUT;i++)
    {
        input_a[i]=rand()%2;
        input_b[i]=rand()%2;
    }
}
void addition_new(char input_a[], char input_b[])
{
    char carry=0;
    int count=0;
    for(int i=BIT_ERROR;i<BIT_NUMBER;i++)
    {
        if(input_a[i]==1) count++;
        if(input_b[i]==1) count++;
        if(carry==1) count++;
        switch(count){
            case 0:

```

```
        result_new[i]=0;
        carry=0;
        break;
    case 1:
        result_new[i]=1;
        carry=0;
        break;
    case 2:
        result_new[i]=0;
        carry=1;
        break;
    case 3:
        result_new[i]=1;
        carry=1;
        break;
    default:
        printf("error occured!");
        break;
    }
    count=0;
}
result_new[BIT_NUMBER]=carry;
for(i=BIT_ERROR-1;i>=0;i--)
{
    if(input_a[i]==1 && input_b[i]==1)
    {
        for(int j=i;j>=0;j--)
        {
            result_new[j]=1;
        }
        break;
    }
    else if(input_a[i]==0 && input_b[i]==0) result_new[i]=0;
    else result_new[i]=1;
}
}

void addition_con(char input_a[], char input_b[])
{
    char carry=0;
    int count=0;
    for(int i=0;i<BIT_NUMBER;i++)
    {
        if(input_a[i]==1) count++;
        if(input_b[i]==1) count++;
    }
}
```

```
        if(carry==1) count++;
        switch(count){
        case 0:
            result_correct[i]=0;
            carry=0;
            break;
        case 1:
            result_correct[i]=1;
            carry=0;
            break;
        case 2:
            result_correct[i]=0;
            carry=1;
            break;
        case 3:
            result_correct[i]=1;
            carry=1;
            break;
        default:
            printf("error occurred!");
            break;
        }
        count=0;
    }
    result_correct[BIT_NUMBER]=carry;
}

void subtraction(char result_correct[], char result_new[])
{
    char carry=0;
    for(int i=0;i<BIT_NUMBER+1;i++)
    {
        if(result_correct[i]==1)
        {
            if(result_new[i]==0 && carry==0)
            {
                difference[i]=1;
                carry=0;
            }
            else if(result_new[i]==1 && carry==1)
            {
                difference[i]=1;
                carry=1;
            }
            else

```

```
        {
            difference[i]=0;
            carry=0;
        }
    }
    else
    {
        if(result_new[i]==0 && carry==0)
        {
            difference[i]=0;
            carry=0;
        }
        else if(result_new[i]==1 && carry==1)
        {
            difference[i]=0;
            carry=1;
        }
        else
        {
            difference[i]=1;
            carry=1;
        }
    }
}

unsigned int binary2integer(char input[])
{
    unsigned int result=0, temp=1;
    if(BIT_NUMBER<31)
    {
        for(int i=0;i<BIT_NUMBER+1;i++)
        {
            result=result+input[i]*temp;
            temp=temp*2;
        }
    }
    else
    {
        for(int i=BIT_NUMBER-31;i<BIT_NUMBER+1;i++)
        {
            result=result+input[i]*temp;
            temp=temp*2;
        }
    }
}
```

```

        return result;
    }
void main()
{
    int msb_dif,msb_cor;
    int t,i,number=0;
    unsigned int cor_result, obt_result, error;
    int n0=0,n1=0,n2=0,n3=0,n4=0,n5=0,n6=0,n7=0,n8=0,n9=0,n10=0;
    double accuracy, prob_0, prob_1, prob_2, prob_3, prob_4, prob_5, prob_6, prob_7, prob_8,
    prob_9, prob_10;
    srand(time(NULL));
    for(t=0;t<SAMPLES;t++)
    {
        msb_dif=0,msb_cor=0;
        random_input_generator(input_a, input_b);
        addition_new(input_a, input_b);
        addition_con(input_a, input_b);
        cor_result=binary2integer(result_correct);
        obt_result=binary2integer(result_new);
        accuracy=(obt_result*1.0)/(cor_result*1.0);
        if(accuracy==th0) n0++;
        if(accuracy>=th1) n1++;
        if(accuracy>=th2) n2++;
        if(accuracy>=th3) n3++;
        if(accuracy>=th4) n4++;
        if(accuracy>=th5) n5++;
        if(accuracy>=th6) n6++;
        if(accuracy>=th7) n7++;
        if(accuracy>=th8) n8++;
        if(accuracy>=th9) n9++;
        if(accuracy>=th10) n10++;
    }
    prob_0=n0/(SAMPLES*1.0);
    prob_1=n1/(SAMPLES*1.0);
    prob_2=n2/(SAMPLES*1.0);
    prob_3=n3/(SAMPLES*1.0);
    prob_4=n4/(SAMPLES*1.0);
    prob_5=n5/(SAMPLES*1.0);
    prob_6=n6/(SAMPLES*1.0);
    prob_7=n7/(SAMPLES*1.0);
    prob_8=n8/(SAMPLES*1.0);
    prob_9=n9/(SAMPLES*1.0);
    prob_10=n10/(SAMPLES*1.0);
    printf("The probability w.r.t %f: %f\n",th0,prob_0);
}

```

