Imperial College London
Department of Electrical and Electronic Engineering

Design of Approximate Overclocked Datapath

Kan Shi

Supervised by George A. Constantinides

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy of Imperial College London
and the Diploma of Imperial College London

Abstract

Embedded applications can often demand stringent latency requirements. While high degrees of parallelism within custom FPGA-based accelerators may help to some extent, it may also be necessary to limit the precision used in the datapath to boost the operating frequency of the implementation. However, by reducing the precision, the engineer introduces quantisation error into the design.

In this thesis, we describe an alternative circuit design methodology when considering trade-offs between accuracy, performance and silicon area. We compare two different approaches that could trade accuracy for performance. One is the traditional approach where the precision used in the datapath is limited to meet a target latency. The other is a proposed new approach which simply allows the datapath to operate without timing closure. We demonstrate analytically and experimentally that for many applications it would be preferable to simply overclock the design and accept that timing violations may arise. Since the errors introduced by timing violations occur rarely, they will cause less noise than quantisation errors.

Furthermore, we show that conventional forms of computer arithmetic do not fail gracefully when pushed beyond the deterministic clocking region. In this thesis we take a fresh look at Online Arithmetic, originally proposed for digit serial operation, and synthesize unrolled digit parallel online arithmetic operators to allow for graceful degradation. We quantify the impact of timing violations on key arithmetic primitives, and show that substantial performance benefits can be obtained in comparison to binary arithmetic. Since timing errors are caused by long carry chains, these result in errors in least significant digits with online arithmetic, causing less impact than conventional implementations.

Acknowledgements

Upon the finish of this thesis, I would like to offer my sincere gratitude to all the people who helped and encouraged me throughout the course of my PhD.

Firstly I would like to thank my supervisor George Constantinides for his remarkable vision in academic research and brilliant supervision of my PhD, without whom this thesis would not have been possible. He has shared his knowledge and ideas, been patient and constructive to my thoughts, and more importantly, nurtured my confidence in my early research career.

I must thank all my colleagues in CAS and all my friends at Imperial. Special thanks are made to David Boland for all his genuine assistances since the first day when I joined CAS. It is truly a great pleasure to work with him. His incredible talents and enthusiasm towards work and life keep motiving me to become a better person. I also would like to thank my awesome friends Huafeng Liu, Shuo Yang and Lan Sun, for consistently making me feel as warm as home in London.

Last of all I sincerely thank my parents Jianhua Shi and Xiaoman Ni, my parents in law Wei Zhang and Shaoxia Yuan, and my wife Xiaozhe Zhang, for their endless love, wisdom and support in my life. Finally to my beloved grandfather Quan Ni, this thesis is for you.

Copyright Declaration

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Circuit performance has increased tremendously over the past decades with the continuous scaling of Complementary Metal-Oxide-Semiconductor (CMOS) technology, such that circuits can occupy less silicon area and operate faster. However, the drastic variations introduced by higher integration densities are anticipated to be the major obstacle when designing reliable, high performance circuits. Typically hardware designers tend to employ conservative safety margins for timing closure and to ensure a uniform functionality of the design across various possible working environments. Nevertheless, continuing with this approach could become very costly in the short term future. As circuit dimensions scale down, it is increasingly difficult to maintain low relative manufacturing variation, meaning that we can no longer expect a uniform circuit performance. Because timing margins in static timing analysis tools increase with process variability, designing for worst-case delay in circuits will become increasingly difficult, expensive, and will result in large yield loss as relative process variation rises.

Hardware accelerator based on Field-Programmable Gate Arrays (FPGAs) serves as an ideal candidate to sustain performance scaling [123]. A large volume of current studies has demonstrated that significant performance gains can be achieved by using FPGAs over software designs across a wide range of applications [39, 121, 126]. However, one of the major factors that limits the performance of these accelerators is that they typically run at much lower clock frequencies than their counterparts such as General Purpose Processors (GPPs) or Graphics Processing Units (GPUs).

In order to boost the operating frequency of a datapath in an FPGA, the standard approaches are either to heavily pipeline the design or reduce the precision throughout the datapath. For the former method, it should be noted that pipelining will not tend to reduce the circuit latency. Actually the latency in terms of clock cycles will increase. As a result, this method will not be applicable to many embedded applications, which typically have strict latency requirements, or in any datapath containing feedback where C-slow retiming [131] is inappropriate. For the second approach, although at the cost of introducing quantisation errors, tuning the precision of datapath to meet desired accuracy criteria can help to meet the performance demands. Due to the freedom of FPGAs to employ customised variable representations, study into exploiting the potential benefits of using the minimum precision necessary to satisfy a design specification, such as the maximum tolerable error, has been an extensive research topic within the FPGA community [25, 75].

Unfortunately, neither of these conventional approaches tends to remove the conservative timing margin. Over the past few years we have seen a growth of research in the field of “approximate computing”. In contrast to the conventional design approach that ensures deterministic circuit performance, approximate computing explores the potential power or performance benefits that can be obtained when designing and/or operating circuits beyond the deterministic region. This topic is expected to be of growing importance in the short term future due to various reasons such as the increasingly stringent timing and power requirements, design complexity and the environmental and process variations, which are all accompanied by the continuous scaling of process technologies [23]. As pointed out by the International Technology Roadmap for Semiconductors (ITRS07) [107], while future technologies may suffer from much poorer timing performance, extra benefits of manufacturing, test and power consumption can be obtained if the tight requirement of absolute correctness is released for devices and interconnect. Current research in this area has typically focused on relaxing the design constraints and the safety margins that are conventionally used.

In this thesis, we demonstrate an alternative approximate computing methodology when considering trade-offs among multiple design specifications, such as accuracy, performance, area and different types of computer arithmetic. Since it is unlikely to create a completely error-free design as any fixed or floating point representations would introduce quantisation errors, we take a radical shift by suggesting that for certain applications it is beneficial to move away from the traditional model of creating a conservative design that is guaranteed to avoid timing violations. Instead, it may be preferable to create a design in which timing violations may only occur rarely. In order to back up this hypothesis, the answers of three main research questions have to be addressed:

1. How can we quantify the occurrence and impact of timing errors on a datapath?
2. Is it possible to minimise the impact of timing errors by carefully selecting the appropriate computer arithmetic and data representations?
3. How can we efficiently implement the selected "overclocking friendly" computer arithmetic on existing hardware platforms such as modern FPGAs?

In this thesis, we attempt to provide answers to all of these questions.

1.1 Thesis Organization

Chapter 2 highlights the theoretical background and reviews the relevant work to this thesis. It initially provides the background of existing techniques for numerical precision optimisation within hardware. It then introduces the reader to the subject of approximate computing. The state-of-the-art approaches and techniques are reviewed in-depth, together with their limitations in comparison to our proposed approach. Finally we discuss the background of an alternative form of computer arithmetic known as “online arithmetic”, which potentially embodies the overclocking friendly feature.

In Chapter 3, we introduce our proposed overclocking approach for approximate datapath design. We evaluate the probabilistic behaviour of basic arithmetic primitives with different datapath precisions, when operating beyond the deterministic clocking region. We demonstrate that for certain applications it is beneficial to move away from the conservative design approach of ensuring timing closure. Alternatively, it may be preferable to allow timing violations to happen, under the knowledge that they are unlikely to occur frequently, because specific input patterns are required to generate errors. We have created probabilistic models for both design scenarios and we show that not only does the overclocking approach allow us to reduce the need for the conservative timing margin, more importantly, our models and experimental results demonstrate that performance benefits can be achieved in comparison to the traditional situation where target latency is limited by choice of precision.

The limitations of the work in Chapter 3 are addressed in Chapter 4. In standard arithmetic operators such as ripple-carry adders, the most significant digit (MSD) is updated last due to carry propagation, therefore if timing violation happens, it will initially affect the MSD of the results. This potentially results in timing errors of large magnitude. In order to tackle this problem, for the first time we employ an alternative form of MSD-first arithmetic known as “online arithmetic”. Similarly to the previous chapter, we develop probabilistic models for basic arithmetic operators such as general purpose multiplier, which is implemented with online arithmetic, when operating beyond the deterministic clocking region. The model is verified with experimental results from an image processing application. We demonstrate that substantial performance benefits can be achieved when compared to the design method using conventional arithmetic as proposed in Chapter 3.

Chapter 5 tackles the major limitation of utilising online arithmetic operators, which is that they normally require a large area overhead for FPGA implementation. In this chapter, we propose novel approaches to implement the key primitives of online arithmetic: adders and multipliers, efficiently on modern FPGA architectures. The proposed new implementation method makes full use of logic resources within an FPGA, such as its 6-input look-up-tables (LUTs) and dedicated carry resources. We demonstrate experimentally that in comparison to a direct Register-Transfer Level (RTL) synthesis, large area savings can be achieved using the proposed architectures. In addition, because an online multiplier generates the MSD first, we develop a method to create a correctly rounded online multiplier with a reduced precision output that is smaller than a traditional multiplier producing the same results.

In the final chapter, Chapter 6, the thesis is summarised and potential avenues for future research are identified.

1.2 Statement of Originality

In this thesis, we make three main original contributions, of which the full discussions and the quantitative descriptions are detailed in the introduction section of their own dedicated chapter. Here we summarise and highlight the key information of the three main contributions as below:

- A novel combination of datapath function with overclocking in a holistic framework to trade off between accuracy, latency and silicon area, together with the detailed modelling methods of probabilistic errors generated in basic arithmetic primitives from two different design scenarios: overclocking the datapath or truncating precisions that used throughout the datapath. (Chapter 3, [110, 109, 112])

- According to our knowledge, the first proposal and evaluation of online arithmetic as an "overclocking friendly" arithmetic, and the modelling methods of overclocking errors for a digit-parallel multiplier with online arithmetic. (Chapter 4, [113])

- Efficient methods to map digit-parallel adders and multipliers with online arithmetic to modern FPGAs, and a novel method to implement correctly rounded online multipliers for a chosen output precision. (Chapter 5, [111, 114])

1.3 Publications

The following papers and articles have been written during the course of this thesis:

1. *Overclocking Datapath for Latency-Error Tradeoff*, Kan Shi, David Boland, and George A. Constantinides, in Proc. IEEE International Symposium on Circuits and Systems (ISCAS), 2013, pp. 2537-2540.

2. *Accuracy-Performance Tradeoffs on an FPGA Through Overclocking*, Kan Shi, David Boland, and George A. Constantinides, in Proc. IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2013, pp. 29-36. **Winner of a HiPEAC paper award.**

3. *Imprecise Datapath Design: An Overclocking Approach*, Kan Shi, David Boland, and George A. Constantinides, ACM Transactions on Reconfigurable Technology

and Systems (TRETS), vol. 8, no. 2, pp. 1-23, 2013. **Invited paper.**

4. *Datapath Synthesis for Overclocking: Online Arithmetic for Latency-Accuracy Trade-offs*, Kan Shi, David Boland, Ed Stott, Samuel Bayliss and George A. Constantinides, in Proc. ACM/IEEE Design Automation Conference (DAC), 2014, pp. 1-6. **Winner of a HiPEAC paper award.**

5. *Efficient FPGA Implementation of Digit Parallel Online Arithmetic Operators*, Kan Shi, David Boland and George A. Contantinides, in Proc. International Conference of Field-Programmable Technology (FPT), 2014, pp. 115-122. **Best paper nominee.**

6. *Evaluation of Design Trade-offs of Adders in Approximate Datapath*, Kan Shi and George A. Constantinides, in HiPEAC Workshop on Approximate Computing (WAPCO), 2015.

Chapter 2

Background

2.1 Introduction

The main focus of this thesis is exploring the trade-off between accuracy and performance when designing a datapath. In this chapter, we discuss relevant work and background theory to help place our work in context. In general, we restrict our discussions to three main research areas as highlighted below.

We initially discuss the work in the area of word-length optimisation in Section 2.2, because reducing the datapath precision will reduce the circuit latency at the cost of introducing quantisation errors into the design. As FPGAs are capable of employing customised variable representations, there has been extensive research into exploiting the potential benefits of using the minimum precision necessary to satisfy a design specification, such as the maximum tolerable error, within the FPGA community. We will use this methodology as the major comparison to our approach throughout this thesis.

It is worth noting that the choice of precision is not the only source of error when designing a datapath. For the past few years we have seen a growth of parallel streams of research which aim at exploring alternative methods to trade accuracy for design efficiency. This strand of research is known as “approximate computing”, and it is motivated by the fact that extra benefits of manufacturing, test, power and timing can be obtained if the tight requirement of absolute correctness is released for devices and interconnects, as indicated by the ITRS in 2007 [107]. This topic is expected to be of growing importance in the future, because we will face new design challenges as the technology scales further. Research in this area will be discussed in Section 2.3.

However, applying approximate computing techniques with conventional computer arithmetic may potentially result in large errors as indicated by previous research [140] and will be highlighted in our study in Chapter 3. This is largely due to carry propagation in conventional arithmetic, in which carry is propagated from the least significant end to the most significant end. In this thesis, we attempt to tackle this issue by using an alternative form of computer arithmetic named “online arithmetic”, which performs all types of computations in a most-significant-digit-first manner. In Section 2.4, we provide a broad overview of online arithmetic. This section lays the foundation of our discussions on utilising online arithmetic in approximate computing and implementing online arithmetic efficiently on modern FPGAs in Chapter 4 and Chapter 5 respectively.

2.2 Word-Length Optimisation

In comparison to implementations based on general purpose processors, the reconfigurable nature of FPGA enables the employment of customised numerical variable representations. The target of optimising the word-length is to represent variables within a design using minimum precision, while meeting the required accuracy and performance constraints [25]. Normally, a word-length optimisation algorithm is designed to provide the trade-offs between computation accuracy and performance. On one hand, more resources and higher cost of data transfer are required for more accurate implementation of a design. On the other hand, the hardware budget is limited due to the consideration of power and performance.

Existing approaches for word-length optimisation can be classified into two main categories: simulation-based and analytical-based. The former approach decides the optimal word-length by comparing simulation outcomes from both the chosen word-length and a very high precision representation, and thereafter checking whether the comparison result meets the error requirement. This straightforward approach is widely employed in industry, since the whole implementation can be treated as a black box, and a very compact data representation can be obtained. However, the major drawback of this approach is that the quality of output highly depends on the selection of input vectors. For large designs, performing exhaustive simulations is infeasible because of the long simulation time, and therefore corner cases may be missed [75].

As for the analytical-based approach, its basic idea is to model the accuracy of computation results in terms of the chosen word-length. In comparison to its counterpart, the analytical-based approach usually provides more conservative determination of word-length and it is better suited for large designs [22]. Existing work employs multiple kinds of modelling methods, including: interval arithmetic (IA) [2, 86], affine arithmetic (AA) [115] and Satisfiability-Modulo Theory (SMT) [68, 69]. The remaining parts of this section briefly review these three approaches.

In interval arithmetic, each variable is bounded with an interval, and the bounds of final computation results can be determined. This approach is straightforward but may generate an extremely loose bound due to the data-dependency problem [75]. That is, the bound will be significantly overestimated when employing the same variable repeatedly [11].

Affine arithmetic is employed to mitigate the data dependency problem. Instead of representing variables through intervals, AA uses affine expressions which keep track of the correlations of intervals. AA is often employed in digital signal processing (DSP) area, especially for analysis of linear time-invariant (LTI) systems [25], because AA will generate tight bounds for fully affine operations. However, AA still produces loose bounds when there are non-affine calculations.

In the SMT-based approach [68, 69], initial variable ranges are firstly determined through existing analytical methods, *i.e.* AA. Then the initial results are refined using SMT with additional constraints. More specifically, the range refinement procedure of a single variable works similarly as the binary search method, which starts with the initial constraints and update the bounds according to the satisfiability of constraints. For instance, in order to find the upper bound of variable z within a certain expression $z = f(x, y)$, an initial constraint $z > 20$ is applied to check whether there is a satisfiable assignment. If not then 20 can be treated as a refined upper bound and 10 can be applied to repeat the process. Otherwise increase 20 to 40 and check the satisfiability again. In comparison to IA or AA, the SMT-based approach will generate more compact bounds while retaining robustness. However the major weakness of this approach is that the growing of instance set can result in very long operating time.

2.3 Approximate Computing

With the progressive shrinking of fabrication technology into the nanometre range, new design challenges such as design complexity and device uncertainty have arisen in order to sustain performance scaling.

Design complexity becomes a critical challenge due to rapid technology shrinking. Since the introduction of Moore's Law in 1965, the speed of transistor has scaled by nearly fiver orders of magnitude [92]. Meanwhile, the transistor density keeps doubling and power consumption is about 50% lower in every technology generation. In this context, circuit designers are able to create complex architectures by exploiting the increasing transistor integration. On the other hand, the ever rising design complexity has become the driving force of significant research efforts to sustain performance scaling. For instance, we have seen the employment of different levels of parallelism in microprocessor architectures, as well as the shifting from single-core to multi-core architecture, and the rise of heterogeneous parallelism [54].

However, as predicted by the ITRS, the size of individual transistors is reaching physical limits with the transistor gate length near 10nm [108]. As a consequence, the operating frequency is expected to increase slowly in the near future. Besides, device scaling is also curtailed by practical power limits, which is also known as "dark silicon". It is predicted that at 8nm more than 50% of fixed-size chip must be powered off [36]. Therefore, alternative approaches for circuit design and operating must be investigated to release the burden of design complexity.

Another design challenge is the uncertainty accompanied by technology scaling. For example, in 22nm technology the oxide and gate length are only five and 42 atomic layers thick, respectively [45]. With critical device dimensions scaling to atomic range, manufacturing identical devices and maintaining uniform and non-deterministic circuit behaviour become increasingly difficult and expensive. According to Pelgrom's Law [98], the mismatches between device behaviour are inversely proportional to the transistor size. This means the impact of environmental and process variations on performance and reliability of integrated circuits is greatly amplified by technology shrinking.

While a uniform behaviour should no longer be expected with current devices and chip components, the conventional approaches of designing and operating digital circuits remain unchanged. A typical solution to these design challenges is to introduce guard bands, which are designed for worst cases to ensure a uniform circuit behaviour across various operating environments and process variations. This approach is commonly used throughout all types of engineering. It is clearly conservative, and consequently results in significant overhead of both performance and energy. Continuing with this approach and designing for worst-case delay in circuits become increasingly expensive as relative process variation rises.

For the past few years, a novel design strategy known as "approximate computing" has been proposed to tackle the aforementioned design challenges. Research in this area suggests that relaxing the absolute accuracy requirements can provide the freedom to create circuits with better performance and/or energy efficiency. Specifically in this section we review two main research directions in the field of approximate computing. In Section 2.3.1 we discuss one design paradigm called "Better Than Worst-Case (BTWC) Design" [6, 23]. While a typical BTWC design usually contains a checker circuit to ensure the correctness of outputs, another stream of research focuses on providing "good enough" results without adding correction mechanism. This approach can lead to even better performance benefits, at the cost of accuracy loss. In addition, there is also research investigating the methods to provide "imprecise architectures" by designing simplified circuits for approximate computing. Research progress in this area is reviewed in Section 2.3.2.

2.3.1 Better Than Worst-Case Design

As suggested from the name, the BTWC design allows the circuits to operate beyond the worst cases by either partially or completely removing the conservative guard bands. In this case, either a higher operating frequency can be achieved for better performance, or the supply voltage can be reduced for better power efficiency. A typical BTWC design is composed of cores and checkers, as shown in Figure 2.1 [6].

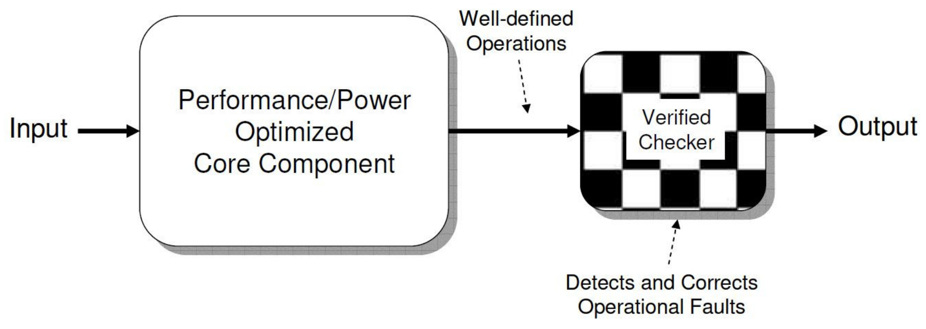

Input
Performance/Power Optimized Core Component
Well-defined Operations
Verified Checker
Output
Detects and Corrects Operational Faults

Figure 2.1: Better Than Worst-Case Design Concept, from [6].

This structure enables the checker to address the design correctness and robustness issues independently, whereas the accuracy requirements of the cores are relaxed such that they can be operated with higher performance. Optionally the system can be recovered at the observation of errors.

As an exemplary design, the Razor project [7, 35] was proposed to shave the conservative timing margins by dynamically scaling the supply voltage and clock frequency beyond the worst-case levels, while monitoring the output error rates by utilising a self-checking circuit. The Razor flip-flop and the pipeline recovery mechanism are illustrated in Figure 2.2. Each Razor flip-flop contains a shadow latch which is triggered by a delayed clock, and it is constrained such that the shadow latch will always provide reliable results. Once the setup time for the main flip-flop is violated due to excessive voltage or frequency scaling, a different result will be sampled by the main flip-flop in comparison to the shadow latch, and an error signal will be generated from the comparator. Razor utilised a pipeline recovery mechanism, as shown in Figure 2.2b, in order to avoid the timing error from corrupting the register and memory state. On occurrence of timing errors, the recovery mechanism will generate forward a bubble and backward flush signal to recover the pipeline from timing failure. This work demonstrated that the benefits brought by removing the safe margin outweigh the cost of monitoring and recovering from errors. For example, 22%, over 30% and up to 64%

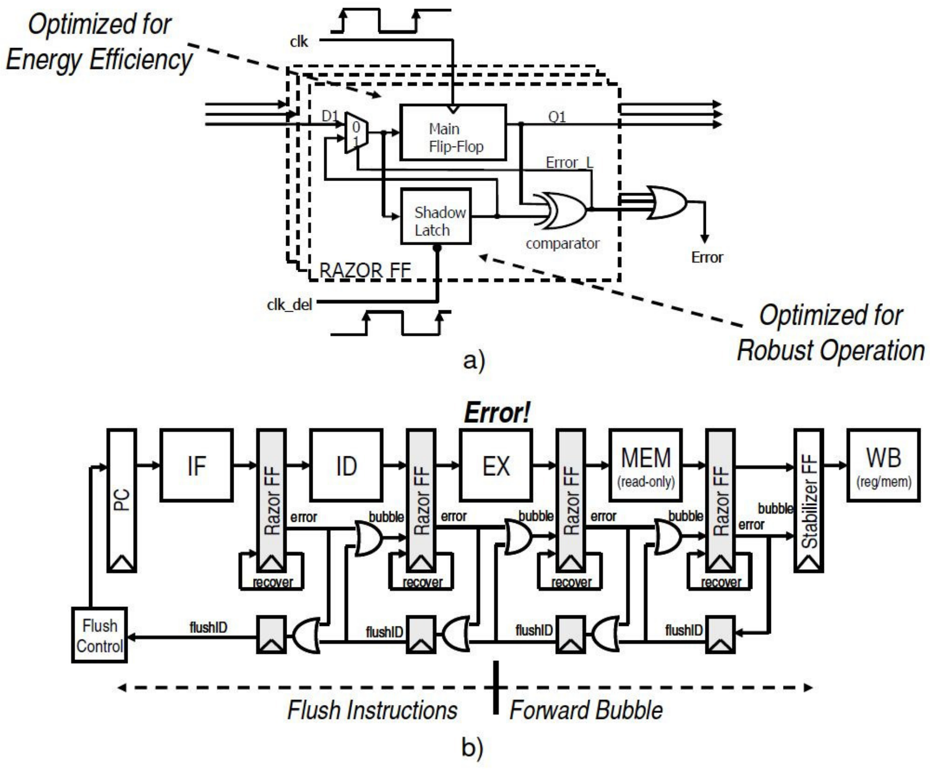


Figure 2.2: Structure of Razor Logic from [7, 35]. (a) the Razor flip-flop. It is utilised to detect timing errors. (b) the pipeline recovery mechanism.

power consumption can be saved with around 0.01%, 1% and 3% error rates at the output, respectively.

Besides the Razor project, related work of BTWC design also includes a similar frequency overscaling technique by operating circuits slightly slower than the critical path delay with dedicated checker circuits to ensure that timing errors will not occur [125] and a processor architecture that decouples the issues of design performance from those of correctness and program verification [5, 16, 130]. In addition, there is work focusing on providing Computer-Aided Design (CAD) solutions, such as synthesis and verification, for BTWC designs. For instance, there is work proposing timing analysis tools that decide the optimum operating frequencies in the non-deterministic region due to process variation [66], and developing synthesis and mapping technologies for BTWC design targeting FPGAs [24, 84].

2.3.2 Unreliable Computing with Unreliable Components

Inherent Application Resilience

Although significant performance and power benefits can be achieved from BTWC design, its major problem is the extra cost, such as silicon area, additional clock cycles and power consumption, introduced by implementing the checker circuits for error detection and recovery. Instead, studies have shown that for a large volume of existing and emerging applications, errors can be potentially tolerated [20]. This feature is also referred to as "inherent application resilience". For these applications, acceptable results can still be generated despite the existence of portions of the application that are computed incorrectly or approximately.

In general, inherent application resilience originates from three main sources, as listed below:

1. *Redundant and noisy input data.* Applications processing real-world data are designed to be robust to noise. For instance in medical applications such as electroencephalogram (EEG) and visual evoked potential (VEP) that detect and process biomedical signals, data sampled from sensors are inevitably contaminated by environmental noise, which will in turn become an important factor that determines the quality of successive computations and final results. In order to achieve noise robustness, data redundancy is normally utilised. On the other hand, the redundancy and noise of input data could potentially form resilience to approximations, as it is not necessary to perform fully accurate computations with noisy data in the first place at the cost of expensive computational efforts.

2. *Algorithmic resilience.* Probabilistic algorithms typically take stochastic inputs and generate outputs with non-deterministic behaviour in terms of correctness and time of operation. Normally a range of outputs can be generated but are

treated equally acceptable. This type of algorithm uses randomness to achieve more efficient computations in comparison to their deterministic competitors. For example, several probabilistic algorithms, such as K-means clustering, iteratively converge to the final solution by sequentially producing approximate answers. In addition, algorithms such as quicksort and simulated annealing embody randomness within the algorithm. The intrinsic algorithmic resilience can remove the necessity for accurate computation to a large extent.