```
printf("The probability w.r.t %f: %f\n",th1,prob_1);
printf("The probability w.r.t %f: %f\n",th2,prob_2);
printf("The probability w.r.t %f: %f\n",th3,prob_3);
printf("The probability w.r.t %f: %f\n",th4,prob_4);
printf("The probability w.r.t %f: %f\n",th5,prob_5);
printf("The probability w.r.t %f: %f\n",th6,prob_6);
printf("The probability w.r.t %f: %f\n",th7,prob_7);
printf("The probability w.r.t %f: %f\n",th8,prob_8);
printf("The probability w.r.t %f: %f\n",th9,prob_9);
printf("The probability w.r.t %f: %f\n",th10,prob_10);
}
```


Appendix C: Hspice netlist of ETAIL

*****INVERTER*****

```
.SUBCKT INVERTER IN OUT VN VP
M1 OUT IN VP VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT IN VN 0 nmos_1p8 L=0.18U W=0.3U
.ENDS
```

*****2-INPUT AND GATE*****

```
.SUBCKT AND2 A B OUT
M1 OUT_B A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT_B B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 OUT_B A TEMP 0 nmos_1p8 L=0.18U W=0.6U
M4 TEMP B 0 0 nmos_1p8 L=0.18U W=0.6U
X1 OUT_B OUT 0 VDD INVERTER
.ENDS
```

*****2-INPUT OR GATE*****

```
.SUBCKT OR2 A B OUT
M1 TEMP A VDD VDD pmos_1p8 L=0.18U W=1.2U
M2 OUT_B B TEMP VDD pmos_1p8 L=0.18U W=1.2U
M3 OUT_B A 0 0 nmos_1p8 L=0.18U W=0.3U
M4 OUT_B B 0 0 nmos_1p8 L=0.18U W=0.3U
X1 OUT_B OUT 0 VDD INVERTER
.ENDS
```

*****3 input AND gate*****

```
.SUBCKT AND3 A B C OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 1 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 1 C VDD VDD pmos_1p8 L=0.18U W=0.6U
M4 1 A 2 0 nmos_1p8 L=0.18U W=0.9u
M5 2 B 3 0 nmos_1p8 L=0.18U W=0.9u
M6 3 C 0 0 nmos_1p8 L=0.18U W=0.9u
XINV4 1 OUT 0 VDD INVERTER
.ENDS
```

*****4 input AND gate*****

```
.SUBCKT AND4 A B C D OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 1 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 1 C VDD VDD pmos_1p8 L=0.18U W=0.6U
M4 1 D VDD VDD pmos_1p8 L=0.18U W=0.6U
M5 1 A 2 0 nmos_1p8 L=0.18U W=1.2u
```

```
M6 2 B 3 0 nmos_1p8 L=0.18U W=1.2u
M7 3 C 4 0 nmos_1p8 L=0.18U W=1.2u
M8 4 D 0 0 nmos_1p8 L=0.18U W=1.2u
XINV5 1 OUT 0 VDD INVERTER
.ENDS
```

*****5 input AND gate*****