3. *Perceptual resilience.* If the end user of an application is human, it may embody inherent application resilience because of the limited sensitivities of human perceptions. In this case, minor fluctuations of outputs can hardly be recognised and/or distinguished by human senses. For example, human hearing can only perceive sounds with frequency range from about 20Hz to 20kHz, and human visual perception systems will unconsciously fill in missing information from images or videos and filter out high frequency patterns. Consequently for applications involving human perception, ensuring 100% computation accuracy can be expensive and unnecessary.

Current Research

Inspired by the existence of inherent application resilience, a novel design concept "unreliable computing with unreliable components" has been proposed and studied over the past few years [93, 94, 95]. Instead of maintaining absolutely correct and reliable computational results, it is more beneficial in terms of performance and energy to only provide "good enough" results that meet given design specifications. This design strategy is particularly interesting and beneficial to applications with inherent error resilience, such as multimedia [4, 106], digital signal processing (DSP) [77, 1], machine learning [14, 21, 19] and so on. In comparison to BTWC design, another advantage of this design approach is that the checker circuit can be eliminated to reduce extra costs.

Current research in this area mainly focuses on designing probabilistic or imprecise circuits at different levels of the Very-Large-Scale Integration (VLSI) design flow, such as software and algorithm level [60, 78, 100], architecture and circuit level [59] and even transistor level [17, 18, 96]. For instance at the programming language level, a tool was proposed [37, 105] to divide variables and objects in a program into precise parts and approximate parts, which can be mapped to different hardware with different speed-grades, supply voltages, *etc.*, respectively. For instance, approximate data can be processed less reliably, whereas the error sensitive part of the program should be operated with guarantees of correctness. This technique enabled a relatively high service quality, in the meantime energy reduction can be obtained due to approximation. It demonstrated up to 50% of energy saving can be achieved. Another study [139] proposed a design framework that allowed circuit designers to explicitly define the approximate portion of the entire system at Hardware Description Language (HDL) level. The design abstractions were presented as extensions to Verilog HDL through high-level annotations. This framework enabled design and reuse of approximate hardware building blocks.

Similar ideas have also been applied on hardware directly. As an example, a non-uniform voltage scaling technique is proposed for the ripple carry adder [65]. In this study, multiple voltage regions are employed for different bits along a carry chain. That is, higher voltage would be applied for computations generating most significant bits, and vice versa. There is also research developing probabilistic CMOS transistors which modelled the relationship between energy consumption and the probability of correct transistor switching [18].

Another main stream of research focuses on designing approximate logic and imprecise arithmetic circuits by mainly simplifying original structure to trade for performance and energy efficiency. For instance, Lu *et al.* proposed a "shrinking" datapath that can be utilised to mimic and speculate the original logic functions [80]. Kulkarni *et al.* described an under-designed 2×2 multiplier unit, of which the worst case was replaced by a normal case based on the straightforward Karnaugh-Map analysis [71]. In both cases, reduction of area and power consumption can be achieved with the cost of accuracy.

In addition, a large volume of current work is targeting approximate adder design. For example, Gupta *et al.* developed approximate adders at the transistor level and compared the energy efficiency of their proposed architectures over truncation of input word-length of conventional structures [46]. Based on the observation that the accuracy demands for different applications might vary, Kahng *et al.* proposed an adder architecture which offered the flexibility to trade accuracy for energy benefits by optionally utilising part of the adder or enabling additional circuitry to correct for errors for high accuracy requirements [61]. Ye *et al.* derived the relationship between achievable quality and computational effort for a reconfigurable adder, which can be tuned based on the quality-effort curve to achieve better trade-offs [140].

2.3.3 Summary

In this section, we initially discussed the background for the generation of approximate computing. We then reviewed two main research directions in the area of approximate computing: better than worst-case design and unreliable computing with unreliable components. For the BTWC design, we discussed the general architecture that combined cores and checkers, based on an exemplary design: Razor logic. For the second approach of approximate computing, we introduced the inherent application resilience that can be utilised to trade absolute accuracy for performance and power benefits, followed by a review of existing work in this area.

From the current literature, we notice several issues when using the methodology of approximate computing. For example in work conducted by Kedem *et al.* [65] as introduced before, utilising several voltage regions within a ripple carry adder will rarely be practical in real situations. Furthermore, the overhead of applying this technique is not addressed. For research targeting the creation of imprecise architectures, it should be noted that using this type of approach means correct results would never be obtained for certain input patterns. Another limitation is that the link between the probability of output correctness and energy savings or performance improvements for designs beyond operator level are rarely analysed in current research. In addition, these techniques normally target Application-Specific Integrated Circuit (ASIC) implementations and cannot be directly applied onto existing hardware platforms such as FPGAs.

In this thesis, we aim to address these issues by proposing an alternative methodology for designing approximate datapath with the consideration of overclocking into the error regime, as will be discussed in detail in Chapter 3.

2.4 Online Arithmetic

A major obstacle of applying approximate computing techniques with conventional computer arithmetic is that it may potentially result in large error magnitudes. This is because in conventional arithmetic, carry propagates from the least significant end to the most significant end, therefore errors generated by the scaling of either voltage or frequency will initially affect the most significant digits. Normally the existing approach of tackling this problem is to either speculate the carry using additional logic [79, 141] or correct the result after computation with actual carry [62, 140]. In comparison to these existing approaches which still employ conventional arithmetic, we attempt to tackle this issue by adopting a different form of computer arithmetic named online arithmetic. In this section, we review and describe basic concepts of online arithmetic, which was introduced by Ercegovac and Trivedi in 1977 [29, 124]. In general, online arithmetic is designed to perform Most Significant Digit First (MSDF) computation. This requires an implementation of redundancy in the number representation system, as will be detailed in Section 2.4.1. Originally online arithmetic was designed for digit serial operations, of which the benefits are highlighted in Section 2.4.2. Finally, we provide an in-depth discussion of online arithmetic together with existing research in this area in Section 2.4.3.

2.4.1 Redundant Number System

Fixed Point Number Representation

For a standard number system with fixed radix r, a number X can be represented by (2.1) where n and m denote the number of integer digits and fractional digits used for representing X, respectively. The selection of x_i can be either from a canonical system, in which we have $x_i \in \{0, 1, \cdots, r-1\}$, or from a noncanonical system, *i.e.* $x_i \in \{-b, \cdots, -1, 0, 1, \cdots, a\}$ where $a + b + 1 \geq r$.

$$X = \sum_{i=-m}^{n-1} x_i r^i \tag{2.1}$$

A system with fixed positive radix r and canonical set of digit values is referred to as a radix-r conventional number system [34]. In comparison, the noncanonical system is normally employed for redundant number representation. A redundant radix-r weighted number system has more than r possible values for one digit, and hence it allows multiple representations of a given number. In other words, there is "redundancy" in number representation. For example, the binary number 0.011 can be represented in redundant form as $0.1\bar{1}1$, $0.10\bar{1}$ or 0.011 among many other possible representations, where $\bar{1} = -1$.

For a redundant number representation, the digit set can be symmetric or asymmetric with different levels of redundancy. For instance, a radix-4 redundant digit set can be one of the following forms:

1. Symmetric with minimal redundancy ($a + b = r$): *e.g.* $\{\bar{2}, \bar{1}, 0, 1, 2\}$;
2. Asymmetric with minimal redundancy: *e.g.* $\{\bar{3}, \bar{2}, \bar{1}, 0, 1\}$;
3. Symmetric with over redundancy ($a + b > r$): *e.g.* $\{\bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3, 4\}$;
4. Asymmetric with over redundancy: *e.g.* $\{\bar{5}, \bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}$.

In the following sections we restrict our discussion to the very first type, because it is used most commonly. This type of data representation is also known as "Signed Digit (SD)" and it was introduced by Avizienis in 1961 [8].

Coding for Signed-Digit Number

For radix-2 SD representation, the value of a binary digit x_i can be selected within the digit set $\{\bar{1}, 0, 1\}$, and x_i is represented using two bits x_i^+ and x_i^-. There exists several codings for x_i using x_i^+ and x_i^-. For instance, two possible codings are listed in Table 2.1.

For the first form, we have $x_i = x_i^+ - x_i^-$. For circuit level implementation, this coding can be achieved by simply using adders. For the second form, the negation of x_i can be directly obtained by inverting the value of the corresponding x_i^+ and x_i^- pair.

Table 2.1: Two Possible Coding Metric for Signed-Digit Number.

(a) Coding Metric 1

x_i^+	x_i^-	x_i
0	0	0
0	1	$\bar{1}$
1	0	1
1	1	0

(b) Coding Metric 2

x_i^+	x_i^-	x_i
0	0	$\bar{1}$
0	1	0
1	0	0
1	1	1

Addition with Signed-Digit Numbers

The most significant feature of a redundant number system is that carry propagation can be eliminated when performing addition. Thus the delay of addition is a fixed value which is independent of the operand word-length. To achieve this, a 2-stage procedure is employed to guarantee that the carry output is independent of carry input. In the first step, the intermediate sum (S_i') and carry (C_{i+1}) is calculated based on the corresponding operands x_i and y_i, as given in (2.2)

$$rC_{i+1} + S_i' = x_i + y_i \tag{2.2}$$

In the second step, S_i' is added with the previous carry output C_i to obtain the final sum digit S_i without generating carry output, as given in (2.3).

$$S_i = S_i' + C_i \tag{2.3}$$

The structure diagram of this process is shown in Figure 2.3. It can be seen that there only exists 1-digit carry propagation. In comparison to adder with conventional arithmetic and data representation, each SD adder unit embodies a more complex

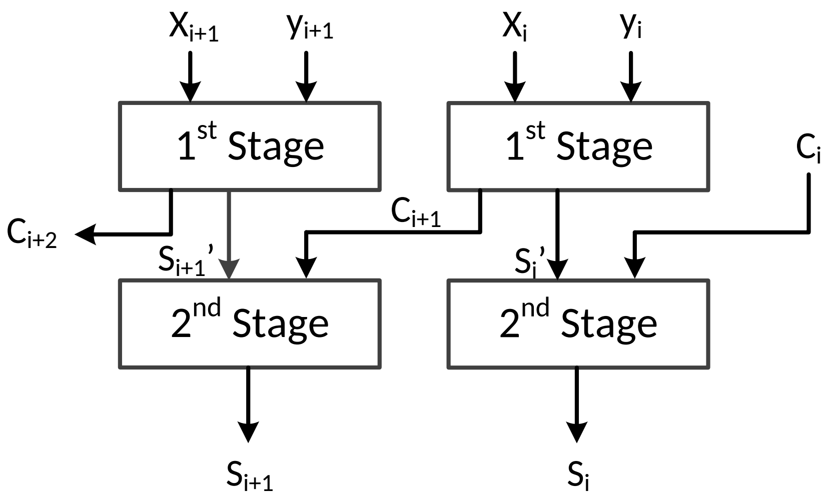


Figure 2.3: Two stages Signed-Digit addition.

structure and correspondingly operates at lower speed than a full adder. However, the delay within the SD adder is independent of the operand word-length, whereas the carry propagation delay in conventional adders is proportional to the operand word-length.

2.4.2 Digit-Serial Arithmetic

In general, digital arithmetic can be classified into digit-parallel arithmetic and digit-serial arithmetic based on how inputs and outputs are processed. For digit-parallel arithmetic, all digits of inputs are applied in parallel, and all digits of outputs are delivered in the same fashion. In this case, all operands can be transferred in one clock cycle, and correspondingly the results can be sampled in one clock cycle. However, transferring data in parallel requires large input and output (I/O) bandwidth for the module. This is potentially the bottleneck when implementing digit-parallel arithmetic, especially for large signal word-length.

For digit-serial arithmetic, both inputs and outputs are processed and delivered one digit per clock cycle. The major advantage of using digit-serial arithmetic is the reduction of the number of signal lines, which in turn can lead to smaller implementations with better performance and energy efficiency.

Generally there are two types of digit-serial modes. The first one is referred to "Least Significant Digit First" (LSDF) mode, which can be commonly found in conventional arithmetic operations. It is worth noting that in the LSDF arithmetic, there is potentially mismatches of the computation directions between different arithmetic units and operations. For instance, the computation results of adders and multipliers are generated from the least significant digit (LSD) to the most significant digit (MSD) due to carry propagation. In comparison, computations such as division and square root, or applications such as analogue to digital (A/D) or digital to analogue (D/A) converters will generate results from MSD to LSD. This inconsistency in computing directions can result in large latency when propagating data among different operations.

The second type of digit-serial mode is "Most Significant Digit First (MSDF)" mode, which is also known as "Online Arithmetic". Using online arithmetic, both the inputs and outputs are processed in a MSD-first manner, enabling parallelism among multiple operations. Therefore the overall latency can be greatly reduced.

2.4.3 Theoretical Background of Online Arithmetic

In order to achieve MSDF mode of operation, two requirements should be satisfied. Firstly, redundant number representation should be utilised to eliminate carry propagation, because the results are generated in the MSDF manner. Secondly, an extra δ input digits are required before generating the first digit of output. If the operator processes one digit per clock cycle, then $(\delta + 1)$ cycles are needed before the MSD of output is delivered, and subsequently the full precision result will be generated in overall $(\delta + 1 + n)$ cycles for n-digit operand word-length. This process is shown in Figure 2.4.

Constant δ is known as the online delay, which can be determined mathematically

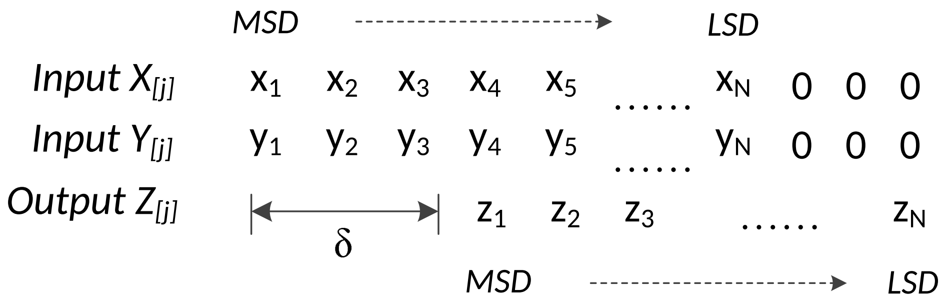


Figure 2.4: Dataflow in digit-serial Online Arithmetic.

Table 2.2: Rated Frequencies of Example Designs.

Operation	δ (radix 2)
$\pm$	2
$\times$	3
$\div$	4
$\sqrt{\ }$	4
Max/Min	0

based on the type of operation, and is independent of the precision of a given operation. For example, the value of δ for a variety of basic operations is summarised in Table 2.2.

In general, an online algorithm operates iteratively. For ease of discussion, assume the circuit input to be normalised to a fixed point number in the range $(-1, 1)$. Based on this premise, the online representation of an N-digit number X is defined in (2.4) where j denotes the iteration number, r denotes the radix, and the value of $X_{[0]}$ is given in (2.5). Notice that the specific methods of determine the value of δ is beyond the scope of this thesis and will not be detailed here.

$$X_{[j]} = X_{[j-1]} + x_{j+\delta} \cdot r^{-\delta-j} \tag{2.4}$$

$$X_{[0]} = \sum_{i=1}^{\delta} x_i r^{-i} \tag{2.5}$$

The digit value x_i belongs to a redundant digit set $\{-a, \cdots, -1, 0, 1, \cdots, a\}$ where $a \in [r/2, r-1]$. a is also known as redundancy factor and it determines the amount of redundancy. Due to redundancy, MSDs of result can be calculated only based on partial information of inputs and then the value of the number can be revised by the following digits, because each number has multiple representations.

For an online algorithm with operands such as X, Y and result Z, the general form of each iteration is given in (2.6) where $W_{[j]}$ denotes the internal vectors required by the algorithm [30].

$$W_{[j]} = f(W_{[j-1]}, X_{[j]}, Y_{[j]}, Z_{[j]}) \tag{2.6}$$

The iteration function $f()$ should use primitive operations such as addition, multiplication by single digit and one digit shift. For each iteration, one digit result is generated based on a selection function $sel()$ with limited precision as given in (2.7), where $\widehat{W}_{[j-1]}$ is a truncated value of $W_{[j-1]}$.

$$z_j = sel(\widehat{W}_{[j-1]}, X_{[j]}, Y_{[j]}) \tag{2.7}$$

In this thesis, the online multiplication algorithm will be used. The algorithm description together with its implementation will be detailed in Chapter 4.

2.4.4 Advantages and Disadvantages of Online Arithmetic

One major advantage of employing online arithmetic is that it enables high performance parallel computations. This is because of several reasons. Firstly, long carry propagation, which is common in conventional arithmetic, is eliminated in online arithmetic, because redundant number systems are used. As discussed previously, online

Table 2.3: Comparison of different forms of arithmetic in terms of time and area complexity.

Arith. Operations.	**Parallel Arith.**		**Online Arith.**		**LSDF Arith.**	
	area	time	area	time	area	time
$\pm$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
$\times$	$\mathcal{O}(n^2)$	$\mathcal{O}(log\ n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
$\div$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	N/A	N/A
$\sqrt{\ }$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	N/A	N/A

delay δ is a constant value for a given arithmetic operation, and it is independent of the precision of the operand word-length. This is of interest particularly for large designs. In comparison, the carry propagation delay in conventional arithmetic is proportional to the precision of operator.

Secondly, the digit-serial nature of online arithmetic offers great potential for overlapping multiple operations to reduce computation latency, and enabling pipelining within operations. This feature compensates the possible speed inefficiency of delivering data serially, and improves the system throughput. In addition, with online arithmetic the interconnection bandwidth can be significantly reduced. Therefore a more area-efficient design can be achieved in comparison to digit-parallel arithmetic.

In Table 2.3, the comparison of digit-parallel arithmetic and digit-serial arithmetic including online arithmetic and LSDF arithmetic is summarised in terms of time and area complexities for basic arithmetic operations [122]. Notice that the complexity values are obtained based on implementation results for each operator and corresponding arithmetic. The N/A in Table 2.3 indicates that certain operations cannot be achieved using LSDF arithmetic.

Another advantage of online arithmetic is that computations can be dynamically terminated based on the required precision. This is because computation results are delivered in an MSD-first manner using online arithmetic. For instance, traditionally

$2N$ digits of results will be generated in a N-digit multiplier, while often only the most significant N digits will be used for successive computations and the N LSDs will be discarded. In contrast, the computation latency can be reduced using online arithmetic, which can be finished immediately once the first N digits of results are obtained. This feature will be further investigated in Chapter 5.

Conversions to conventional representations are not needed if both inputs and outputs are in online representation. This is common when online operators are utilised within a datapath to generate intermediate results. For those cases where conversion is necessary, online representation provides a more efficient conversion approach than redundant parallel representation [28, 32]. This is because for redundant conversion, a carry-propagation adder is required, which limits the overall performance. However, the conversion from online representation can be performed using an on-the-fly method without carry propagation [28]. In terms of resource overhead, the conversion requires two extra registers, the word-lengths of which are equal to the word-length of the inputs to the operator. For the latency overhead, the conversion time equals to two logic levels delay plus a register shift/load time [28].

The major disadvantage of using online arithmetic is the extra area paid by using redundant number representation. We will address this issue in Chapter 5 by proposing an area-efficient implementation of online arithmetic on modern FPGAs.

2.4.5 Current Research

General Purpose Multiplier with Redundant Binary Number System

There has been a significant amount of research into employing redundant number systems for high-speed binary multipliers over the last few decades. This strand of research serves as a great example that when a performance bottleneck is encountered, people turn to explore the possibility of utilising alternative number systems in conventional arithmetic to boost system performance.

Specifically for conventional binary (CB) multiplication, two typical steps will be performed:

(1) generation of partial products based on inputs;

(2) accumulation of partial products and delivery of output.

The speed of multiplication can be improved by reducing the number of partial products or/and by speeding up the generation as well as summation of partial products. In order to accelerate the accumulation of partial products, redundant binary (RB) representation is used as an efficient alternative, because there is no carry-propagation in RB addition. Instead of accumulating partial products in the CB format, an extra step is inserted into the multiplication flow by converting the data from CB to RB. Consequently a RB-to-CB converter is needed to reverse the former process when the accumulation is completed. In this case, there are totally three stages for RB multiplication:

(1) generation RB partial products;

(2) summation of RB partial products;

(3) RB-to-NB conversion and deliver results.

It is worth noting that the final stage is added so that the results can be directly utilised by successive operations, which are normally designed based on the CB number system [52].

The first attempt of implementing RB for fast multiplication was presented by Takagi *et al.* [117, 49]. In their work, the aforementioned 3-stage process was proposed, and the accumulation of partial products was performed based on the RB adder tree. It was demonstrated that the time for N-digit multiplication was $\mathcal{O}(\log_2 N)$.

A modified RB adder and conversion circuit were presented by Makino *et al.* [81]. The modification was based on observation that the conversion circuits (both CB-to-RB and RB-to-CB) limited the operating speed, because carry-propagate was required in this process. However, the major drawback of this work is that an additional vector was introduced during conversion, which increased delay and area overhead. Moreover it placed extra complexity on layout. But this work still lays the foundation for numerous future works.

To-date, current work in this area mainly focus on several aspects such as improving the conversion speed from RB to CB [51, 53], exploring novel encoding metric to reduce the number of partial products [58], and more efficient architecture for redundant binary addition [64, 67].

Online Arithmetic Operators and Applications

There has been a significant amount of research focusing on developing arithmetic operators using online arithmetic, and utilising online arithmetic in a variety of applications.

Addition serves as a key building block for other arithmetic operations. Besides the standard implementation of online addition based on redundant addition as discussed before, alternative structures for online adders [57, 76] were proposed to simplify design complexity in terms of the number of transistors within an adder cell. Correspondingly the performance can be increased in comparison to previous designs. In addition, a binary online adder that took multiple operands with online representation was developed [128, 129]. This structure was designed to cope with streaming or vector inputs, which were common in numerous DSP applications. This work was extended by either taking binary multioperand with multiple data representations, including SD and two's complement [87], or taking decimal multioperand with online representation [43, 127].

Online arithmetic is also implemented with other arithmetic operations, such as multiplication [32, 119, 124], division [33, 85, 120], multiply-accumulation [90], polynomial approximation [10] and Coordinate Rotation DIgital Computer (CORDIC) [48, 116]. In addition, libraries of online arithmetic operators were also proposed [9, 74].

Online arithmetic has been widely used to implement recursive filtering algorithms, digital filters [12, 31, 38] such as Finite Impulse Response (FIR) filter and Infinite Impulse Response (IIR) filter, Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) [74, 91, 83], Discrete Cosine Transform (DCT) [74, 73] and so on. Since online arithmetic performs computations in a MSD first manner, consistent order of data consumption and production can be achieved among multiple operations. This leads to area efficient designs because the extra hardware that is used to reform data order can be removed, which is common with conventional arithmetic. In addition, multiple computations can be overlapped and pipelined at the digit level to increase system frequency and throughput.

Online arithmetic can also be used to control the computation precision dynamically to produce both performance and area benefits. Research has shown exemplary designs in Wideband Code Division Multiple Access (WCDMA) communication system [102]. In comparison to conventional arithmetic which delivers the MSD at the end of computations, online arithmetic computes in MSD-first manner, thus providing a natural way of truncating the computation results without introducing severe output errors. Research has also shown that in detection algorithms [103], computation can be dynamically terminated when the first non-zero MSD (sign) is calculated, instead of waiting for the end of entire computation. This potentially leads to efficient implementation of finite precision embedded system with power and performance advantages.

FPGA Implementation of Online Arithmetic

Research effort has also given into implementing online arithmetic using FPGAs at operator level and algorithm level. For example at operator level, lots of studies focus on efficient FPGA implementation of both digit-serial and digit-parallel online adders [15, 55, 63, 99]. Since FPGAs are normally optimised for conventional arithmetic, these works attempt to map the online adder onto sophisticated FPGA resources, such as the fast carry logic of Xilinx FPGAs, in order to compensate the area overhead brought by using redundant number system. There is also work developing multioperand adders on FPGAs based on compressor trees [56, 82], redefining the arithmetic structure within FPGAs for efficient operation of online arithmetic [13, 97], and implementation of floating point online adders on FPGAs [70]. In addition, FPGA-based online arithmetic operator libraries were proposed targeting signal processing applications [74, 42]. The operator library could be incorporated into conventional FPGA development flow, and could be used as an alternative to existing arithmetic operators.

At algorithm and application level, existing studies mainly focus on DSP applications [41, 73, 83, 88] and control algorithms [26]. In addition, a modified FPGA structure for low power online arithmetic operation was developed [122]. This highly-specific architecture combined the advantages brought by online arithmetic as highlighted before, and could help to reduce the time for logic mapping, placing and routing.

2.4.6 Summary

In this section, we introduced online arithmetic in terms of its basic concepts and theoretical background. Online arithmetic is designed to perform computations in digit serial with MSD first, which is achieved by using a redundant number system. We then discussed two types of digit serial computations, LSD first and MSD first, and highlighted the advantages and disadvantages of both types. Finally we reviewed current research in this area in three categories: the first one is designing general purpose multipliers with a redundant number system to accelerate the summation of partial products. This research area shows the potential performance benefits of using redundant representation. The following two categories focused on designing online arithmetic primitives and applications, and FPGA implementation of online arithmetic modules.

From current literature, it can be seen that the MSD-first nature of online arithmetic offers significant potential to build a high performance embedded system. Although it was originally designed for digit serial operation, online arithmetic can be implemented in a unrolled digit parallel manner to trade area for better performance. In this case, we suggest that if timing violation happens, timing errors will initially affect LSDs of results, which will potentially lead to significant reduction in error magnitude. This hypothesis will be addressed in detail in Chapter 4.

We also notice a major drawback of implementing online arithmetic is that it requires large area overhead due to the utilisation of a redundant number system. This issue is severe especially for FPGA designs, because current FPGA structures from both main vendors are optimised for conventional arithmetic operations. Although we see work investigating area efficient FPGA implementation of online primitives, none of these were targeting modern FPGAs. This issue will be the main focus of Chapter 5.

Chapter 3

Overclocked Datapath with Conventional Arithmetic

3.1 Introduction

As stated in Section 2.2 of the previous chapter, we have seen the potential for achieving performance benefits by optimising the precision throughout datapath to meet a desired accuracy criterion. This idea stems from the fact that in general it is rarely possible to create a completely error-free design, since any fixed or floating point representation introduces quantisation errors into an algorithm. We also note that the choice of precision is not the only source of error in a datapath, and parallel streams of research reviewed in Section 2.3 in Chapter 2 have explored the approximate computing techniques to trade accuracy for performance or energy efficiency.