```
.SUBCKT AND5 A B C D E OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 1 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 1 C VDD VDD pmos_1p8 L=0.18U W=0.6U
M4 1 D VDD VDD pmos_1p8 L=0.18U W=0.6U
M5 1 E VDD VDD pmos_1p8 L=0.18U W=0.6U
M6 1 A 2 0 nmos_1p8 L=0.18U W=1.5u
M7 2 B 3 0 nmos_1p8 L=0.18U W=1.5u
M8 3 C 4 0 nmos_1p8 L=0.18U W=1.5u
M9 4 D 5 0 nmos_1p8 L=0.18U W=1.5u
M10 5 E 0 0 nmos_1p8 L=0.18U W=1.5u
XINV6 1 OUT 0 VDD INVERTER
.ENDS
```

*****5 input OR gate*****

```
.SUBCKT OR5 A B C D E OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=3.0u
M2 2 B 1 VDD pmos_1p8 L=0.18U W=3.0u
M3 3 C 2 VDD pmos_1p8 L=0.18U W=3.0u
M4 4 D 3 VDD pmos_1p8 L=0.18U W=3.0u
M5 5 E 4 VDD pmos_1p8 L=0.18U W=3.0u
M6 5 A 0 0 nmos_1p8 L=0.18U W=0.3U
M7 5 B 0 0 nmos_1p8 L=0.18U W=0.3U
M8 5 C 0 0 nmos_1p8 L=0.18U W=0.3U
M9 5 D 0 0 nmos_1p8 L=0.18U W=0.3U
M10 5 E 0 0 nmos_1p8 L=0.18U W=0.3U
XINV9 5 OUT 0 VDD INVERTER
.ENDS
```

*****FOUR-BIT CARRY-LOOKAHEAD BLOCK*****

```
.SUBCKT CLA4 C0 A3 A2 A1 A0 B3 B2 B1 B0 COUT4
X1 A0 B0 G0 AND2
X2 A1 B1 G1 AND2
X3 A2 B2 G2 AND2
X4 A3 B3 G3 AND2
X5 A0 B0 P0 OR2
X6 A1 B1 P1 OR2
```

X7 A2 B2 P2 OR2

X8 A3 B3 P3 OR2

X18 G2 P3 T6 AND2

X19 G1 P2 P3 T7 AND3

X20 G0 P1 P2 P3 T8 AND4

X21 C0 P0 P1 P2 P3 T9 AND5

X22 G3 T6 T7 T8 T9 COUT4 OR5

.ENDS

*****28-TRANSISTORS FULL ADDER*****

.SUBCKT FA A B CIN SUM COUT

M1 T1 A VDD VDD pmos_1p8 L=0.18U W=0.6U

M2 T1 B VDD VDD pmos_1p8 L=0.18U W=0.6U

M3 T2 B T1 VDD pmos_1p8 L=0.18U W=0.6U

M4 X A T2 VDD pmos_1p8 L=0.18U W=0.6U

M5 X CIN T1 VDD pmos_1p8 L=0.18U W=0.6U

M6 X CIN T3 0 nmos_1p8 L=0.18U W=0.3U

M7 T3 A 0 0 nmos_1p8 L=0.18U W=0.3U

M8 T3 B 0 0 nmos_1p8 L=0.18U W=0.3U

M9 X A T4 0 nmos_1p8 L=0.18U W=0.3U

M10 T4 B 0 0 nmos_1p8 L=0.18U W=0.3U

M11 T5 CIN VDD VDD pmos_1p8 L=0.18U W=0.6U

M12 T5 A VDD VDD pmos_1p8 L=0.18U W=0.6U

M13 T5 B VDD VDD pmos_1p8 L=0.18U W=0.6U

M14 T6 A T5 VDD pmos_1p8 L=0.18U W=0.6U

M15 T7 B T6 VDD pmos_1p8 L=0.18U W=0.6U

M16 T8 CIN T7 VDD pmos_1p8 L=0.18U W=0.6U

M17 T8 X T5 VDD pmos_1p8 L=0.18U W=0.6U

M18 T8 X T9 0 nmos_1p8 L=0.18U W=0.3U

M19 T9 A 0 0 nmos_1p8 L=0.18U W=0.3U

M20 T9 B 0 0 nmos_1p8 L=0.18U W=0.3U

M21 T9 CIN 0 0 nmos_1p8 L=0.18U W=0.3U

M22 T8 CIN T10 0 nmos_1p8 L=0.18U W=0.3U

M23 T10 A T11 0 nmos_1p8 L=0.18U W=0.3U

M24 T11 B 0 0 nmos_1p8 L=0.18U W=0.3U

M25 SUM T8 VDD VDD pmos_1p8 L=0.18U W=0.6U

M26 SUM T8 0 0 nmos_1p8 L=0.18U W=0.3U

M27 COUT X VDD VDD pmos_1p8 L=0.18U W=0.6U

M28 COUT X 0 0 nmos_1p8 L=0.18U W=0.3U

.ENDS

*****FOUR-BIT RIPPLE-CARRY BLOCK*****

.SUBCKT RCA4 C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4

```
X1 A0 B0 C0 S0 C1 FA
X2 A1 B1 C1 S1 C2 FA
X3 A2 B2 C2 S2 C3 FA
X4 A3 B3 C3 S3 C4 FA
.ENDS
```

```
*****FOUR-BIT ADDER CELL*****
.SUBCKT ADDER_CELL4 C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4
X1 0 A3 A2 A1 A0 B3 B2 B1 B0 C4 CLA4
X2 C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4A RCA4
.ENDS
```

```
*****MAIN CIRCUIT*****
X1 0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4 ADDER_CELL4
X2 C4 A7 A6 A5 A4 B7 B6 B5 B4 S4 S5 S6 S7 C8 ADDER_CELL4
X3 C8 A11 A10 A9 A8 B11 B10 B9 B8 S8 S9 S10 S11 C12 ADDER_CELL4
X4 C12 A15 A14 A13 A12 B15 B14 B13 B12 S12 S13 S14 S15 C16 ADDER_CELL4
X5 C16 A19 A18 A17 A16 B19 B18 B17 B16 S16 S17 S18 S19 C20 ADDER_CELL4
X6 C20 A23 A22 A21 A20 B23 B22 B21 B20 S20 S21 S22 S23 C24 ADDER_CELL4
X7 C24 A27 A26 A25 A24 B27 B26 B25 B24 S24 S25 S26 S27 C28 ADDER_CELL4
X8 C28 A31 A30 A29 A28 B31 B30 B29 B28 S28 S29 S30 S31 C32 RCA4
```

Appendix D: C code for testing the accuracy of ETAI

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define TOTAL_NUMBER 32
#define BLOCK_SIZE 4
#define BLOCK_LOW 8
#define SAMPLES 10000
#define th0 1.0
#define th1 0.99
#define th2 0.98
#define th3 0.97
#define th4 0.96
#define th5 0.95
#define th6 0.94
#define th7 0.93
#define th8 0.92
#define th9 0.91
#define th10 0.90
int carry(int A, int B, int CIN)
{
    int COUT;
    if((A+B+CIN)>(pow(2,BLOCK_SIZE)-1)) COUT=1;
    else COUT=0;
    return COUT;
}
int sum(int A, int B, int CIN)
{
    int S;
    S=A+B+CIN;
    if(S>(pow(2,BLOCK_SIZE)-1)) S=S-pow(2,BLOCK_SIZE);
    return S;
}
void main()
{
    int i=0, j=0, t=0, ground=0, n0=0, n1=0, n2=0, n3=0, n4=0, n5=0, n6=0, n7=0, n8=0, n9=0,
    n10=0;
    long temp_A=0, temp_B=0, obt_result=0, cor_result=0;
    char A[TOTAL_NUMBER]={0}, B[TOTAL_NUMBER]={0};
    int cell_A[TOTAL_NUMBER/BLOCK_SIZE]={0},
    int cell_B[TOTAL_NUMBER/BLOCK_SIZE]={0},
    int cin[TOTAL_NUMBER/BLOCK_SIZE+1]={0},
```

```

int cell_S[TOTAL_NUMBER/BLOCK_SIZE]={0};
double accuracy, prob_0, prob_1, prob_2, prob_3, prob_4, prob_5, prob_6, prob_7, prob_8,
prob_9, prob_10;
srand(time(NULL));
for(t=0;t<SAMPLES;t++)
{
    temp_A=0, temp_B=0, obt_result=0, cor_result=0;
    for(i=0; i<(TOTAL_NUMBER/BLOCK_SIZE); i++)
    {
        cell_A[i]=0;
        cell_B[i]=0;
    }
    for(i=0; i<TOTAL_NUMBER; i++)
    {
        A[i]=rand()%2;
        B[i]=rand()%2;
    }
    for(i=0; i<(TOTAL_NUMBER/BLOCK_SIZE); i++)
    {
        for(j=0; j<BLOCK_SIZE; j++)
        {
            cell_A[i]=cell_A[i]+A[BLOCK_SIZE*i+j]*pow(2,j);
            cell_B[i]=cell_B[i]+B[BLOCK_SIZE*i+j]*pow(2,j);
        }
    }
    for(i=0; i<BLOCK_LOW; i++)
    {
        cin[i+1]=carry(cell_A[i], cell_B[i], ground);
    }
    for(i=BLOCK_LOW; i<(TOTAL_NUMBER/BLOCK_SIZE); i++)
    {
        cin[i+1]=carry(cell_A[i], cell_B[i], cin[i]);
    }
    for(i=0; i<(TOTAL_NUMBER/BLOCK_SIZE); i++)
    {
        cell_S[i]=sum(cell_A[i], cell_B[i], cin[i]);
        obt_result=obt_result+cell_S[i]*pow(2,BLOCK_SIZE*i);
    }
    obt_result=obt_result+cin[TOTAL_NUMBER/BLOCK_SIZE]*pow(2,TOTAL_NUMB
ER);
    for(i=0; i<TOTAL_NUMBER; i++)
    {
        temp_A=temp_A+A[i]*pow(2,i);
        temp_B=temp_B+B[i]*pow(2,i);
    }
}