In this chapter, we describe an alternative circuit design methodology when considering the trade-offs between accuracy, performance and silicon area. In general we examine two design scenarios: one is the conventional design scenario where precision of the datapath is truncated to ensure timing closure; the other is the overclocking scenario where the original datapath word-length is maintained, but the circuit is operated with frequencies higher than rated values. We initially present probabilistic models of errors generated in this process for two different adder structures: the Ripple Carry Adder (RCA) and the Carry-Select Adder (CSA), as discussed in Section 3.2 and Section 3.3, respectively. The decision of the optimum adder structure under the trade-offs between latency, accuracy and silicon area, is then analysed in Section 3.4. This is followed by a description of the probabilistic error models for another arithmetic operator, the Constant Coefficient Multiplier (CCM), in Section 3.5. Section 3.7 discusses a practical experimental setup on an FPGA to verify our models and test the proposed design methodology. We then demonstrate the benefits of our proposed approach in Section 3.6 and Section 3.8.

A summary of the main contributions of this chapter are listed below:

- Detailed modelling methods for probabilistic errors generated in basic arithmetic primitives from two design scenarios to trade accuracy for performance.
- Exploring the accuracy-performance-area trade-offs for different implementations of adders to identify the optimum architecture for a certain arithmetic function under a given requirement of silicon area.
- Demonstration of the proposed findings analytically and experimentally from FPGA implementations that allowing rare timing violations to happen leads to a reduction of average errors or an improvement of performance over truncating the datapath precision to meet timing.

3.2 Ripple Carry Adder

Adders serve as a key building block for arithmetic operations, and other major arithmetic operators such as multipliers and dividers can be implemented using adders. In general, the ripple carry adder is the most straightforward and widely used adder structure. As such, the philosophy of our approach is first exemplified with the analysis of an RCA. In this section we discuss the modelling method for both truncation error and overclocking error in an RCA. We later describe how this methodology can be extended to other arithmetic operators such as CSAs and CCMs, which are also commonly used in DSP applications and numerical algorithms, in Section 3.3 and Section 3.5 respectively.

3.2.1 Adder Structures in FPGAs

An n-bit RCA is composed of n serial connected Full Adders (FAs). Typically the maximum frequency of a RCA is determined by the longest carry propagation delay. Consequently, modern FPGAs offer built-in architectures for very fast ripple carry addition. For instance, the Altera Cyclone series uses fast tables [3] while the Xilinx Virtex series employs dedicated multiplexers and encoders for the fast carry logic [135]. Figure 3.1 illustrates the structure of an n-bit RCA in the Xilinx Virtex-6 FPGA by utilising the built-in fast carry logic.

While the fast carry logic reduces the time of each individual carry propagation delay, the overall delay of carry propagation will eventually overwhelm the delay of sum generation of each LUT with increasing operand word-lengths. We perform our initial analysis based on the assumption that the carry propagation delay of each FA is a constant value μ, which is a combination of logic delay and routing delay. Hence the critical path delay of the RCA is given by (3.1), as shown in Figure 3.1.

$$\mu_{RCA} = n \cdot \mu \tag{3.1}$$

For an n-bit RCA, it follows that if the sampling period T_S is greater than μ_{RCA},

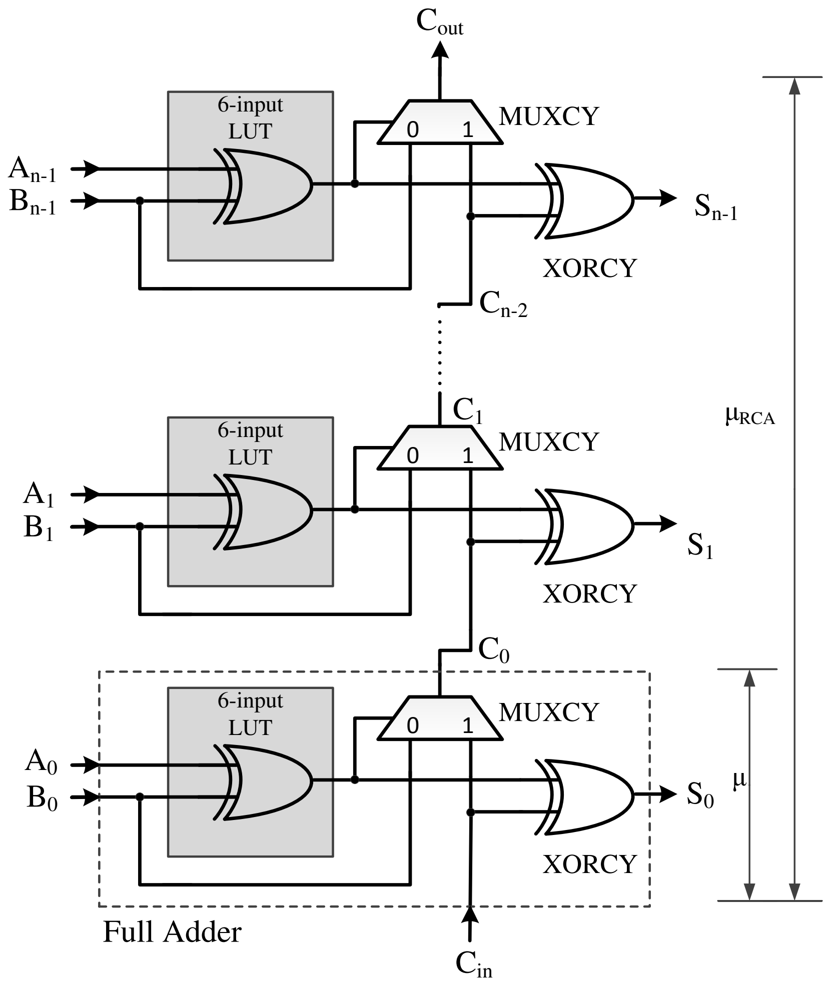


Figure 3.1: An n-bit ripple carry adder in Xilinx Virtex-6 FPGA.

correct results will be sampled. If, however, faster-than-rated sampling frequencies are applied such that $T_S < \mu_{RCA}$, intermediate results will be sampled, potentially generating errors.

In the following sections, we consider two methods that would allow the circuit to run at a frequency higher than $1/\mu_{RCA}$. The first is a traditional circuit design approach where operations are executed without timing violations. To this end, the operand word-length is truncated in order to meet the timing requirement. As a consequence, this process would result in truncation or roundoff error. In the second method, we propose a new design scenario in which circuits are implemented with greater word-length, but are clocked beyond the safe region so that timing violations sometimes

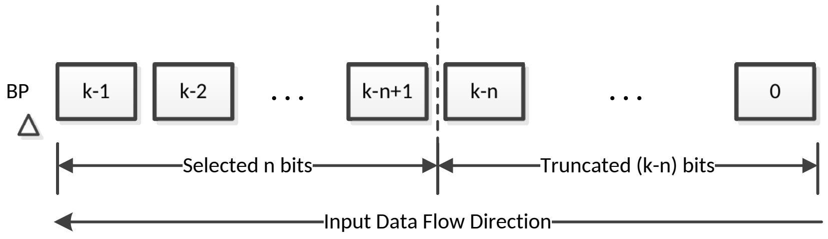


Figure 3.2: Generation of Truncation Error.

occur. Errors may also be generated during this process, and they are referred to as “overclocking error”.

3.2.2 Probabilistic Model of Truncation Error

For ease of discussion, we assume that the input to our circuit is a fixed point number scaled to lie in the range $[-1, 1)$. In our initial analysis, we assume every bit of each input is uniformly and independently generated. However, this assumption will be relaxed in Section 3.6 and Section 3.8 where the predictions are verified using real data from several image benchmarks. In our analysis, the errors at the output are evaluated in terms of both the absolute value and the probability of their occurrence. These two metrics are combined as the error expectation.

If the input signal of a circuit contains k bits, truncation error occurs when the input signal is truncated from k bits to n bits, as shown in Figure 3.2, where BP denotes the binary point. Under our previous assumption that all bits are independently generated following uniform distribution, the mean value of the truncated bits at signal input (E_{Tin}) is given by (3.2).

$$E_{Tin} = \frac{1}{2}\sum_{i=n+1}^{k} 2^{-i} = 2^{-n-1} - 2^{-k-1} \tag{3.2}$$

Since there are two mutually independent inputs to the RCA, the overall expectation of truncation error for the RCA is given by (3.3).

$$E_T = \begin{cases} 2^{-n} - 2^{-k}, & \text{if } n < k \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

3.2.3 Generation of Overclocking Error

For an n-bit RCA, in order to avoid timing violations it is always ensured the sampling period T_S is greater than μ_{RCA}. In this case, correct results will be sampled. In the overclocking scenario we have $T_S < \mu_{RCA}$, hence intermediate results might be sampled and this potentially generates overclocking error. For instance consider a 4-bit addition $0.1111 + 0.0001$: a carry is initially generated at the least significant bit (LSB), it will be propagated to the most significant end and finally annihilated at the most significant bit (MSB). This process is shown in Figure 3.3. We see that it takes 4μ before the final result is stabilised. For this example, if $T_S \geq 4\mu$, the correct result will be sampled, as illustrated by the solid line of T_S. However, if $T_S < 4\mu$, as indicated by the dotted line of T_S, the intermediate result 0 will be sampled and thus an error value of 1 is generated.

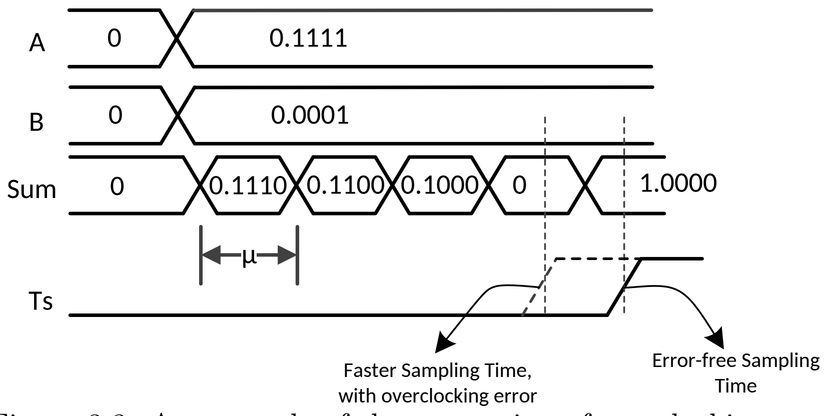


Figure 3.3: An example of the generation of overclocking error.

If T_S is given, we define the maximum length of error-free carry propagation by (3.4) where f_S denotes the sampling frequency. For instance, if $T_S = 3.5\mu$, according to (3.4) we have $b = 4$ which means the maximum length of error-free carry propagation under this situation is 4-bit. In other words, timing violation will happen on any carry chain with length greater than 4 given $T_S = 3.5\mu$.

$$b := \left\lceil \frac{T_S}{\mu} \right\rceil = \left\lceil \frac{1}{\mu \cdot f_S} \right\rceil \tag{3.4}$$

However, since the length of an actual carry chain during execution is dependent upon input patterns, in general, the worst case may occur rarely. As a result, in order to determine when this timing constraint is not met, and the size of the error in this case, we make use of standard results [101] which decide carry generation, propagation and annihilation at bit i based on the inputs A_i and B_i as below. The corresponding summation results S_i can also be calculated by (3.5) where $\oplus$ denotes the exclusive or operation.

$$S_i = A_i \oplus B_i \oplus C_{i-1} \tag{3.5}$$

1. Carry generation: if $A_i = B_i = 1$, then a new carry chain is generated at bit i, and we have $S_i = C_{i-1}$;
2. Carry propagation: given a carry is generated from previous bit, *i.e.* $C_{i-1} = 1$, and if $A_i \neq B_i$, then the carry propagates for this carry chain at bit i, and we have $S_i = \overline{C_{i-1}} = 0$;
3. Carry annihilation: if $A_i = B_i$ (either 0 or 1), the current carry chain annihilates at bit i, and we have $S_i = C_{i-1} = 1$.

3.2.4 Absolute Value of Overclocking Error

For an n-bit RCA, let C_{tm} denote the carry chain generated at bit t with the length of m bits. For a certain f_S, the maximum length of error-free carry propagation, b, is determined through (3.4). The presence of overclocking error requires $m > b$ and $t \leq n - b$. In addition, the length of any carry chain cannot be greater than n. To sum up, we obtain the bounds of parameters t and m as given by (3.6) and (3.7):

$$0 \leq t \leq n - b \tag{3.6}$$

$$b < m \leq n + 1 - t \tag{3.7}$$

For C_{tm}, correct results will be generated from bit t to bit $t + b - 1$, whereas the potential overclocking errors may generate from bit $t + b$ to bit n. Therefore the absolute value of error seen at the RCA output, normalised to the MSB (2^n), is given by (3.8), where $\widehat{S}_i$ and S_i denote the actual and error-free output of bit i respectively.

$$e_{tm} = \frac{\left|\sum_{i=t+b}^{n} (\widehat{S}_i - S_i) \cdot 2^i\right|}{2^n} \tag{3.8}$$

The value of S_i and $\widehat{S}_i$ can be determined based on the previous statements that determine carry generation, propagation and annihilation in Section 3.2.3. More specifically, in the error-free case, the carry will propagate correctly from bit t to bit $t + m - 1$, and we will obtain $S_{t+b} = S_{t+b+1} = \cdots = S_{t+m-2} = 0$ for carry propagation, and $S_{t+m-1} = 1$ for carry annihilation. However, when a timing violation occurs, the carry will not propagate through all these bits. Hence we have the actual outputs $\widehat{S}_{t+b} = \widehat{S}_{t+b-1} = \cdots = \widehat{S}_{t+m-2} = 1$ for carry propagation, and $\widehat{S}_{t+m-1} = 0$ for carry annihilation. Substituting these values into (3.8) we have:

$$\begin{aligned} e_{tm} &= \frac{\left|2^{t+m-1} - 2^{t+m-2} - \cdots - 2^{t+b}\right|}{2^n} = \frac{2^{t+m} - \sum_{i=t+b}^{t+m-1} 2^i}{2^n} \\ &= \frac{2^{t+m} - 2^{t+b}(2^{m-b} - 1)}{2^n} \end{aligned}$$

And this finally yields (3.9). Interestingly as seen in (3.9), the magnitude of overclocking error has no dependence on the length of carry chain m.

$$e_{tm} = 2^{t+b-n} \tag{3.9}$$

3.2.5 Probability of Overclocking Error

The carry chain C_{tm} occurs when there is a carry generating at bit t, a carry annihilates at bit $t + m - 1$ and carry propagates in between. Consequently, its probability P_{tm} is given by (3.10).

$$P_{tm} = P_{(A_t=B_t=1,\ A_i \neq B_i,\ A_{t+m-1}=B_{t+m-1})},\ i \in [t+1, t+m-1) \tag{3.10}$$

Under the assumption that both inputs A and B are mutually independent and uniformly distributed, (3.10) can be modified as given in (3.11). Also we have $P_{(A_i=B_i=1)} = 1/4$, $P_{(A_i \neq B_i)} = 1/2$ and $P_{(A_i=B_i)} = 1/2$. By substituting these values into (3.11) we obtain the expression of P_{tm} in (3.12). Note that (3.12) takes into account the carry annihilation always occurs when $t + m - 1 = n$.

$$P_{tm} = P_{(A_t=B_t=1)} P_{(A_{t+m-1}=B_{t+m-1})} \cdot \prod_{i=t+1}^{t+m-2} P_{(A_i \neq B_i)} \tag{3.11}$$

$$P_{tm} = \begin{cases} (1/2)^{m+1} & \text{if } t+m-1 < n \\ (1/2)^{m} & \text{if } t+m-1 = n \end{cases} \tag{3.12}$$

In addition, the probabilistic models for the magnitude and probability of overclocking errors can be easily generalised for any input distributions with mutually independent bits, since (3.8) and (3.9) are not related to the input distribution, while (3.11) can be employed as long as the value of $P_{(A_i=B_i=1)}$ and $P_{(A_i \neq B_i)}$ is known. If there is dependency between input bits, (3.10) can still be utilised whereas the value of probability $P_{(A_i=B_i=1)}$ and $P_{(A_i \neq B_i)}$ may change. For instance, if the dependency of inputs A_i and B_i is unknown, we have $P_{(A_i=1)} = E_{(A_i)} = 1/2$ and $P_{(B_i=1)} = E_{(B_i)} = 1/2$ separately for uniform distribution, where $E_{(A_i)}$ and $E_{(B_i)}$ denote the expectation of A_i and B_i. The probability of $A_i = B_i = 1$ is then given by (3.13) [104] where $\sigma_{(A_i)}$ denotes the standard variation of A_i and ρ_i denotes the temporal correlation between the two inputs at the i^{th} bit. $\rho_i = 0$ corresponds to the case where bits are uncorrelated.

$$P_{(A_i=B_i=1)} = E_{(A_i)}E_{(B_i)} + \rho_i \sigma_{(A_i)} \sigma_{(B_i)} \tag{3.13}$$

It can be seen that if $\rho_i = 0$ then $P_{(A_i=B_i=1)} = 1/4$ as used to derive (3.12). Otherwise ρ_i should be determined before employing our models.

3.2.6 Expectation of Overclocking Error

For an n-bit RCA, the expectation of overclocking error can be expressed by (3.14), where t is bounded by (3.6) and m is bounded by (3.7).

$$E_O = \sum_t \sum_m P_{tm} \cdot e_{tm} \tag{3.14}$$

Substituting P_{tm} and e_{tm} from (3.9) and (3.12) respectively into (3.14), under the situation that $b \leq n$ we have:

$$
\begin{aligned}
E_O &= \sum_t \sum_m P_{tm} \cdot e_{tm} \\
&= \sum_{t=0}^{n-b} 2^{t+b-n} \cdot \left(\sum_{m=b+1}^{n-t} (1/2)^{m+1} + (1/2)^{n-t+1} \right) \\
&= \sum_{t=0}^{n-b} 2^{t+b-n} \cdot \left((1/2)^{b+1} - (1/2)^{n-t+1} + (1/2)^{n-t+1} \right) \\
&= \sum_{t=0}^{n-b} 2^{t+b-n} \cdot 2^{-b-1} \\
&= 2^{-b} - 2^{-n-1}
\end{aligned}
$$

In a general case E_O can be obtained by (3.15).

$$
E_O = \begin{cases} 2^{-b} - 2^{-n-1}, & \text{if } b \leq n \\ 0, & \text{otherwise} \end{cases} \tag{3.15}
$$

3.2.7 Comparison between Two Design Scenarios

In traditional scenario, for a given f_S, the word-length of RCA must be truncated to $n = b - 1$ in order to meet the required latency. The expectation of the truncation error in the traditional scenario is then given by (3.16) according to (3.3).

$$
E_{trad} = 2^{-b+1} - 2^{-k} \tag{3.16}
$$

Overclocking errors are allowed to happen in our proposed overclocking design scenario, and the original word-length is maintained. Therefore the word-length of RCA is set to be equal to the input word-length, that is, $n = k$. Hence we obtain (3.17) according to (3.15).

$$E_{new} = 2^{-b} - 2^{-k-1} \tag{3.17}$$

Comparing (3.17) and (3.16), we have (3.18).

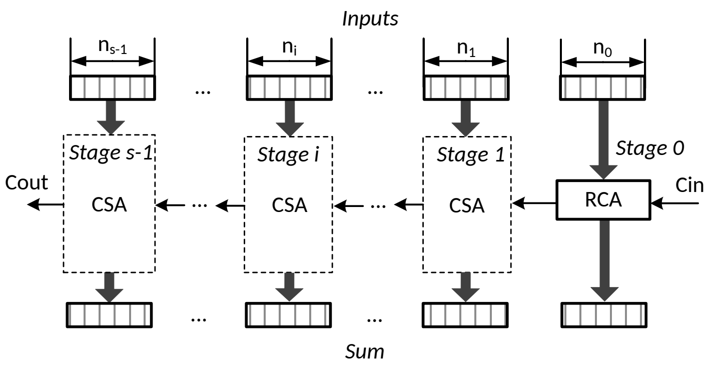
$$\frac{E_{new}}{E_{trad}} = \frac{2^{-b} - 2^{-k-1}}{2^{-b+1} - 2^{-k}} = \frac{1}{2} \tag{3.18}$$

This equation indicates that by allowing timing violations, the overall error expectation of RCA outputs drops by a factor of two in comparison to the traditional scenario. This provides the first hint that our approach is useful in practice.

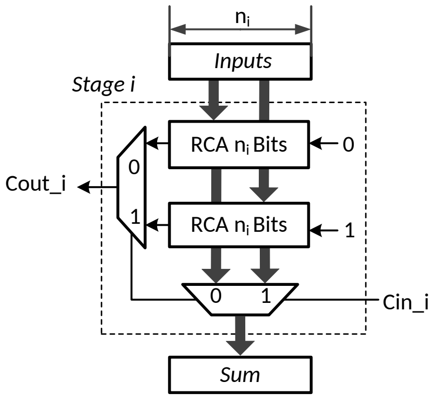
3.3 Carry Select Adder

As seen in (3.18), a 2× reduction in error expectation can be achieved by using the proposed approach in RCA. However, implementing our approach costs extra silicon area in comparison to the conventional approach, because we keep the original word-length k in the overclocking scenario instead of truncating to only $b-1$ bits in the traditional scenario. Besides our approach, there are existing techniques that could also be employed to trade extra silicon area for low latency. For instance, alternative adder architectures, such as carry select adder (CSA), have been proposed to boost performance.

In a CSA, the carry chain is divided into multiple overlapped stages, where each stage contains two RCAs and two multiplexers, as shown in Figure 3.4(a) and Figure 3.4(b), respectively. For a given input, two additions are performed simultaneously within a single stage with carry inputs setting to zero and one separately. One of these two results is then selected according to the actual carry input. Although this structure brings performance benefits, it costs extra hardware resources in comparison to a standard RCA mainly because the carry chain is duplicated, also multiplexers are introduced to select correct results. However in FPGA technology, implementing multiplexers are expensive in terms of area. In order to combine all of the aforementioned design considerations, we explore the trade-offs between silicon area, accuracy and latency of RCA and CSA in this section.



(a) A CSA with s stages.



(b) The i^{th} stage in the CSA.

Figure 3.4: The structure of CSA.

3.3.1 Timing Models for Carry Select Adder

We initially model the CSA in order to understand the relationship between the operating frequency and the maximum word-length of the CSA. This information can then be employed to determine the truncation error based on the models presented in Section 3.2.2.

In our analysis, the stage delay of the i^{th} stage in the CSA refers to the combination of the i-bit carry propagation delay and the delay of multiplexing the carry from the carry input of the i^{th} stage to the carry output of the CSA. For a CSA with s stages ($s \geqslant 2$), let the stage delay be denoted by d_i where $i \in [s-1, 0]$, and d_{s-1} and d_0 represent the delay of the most significant stage and the least significant stage, respectively. We still follow the previous assumption that the critical path delay of the CSA is due to carry propagation and multiplexing the carry output, instead of generating the sum outputs. It should be noted that unlike other stages, the least significant stage of the CSA can be built only by one RCA without multiplexers, since it is directly driven by the carry input. Hence we can obtain the delay of the i^{th} stage as presented in (3.19), where μ_c, μ_{mux} and n_i denote the delay of 1-bit carry propagation, the delay of multiplexing and the word-length of the i^{th} stage of the CSA, respectively.

$$d_i = \begin{cases} n_i \cdot \mu_c + (s-i) \cdot \mu_{mux}, & \text{if } i \in [1, s-1] \\ n_0 \cdot \mu_c + (s-1) \cdot \mu_{mux}, & \text{if } i = 0 \end{cases} \tag{3.19}$$

Under the timing-driven design environment, the delay of each stage of CSA is equalised in order to achieve the fastest operation, as presented in (3.20).

$$d_{s-1} = d_{s-2} = \cdots = d_0 \tag{3.20}$$

In this case, combining (3.19) and (3.20) yields n_i in (3.21), which is represented by the word-length of the most significant stage n_{s-1}.

$$n_i = \begin{cases} n_{s-1} - (s-1-i) \cdot \frac{\mu_{mux}}{\mu_c}, & \text{if } i \in [1, s-1] \\ n_{s-1} - (s-2) \cdot \frac{\mu_{mux}}{\mu_c}, & \text{if } i = 0 \end{cases} \tag{3.21}$$

Sum up n_i to give the total word-length of the CSA in (3.22).

$$n_{CSA} = \sum_{i=0}^{s-1} n_i = s \cdot n_{s-1} - \frac{\mu_{mux}}{\mu_c} \cdot \frac{(s+1)(s-2)}{2} \tag{3.22}$$

Conventionally under a given frequency constraint, the word-length of RCA is truncated by using $n_{RCA} = b - 1$ bits, where b is determined by (3.4). Similarly for CSA, the word-length of each stage should be selected in order to satisfy (3.23).

$$\forall i \in [0, s-1] \quad , \quad d_i \leqslant \tfrac{1}{f_s} \tag{3.23}$$

Therefore for the same frequency requirement, we can form the relationship between the delay of the most significant stage of CSA and the delay of RCA, as given by (3.24),

$$\mu_c \cdot (b-1) = n_{s-1} \cdot \mu_c + \mu_{mux} \tag{3.24}$$

Substitute (3.24) into (3.22) to replace n_{s-1}, we derive the representation of the word-length of CSA in terms of b, as given in (3.25).

$$n_{CSA} = s \cdot (b-1) - \frac{\mu_{mux}}{\mu_c} \cdot \frac{(s+2)(s-1)}{2} \tag{3.25}$$

3.3.2 Accuracy Benefits and Area Overhead in CSA

We first verify the models for the RCA and CSA in terms of the maximum word-length under the given operating frequencies. For the CSA, the ratio μ_{mux}/μ_c can be computed experimentally. We perform post place-and-route simulations on the CSA with two stages using a Xilinx Virtex-6 FPGA. The delay of the i^{th} stage d_i in (3.19) is recorded with respect to different word-lengths. The total word-length of CSA can be predicted through (3.25). In addition, the maximum word-lengths of the 2-stage CSA and RCA are obtained experimentally by increasing n_{CSA} and n_{RCA} respectively until errors are observed at the output. The comparison between the modelled value and the empirical results is illustrated in Figure 3.5. It can be seen that our models for both RCA and CSA match well with the experimental results.