```

```

    }
    cor_result=temp_A+temp_B;
    if(cor_result==0) accuracy=1.0;
    else accuracy=(obt_result*1.0)/(cor_result*1.0);
    if(accuracy==th0) n0++;
    if(accuracy>=th1) n1++;
    if(accuracy>=th2) n2++;
    if(accuracy>=th3) n3++;
    if(accuracy>=th4) n4++;
    if(accuracy>=th5) n5++;
    if(accuracy>=th6) n6++;
    if(accuracy>=th7) n7++;
    if(accuracy>=th8) n8++;
    if(accuracy>=th9) n9++;
    if(accuracy>=th10) n10++;
}
prob_0=n0/(SAMPLES*1.0);
prob_1=n1/(SAMPLES*1.0);
prob_2=n2/(SAMPLES*1.0);
prob_3=n3/(SAMPLES*1.0);
prob_4=n4/(SAMPLES*1.0);
prob_5=n5/(SAMPLES*1.0);
prob_6=n6/(SAMPLES*1.0);
prob_7=n7/(SAMPLES*1.0);
prob_8=n8/(SAMPLES*1.0);
prob_9=n9/(SAMPLES*1.0);
prob_10=n10/(SAMPLES*1.0);
printf("The probability w.r.t %f: %f\n",th0,prob_0);
printf("The probability w.r.t %f: %f\n",th1,prob_1);
printf("The probability w.r.t %f: %f\n",th2,prob_2);
printf("The probability w.r.t %f: %f\n",th3,prob_3);
printf("The probability w.r.t %f: %f\n",th4,prob_4);
printf("The probability w.r.t %f: %f\n",th5,prob_5);
printf("The probability w.r.t %f: %f\n",th6,prob_6);
printf("The probability w.r.t %f: %f\n",th7,prob_7);
printf("The probability w.r.t %f: %f\n",th8,prob_8);
printf("The probability w.r.t %f: %f\n",th9,prob_9);
printf("The probability w.r.t %f: %f\n",th10,prob_10);
}

```

Appendix E: Hspice netlist of ETAIIM

*****INVERTER*****

```
.SUBCKT INVERTER IN OUT VN VP
M1 OUT IN VP VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT IN VN 0 nmos_1p8 L=0.18U W=0.3U
.ENDS
```

*****2-INPUT AND GATE*****

```
.SUBCKT AND2 A B OUT
M1 OUT_B A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 OUT_B B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 OUT_B A TEMP 0 nmos_1p8 L=0.18U W=0.6U
M4 TEMP B 0 0 nmos_1p8 L=0.18U W=0.6U
X1 OUT_B OUT 0 VDD INVERTER
.ENDS
```

*****2-INPUT OR GATE*****

```
.SUBCKT OR2 A B OUT
M1 TEMP A VDD VDD pmos_1p8 L=0.18U W=1.2U
M2 OUT_B B TEMP VDD pmos_1p8 L=0.18U W=1.2U
M3 OUT_B A 0 0 nmos_1p8 L=0.18U W=0.3U
M4 OUT_B B 0 0 nmos_1p8 L=0.18U W=0.3U
X1 OUT_B OUT 0 VDD INVERTER
.ENDS
```