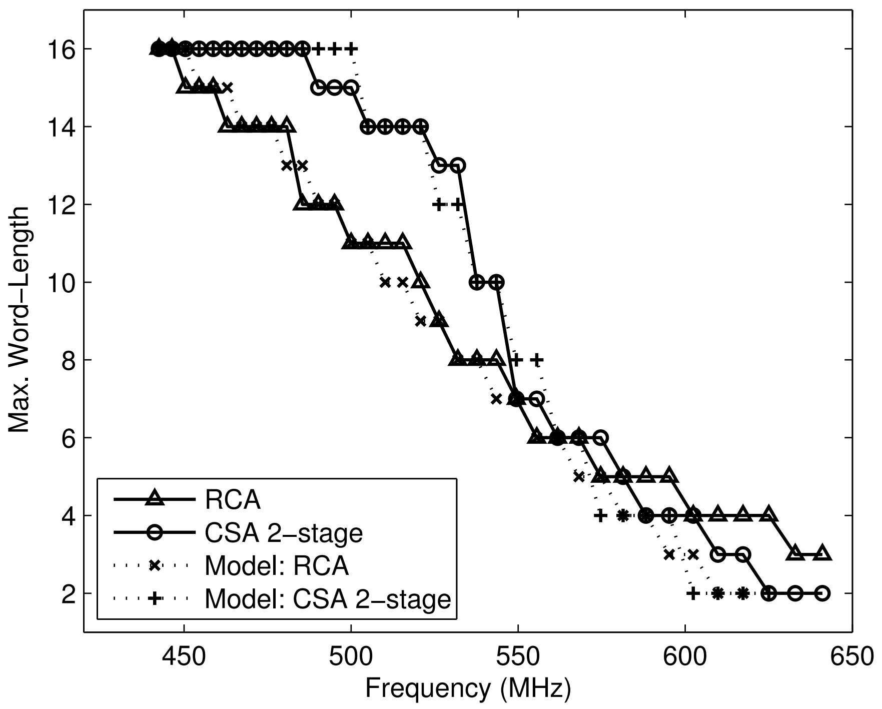


Figure 3.5: The modelled value and experimental results of the maximum word-length of RCA and CSA with respect to various frequency requirements.

Figure 3.5 also highlights that for a relatively relaxed frequency constraint, the CSA achieves greater word-length than the RCA. This is because the stage parallelism in the CSA enables a greater overall word-length, even though the multiplexer delay limits the word-length of each stage in the CSA when compared to the RCA. When more stringent latency requirements are applied, however, the word-length of each stage is largely limited such that the multiplexer delay becomes comparable to the delay of the carry chain, and this inhibits the benefits of parallelism.

However, the accuracy benefits brought by CSA comes at the cost of a large area overhead. Figure 3.6 depicts the number of LUTs in the FPGA used for an RCA, a 2-stage CSA and a 3-stage CSA. It can be seen that in order to meet a given frequency, the 3-stage CSA consumes 2.4× to 3.7× more area than RCA, while the 2-stage CSA

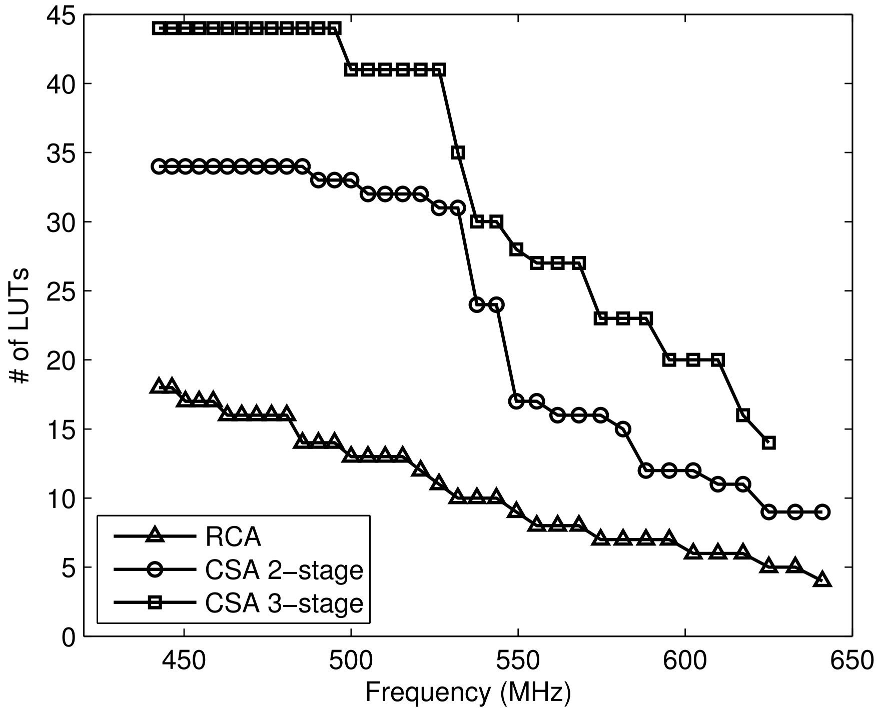


Figure 3.6: Usage of LUTs for an RCA and a CSA with two stages and three stages with respect to various frequency requirements.

requires 1.7× to 3.1× extra area. This finding poses a question of which is the best adder structure for a specific area budget.

3.4 Choosing the Optimum Adder Structure

In Section 3.2, we discussed two design scenarios for the RCA when considering timing constraints. In this section, we expand our analysis by incorporating silicon area as another evaluation metric, and investigate the accuracy, latency and area trade-offs for different adder structures. In the conventional design scenario, the word-length of RCA and CSA is limited by the given frequency constraint, or/and the available hardware resources. The precision loss potentially generates large errors even without timing violations. However in the overclocking design scenario, we use RCA with the maximum possible word-length under a given area budget, whilst timing constraints are allowed to be violated. This process might result in timing errors as well as truncation errors due to area limitation. We compare these two scenarios with different design goals with the aim of finding the optimum design methodology under each situation, instead of ranking of all possible design approaches.

3.4.1 Determination of the Optimum Adder Structure for Given Frequency Requirements

First, suppose an algorithm designer wishes to create a circuit that can run at a given frequency in a given resource constraint while achieving the minimum possible output error. A decision must then be taken over which adder structure achieves this minimum, and whether the structure should be overclocked.

As an example slice through the design space, in Figure 3.7 we record the mean value of error at outputs with respect to different operating frequencies for both design scenarios when the number of the available LUTs is set to 25. The results are obtained from post place-and-route simulations on a Xilinx Virtex-6 FPGA. In this graph, we have labelled the optimum adder structure and the corresponding design scenario. Notice that in this and all the following experiments within this section, in order to apply our models the input data are all randomly generated following independent uniform distributions.

We first notice from Figure 3.7 that for all frequency values, the overclocked RCA achieves no larger error expectation than the RCA with truncated operand word-lengths, as predicted by the models for the RCA in Section 3.2.7. It can also be observed that the CSA cannot be implemented with the original word-length due to the limited area budget and this leads to large truncation errors. Although the CSA with four stages is best for some frequencies, the overclocked RCA is still the optimum

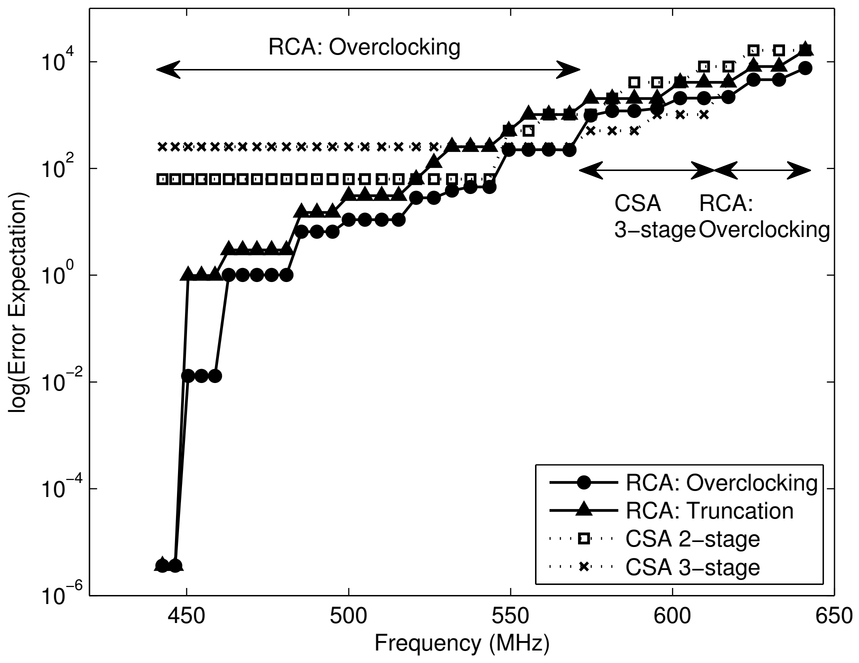


Figure 3.7: A comparison between two design scenarios when the number of available LUTs is 25. The RCA and CSA are investigated in the conventional scenario, while the RCA is explored in the overclocking scenario.

design when high operating frequencies are required.

We then perform similar experiments with a variety of area constraints. The optimum design methods with respect to different operating frequencies and area consumptions are demonstrated in Figure 3.8. From this figure, several observations can be made. Firstly, if the frequency requirement is not large whilst the available area is large enough to implement a CSA in full precision, it will be the optimum design. This is expected from our earlier analysis in Figure 3.5. Secondly, the 2-stage CSA is better than the 3-stage CSA and the 4-stage CSA for low frequency requirements, as it consumes less area although they all achieve the same error expectation. Thirdly, it is possible that only part of the CSA can be implemented under a tighter area budget, whereas the RCA still keeps full precision. In this case, area becomes the dominant factor and precision is lost for the CSA, meaning the RCA with overclocking is the

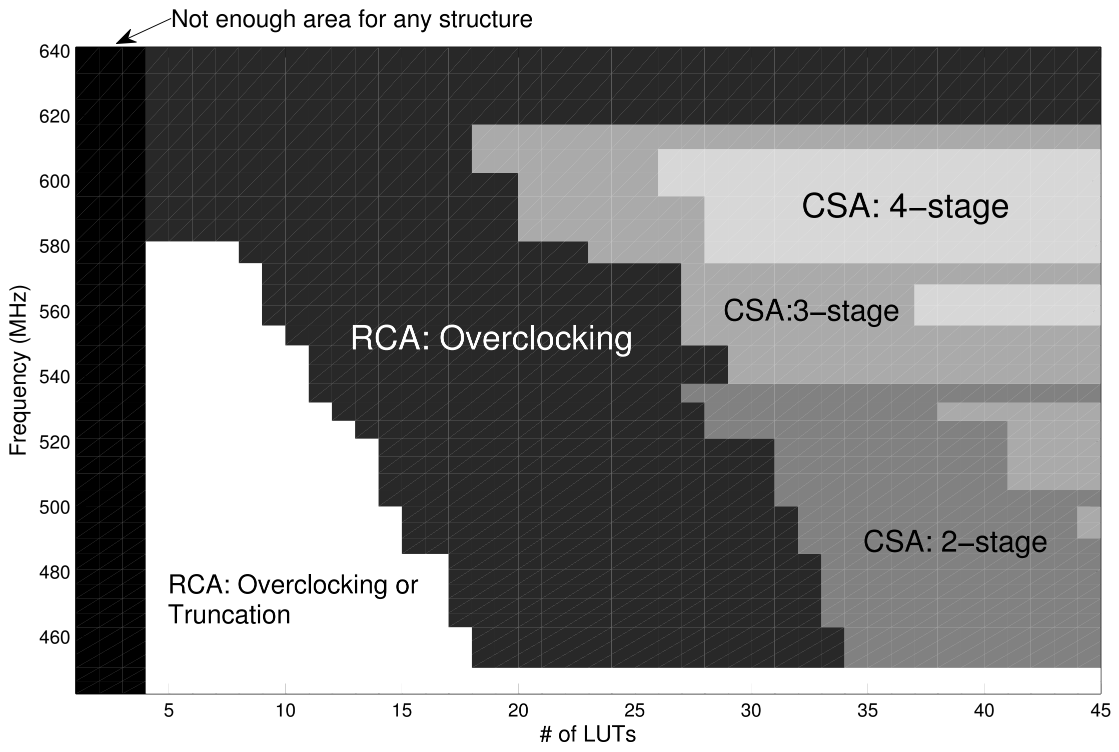


Figure 3.8: A demonstration of the optimum design methodology which achieves the minimum error at outputs with respect to a variety of frequency and area constraints.

optimum design method across almost the whole frequency domain in this situation. Finally, for very stringent area constraints, the word-length of RCA is also limited. This results in truncation errors initially for all design scenarios. However, the RCA with overclocking can still be employed as the optimum design, because it loses less precision than the CSA under the same area constraint.

3.4.2 Determination of the Optimum Adder Structure for Given Accuracy Requirements

If the design goal is to operate the circuit as fast as possible with the minimum area whilst a certain error budget can be tolerated, the optimum design methodology can be decided as illustrated in Figure 3.9. In this situation, the error specifications are evaluated in terms of Mean Relative Error (MRE), as given by (3.26), where

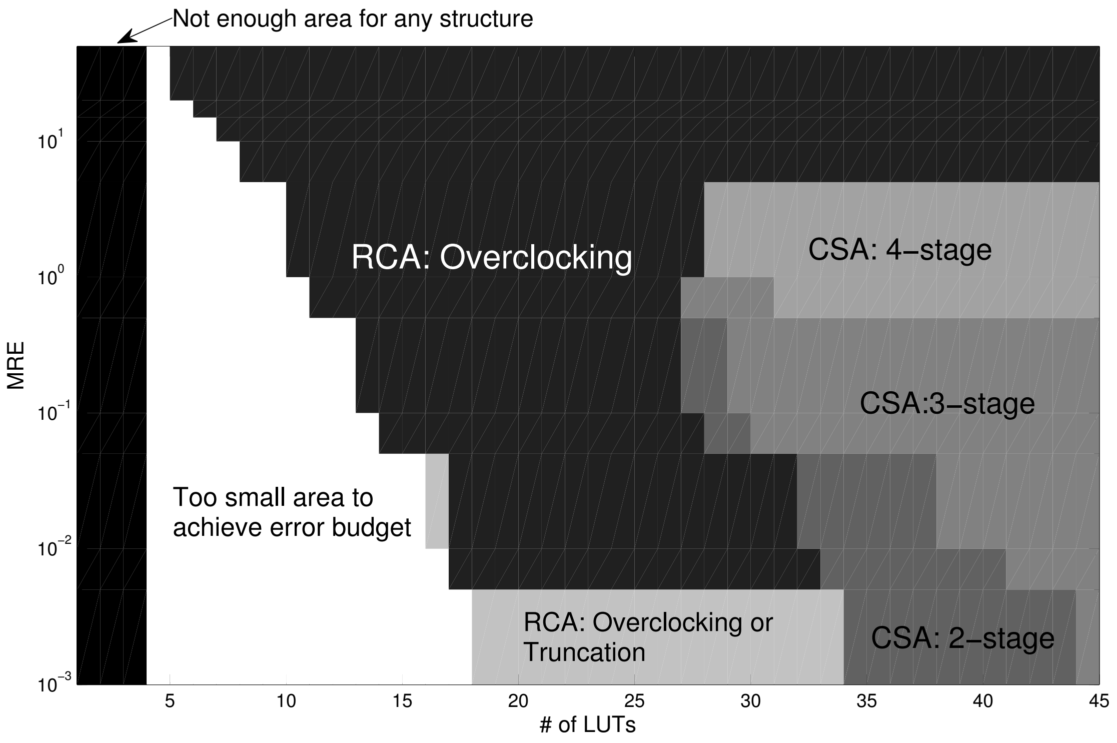


Figure 3.9: A demonstration of the optimum design methodology which runs at the fastest frequency with respect to a variety of accuracy and area constraints.

E_{error} and E_{out} refer to the expected value of error and the expected value of outputs, respectively.

$$MRE = \left| \frac{E_{error}}{E_{out}} \right| \times 100\% \qquad (3.26)$$

In our experiments within this section, MRE is set ranging from 0.001% to 50%. For a given value of MRE, the design with the maximum operating frequency is selected as the optimum design. Moreover, the smallest design is the optimum one if multiple structures operate at the same frequency, with a certain area requirement. Figure 3.9 can thus be obtained based on these criteria.

For a tight accuracy requirement, *e.g.* $MRE < 0.005\%$, CSA serves as the optimum design choice for large area constraints, as it intrinsically operates faster than RCA. Once again, when the area budget shrinks, the RCA performs best because the precision of the CSA is limited. Similarly to the results in Figure 3.8, we see that the overclocked RCA achieves the fastest operating frequencies under most area constraints when the accuracy requirement is released.

3.4.3 Design Guidance

To sum up, our experiments reveal that for both design goals, CSA is not competitive with the overclocking approach unless one has very relaxed area budget. As a result, for the remainder of this chapter, we will focus on the RCA in our following analysis and experiments.

3.5 Constant Coefficient Multiplier

As another key primitive of arithmetic operation, constant coefficient multiplier (CCM) can be implemented using RCA and shifters. For example, a constant coefficient multiplication $B = 9A$ is equivalent to $B = A + 8A = A + (A << 3)$, which can be built using one RCA and one shifter, as demonstrated in Figure 3.10. We first focus on this single RCA and single shifter structure, before describing how more complex structures consisting of multiple RCAs and multiple shifters can be built in accordance with this baseline structure in Section 3.5.5.

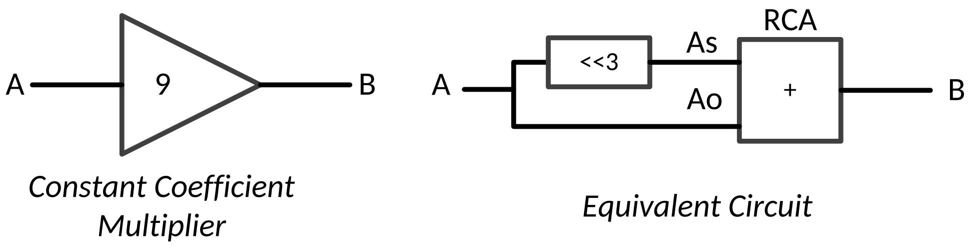


Figure 3.10: Constant coefficient multiplier: symbol and equivalent circuit.

In this CCM structure, let the two inputs of the RCA be denoted by A_S and A_O

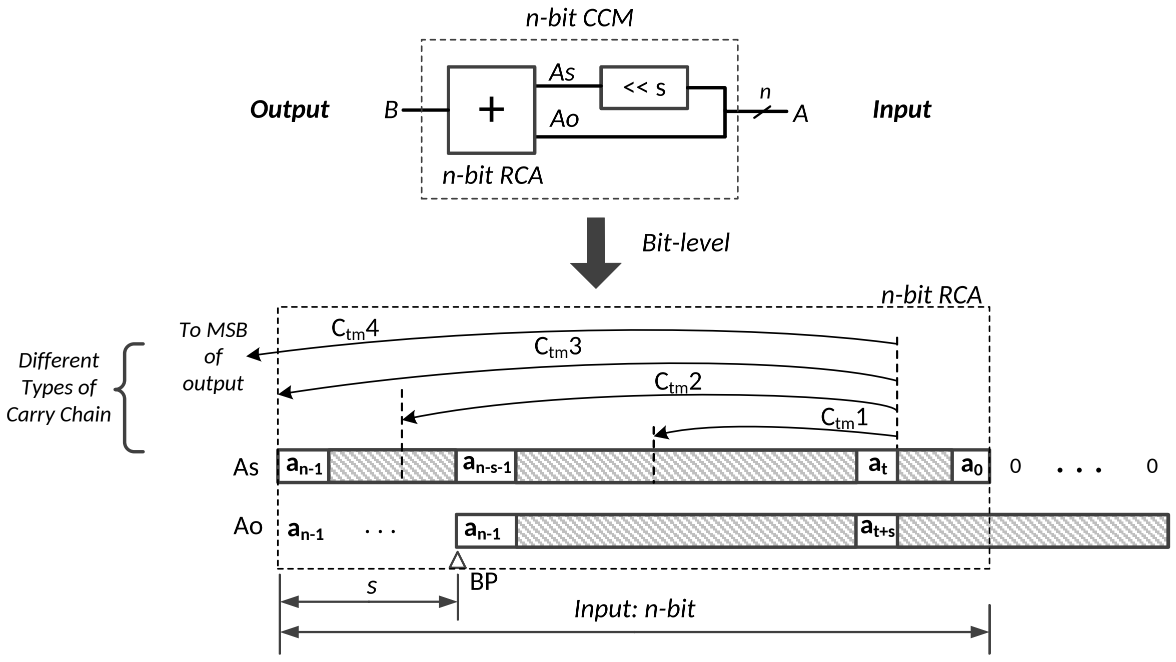


Figure 3.11: Four possible carry chain types in a constant coefficient multiplier with n-bit inputs. The notion s denotes the shifted bits and BP denotes the binary point.

respectively, which are both two's complement numbers. A_S denotes the "shifted signal", with zeros padded after LSB, while A_O denotes the "original signal" with MSB sign extension. For an n-bit input signal, it should be noted that an n-bit RCA is sufficient for this operation, because no carry will be generated or propagated when adding with zeros, as shown in Figure 3.11.

3.5.1 Probabilistic Model of Truncation Error

Let E_{Tin} and E_{Tout} denote the expectation of truncation error at the input and output of the CCM respectively. We then have (3.27), where coe denotes the coefficient value of the CCM, and E_{Tin} can be obtained according to (3.3).

$$E_{Tout} = |coe| \cdot E_{Tin} \tag{3.27}$$

3.5.2 Absolute Value of Overclocking Error

For a CCM, the absolute value of overclocking error of carry chain C_{tm} is increased by a factor of 2^s due to shifting, when compared to RCA. Therefore e_{tm} in CCM can be modified from (3.9) to give (3.28).

$$e_{tm} = 2^{t+b-n+s} \tag{3.28}$$

3.5.3 Probability of Overclocking Error

Due to the dependencies in a CCM, carry generation requires $a_t = a_{t+s} = 1$, while the propagation and annihilation of a carry chain is best considered separately for four types of carry chain generated at bit t. We label these by $C_{tm}1$ to $C_{tm}4$ in Figure 3.11, defined by the end region of the carry chain. For $C_{tm}1$, we have the following statements:

- Carry propagation: $a_i \neq a_{i+s}$, where $i \in [t+1, n-s-2]$;
- Carry annihilation: $a_j = a_{j+s}$, where $j \in [t+1, n-s-1]$.

Similarly for $C_{tm}2$, we have:

- Carry propagation: $a_i \neq a_{n-1}$, where $i \in [n-s-1, n-3]$; or $a_i \neq a_{i+s}$, where $i \in [t+1, n-s-2]$;
- Carry annihilation: $a_j = a_{n-1}$, where $j \in [n-s-1, n-2]$.

For the first two types of carry chain $C_{tm}1$ and $C_{tm}2$, the probability of carry propagation and annihilation is 1/2 and the probability of carry generation is 1/4, under the premise that all bits of input signal are mutually independent. Therefore (3.29) can be obtained by substituting the probability value into (3.11).

$$P_{tm} = (1/2)^{m+1}\,, \quad \text{if } t + m - 1 \leqslant n - 2 \tag{3.29}$$

For carry annihilation of $C_{tm}3$, it requires $a_{n-1} = a_{n-1}$, which is always true. Thus the probability of $C_{tm}3$ is given by (3.30).

$$P_{tm} = (1/2)^{m}\,, \quad \text{if } t + m - 1 = n - 1 \tag{3.30}$$

$C_{tm}4$ represents carry chain annihilates over a_{n-1}, therefore carry propagation requires $a_{n-1} \neq a_{n-1}$. This means $C_{tm}4$ never occurs in a CCM.

Altogether, P_{tm} for a CCM is given by (3.31).

$$P_{tm} = \begin{cases} (1/2)^{m+1} & \text{if } t + m - 1 < n - 1 \\ (1/2)^{m} & \text{if } t + m - 1 = n - 1 \end{cases} \tag{3.31}$$

3.5.4 Expectation of Overclocking Error

Since the carry chain of a CCM will not propagate over a_{n-1}, the upper bound of parameter t and m should be modified from (3.6) and (3.7) to give (3.32) and (3.33).

$$0 \leqslant t \leqslant n - b - 1 \tag{3.32}$$

$$b < m \leqslant n - t \tag{3.33}$$

Finally, by substituting (3.31) and (3.28) with modified bounds of t and m into (3.14), we obtain the expectation of overclocking error for a CCM to be given by

(3.34).

$$E_O = \begin{cases} 2^{s-b-1} - 2^{s-n-1}, & \text{if } b \leq n-1 \\ 0, & \text{otherwise} \end{cases} \tag{3.34}$$

3.5.5 CCM with Multiple RCAs and Shifters

In the case where a CCM is composed of two shifters and one RCA, such as operation $B = 20A = (A << 2) + (A << 4)$, let the shifted bits be denoted as s_1 and s_2 respectively. Hence the equivalent s in (3.34) can be obtained through (3.35).

$$s = |s_1 - s_2| \tag{3.35}$$

For those operations such as $B = 37A = (A << 5) + (A << 2) + (A << 1)$, the CCM can be built using a tree structure. Each root node is the baseline CCM and the errors are propagated through an adder tree, of which the error can be determined based on our previous RCA model.

3.6 Case Study: FIR Filter

3.6.1 Experimental Setup and Model Verification

In Section 3.2 and Section 3.5 we made assumptions that the delay is only generated due to the carry propagation whereas the generation of the output costs no delay. In this section, we first verify the proposed design methodology with overclocking according to this assumption by using an FIR filter, as shown in Figure 3.12. Results shown here are obtained from behavioural model simulations where we only consider carry propagation delay within the design. Results from actual FPGA measurements

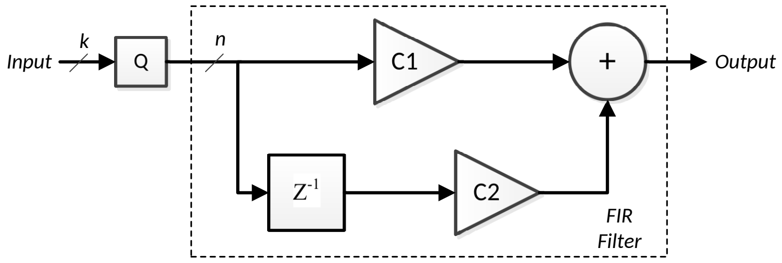


Figure 3.12: A simple FIR filter.

will be described in the next section.

As seen in Figure 3.12, the word-length of the input signal is truncated through the quantiser (Q) in the traditional scenario, in order to achieve the desired latency between input and output. However, in the proposed new scenario, the operating frequency is over-scaled while maintaining the original input word-length. We explore the best trade-off between latency and error based on these two scenarios.

In this experiment, two types of input signal are employed. The first type is 8-bit data which are randomly sampled from a uniform distribution, which we refer to as “uniform independent inputs”. The second type, denoted as “real inputs”, is 8-bit pixel values of several 512×512 image benchmarks.

We first assess the accuracy of our proposed models of error. The amount of input signal truncation for the traditional scenario varies from one bit to seven bits. When the circuit is truncated, it allows the circuit to operate at a higher frequency than the rated frequency, up to $2.75\times$, which corresponds to the maximum frequency required for 1-bit input word-length. Results in Figure 3.13 demonstrate that our models for both overclocking error and truncation error match well with the ideal case. However, we observe a small deviation when using real inputs, since the real data does not exactly follow the uniform distribution or the independent assumption. In Figure 3.13 we only plot results for the “Lena” benchmark image as an example.

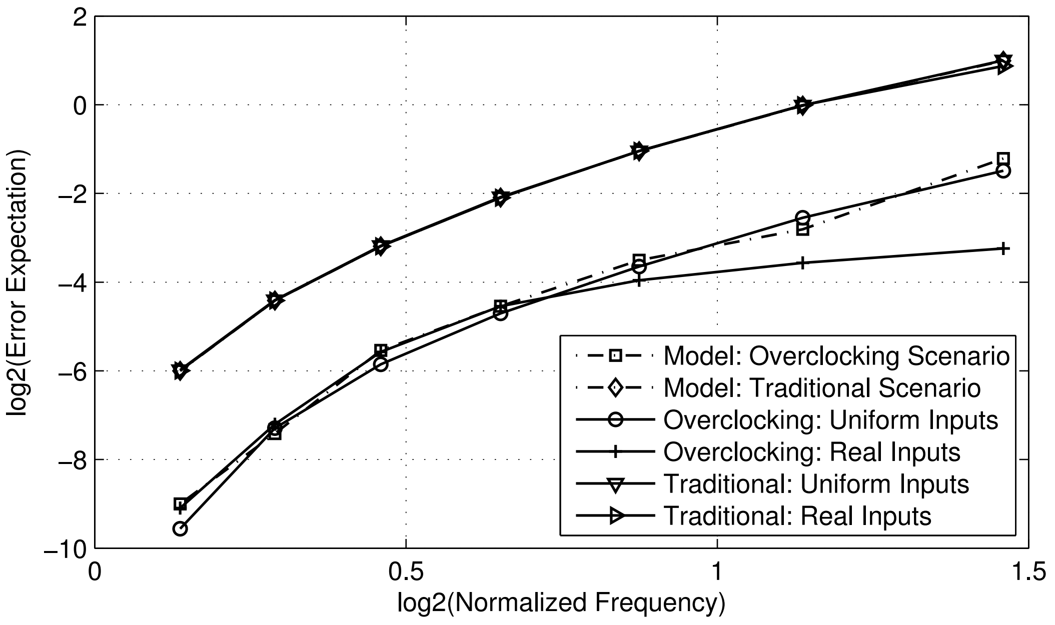


Figure 3.13: Verification of proposed models under timing assumptions.

3.6.2 Error Expectation

Table 3.1 summarises the experimental results obtained by uniform independent inputs together with four benchmark images. First we see that for all input types, error expectation is reduced in the overclocking scenario, as expected by our model, with the geometric mean of reduction varying from 5.6× to 36.7×. In addition, we observe that in practice with real image data as inputs, larger differences of error expectation are achieved. This is because for real data, long carry chains occur with smaller probabilities than those with uniform independent data.

Table 3.1: Ratio of Error Expectation: Traditional Over New.

Inputs	**Normalised Frequency**							**Geo.**
	1.10	1.22	1.38	1.57	1.83	2.20	2.75	**Mean**
Uniform	7.11	4.17	5.06	5.35	5.86	5.92	5.83	5.55
Lena	8.55	6.93	5.20	5.45	7.54	11.74	17.30	8.24
Sailboat	35.31	15.19	11.41	9.70	9.39	10.92	11.69	13.24
Pepper	17.91	10.79	8.84	7.67	8.13	10.39	16.49	10.90
Tiffany	204.5	50.52	28.78	21.94	18.57	22.14	33.69	36.74

3.6.3 Signal-to-Noise Ratio

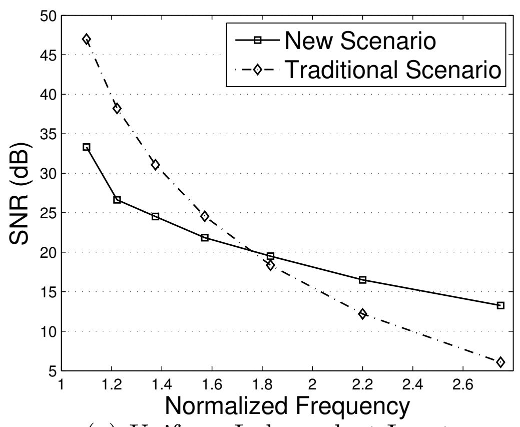
Although Signal-to-Noise Ratio (SNR) is not modelled in our previous analysis, it is an important metric to assess performance especially for DSP applications, and the value of SNR can be obtained through experiments. Figure 3.14 demonstrates SNR for uniform independent inputs and the “Tiffany” image with increasing frequencies. For the former input type, higher SNR is obtained in the traditional scenario where f_s is increased from 1.1× to 1.6× of rated frequency, while the overclocking scenario outperforms when f_S is increased to over 1.8× of rated frequency. This is because SNR is inversely proportional to square of the error magnitude, as specified in (3.36), where A_{signal} and A_{noise} denote the magnitude of output signal and noise, respectively.

$$SNR \propto \left(\frac{A_{signal}^2}{A_{noise}^2} \right) \tag{3.36}$$

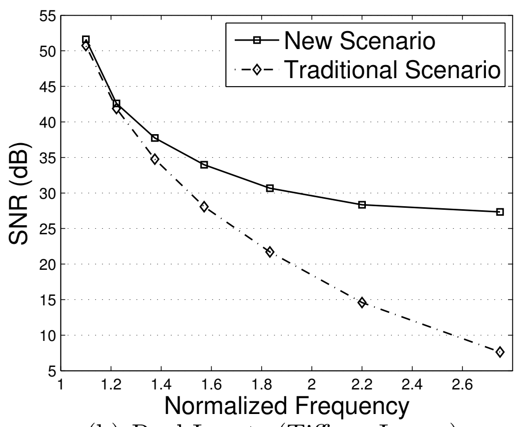
In the traditional scenario, small f_S corresponds to limited truncation of LSBs, which in turn lead to small noise power. However in the overclocking scenario, timing error is generated in the MSBs, therefore large noise power is expected at early stage of overclocking. However, as f_S is further increased, the corresponding amount of truncation rises, resulting in smaller SNR compared to the overclocking scenario.

For inputs with real image data, SNR is higher for all frequencies under the overclocking scenario, since long carry chains occur rarely with real image inputs. This information could be used to achieve better performance in the overclocking scenario. For example, suppose a user only require an SNR of 30dB or less. In this case, one could operate at a frequency of 2.1× over rated frequency under the overclocking scenario but only a 1.5× frequency speed-up can be achieved under the traditional scenario.

Table 3.2 presents the differences of SNR (dB) obtained by uniform inputs and



(a) Uniform Independent Inputs.



(b) Real Inputs (Tiffany Image).

Figure 3.14: Comparison between overclocking scenario and traditional scenario in terms of SNR.

four benchmark images. Similar to error expectation, we also see that the average improvement of SNR is higher for real image data than the uniform data.

Table 3.2: Difference of SNR (dB): New Over Traditional

Inputs	**Normalised Frequency**						
	1.10	1.22	1.38	1.57	1.83	2.20	2.75
Uniform	-13.68	-11.57	-6.56	-2.72	1.14	4.30	7.18
Lena	-12.90	-9.20	-6.78	-2.71	2.83	9.05	14.41
Sailboat	-6.75	-5.11	-2.06	1.04	4.56	8.76	11.33
Pepper	-9.71	-6.99	-3.71	-0.56	3.46	8.29	14.67
Tiffany	0.89	0.70	2.96	5.90	8.97	13.75	19.71

3.7 Test Platform on An FPGA

We also perform experiments on an actual hardware platform. Similarly we compare the traditional scenario and the overclocking scenario and explore the trade-offs between latency and accuracy. The benefits of the proposed methodology are demonstrated over a set of DSP example designs, which are implemented on the Xilinx ML605 board with a Virtex-6 XC6VLX240T-1FFG1156 FPGA.

3.7.1 Experimental Setup

We initially build up an FPGA test framework. The general architecture is depicted in Figure 3.15. The main body of the test framework consists of the circuit under test (CUT), the test frequency generator and the control logic, as shown in the top dotted box in Figure 3.15. The I/Os of the CUT are registered by the launch registers (LRs) and the sample registers (SRs), which are all triggered by the test clock. Input test vectors are stored in the on-chip memory during initialisation. The results are sampled using Xilinx ChipScope. Finally, in the host computer we perform an offline comparison between the error-free data, which are the outputs of the original circuit sampled at the rated frequency, and the outputs of the overclocked as well as the truncated designs using the same input vectors.

The test frequency generator is implemented using two cascaded Mixed-Mode Clock Managers (MMCMs) [134], created using Xilinx Core Generator [137]. The frequency of the MMCM clock output can be dynamically reconfigured during run time [72, 118], such that it is not necessary to re-implement the entire design, which may introduce timing differences between multiple tests. Besides the outputs, the corresponding input vectors and memory addresses are also recorded into the comparator for cross comparison, as can be seen in Figure 3.15, in order to ensure that the recorded errors arise from overclocking the CUT rather than the surrounding circuitry when high test frequencies are applied.

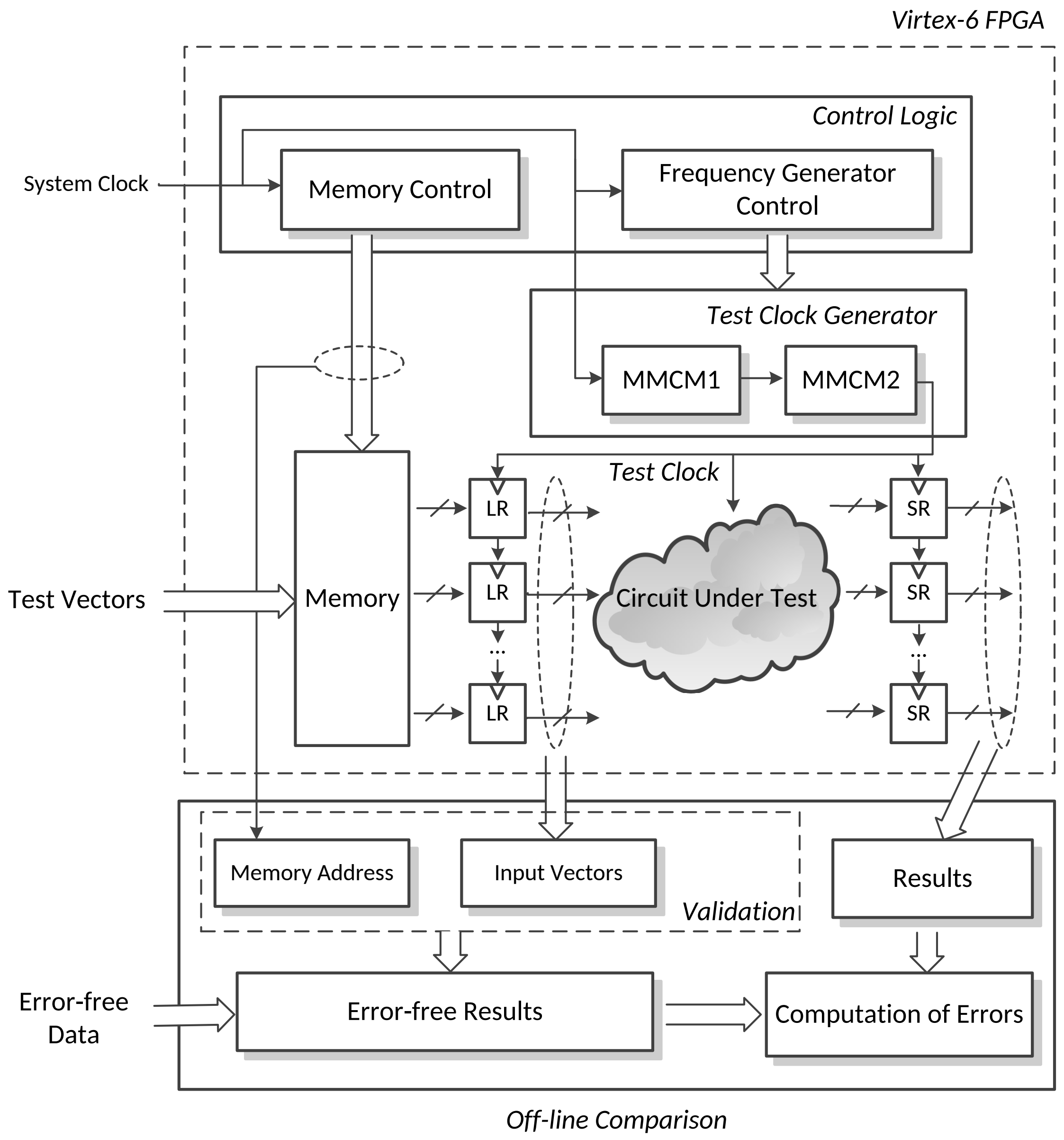


Figure 3.15: Test framework, which is composed of a measurement architecture (the top dotted box) on an FPGA and an off-line comparator using software. The error-free data are obtained by either pre-computation or initial run with rated frequencies.

3.7.2 Benchmark Circuits

Three types of DSP designs are tested: digital filters (FIR, IIR and Butterworth), a Sobel edge detector and a direct implementation of a Discrete Cosine Transformation (DCT). The filter parameters are generated using MATLAB filter design toolbox, as listed below:

- FIR filter (5^{th} order): $0.023, 0.095,\ 0.431,\ 0.431,\ 0.095,\ 0.023$.
- IIR filter (7^{th} order):
  1. Numerator: $0.014, 0.042, 0.082, 0.109, 0.109, 0.081, 0.042, 0.014$.
  2. Denominator: $1.000, 1.802, 2.273, 1.584, 0.805, -0.238, 0.046, -0.003$.
- Butterworth filter (9^{th} order):
  1. Numerator: $0.001, 0.009, 0.038, 0.089, 0.134, 0.134, 0.089, 0.038, 0.009, 0.001$.
  2. Denominator: $1.000, 1.791, 2.531, 2.118, 1.370, 0.609, 0.199, 0.043, 0.005, 0.001$.

Table 3.3 summarises the rated operating frequency of each implemented design reported by Xilinx ISE14.1 when the word-length of input signal is 8-bit. Notice that all designs are implemented using LUTs only without DSPs for fairer comparisons.

Table 3.3: Rated Frequencies of Example Designs.

Design	Frequency (MHz)	Description
FIR Filter	126.2	5^{th} order
Sobel Edge Detector	196.7	3×3
IIR Filter	140.3	7^{th} order
Butterworth Filter	117.1	9^{th} order
DCT	176.7	4-point

Similar to the experiments described in Section 3.6, two sources of the input data are generated in this experiment. One is "uniform independent inputs", which are randomly sampled from a uniform distribution of 8-bit numbers. The other is called "real inputs", which are 8-bit pixel values of the 512×512 Lena image.

3.7.3 Correcting for Conservative Timing Margin

Generally, the operating frequency reported by electronic design automation (EDA) tools tends to be conservative to ensure the correct functionality under a wide range of operating environments and workloads. In a practical situation, this may result in a large gap between the predicted frequency and the actual frequency under which the correct operation is maintained [44].

For example, the frequencies predicted by Xilinx ISE and the actual frequencies of a 5^{th} order FIR filter using different word-lengths are depicted in Figure 3.16. The "actual" maximum frequencies are obtained by increasing the operating frequency from the rated value until errors are observed at the output; the maximum operating frequency with correct output is recorded for the corresponding word-length. As can be seen in Figure 3.16, according to our experiments the circuit can operate without errors at a much higher frequency in practice than predicted by the tool. A maximum speed differential of $3.2\times$ is obtained when the input signal is 5-bit.

In our experiments as will be discussed in Section 3.8, the conservative timing margin will be removed in the traditional scenario for a fairer comparison to the overclocking scenario. To do this, for each truncated word-length, we select the maximum frequency at which we see no overclocking error on the FPGA board in our lab. For example, in Figure 3.16, the operating frequency of the design when the word-lengths are truncated to 8 bits, 5 bits and 2 bits are 400MHz, 450MHz and 500MHz respectively.

Figure 3.16 also demonstrates that when the circuit is truncated, it allows the circuit to operate at a higher frequency than the frequency of full precision implementation. However, we can also see that the maximum operating frequency keeps almost con-

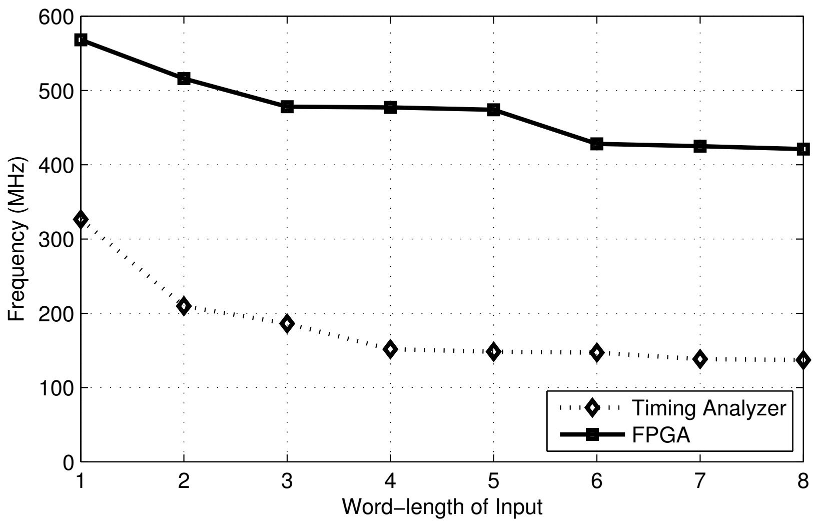


Figure 3.16: The maximum operating frequencies for different input word-lengths of an FIR filter. The dotted line depicts the rated frequency reported by the timing analysis tool from Xilinx ISE 14.1. The solid line is obtained through FPGA tests using our platform.

stant when the operand word-length reduces from 8 to 6 or from 5 to 3 in both the experimental results and those of timing analyser. This will cause a slight deviation between our analytical model which assumes that the single bit carry propagation delay to be a constant value, as discussed in (3.16) with expression $n = b - 1$. This deviation will be influenced by many factors such as how the architecture has been packed onto logic elements on FPGAs such as LUTs and Slices. In addition, process variation might cause non-uniform interconnection delays [132]. Nevertheless, we shall see that our models remain close to the true empirical results in Section 3.8.

3.7.4 Computing Model Parameters

The accuracy of our proposed models is examined with practical results on Virtex-6 FPGA. In order to apply the proposed models, the model parameters should be determined first. In general there are two types of parameters in the models of overclocking error. The first is based on the circuit architecture. For example, the word-length of RCAs and CCMs (n), the shifted bits of the shifters in CCM (s), and the word-length of the input signal (k). This is determined through static analysis. The second depends on timing information, such as the single bit carry propagation delay μ. In order to keep consistency with the assumption made in models that μ is a fixed value, it is obtained according to the actual FPGA measurement results.

In the FPGA tests, initially the maximum error-free frequency f_0 is applied. In this case we have (3.37) where d_c is a constant value which denotes the interconnection delay, and $d_0 = 1/f_0$. The frequency is then increased such that (3.38) is obtained. This process repeats until the maximum frequency f_{n-1} is applied in (3.39). Based on these frequency values, μ can be determined.

$$d_0 = n\mu + d_c \tag{3.37}$$

$$d_1 = (n-1)\mu + d_c \tag{3.38}$$

$$\cdots$$

$$d_{n-1} = \mu + d_c \tag{3.39}$$

3.8 Results from FPGA Measurements

3.8.1 Case study: FIR Filter

In this section, the aforementioned timing assumptions in our models are relaxed by using results from real FPGA measurements. We first assess the accuracy of our proposed models of errors. We record the value of MRE based on (3.26) in a 5^{th} order

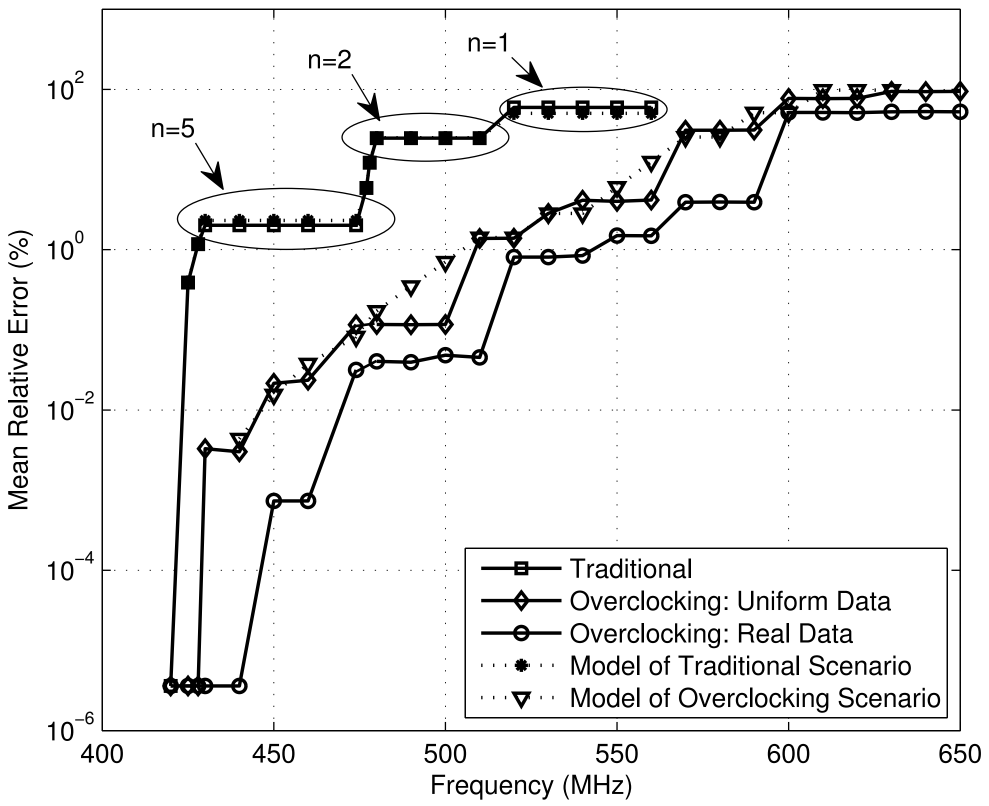
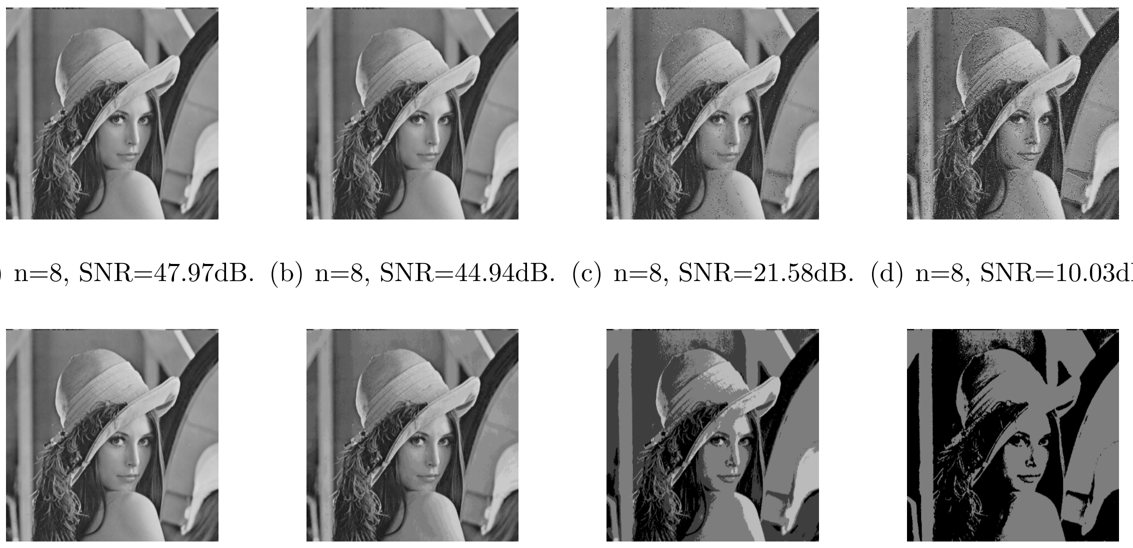


Figure 3.17: A demonstration of two design perspectives with a 5th order FIR filter, which is implemented on Virtex-6 FPGA. The modelled values of both overclocking errors and truncation errors are presented as dotted lines. The actual FPGA measurements are depicted using solid lines. Two types of inputs are employed in the overclocking scenario: the uniformly distributed data and the real image data from Lena.

FIR with respect to different operating frequencies. The modelled values of both overclocking error and truncation error of the FIR filter are presented in Figure 3.17 (dotted lines), as well as the actual measurements on the FPGA (solid lines) with two types of input data. The results demonstrate that our models match well with the practical results obtained using the uniform independent inputs.

According to Figure 3.17, output errors are reduced in the overclocking scenario for both input types in comparison to the traditional scenario, as the analytical model validates. In addition, we see that using real data, more significant reduction of MRE



(a) n=8, SNR=47.97dB. (b) n=8, SNR=44.94dB. (c) n=8, SNR=21.58dB. (d) n=8, SNR=10.03dB.

(e) n=7, SNR=26.02dB. (f) n=5, SNR=21.76dB. (g) n=2, SNR=6.57dB. (h) n=1, SNR=3.95dB.