*****3 input AND gate*****

```
.SUBCKT AND3 A B C OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 1 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 1 C VDD VDD pmos_1p8 L=0.18U W=0.6U
M4 1 A 2 0 nmos_1p8 L=0.18U W=0.9u
M5 2 B 3 0 nmos_1p8 L=0.18U W=0.9u
M6 3 C 0 0 nmos_1p8 L=0.18U W=0.9u
XINV4 1 OUT 0 VDD INVERTER
.ENDS
```

*****4 input AND gate*****

```
.SUBCKT AND4 A B C D OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 1 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 1 C VDD VDD pmos_1p8 L=0.18U W=0.6U
M4 1 D VDD VDD pmos_1p8 L=0.18U W=0.6U
M5 1 A 2 0 nmos_1p8 L=0.18U W=1.2u
```



```
M6 2 B 3 0 nmos_1p8 L=0.18U W=1.2u
M7 3 C 4 0 nmos_1p8 L=0.18U W=1.2u
M8 4 D 0 0 nmos_1p8 L=0.18U W=1.2u
XINV5 1 OUT 0 VDD INVERTER
.ENDS
```

*****5 input AND gate*****

```
.SUBCKT AND5 A B C D E OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=0.6U
M2 1 B VDD VDD pmos_1p8 L=0.18U W=0.6U
M3 1 C VDD VDD pmos_1p8 L=0.18U W=0.6U
M4 1 D VDD VDD pmos_1p8 L=0.18U W=0.6U
M5 1 E VDD VDD pmos_1p8 L=0.18U W=0.6U
M6 1 A 2 0 nmos_1p8 L=0.18U W=1.5u
M7 2 B 3 0 nmos_1p8 L=0.18U W=1.5u
M8 3 C 4 0 nmos_1p8 L=0.18U W=1.5u
M9 4 D 5 0 nmos_1p8 L=0.18U W=1.5u
M10 5 E 0 0 nmos_1p8 L=0.18U W=1.5u
XINV6 1 OUT 0 VDD INVERTER
.ENDS
```

*****5 input OR gate*****

```
.SUBCKT OR5 A B C D E OUT
M1 1 A VDD VDD pmos_1p8 L=0.18U W=3.0u
M2 2 B 1 VDD pmos_1p8 L=0.18U W=3.0u
M3 3 C 2 VDD pmos_1p8 L=0.18U W=3.0u
M4 4 D 3 VDD pmos_1p8 L=0.18U W=3.0u
M5 5 E 4 VDD pmos_1p8 L=0.18U W=3.0u
M6 5 A 0 0 nmos_1p8 L=0.18U W=0.3U
M7 5 B 0 0 nmos_1p8 L=0.18U W=0.3U
M8 5 C 0 0 nmos_1p8 L=0.18U W=0.3U
M9 5 D 0 0 nmos_1p8 L=0.18U W=0.3U
M10 5 E 0 0 nmos_1p8 L=0.18U W=0.3U
XINV9 5 OUT 0 VDD INVERTER
.ENDS
```

*****FOUR-BIT CARRY-LOOKAHEAD BLOCK*****

```
.SUBCKT CLA4 C0 A3 A2 A1 A0 B3 B2 B1 B0 COUT4
X1 A0 B0 G0 AND2
X2 A1 B1 G1 AND2
X3 A2 B2 G2 AND2
X4 A3 B3 G3 AND2
X5 A0 B0 P0 OR2
X6 A1 B1 P1 OR2
```