Figure 3.18: Output images of the FIR filter for both overclocking scenario (top row) and traditional scenario (bottom row) when operating frequencies are 425MHz, 450MHz, 480MHz and 520MHz, respectively (from left to right).

are achieved, and that no errors are observed when frequency is initially increased. This is because for real data, long carry chains are typically generated with even smaller probabilities, and the longest carry chain rarely occurs.

The output images of the FIR filter for both of the two scenarios with increasing frequencies are presented in Figure 3.18, from which we can clearly see the differences between the errors generated in these two scenarios. In the overclocking scenario, we observe errors in the MSBs for certain input patterns. This leads to "salt and pepper noise", as shown on the images in the top row of Figure 3.18. In the traditional scenario, truncation causes an overall degradation of the whole image, as can be seen in the bottom row of Figure 3.18. Furthermore, it is worth noting that recovering from the latter type of error is more difficult, since it is generated due to precision loss.

In addition, we record the probability distribution of different length of carry chains using an 8-bit RCA with both data types, as shown in Figure 3.19. As the input data is

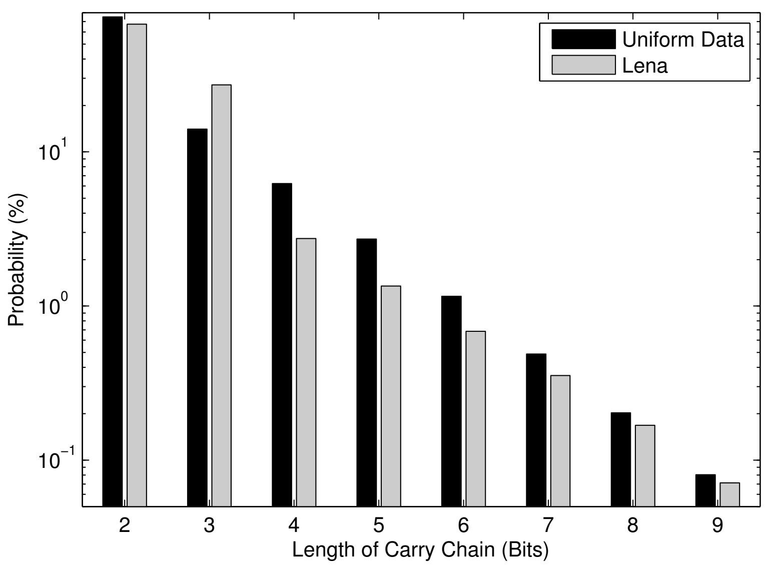


Figure 3.19: Probability distribution of different length of carry chains in a 8-bit RCA. For the uniform data, two inputs of RCA are randomly sampled from a uniform distribution. For the image data Lena, one input of RCA uses the original data and the other uses the delayed original data for several clock cycles.

8-bit, the longest possible carry chain is of length 9-bit. It can be observed that when using the Lena data, the probability is only higher than that using uniform data for carry chain with a length of 3-bit. While in other situations, using uniform data leads to higher probability, especially for long carry chain length. This finding explicitly demonstrates that longer carry chain happens with an even smaller probability when utilising real data.

3.8.2 Potential Benefits in Datapath Design

As we mentioned in Section 3.4, our results could be of interest to a circuit designer in two ways. Typically, either the designer will want to create a circuit that can run at a given frequency with the minimum possible value of MRE, or the algorithm designer will wish to run as fast as possible whilst maintaining a specific error tolerance. For the first design target, the experimental results for all five example designs on FPGA are

Table 3.4: Relative Reduction of MRE (%) in Overclocking Scenario for Various Normalised Frequencies Based on (3.40) for Two Types of Input Data: Uniform Data (Uni) and Real Image Data from Lena (Lena).

Norm.	**FIR**		**Sobel**		**IIR**		**Butterworth**		**DCT4**		**Geo.Mean**	
Freq.	Uni	Lena	Uni	Lena	Uni	Lena	Uni	Lena	Uni	Lena	Uni	Lena
1.04	99.85	100.00	99.51	99.74	72.28	90.09	79.03	100.00	83.58	98.06	86.14	97.50
1.08	98.93	99.97	96.26	93.75	71.64	90.50	78.81	100.00	83.26	98.45	85.15	96.46
1.12	94.27	98.82	96.25	93.62	73.63	88.25	81.88	84.87	89.44	99.56	86.68	92.84
1.16	99.66	99.91	73.73	93.62	73.10	89.92	79.30	84.23	79.07	99.44	80.44	93.23
1.20	96.03	99.90	81.55	81.52	70.76	75.67	64.96	84.50	N/A*	N/A*	77.46	84.95
1.24	98.46	99.32	81.43	81.67	70.47	76.12	63.66	84.23	N/A*	N/A*	77.44	84.92
1.28	95.39	99.29	60.41	78.24	N/A*	N/A*	54.38	75.15	N/A*	N/A*	67.92	83.58
1.32	95.37	98.75	N/A*	N/A*	N/A*	N/A*	N/A*	N/A*	N/A*	N/A*	95.37	98.75

* Current frequency cannot be achieved in the traditional scenario. These points are excluded from the calculation of geometric means.

summarised in Table 3.4 in terms of the relative reduction of MRE as given in (3.40) where MRE_{trd} and MRE_{ovc} denote the value obtained in the traditional scenario and in the overclocking scenario, respectively.

$$\frac{MRE_{trd} - MRE_{ovc}}{MRE_{trd}} \times 100\% \tag{3.40}$$

In this table, the frequency is normalised to the maximum error-free frequency for each design when the input signal is 8-bit. The N/A in Table 3.4 refers to the situations where a certain frequency simply cannot be achieved using the traditional scenario, and these points are excluded from the calculation of geometric means. It can be seen that a significant reduction of MRE can be achieved using the proposed overclocking scenario, and the geometric mean reduction varies from 67.9% to 95.4% using uniform input data. Even larger differences of MRE can be observed when testing with real image data for each design, ranging from 83.6% to 98.8%, as expected given the results shown in Figure 3.17.

For the second design target, higher operating frequencies can be obtained if certain

Table 3.5: Frequency Speedups (%) in Overclocking Scenario Under Various Error Budgets for Two Types of Input Data: Uniform Data (Uni) and Real Image Data from Lena (Lena).

Error	**FIR**		**Sobel**		**IIR**		**Butterworth**		**DCT4**		**Geo.Mean**	
Budget	Uni	Lena	Uni	Lena	Uni	Lena	Uni	Lena	Uni	Lena	Uni	Lena
0.05%	4.76	21.43	6.82	6.82	0.95	1.26	12.40	24.03	0.72	0.96	3.07	5.32
0.5%	19.05	21.43	13.64	6.82	0.63	10.06	24.03	24.03	0.48	12.44	4.52	13.45
1%	19.05	28.57	13.64	18.18	10.06	16.35	24.03	24.03	0.48	12.44	7.86	19.10
5%	19.15	25.53	18.18	18.18	0.54	0.82	0.63	0.94	7.66	12.44	3.91	5.36
10%	19.15	25.53	10.64	10.64	0.54	1.09	6.92	13.21	4.91	4.911	5.19	7.19
20%	19.15	25.53	5.77	15.39	8.70	8.70	3.26	3.26	6.70	10.88	7.32	10.39
50%	15.69	15.69	10.53	19.30	42.50	50.00	46.74	54.89	15.06	19.25	21.8	27.59

error budget can be tolerated. We record the frequency speed-ups for each benchmark circuit in Table 3.5, when the specified error tolerance varies from 0.05% to 50%. For all designs, we see that the overclocking scenario still outperforms the traditional scenario for each MRE budget in terms of operating frequency. Likewise, the frequency speed-up is higher for real image inputs than uniform inputs. The geometric mean of frequency speed-ups of 3.1% to 21.8% can be achieved by using uniform data, while 5.3% to 27.6% when using real image data.

3.8.3 Area Overhead of Our Approach

For both aforementioned design goals, it should be noted that a benefit of using the traditional approach is that reducing datapath precision would potentially lead to a smaller design, in comparison to our proposed overclocking scenario which maintains the original word-length throughout the datapath. Consequently, the area overheads of our approach are summarised in Table 3.6 and Table 3.7 for both design goals. In our experiments, the area overhead is evaluated in terms of the number of LUTs and Slices in the FPGA technology, while similar metrics, *e.g.* the number of logic gates and transistors, can also be employed if applying our approach into other hardware

Table 3.6: Area Overhead of Our Approach with respect to Various Frequency Requirements

Normalised	**FIR**		**Sobel**		**IIR**		**Butterworth**		**DCT4**		**Geo.Mean**	
Frequency	LUT	Slice	LUT	Slice	LUT	Slice	LUT	Slice	LUT	Slice	LUT	Slice
1.04	1.24	1.05	1.12	1.22	1.26	1.42	1.16	1.11	1.13	1.00	1.18	1.15
1.08	1.24	1.05	1.03	1.08	1.26	1.42	1.16	1.11	1.11	1.15	1.16	1.16
1.12	1.24	1.05	1.03	1.08	1.26	1.42	1.37	1.54	2.30	2.17	1.38	1.40
1.16	2.95	2.56	1.03	1.08	1.61	1.71	1.99	2.03	4.60	2.89	2.14	1.95
1.20	2.95	2.56	1.66	1.39	2.77	3.48	3.17	3.50	N/A*	N/A*	2.56	2.57
1.24	4.42	3.15	4.00	3.55	7.87	6.73	6.92	5.25	N/A*	N/A*	5.57	4.46
1.28	4.42	3.15	4.00	3.55	N/A*	N/A*	6.92	5.25	N/A*	N/A*	4.97	3.89
1.32	4.42	3.15	N/A*	N/A*	N/A*	N/A*	N/A*	N/A*	N/A*	N/A*	4.42	3.15

* Current frequency cannot be achieved in the traditional scenario. These points are excluded from the calculation of geometric means.

Table 3.7: Area Overhead of Our Approach with respect to Given Error Budgets.

Error	**FIR**		**Sobel**		**IIR**		**Butterworth**		**DCT4**		**Geo.Mean**	
Budget	LUT	Slice	LUT	Slice	LUT	Slice	LUT	Slice	LUT	Slice	LUT	Slice
0.05%	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
0.5%	1.02	1.15	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.03
1%	1.02	1.15	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.03
5%	1.24	1.05	1.00	1.00	1.26	1.42	1.65	1.11	1.00	1.00	1.13	1.11
10%	1.24	1.05	1.12	1.22	1.26	1.42	1.65	1.11	1.11	1.15	1.18	1.18
20%	1.24	1.05	1.03	1.08	1.26	1.42	6.92	5.25	3.45	2.17	2.07	1.79
50%	2.95	2.56	4.00	3.55	7.87	6.73	6.92	5.25	3.45	2.12	4.67	3.70

platforms. It can be seen that for a given frequency requirement with minimum possible errors, the geometric mean of area overhead ranges from 1.16× and 1.15× to 5.57× and 4.46× for LUTs and Slices, respectively. This result illustrates that although using traditional approach would lead to smaller design, the area benefits are less attractive than the performance or accuracy benefits brought by utilising the our proposed new approach, especially for not stringent frequency requirements.

An even smaller area overhead can be found for a specified error budget with the maximum clock frequencies. As seen in Table 3.6, the geometric mean of area overhead in terms of LUTs and Slices is from 1.00× to 4.67× and from 1.00× to 3.70× respectively. Notice that an area overhead of 1.00× means the original precision is employed in both the overclocking scenario and the traditional scenario. However the former achieves frequency speed-ups as seen in Table 3.7, because a specific error budget can be tolerated and the design can be overclocked.

In general, a very small area overhead of our approach can be observed when the design is initially operated beyond the safe region with higher frequencies or with a small error budget (*e.g.* below 10%). In addition, at even higher operating frequencies or with larger error specifications, our approach achieves significant performance or accuracy improvements at the cost of a relatively small extra silicon area.

3.9 Conclusion and Discussion

In this section, we have explored the probabilistic behaviour of several key arithmetic primitives in an FPGA when operating beyond the deterministic clocking region. Two design scenarios that might generate errors: violating timing constraints as well as reducing precisions, have been investigated. It has been found that simply allowing timing violations to happen would potentially lead to either reductions in average errors or speed-ups in clock frequencies, in comparison to the traditional approach where the target timing is met by limiting the precisions throughout the datapath. To verify this hypothesis, probabilistic models have been developed for errors generated from both design methodologies, and empirical results on a Virtex-6 FPGA have also been obtained over a set of DSP applications. Besides, the silicon area has also been taken into account for the trade-offs. We have shown that applying our approach on basic arithmetic primitives would potentially outperform utilising the traditional approach on either the same architectures or structures specifically designed for high performance, of which the precision loss is even greater under a tight area budget.

In this thesis, the proposed approach is targeted at datapath logic. This is because the critical path of complex signal processing systems often resides in the datapath, in which case our proposed approach would directly improve the clock frequency. In the case where the critical path lies elsewhere, it may still be possible to improve the overall operating speed of the system-level design solely by this approach, by using multiple clock domains, which may be practical in the case of large sections of datapath with simple control structures (*e.g.* the filters we present in this chapter) that are then interfaced to more complex control at IP boundaries.

In addition, the timing model is proposed with the purpose of capturing the notion of overclocking, therefore we focus on the fundamental feature of arithmetic datapath, *i.e.* carry propagation, to build up the models. We have then reverted to experimental results to verify whether the model sufficiently captures the complexity of a real implementation. For the circuits we have used, the model prediction matches well with the empirical results. These benchmarks are sufficiently large to have confidence that the presence of more than one basic arithmetic core does not have a significant impact on the overall behaviour.

On the other hand, there are numerous theoretical limits of the proposed model especially for practical usage. For example, in a practical FPGA implementation the carry propagation delay μ is unlikely to be a constant number as assumed in our model. In addition, our model isolates the targeting arithmetic operators from the datapath, whereas the error propagation between multiple operators as well as the corresponding impact on the overall accuracy are not addressed theoretically. Overcoming the theoretically limitations of the proposed models potentially serves as a promising future research direction and will be discussed in Chapter 6.

Chapter 4

Overclocking Datapath with Online Arithmetic

4.1 Introduction

In the previous chapter, a novel datapath design methodology using overclocking has been described in detail. We have demonstrated that allowing timing violations to happen can be beneficial when considering trade-offs between latency, accuracy and silicon area, because the worst case scenario only happens with a very small probability. However, we notice that one main obstacle of applying this approach is that once timing violation happens, large magnitude errors potentially occur. This is because, in standard arithmetic operators such as ripple-carry adders, the most significant digit (MSD) is updated *last* due to carry propagation, so timing violation initially affects the MSD of the result.

In this chapter, for the first time we attempt to solve this problem by adopting an alternative form of computer arithmetic named online arithmetic. In online arithmetic, all operations are preformed in an MSD-first manner. We suggest that in comparison to conventional arithmetic, operators employing this arithmetic are less sensitive to overclocking, because timing errors only affect the least significant digits (LSDs). To support this hypothesis, we evaluate the probabilistic behaviour of basic arithmetic primitives with different types of arithmetic when operating beyond the deterministic clocking region. A probabilistic model of overclocking error for an online arithmetic multiplier is proposed. We back up the model with experimental results from an image processing application. We demonstrate that our novel design methodology that combines overclocking and online arithmetic can lead to substantial performance benefits in comparison to the design method using conventional arithmetic.

In summary, the main contributions of this chapter are as follows:

- to our knowledge, the first consideration and evaluation of online arithmetic as an "overclocking friendly" computer arithmetic;
- probabilistic models of overclocking errors for a digit parallel online multiplier;
- analytical and empirical results which demonstrate that the impact of timing violations can be ameliorated with online arithmetic and that significant performance benefits can be achieved.

In the reminder of this chapter, we firstly describe the background of online arithmetic in Section 4.2, including its main features, together with the implementation of basic online arithmetic operators such as adder and multiplier. We then discuss the modelling of overclocking error for the online multiplier in Section 4.3. This is followed by a case study on image filter in Section 4.4, which demonstrates the benefits of applying our approach. Finally we draw the conclusion of this chapter in Section 4.5.

4.2 Background

In Chapter 2 we have provided an in-depth review of theoretical background of online arithmetic. In this section, we limit our discussion specifically on digit parallel online adder and online multiplier in Section 4.2.1 and Section 4.2.2, respectively. For ease of discussion, in the following of this chapter we only discuss online arithmetic algorithms with binary radix, as the it is used most commonly in computer arithmetic.

4.2.1 Binary Online Addition

Adders serve as a critical building block for arithmetic operations. The structure of an N-digit binary digit-parallel online adder is shown in Figure 4.1 [34], where all digits are represented with signed digit (SD) numbers with digit set $\{\bar{1}, 0, 1\}$. In this figure, the module "3:2" denotes a 3:2 compressor, which takes three inputs and generates two outputs, and is logically equivalent to a full adder (FA). Notice that this structure is identical to an adder that employs a redundant number system [89]. As highlighted in Chapter 2, a major advantage with the redundant number system over the standard ripple-carry based arithmetic is that the propagation of carry is eliminated, resulting in a precision-independent computation time for addition. As seen in Figure 4.1, the computation delay of this adder is only two FA delays for any operand word-length, with the cost of one extra FA for each digit of operands. Therefore this adder structure has been widely used as a part of other arithmetic operators such as multipliers to accelerate the sum of partial products [49, 117].

4.2.2 Binary Online Multiplication

Multiplication is another key primitive of arithmetic operations. Typically the online multiplication is performed in an iterative digit-serial manner, as illustrated in

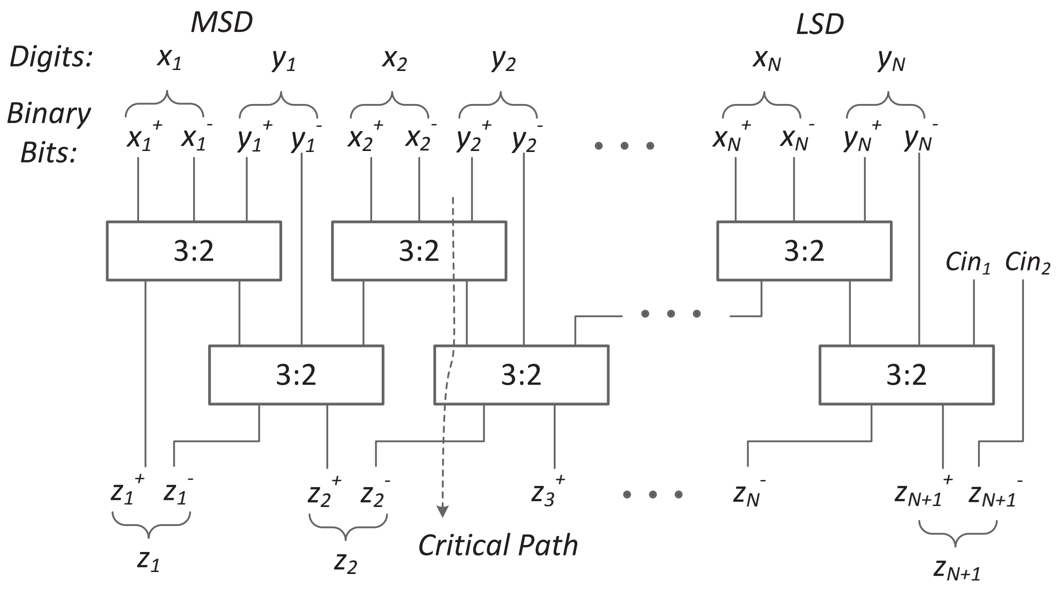

MSD
LSD
Digits:
x_1 y_1 x_2 y_2 x_N y_N
Binary Bits:
x_1^+ x_1^- y_1^+ y_1^- x_2^+ x_2^- y_2^+ y_2^- x_N^+ x_N^- y_N^+ y_N^-
3:2
Cin_1 Cin_2
z_1^+ z_1^- z_2^+ z_2^- z_3^+ z_N^- z_{N+1}^+ z_{N+1}^-
z_1 z_2 z_{N+1}
Critical Path

Figure 4.1: An N-digit binary digit-parallel online adder. Both inputs and outputs are represented using SD representation. “3:2” denotes a 3:2 compressor.

Algorithm 1 where both inputs and outputs are N-digit SD numbers as represented in (4.1) for $j \in [-\delta, 2N-1]$ and r denotes the radix [34]. For a given iteration j, the product digit z_j is generated through a selection function $sel()$. For any radix r and chosen digit set, there exits an appropriate selection method and a value of δ which ensure the convergence of the algorithm. The mathematical analysis of determining δ and the selection function is beyond the scope of this thesis, and the full details can be fond in [34]. Specifically for the case of binary radix, we have $\delta = 3$, and $sel()$ is given by (4.2). It can be seen that the selection can be made only based on one fractional bit together with integer bits of $W_{[j]}$.

$$X_{[j]} = \sum_{i=1}^{j+\delta} x_i r^{-i},\ Y_{[j]} = \sum_{i=1}^{j+\delta} y_i r^{-i},\ Z_{[j]} = \sum_{i=1}^{j} z_i r^{-i} \tag{4.1}$$

Algorithm 1 Online Multiplication

```
1: Initialisation: $X_{[-\delta]} = Y_{[-\delta]} = P_{[-\delta]} = 0$
2: for $j = -\delta, \ -\delta+1, \ \cdots, \ 2N-1$ do
3:     $H_{[j]} \leftarrow r^{-\delta}\left(x_{j+\delta+1} \cdot Y_{[j+1]} + y_{j+\delta+1} \cdot X_{[j]}\right)$
4:     $W_{[j]} \leftarrow P_{[j]} + H_{[j]}$
5:     $z_j \ \leftarrow sel(W_{[j]})$
6:     $P_{[j+1]} \leftarrow r\left(W_{[j]} - Z_{[j]}\right)$
7: end for
```

$$sel(W_{[j]}) = \begin{cases} 1 & \text{if } W_{[j]} \geqslant \frac{1}{2} \\ 0 & \text{if } -\frac{1}{2} \leqslant W_{[j]} < \frac{1}{2} \\ \bar{1} & \text{if } W_{[j]} < -\frac{1}{2} \end{cases} \tag{4.2}$$

It is worth noting that the precision of the product Z can be determined by directly truncating from the LSDs, since there is no carry propagation from the LSD to the MSD. This is useful especially when the multiplication product is used by successive computations, where often only the most significant N digits of the results are used. In comparison, the product is generated starting from the LSD with conventional arithmetic. Hence the successive operations cannot commence until the result with full-precision is generated. This feature of online arithmetic has inspired research about dynamically control of computation precision in numerous applications [102].

Algorithm 1 can be synthesized into a unrolled digit parallel structure as shown in Figure 4.2(a). In order to generate results with full precision, there are in total $2N + \delta$ stages. The online inputs are generated from the appending logic according to (4.1). For each stage, an online adder is used as shown in Figure 4.2(b). The SDVM module performs the Signed Digit Vector Multiplication, that is, it generates the results of $(x_{j+\delta+1} \cdot Y_{[j+1]})$ or $(y_{j+\delta+1} \cdot X_{[j]})$ in Algorithm 1. For the binary radix,

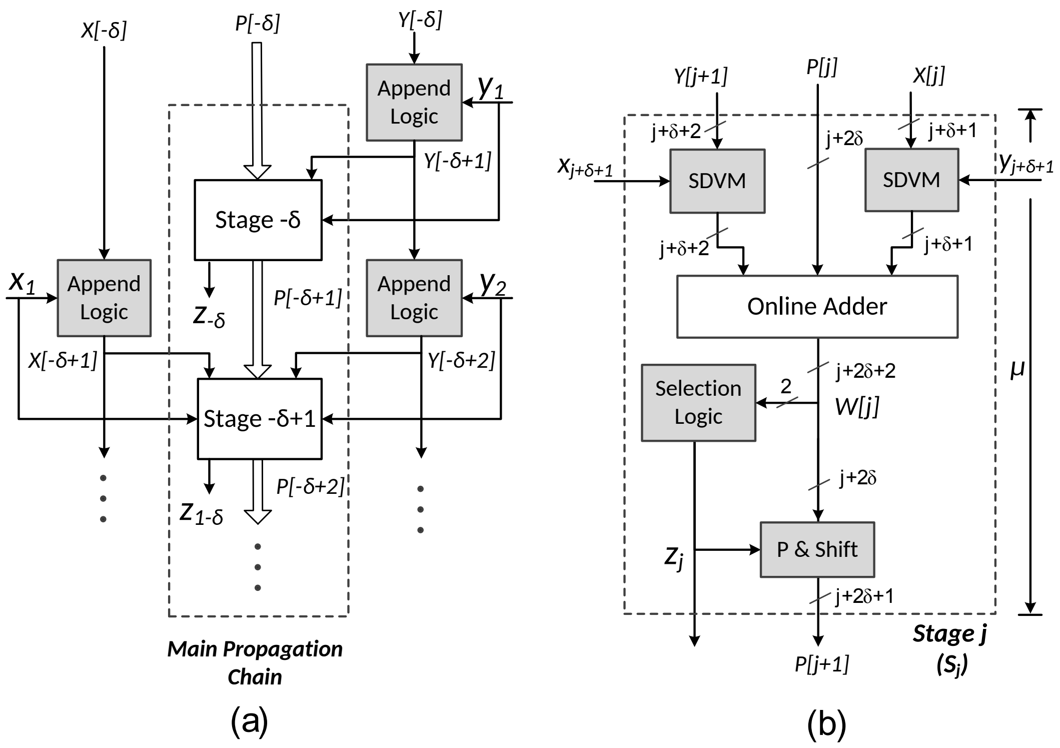


Figure 4.2: (a) A digit-parallel implementation of the online multiplication algorithm (b) Structure of one stage of online multiplier, the maximum digit widths of signals are labelled.

the implementation of SDVM is straightforward, because in this case we have $x_j, y_j \in \{\bar{1}, 0, 1\}$, hence for instance the output of $(y_{j+\delta+1} \cdot X_{[j]})$ can be 0, $X_{[j]}$ or $-X_{[j]}$ when y is equal to 0, 1 and -1 respectively.

Instead of duplicating the digit-serial implementation $2N + \delta$ times, each stage in the digit-parallel architecture can be optimised for area reduction. For example, the SDVM modules and the appending logic are not required in the last δ stages because the inputs are zero. This leads to a smaller online adder in these stages. Similarly for the first δ stages the selection logic can be removed, as these stages generate no output digits and the first digit of the result is generated at S_0.

4.3 Probabilistic Model of Overclocking Error

As the delay of online adder is small, the occurrence of timing violation requires very fast frequency. Hence we only model the overclocking error for the radix-2 online multiplier OM. From Figure 4.2 we observe two types of delay chains. One is caused by generation and propagation of $P_{[j]}$ among different stages. The other is the generation of online inputs $X_{[j]}$ and $Y_{[j]}$. Since according to (4.1) the appending logic is basically wires and simple combinational logic [28], the overall latency will eventually be determined by the delay of the $P_{[j]}$ path, especially with increasing operand word-lengths. As such, we initially model the delay of each stage within an OM to be a constant value μ, as shown in Figure 4.2(b). We also assume in our model that the generation of online inputs costs no delay.

Let μ_{OM} denote the worst-case delay of an OM, thus we have $\mu_{OM} = (2N + \delta)\mu$. Similar to the modelling process of overclocking error of an RCA as detailed in the previous chapter, it follows that if the clock period T_S is greater than μ_{OM}, correct results will be sampled. If, however, faster-than-rated sampling frequencies are applied such that $T_S < \mu_{OM}$, timing violations might happen and intermediate results will be sampled, potentially generating overclocking errors. For a given T_S, we re-use the previous definition in (3.4) of Chapter 3 that parameter b denotes the maximum length of error-free propagation, as given by (4.3) where f_S denotes the operating frequency. In the model we always ensure that the first digit of the product will be generated correctly, *i.e.* $b > \delta$.

$$b := \left\lceil \frac{T_S}{\mu} \right\rceil = \left\lceil \frac{1}{\mu \cdot f_S} \right\rceil \tag{4.3}$$

We now determine when this timing constraint is not met, and the size of error in this case. Similar to Chapter 3, we model the overclocking errors with the following assumptions:

1. Every digit of each input to our circuit is uniformly and independently generated with the digit set $\{\bar{1}, 0, 1\}$.

2. All internal signals of the circuit is reset to zero before the first iteration in Algorithm 1 starts.

However, all these assumptions will be relaxed in Section 4.4 where data from real image benchmarks are used as inputs, and results are obtained from simulations after implementing the design on an FPGA.

4.3.1 Probability of Timing Violations

For an N-digit online multiplier, there are overall $2N + \delta$ stages for digit parallel implementation. We use τ to denote the stage number, that is, the τ^{th} stage is referred to as S_τ. The value of τ is then bounded by (4.4). Also let a propagation chain be generated at stage S_τ with the length of $d(\tau)$ digits. Because the presence of timing violation requires $d(\tau) > b$, and the chain cannot propagate over S_{2N-1}, we cam obtain the bound of parameter $d(\tau)$ as given by (4.5).

$$-\delta \leqslant \tau \leqslant 2N - 1 - b = \tau_{max} \tag{4.4}$$

$$b < d(\tau) \leqslant 2N - 1 - \tau \tag{4.5}$$

However, the actual length of a "carry" chain is dependent upon input patterns. We now examine and model the relationship between $d(\tau)$ and the input patterns that correspond to the generation, propagation and annihilation of the carry chain. In addition, for a given carry chain of length $d(\tau)$, we quantify its probability based on inputs, such that the probability of timing violation, *i.e.* the probability of $d(\tau) > b$, can be determined using our model.

Let the specific input pattern of stage S_τ be represented by $C(\tau)$, which can be classified into four types based on whether a digit of inputs x and y is zero, as listed in (4.6). Under the assumption that all digits of input signal are mutually independent and uniformly distributed, the probabilities of C_1 to C_4 equal to 1/9, 4/9, 2/9 and 2/9 respectively.

$$C(\tau) = \begin{cases} \text{Case 1 } (C_1): & x_{\tau+\delta+1} = 0,\ y_{\tau+\delta+1} = 0 \\ \text{Case 2 } (C_2): & x_{\tau+\delta+1} \neq 0,\ y_{\tau+\delta+1} \neq 0 \\ \text{Case 3 } (C_3): & x_{\tau+\delta+1} \neq 0,\ y_{\tau+\delta+1} = 0 \\ \text{Case 4 } (C_4): & x_{\tau+\delta+1} = 0,\ y_{\tau+\delta+1} \neq 0 \end{cases} \tag{4.6}$$

As we assume that all internal signals are reset to zero initially, we have $H_{[j]} = P_{[j]} = Z_{[j]} = 0$. Therefore before the first iteration of Algorithm 1 starts, Algorithm 1 can be simplified to give (4.7) for $j = \tau > -\delta$. This equation is used to determine whether a carry chain will be generated at S_τ, for a given value of $C(\tau)$.

$$\begin{aligned} P_{[\tau+1]} &= r \cdot W_{[\tau]} \\ &= r^{-\delta+1}(x_{\tau+\delta+1}Y_{[\tau+1]} + y_{\tau+\delta+1}X_{[\tau]}) \end{aligned} \tag{4.7}$$

If $C(\tau) = C_1$, no chain will be generated at S_τ because in this case $P_{[\tau+1]} = 0$. In other cases, $P_{[\tau+1]} \neq 0$ and a new chain will be generated at S_τ. We denote the number of the leading non-zero digits of $P_{[\tau+1]}$ be D. When $P_{[\tau+1]}$ propagates to the next stage, from Algorithm 1 we notice that the value of D is reduced by one, because of the one digit shifting left when computing $P_{[\tau+1]} = r(W_{[\tau]} - z_\tau)$. Also it is worth noting that this process is independent of inputs, thus it will repeat in the following stages until

$D = 1$, when this carry chain is finally annihilated. Hence we see that the value of D can affect the chain length $d(\tau)$, which we consider more specifically and separately for $C(\tau) = C_2,\ C_3$ and C_4.

If $C(\tau) = C_2$, D equals to the maximum word-length of $P_{[\tau+1]}$ as shown in Figure 4.2(b), *i.e.* $D = \tau + 2\delta + 1$. Combining this value of D and (4.5) gives the maximum chain length $d(\tau)$ in (4.8).

$$\begin{aligned} d(\tau)_{max} &= Min(D, 2N - 1 - \tau) \\ &= Min(\tau + 2\delta + 1, 2N - 1 - \tau) \end{aligned} \tag{4.8}$$

For C_3 and C_4, the carry generation, propagation and annihilation behaviour of both cases are statistically identical. Hence we only discuss $C(\tau) = C_3$, and the conclusion can be directly applied for $C(\tau) = C_4$. For $C(\tau) = C_3$, (4.7) is modified to give (4.9), because $y_{\tau+\delta+1} = 0$. Due to the same reason we also have $Y_{[\tau+1]} = Y_{[\tau]}$. According to (4.9), D can be determined only based on the number of the leading non-zero digits of $Y_{[\tau]}$.

$$P_{[\tau+1]} = r^{-\delta+1}(x_{\tau+\delta+1}Y_{[\tau+1]}) = r^{-\delta+1}(x_{\tau+\delta+1}Y_{[\tau]}) \tag{4.9}$$

If the carry chain is generated from the first stage, we have $\tau = -\delta$. Substituting this into (4.9) yields $P_{[-\delta+1]} = r^{-\delta+1}(x_1 y_1)$. Therefore if $C(-\delta) = C_2$, we have $d(-\delta) = d(-\delta)_{max}$ as given in (4.8), otherwise $d(-\delta) = 0$.

If the carry chain is generated from other stages where $\tau > -\delta$, D and $d(\tau)$ should be determined based on two possible judgements for $\tau' = \tau - 1$ as stated below:

1. If $y_{\tau'+\delta+1} \neq 0$, D equals to the maximum word-length of $Y_{[\tau'+1]}$. Thus we have $d(\tau) = d(\tau')_{max} - 1$.

2. If $y_{\tau'+\delta+1} = 0$, then $Y_{[\tau']} = Y_{[\tau'-1]}$. This is a recursive process and these two judgements should be performed recursively for $\tau'' = \tau' - 1$. This process will terminate when either the first judgement is satisfied or the first stage is reached, *i.e.* $\tau = -\delta$.

Now we have examined all possible cases of $C(\tau)$ and the corresponding value of $d(\tau)$ for any given value of τ. Since the occurrence of timing violation requires $d(\tau) > b$, the probability of timing violation can be determined by summing up the probabilities of all carry chains that satisfy $d(\tau) > b$. This process can be summarised into Algorithm 2, of which the general idea is to examine all possible stages that timing violations may occur, together with all possible input patterns for each stage. This algorithm returns $Pr(T_S)$, which denotes the probability that timing violations happen in an N-digit online multiplier as a function of T_S. In this algorithm, b and τ_{max} are obtained from (4.3) and (4.4), respectively. The probability of the input pattern for stage i is denoted by $Pr(C(i))$, which can be determined through (4.6).

Algorithm 2 Probability of Timing Violations

```
1: $Pr(T_S) \leftarrow 0$;
2: for all Stages $\tau \in \{-\delta, -\delta+1, \cdots, \tau_{max}\}$ do
3:     for all Input cases $C(\tau) \in \{C_1, \cdots, C_4\}$ do
4:         Determine $d(\tau)$ according to $C(\tau)$;
5:         if $d(\tau) > b$ then
6:             $Pr(T_S) \leftarrow Pr(T_S) + \prod_{i=-\delta}^{\tau} Pr(C(i))$;
7:         end if
8:     end for
9: end for
10: return $Pr(T_S)$
```

4.3.2 Magnitude of Overclocking Error

In the presence of timing violations, multiple chains might not be correctly propagated, resulting in overclocking errors generated from LSDs. In order to determine in which digits overclocking errors may generate, we start by locating the first output digit z_λ that contains error, and hence errors are expected to occur for all digits from digit z_λ to digit z_{2N}. For a given T_S and a minimum value of τ such that the maximum length of the carry chain generated at S_τ satisfies $d(\tau)_{max} > b$, this carry chain will be annihilated at the stage that delivers output digit z_λ. Therefore we have $\lambda = \tau + d_{max}(\tau) + 1$. As such, the magnitude of overclocking error can be expressed by (4.10), where z_i and $z_i{}'$ denote the correct value and the actual value of the output digit of S_i, and $\varepsilon_i \in \{\pm 2^{-i}, \pm 2^{-i+1}, 0\}$ since z is represented using digit set $\{\bar{1}, 0, 1\}$.

$$|\varepsilon| = \left| \sum_{i=\lambda}^{2N} 2^{-i}(z_i - z_i{}') \right| = \left| \sum_{i=\lambda}^{2N} \varepsilon_i \right| \tag{4.10}$$

However, it is worth noting that using online arithmetic, the change of output digits does not necessarily result in a different number as it is represented using a redundant form. This will lead to smaller error magnitude in practice. For instance, due to timing violation two digits of a computation output is changed from $0.10\bar{1}$ to $0.1\bar{1}1$. Nevertheless, no error is generated in this case, as both numbers represent 0.011 in the two's complement form. In contrast, errors will always be generated with conventional arithmetic if the output digits are changed, because each number has a unique representation.

4.3.3 Expectation of Overclocking Error

For a given T_S, the expectation of overclocking error can be obtained by combining $Pr(T_S)$ from Algorithm 2 and $|\varepsilon|$ from (4.10), as presented in (4.11). We first verify the proposed model against the results from behavioural simulations with the aforementioned assumptions on timing and inputs, as shown in Figure 4.3(a) and Figure 4.3(b) for 8-digit OM and 12-digit OM, respectively. T_S is normalised with respect to the value from structural timing analysis, *i.e.* $(2N+\delta)\mu$. It can be seen that the modelled value match well with the simulation results. We also verify the models against the FPGA timing simulation results without timing assumptions, as illustrated in Figure 4.3(c) and Figure 4.3(d). We notice that the general behaviour of the model is similar to the FPGA results, however it does not capture the small increments in errors because we only model the coarse delay which is in units of μ. Whereas the other sources of delay, such as the generation of the online form of inputs and result digits, are not included in our model.

$$E_{ovc} = Pr(T_S) \cdot |\varepsilon| \tag{4.11}$$

From Algorithm 2, instead of accumulating $Pr(T_S)$, the probability of each individual chain delay $Pr(d)$ is shown in Figure 4.4 for a variety of word-lengths of OM, together with the corresponding error magnitude $|\varepsilon(d)|$ and the product of both in terms of error expectation $E(d)$, where parameter d represents the chain delay. For a given T_S, the overall error expectation in (4.11) can also be obtained from (4.12).

$$E_{ovc} = \sum_{d>b} Pr(d) \cdot |\varepsilon(d)| = \sum_{d>b} E(d) \tag{4.12}$$

From Figure 4.4 several observations can be made. Firstly, for all operand word-lengths the magnitude of overclocking error decreases exponentially with longer chain lengths. This is in contrast to designs with conventional arithmetic, and justifies the advantage of using online arithmetic in the first place that timing violation firstly affects LSDs with online arithmetic. Secondly, interestingly we see that carry chains

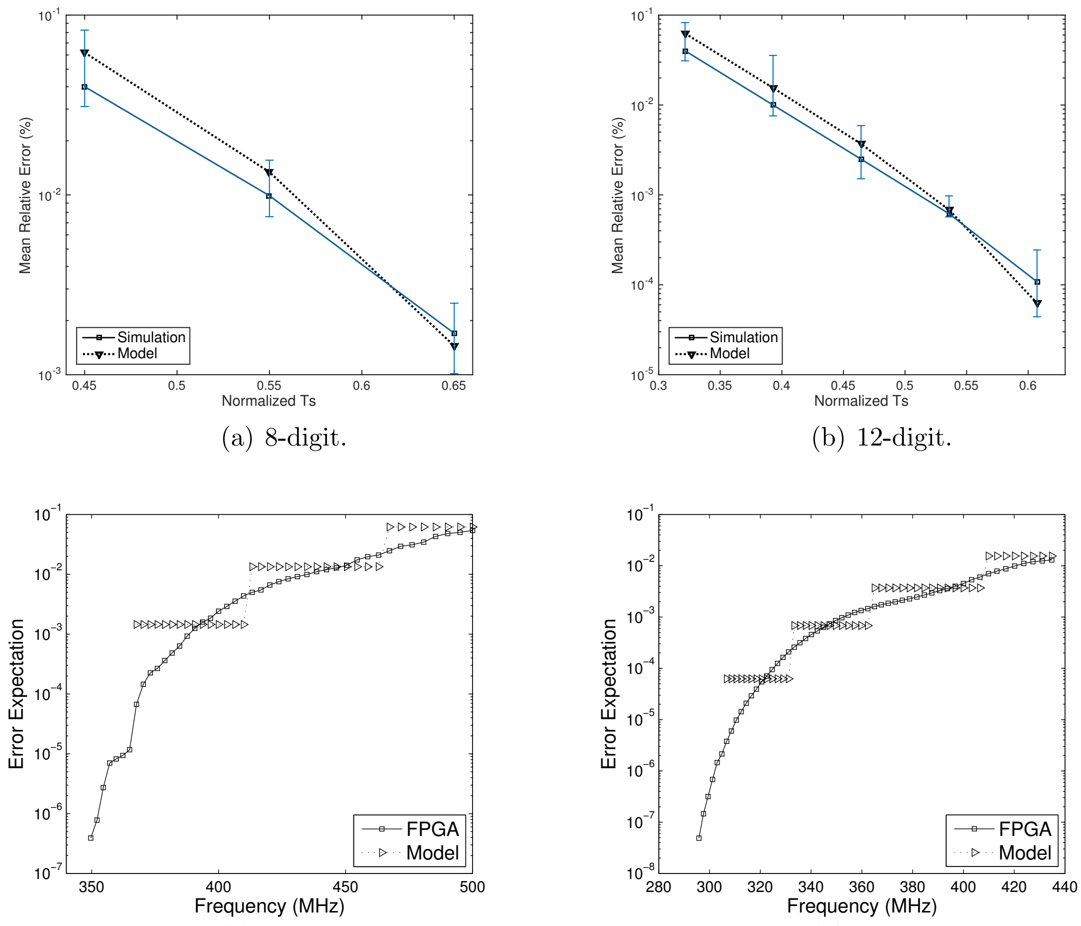
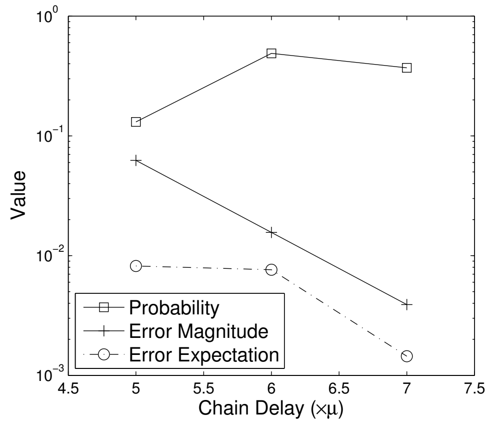
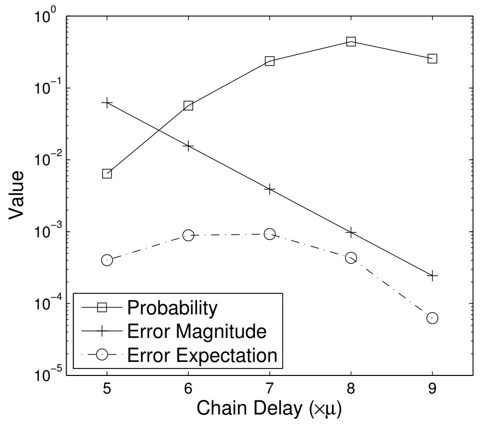


Figure 4.3: Expectation of overclocking error for online multipliers: verification of the proposed model against behavioural simulations with timing assumptions (top row) and FPGA timing simulations (bottom row).

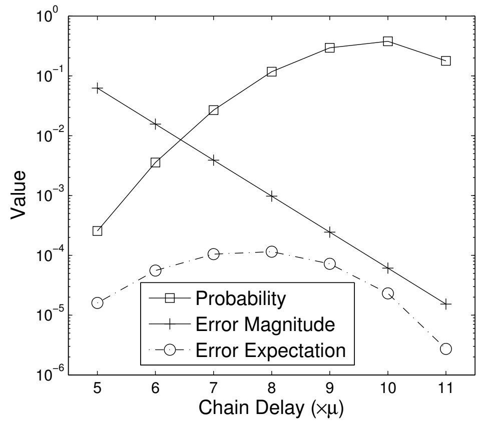
with longer delay would happen with greater probabilities in an OM. This is because for the unrolled radix-2 OM, $d(\tau)$ is only dependent upon the inputs for chain generation, whereas carry propagation and annihilation will not be affected by inputs, as highlighted in Section 4.3.1. Therefore long carry chains will be generated as long as both input digits are nonzero with the probability 4/9. In comparison to the traditional arithmetic, carry generation, propagation and annihilation are all decided by specific input patterns. This limits the overall probability of long chains. However, we also notice that for chains with long delays, the increasing in likelihood of error is



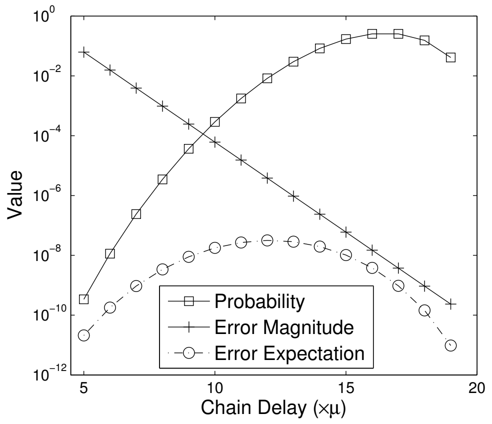
(a) 8-digit.



(b) 12-digit.



(c) 16-digit.



(d) 32-digit.

Figure 4.4: Probabilities of different chain delay, the corresponding magnitude of overclocking error and the combination of both in terms of error expectation in the radix-2 OM with different word-lengths.

outweighed by the decrease in magnitude of error. Therefore the combination of both results in a decline in error expectation. In contrast, one would expect error expectation to grow faster as the amount of overclocking is increased when using traditional arithmetic because errors occur in MSDs.

4.4 Case Study: Image Filter

4.4.1 Experimental Setup

The benefits of the proposed methodology are demonstrated by using a 3×3 Gaussian image filter with two types of computer arithmetic. One is the standard binary arithmetic with all data represented in the two's complement form. In order to implement the image filter, speed-optimised adders and multipliers are created using Xilinx Core Generator [136]. The other is the radix-2 online arithmetic, with the basic building blocks described in Section 4.2.1 and Section 4.2.2. In the online design, all signals are represented in the online format. In order to achieve the desired latency between input and output, both designs are overclocked and the errors seen at the output are recorded. The results are obtained from a Xilinx Virtex-6 FPGA through post place-and-route simulations. Same to the previous chapter, the differences between the actual results and error-free results are still evaluated in terms of mean relative error (MRE), as given by (3.26) and it denotes the ratio between the mean value of error and mean value of outputs.

Since the redundant digit set is used in online arithmetic, a signed integer number within $[-2^N, 2^N - 1]$ can be represented using N digits. However the representation range is $[-(2^{N-1}, 2^{N-1} - 1)]$ with N digits using two's complement number system. For a valid comparison, an N-digit image filter using online arithmetic is employed to compare against an $(N + 1)$-digit design using traditional arithmetic.

In our experiments, two types of input data are utilised. One is randomly sampled from a uniform distribution of N-digit numbers. This type is referred to as "Uniform Independent (UI) inputs". The other is called "real inputs", which are the pixel values of several 512×512 benchmark images.

4.4.2 The Impact of Overclocking

The MRE values of the image filter with traditional arithmetic (dotted lines) and online arithmetic (solid lines) when $N = 8$ are illustrated in Figure 4.5. Note that we plot the "actual" maximum frequencies which are obtained experimentally increasing the operating frequency from the rated value until errors are observed at the outputs. This is to remove the conservative timing margin for a fairer comparison once overclocking is introduced.

According to the timing report, the rated operating frequencies are 168.7MHz and 148.3MHz for design with conventional arithmetic and design with online arithmetic, respectively. However as seen in Figure 4.5, both designs are actually running at faster frequencies without timing errors. This is in accordance to the results shown in the previous chapter, and is due to the removal of the conservative guard bands. For the UI inputs, design with online arithmetic actually operates at a higher frequency without timing violations in comparison to the design using traditional arithmetic. This error-free frequency is even larger when using the real image data as inputs, because the real data do not exactly follow the uniform distribution or the independent assumption, and the worst cases happen with even smaller possibilities. In Figure 4.5 the "Lena" benchmark image is used as the real inputs.

If errors can be tolerated, we may allow timing violations to happen for better performance. The sensitivity of overclocking error for a given arithmetic can be evaluated by the data slope in Figure 4.5. For instance under an error budget of 1% MRE, using the UI inputs the frequency of the traditional design can be improved by 3.89% with respect to the maximum frequency without errors, whereas the design with online arithmetic can be overclocked by 6.85%. This indicates that online arithmetic is less sensitive to overclocking, as illustrated in Section 4.3.3. The difference is even greater using real image data: 13.74% frequency speed-up using online arithmetic against

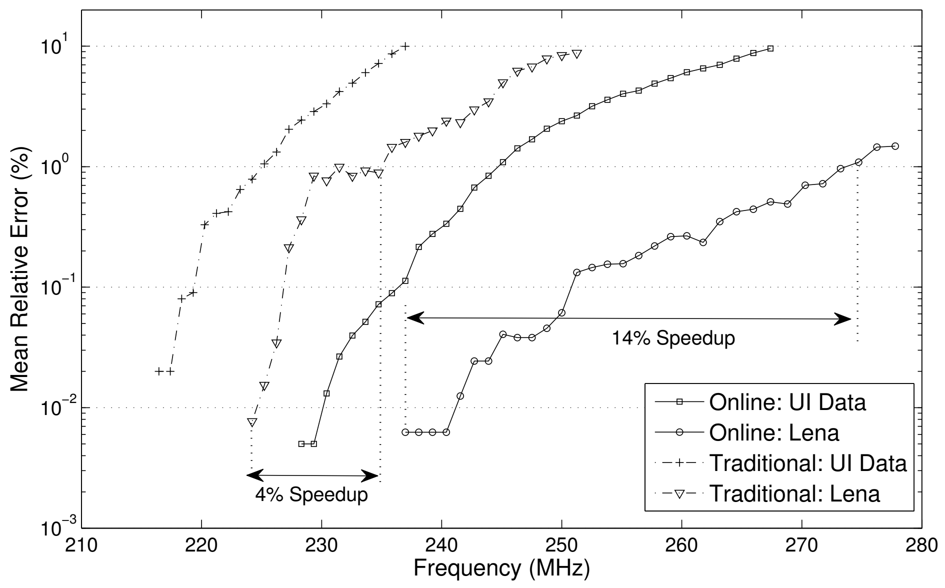
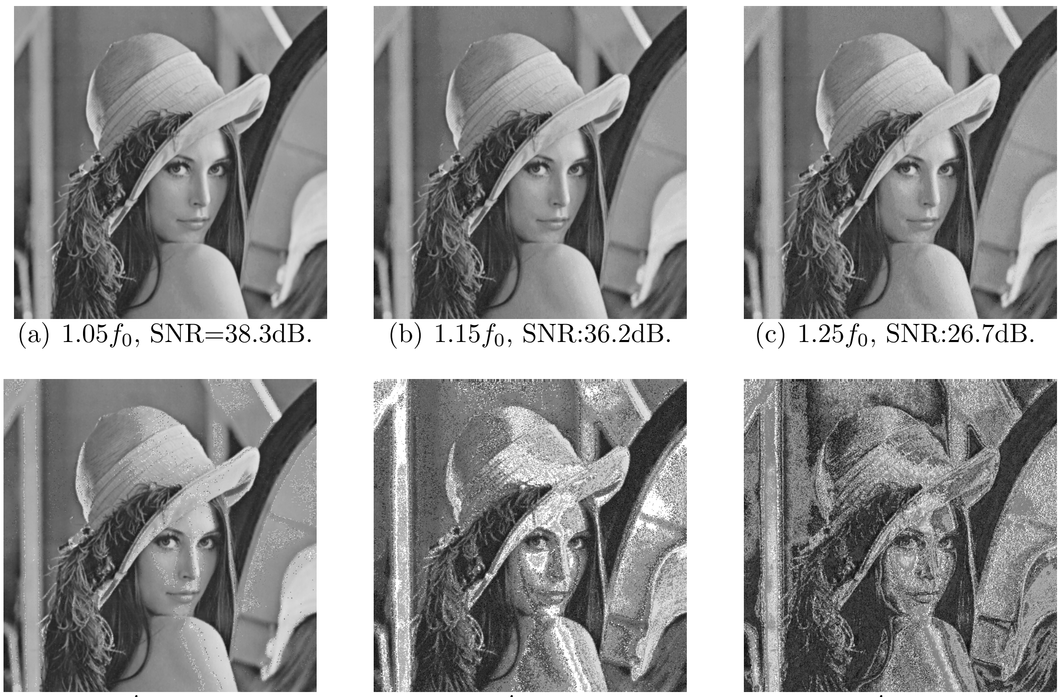


Figure 4.5: Overclocking error in an image filter with two types of computer arithmetic: online arithmetic and standard binary arithmetic, of which the rated frequencies are 148.3MHz and 168.7MHz, respectively, according to the timing analysis tool.