X7 A2 B2 P2 OR2

X8 A3 B3 P3 OR2

X18 G2 P3 T6 AND2

X19 G1 P2 P3 T7 AND3

X20 G0 P1 P2 P3 T8 AND4

X21 C0 P0 P1 P2 P3 T9 AND5

X22 G3 T6 T7 T8 T9 COUT4 OR5

.ENDS

*****28-TRANSISTORS FULL ADDER*****

.SUBCKT FA A B CIN SUM COUT

M1 T1 A VDD VDD pmos_1p8 L=0.18U W=0.6U

M2 T1 B VDD VDD pmos_1p8 L=0.18U W=0.6U

M3 T2 B T1 VDD pmos_1p8 L=0.18U W=0.6U

M4 X A T2 VDD pmos_1p8 L=0.18U W=0.6U

M5 X CIN T1 VDD pmos_1p8 L=0.18U W=0.6U

M6 X CIN T3 0 nmos_1p8 L=0.18U W=0.3U

M7 T3 A 0 0 nmos_1p8 L=0.18U W=0.3U

M8 T3 B 0 0 nmos_1p8 L=0.18U W=0.3U

M9 X A T4 0 nmos_1p8 L=0.18U W=0.3U

M10 T4 B 0 0 nmos_1p8 L=0.18U W=0.3U

M11 T5 CIN VDD VDD pmos_1p8 L=0.18U W=0.6U

M12 T5 A VDD VDD pmos_1p8 L=0.18U W=0.6U

M13 T5 B VDD VDD pmos_1p8 L=0.18U W=0.6U

M14 T6 A T5 VDD pmos_1p8 L=0.18U W=0.6U

M15 T7 B T6 VDD pmos_1p8 L=0.18U W=0.6U

M16 T8 CIN T7 VDD pmos_1p8 L=0.18U W=0.6U

M17 T8 X T5 VDD pmos_1p8 L=0.18U W=0.6U

M18 T8 X T9 0 nmos_1p8 L=0.18U W=0.3U

M19 T9 A 0 0 nmos_1p8 L=0.18U W=0.3U

M20 T9 B 0 0 nmos_1p8 L=0.18U W=0.3U

M21 T9 CIN 0 0 nmos_1p8 L=0.18U W=0.3U

M22 T8 CIN T10 0 nmos_1p8 L=0.18U W=0.3U

M23 T10 A T11 0 nmos_1p8 L=0.18U W=0.3U

M24 T11 B 0 0 nmos_1p8 L=0.18U W=0.3U

M25 SUM T8 VDD VDD pmos_1p8 L=0.18U W=0.6U

M26 SUM T8 0 0 nmos_1p8 L=0.18U W=0.3U

M27 COUT X VDD VDD pmos_1p8 L=0.18U W=0.6U

M28 COUT X 0 0 nmos_1p8 L=0.18U W=0.3U

.ENDS

*****FOUR-BIT RIPPLE-CARRY BLOCK*****

.SUBCKT RCA4 C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4

```
X1 A0 B0 C0 S0 C1 FA
X2 A1 B1 C1 S1 C2 FA
X3 A2 B2 C2 S2 C3 FA
X4 A3 B3 C3 S3 C4 FA
.ENDS
```

*****FOUR-BIT ADDER CELL FOR LOWER BITS*****

```
.SUBCKT ADDER_CELL4_LOW C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4
X1 0 A3 A2 A1 A0 B3 B2 B1 B0 C4 CLA4
X2 C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4A RCA4
.ENDS
```

*****FOUR-BIT ADDER CELL FOR HIGHER BITS*****

```
.SUBCKT ADDER_CELL4_HIGH C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4
X1 C0 A3 A2 A1 A0 B3 B2 B1 B0 C4 CLA4
X2 C0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4A RCA4
.ENDS
```

*****MAIN CIRCUIT*****

```
X1 0 A3 A2 A1 A0 B3 B2 B1 B0 S0 S1 S2 S3 C4 ADDER_CELL4_LOW
X2 C4 A7 A6 A5 A4 B7 B6 B5 B4 S4 S5 S6 S7 C8 ADDER_CELL4_LOW
X3 C8 A11 A10 A9 A8 B11 B10 B9 B8 S8 S9 S10 S11 C12 ADDER_CELL4_LOW
X4 C12 A15 A14 A13 A12 B15 B14 B13 B12 S12 S13 S14 S15 C16 ADDER_CELL4_LOW
X5 C16 A19 A18 A17 A16 B19 B18 B17 B16 S16 S17 S18 S19 C20 ADDER_CELL4_LOW
X6 C20 A23 A22 A21 A20 B23 B22 B21 B20 S20 S21 S22 S23 C24 ADDER_CELL4_HIGH
X7 C24 A27 A26 A25 A24 B27 B26 B25 B24 S24 S25 S26 S27 C28 ADDER_CELL4_HIGH
X8 C28 A31 A30 A29 A28 B31 B30 B29 B28 S28 S29 S30 S31 C32 RCA4
```