4.04% with traditional arithmetic, because fewer long chains are generated with real inputs.

The output images for both design scenarios are presented in Figure 4.6. Images obtained from design with online arithmetic are shown in the top row, while output images obtained from design with conventional arithmetic are illustrated in the bottom row. In Figure 4.6, f_0 and f_0' denote the maximum error-free frequencies for the online design and the conventional design, respectively. Both f_0 and f_0' are obtained from our experiments instead of timing reports. In addition, SNR of each image is also recorded. Since the overclocking errors are in the LSDs of the results with online arithmetic, this leads to small error magnitude and consequently the degradation on the image can hardly be observed. In contrast, timing violations cause errors in the MSDs with traditional arithmetic. This leads to "salt and pepper noise", which causes severe quality loss of the images especially for high frequencies, as can be clearly observed from the bottom row of Figure 4.6. Meanwhile, errors in the MSDs would result in large noise power and therefore the SNR for the traditional design is small.


(a) $1.05f_0$, SNR=38.3dB. (b) $1.15f_0$, SNR:36.2dB. (c) $1.25f_0$, SNR:26.7dB.

(d) $1.05{f_0}'$, SNR:21.5dB. (e) $1.15{f_0}'$, SNR:9.0dB. (f) $1.25{f_0}'$, SNR:8.5dB.

Figure 4.6: Output images of image filter using online arithmetic (top row) and traditional arithmetic (bottom row).

4.4.3 Potential Benefits in Circuit Design

Similarly to Section 3.8.2, our results could be of interest to a circuit designer in two ways. By choosing different arithmetic and data representations, either a circuit can be designed to operate at a certain frequency with the minimum possible MRE, or a given error budget can be met with the fastest achievable frequency. For the first case, the experimental results obtained by UI inputs and four benchmark images are summarised in Table 4.1 in terms of the relative reduction of MRE, and in Table 4.2 for the differences of SNR, which are calculated by (4.13) where SNR_{online} and SNR_{conv} denote the SNR obtained from the design with online arithmetic and conventional arithmetic, respectively. In both tables the frequency is normalised to the maximum error-free frequency for each arithmetic, *i.e.* f_0 and ${f_0}'$ in Figure 4.6. From our results, a significant reduction of MRE can be observed using online arithmetic for all input types. The geometric mean reduction of MRE is 89.2% using UI data. Even larger differences of MRE reduction can be achieved when testing with real image data, varying from 97.3% to 98.2%, as long chains occur with smaller probabilities with real data. Similarly from Table 4.2, the improvements in SNR are ranging from $21.4dB$ to $43.9dB$.

$$SNR_{diff} = SNR_{online} - SNR_{conv} \tag{4.13}$$

For the second design perspective, Table 4.3 illustrates the frequency speed-ups with different input types when specific error budgets can be tolerated. Note that the “N/A” in this table means certain error budget cannot be met with traditional arithmetic at all. We see that for all input types, design using online arithmetic still outperforms the traditional design for each MRE budget in terms of operating frequency. Likewise the geometric mean of frequency speed-ups is larger for real image inputs.

Table 4.1: Relative Reduction of MRE with Online Arithmetic for Various Normalised Frequencies.

Inputs	**Normalised Frequency**					**Geo.**
	1.05	1.10	1.15	1.20	1.25	**Mean**
Uniform	94.5%	89.1%	90.1%	88.3%	84.3%	89.2%
Lena	99.3%	99.2%	98.9%	97.7%	94.9%	97.9%
Pepper	99.7%	98.3%	98.1%	97.2%	95.1%	97.7%
Sailboat	99.5%	97.9%	97.3%	96.8%	95.1%	97.3%
Tiffany	99.9%	97.6%	98.4%	97.8%	97.2%	98.2%

Table 4.2: Improvement of SNR (dB) with Online Arithmetic for Various Normalised Frequencies, Calculated Using (4.13).

Inputs	**Normalised Frequency**				
	1.05	1.10	1.15	1.20	1.25
Lena	44.6	36.3	33.2	29.1	22.9
Pepper	35.7	28.3	28.7	25.9	24.1
Sailboat	33.7	27.5	26.3	25.0	21.7
Tiffany	43.9	25.9	29.5	25.6	24.5

Table 4.3: Relative Improvement in Frequency with Online Arithmetic for Various Error Budgets.

Inputs	**Error Budget**				**Geo.**
	0.01%	0.1%	1%	10%	**Mean**
Uniform	N/A	4.59%	8.78%	12.83%	8.03%
Lena	7.96%	11.50%	16.39%	13.71%	11.88%
Pepper	6.22%	8.87%	18.68%	15.94%	11.32%
Sailboat	5.71%	8.87%	16.93%	15.29%	10.70%
Tiffany	2.35%	6.90%	18.58%	12.22%	7.79%

4.4.4 Area Overhead of The Proposed Approach

Transferring from conventional arithmetic to online arithmetic comes at the cost of extra usage of silicon area. The area comparison between the above two designs is illustrated in Table 4.4 in terms of the number of LUTs and Slices in FPGA technology. While we notice that our approach does cause increment in area utilisation, it should be noticed that the FPGA architecture is optimised for conventional arithmetic. For instance, the Xilinx Virtex series FPGAs employ dedicated multiplexers and XOR gates for very fast ripple carry addition [135]. In comparison, our designs are not specified optimised for FPGA architecture. Besides, at least two orders of magnitude error reduction is obtained for a given frequency in our design, as shown in Figure 4.5. Although the differences in both error and area can be compensated by using more digits with traditional arithmetic, this will result in longer delay and an even larger gap in frequency between two designs.

Table 4.4: Area Comparison between Designs with Two Types of Computer Arithmetic.

Metric	**Arithmetic Type**		**Overhead**
	Conventional	Online	
LUT	912	1896	2.08
Slice	324	525	1.62

4.5 Conclusion

In this chapter, we have studied the probabilistic behaviour of key arithmetic primitives with different types of computer arithmetic when operating beyond the deterministic clocking region. For the first time we attempt to combine online arithmetic with approximate datapath design in order to achieve a graceful degradation under overclocking. Through the usage of analytical models and empirical FPGA results, we have demonstrated that significant error reduction and performance improvement can be achieved by using the "overclocking friendly" online arithmetic. Furthermore, we have identified that the major drawback of implementing online arithmetic on existing hardware platform such as FPGAs is that it normally requires large area overhead. This issue will be the main focus of the next Chapter.

Chapter 5

Efficient FPGA Implementation of Online Arithmetic Operators

5.1 Introduction

In the previous chapter, we have demonstrated the "overclocking friendly" feature of online arithmetic. We have shown that digit parallel online operators allow graceful degradation when operating the circuit beyond the deterministic clocking region, because timing errors only occur at LSDs of the results. In comparison, traditional forms of computer arithmetic do not fail gracefully when timing violations happen, since timing errors tend to affect MSDs of the results.

Despite its attractiveness, we have also pointed out in the end of previous chapter that the usage of online arithmetic on existing hardware platforms such as FPGAs is still limited due to a large area overhead. For the implementation of standard arithmetic on FPGAs, because it is inefficient and slow to map arithmetic circuits solely onto look-up-tables (LUTs), both main commercial FPGA vendors have introduced dedicated carry logic [135] to address this issue. Significant prior work explores methods that can effectively map circuits to the carry logic for performance improvements [99], or proposes alternative carry logic for FPGAs [50, 40]. Hard DSP blocks are also included in modern architectures to improve area efficiency and performance of arithmetic operations such as multiplication and multiply accumulation. However, most of these approaches are designed to accelerate computations with the conventional form of arithmetic.

For online arithmetic, a major research direction is about its efficient FPGA implementation: a brief review of relevant work in this area is presented in Section 5.2. Specifically, there are existing approaches to build general purpose online adders specifically for FPGAs with 4-input LUTs (4-LUT) and with two LUTs within a slice or a logic element (LE), such as the Xilinx Spartan 3 series FPGAs [133] and the Altera Cyclone III FPGA [3]. However, as will be discussed in Section 5.2, these approaches cannot be directly and/or efficiently applied on many modern FPGAs, which typically have 6-input LUTs (6-LUT) and four LUTs in a slice, such as the Xilinx Virtex series FPGAs and all Xilinx 7 series FPGAs [138]. Moreover, the move to the 6-LUT on the basic architecture opens new opportunities for further optimisation, which have not been investigated by previous studies.

In this chapter, this issue will be addressed. In Section 5.2, we propose a novel approach to map digit parallel online adders onto FPGAs with 6-LUT. The mapping is based on the fast carry logic and ensures the available logic resources within a slice are fully utilised. We demonstrate experimentally on a Xilinx Virtex-6 FPGA that the proposed designs achieve significant area reduction and performance gain over their original implementations. For online adders, our method achieves slice savings of over 67% and frequency speed-ups of over 1.2× in comparison to a direct implementation. In comparison to the standard ripple-carry adder, the area overhead drops from up to 8.41× to 1.88× using the proposed online adder.

In Section 5.3, we expand the evaluations of the optimum adder structure in Chapter 3 by adding our proposed online adder into the comparison. We take into account multiple design constrains and options, such as accuracy, operating frequency, silicon area, different types of adder structure, and whether the design is overclocked or not. We demonstrate that for limited area budgets, applying the overclocking approach to RCA can be more beneficial than using faster adders to achieve similar latency. Whereas with more relaxed area constraints, using our proposed online adder outperforms in terms of accuracy or performance.

In Section 5.4, we optimise the online multiplication algorithm to yield an efficient digit parallel online multiplier for FPGAs. The performance of the proposed online multiplier is demonstrated experimentally on a Xilinx Virtex-6 FPGA. We show that the slice utilisation of the proposed online multiplier reduces by over 69%, with a frequency speed-up over 1.5× in comparison to a direct implementation. When compared to the multiplier with standard arithmetic, the direct implementation has an area overhead up to 8.11×, which drops to 1.84× when using the proposed new architecture.

In addition, our approach has the advantage of being able to create a correctly rounded multiplier with a reduced output precision using less hardware than a full multiplier, which multiplies two N-digit numbers and produces an output of $2N$ digits. However, in many cases the full precision product is not required. In such cases, a reduced precision output is created by either truncating or rounding the LSDs. Although there has been research into saving area by avoiding the creation of unnecessary LSDs [27], these techniques still often require extensive simulation to guarantee correctness. In contract, since online multipliers compute the output from the MSD first, hardware can be saved by not generating the LSDs in the first place. As an example, if we are only interested in the most significant half of the product, according to our experiments we can create a digit parallel online multiplier that saves up to 56% slice usage in comparison to a standard multiplier with truncated output.

Finally in Section 5.5, we re-implement the benchmark circuits employed in Chapter 3 with online arithmetic on a Virtex-6 FPGA, and evaluate the performance benefits of using online arithmetic operators based on FPGA measurements.

In summary, the main contributions of this chapter are:

- Efficient methods to map online adders and online multipliers to modern FPGAs;
- Comparison of online adders to traditional arithmetic adders when considering a variety of design trade-offs;
- A method to implement correctly rounded online multipliers for a chosen output precision;
- Demonstration of the performance improvements of online operators in terms of area and frequency.

5.2 Digit Parallel Online Adder for FPGAs

The structure of a digit parallel online adder is detailed in Chapter 4. In this section, we aim to investigate an area-efficient FPGA implementation of online adder.

5.2.1 Related Work

As discussed in Chapter 2, there has been previous work on FPGA implementation of digit parallel online adders. From the literature, the existing approaches can be classified into three types:

1. Efficient mapping of the digit parallel online adder onto sophisticated FPGA resources [63, 55];

2. Addition of multiple operands by designing compressor trees based on bit counters [56, 97];

3. Modifying existing FPGA architecture for more efficient digit parallel online addition [13] and for specific applications [47].

Type 2 examines the use of counters to sum k N-digit numbers where $k > 2$. Instead, we focus on online addition with two operands which performs $Z = X+Y$, as discussed in the previous chapter. In comparison to Type 3, we attempt to obtain the best performance from mainstream FPGAs instead of suggesting ways to modify the FPGA fabric. Therefore both Type 2 and Type 3 are beyond the consideration of this thesis.

For Type 1, existing work takes advantage of the built-in carry resources in FPGAs. Conventionally the ASIC implementation of online adders is based on 4:2 compressors, as outlined in Figure 5.1 within the grey background. However, directly applying this approach in FPGAs could be less efficient. This is because there is no carry propagation between the two FAs within a 4:2 compressor, and the net delay between them can be large. Instead, Kamp *et al.* [63] and Hormigo *et al.* [55] described very similar mapping techniques for online adders with signed digit (SD) representations and carry save (CS) representations, respectively. They share the common idea of building the online adder based on the logic block (`LB`) as highlighted within the dotted circle in Figure 5.1. In this case, the fast carry logic in the FPGA can be employed, and the delay between the two 3:2 compressors can be greatly reduced.

However, we notice the major limitations of both approaches are that they only target on FPGAs with 4-input LUTs (4-LUT) and two LUTs within a logic slice, such as the Xilinx Spartan series FPGAs and the Altera Cyclone series FPGAs. This is

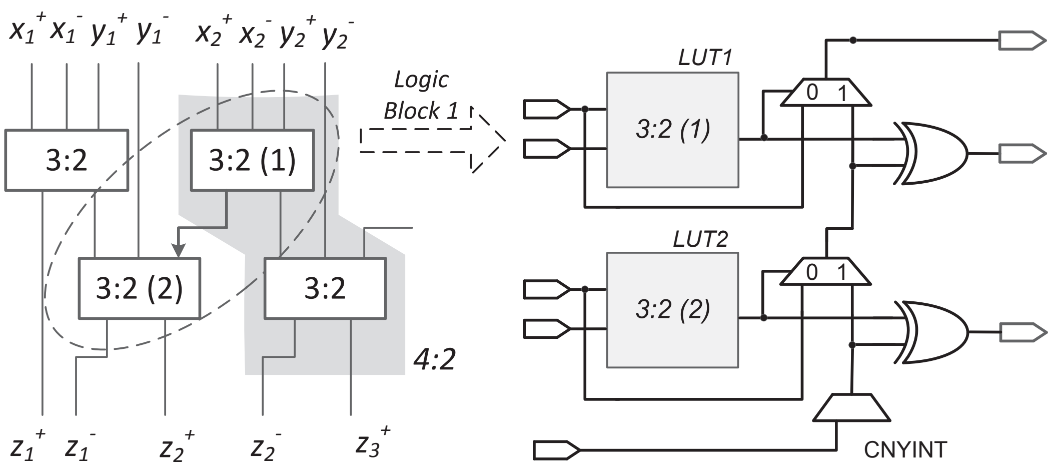

x_1^+ x_1^- y_1^+ y_1^- x_2^+ x_2^- y_2^+ y_2^-
3:2
3:2 (1)
3:2 (2)
3:2
4:2
z_1^+ z_1^- z_2^+ z_2^- z_3^+
Logic Block 1
LUT1
3:2 (1)
0 1
LUT2
3:2 (2)
0 1
CNYINT

Figure 5.1: Map the online adder onto Spartan FPGAs using the fast-carry resources. The grey background highlights the 4:2 compressor. Dotted circle indicates the logic block (LB) which can be mapped to the FPGA using the carry resources.

naturally reasonable because one `LB` can be mapped to a single slice. Nevertheless, for modern FPGAs with 6-input LUTs (6-LUT) and four LUTs in a slice, such as the Xilinx Virtex series FPGAs, all Xilinx 7 series FPGAs as well as Xilinx Zynq All programmable System-on-Chip (SoC), directly applying these approaches will result in either resource waste or wrong results. For instance, if two `LB`s are mapped to a slice with four LUTs as seen in Figure 5.2, the outputs of `LB1` should not be used because its carry input cannot be explicitly initialised. Due to the same reason, the output of `XORCY2` should not be used.

5.2.2 Proposed Mapping Method

To tackle this problem, we first present an alternative structure of the online adder to enable an efficient FPGA mapping. The structure of a 4-digit online adder is given as an example in Figure 5.3. In this equivalent structure, the first 3:2 compressor in each 4:2 compressor is split into two parts, which only generate carry and sum respectively,

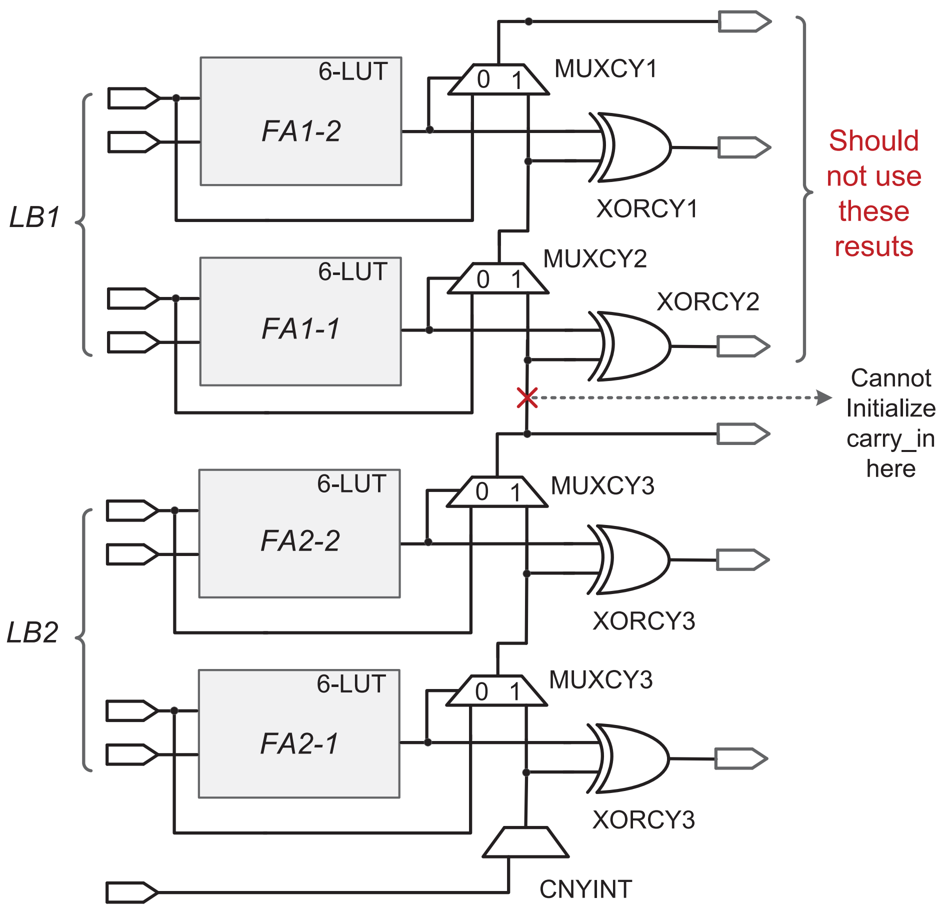


Figure 5.2: An example illustrating that the mapping method in [63] should not be directly applied on a Virtex-6 FPGA, because the carry input of `LB1` cannot be explicitly initialised.

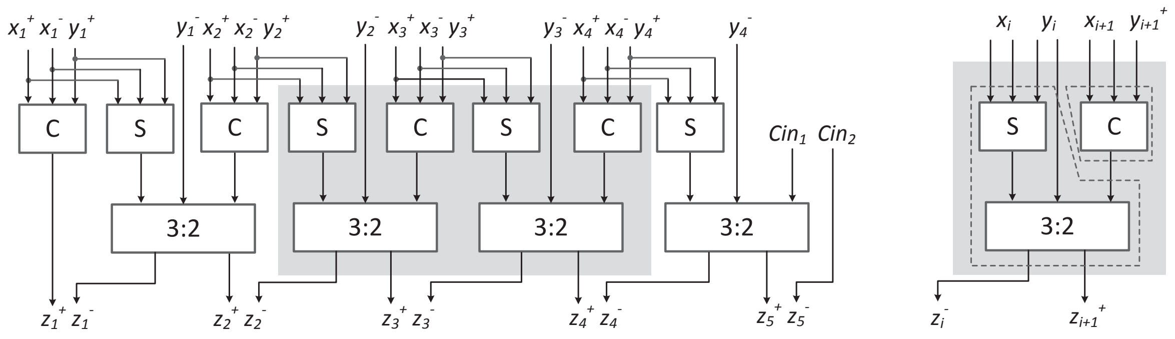


Figure 5.3: Alternative structure of online adder. Left: 4-digit online adder. The shaded part refers to the two logic blocks (LBs) that can be mapped onto one slice. Right: one LB. The dotted boxes outline the logic that can be mapped onto each LUT and the corresponding fast carry logic.

as shown on the right of Figure 5.3. In this case they can be mapped individually on two LUTs. The second 3:2 compressor, which generates the outputs, is unchanged and can be implemented using the fast carry logic.

The detailed slice mapping of the two `LB`s and the logic in each LUT are shown in Figure 5.4. The I/O signals are identical to the previous example in Figure 5.3. The 6-LUT can be configured with two different output ports `O6` and `O5`. For `LB1`, the carry input can be initialised by setting the `O6` of LUT2 to zero. Thus the output of `MUXCY2` is always `O5` of LUT2, and the carry from `LB2` will not affect the result of `LB1`. The logic to generate the MSD and the LSD can be combined into one slice as observed from Figure 5.3. Therefore using this mapping method the resources within one slice can be fully utilised, leading to a significant area reduction, while maintaining the performance advantage of online adder.

In this mapping approach, the resource usage of the online adder in terms of the number of LUTs and slices with respect to the operand word-lengths (N) can be calculated as given in (5.1) and (5.2), respectively.

$$OA\_LUT = 2N \tag{5.1}$$

$$OA\_slice = 1 + \left\lceil \frac{N-2}{2} \right\rceil \tag{5.2}$$

5.2.3 Performance Analysis

We now evaluate the performance of our proposed online adder. In our experiments, we compare the proposed online adder (`OA_new`) against the original design which is implemented using functional descriptions in Verilog HDL based on 4:2 compressors (`OA_behv`). We also compare these architectures against a ripple carry adder (`RCA`),

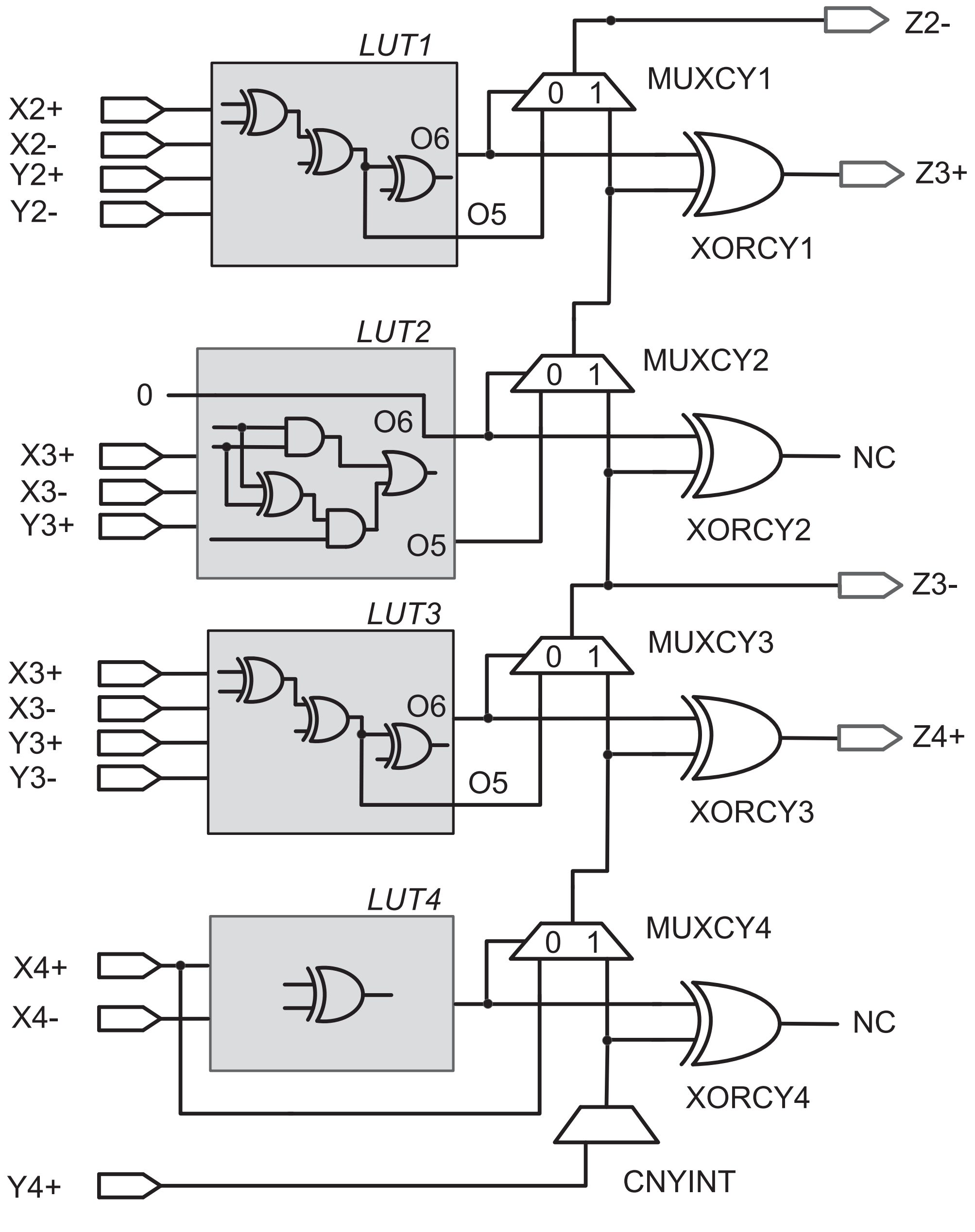


Figure 5.4: Implementation of two logic blocks (LBs) in one FPGA slice which contains four 6-LUTs. NC stands for “Not Care”.

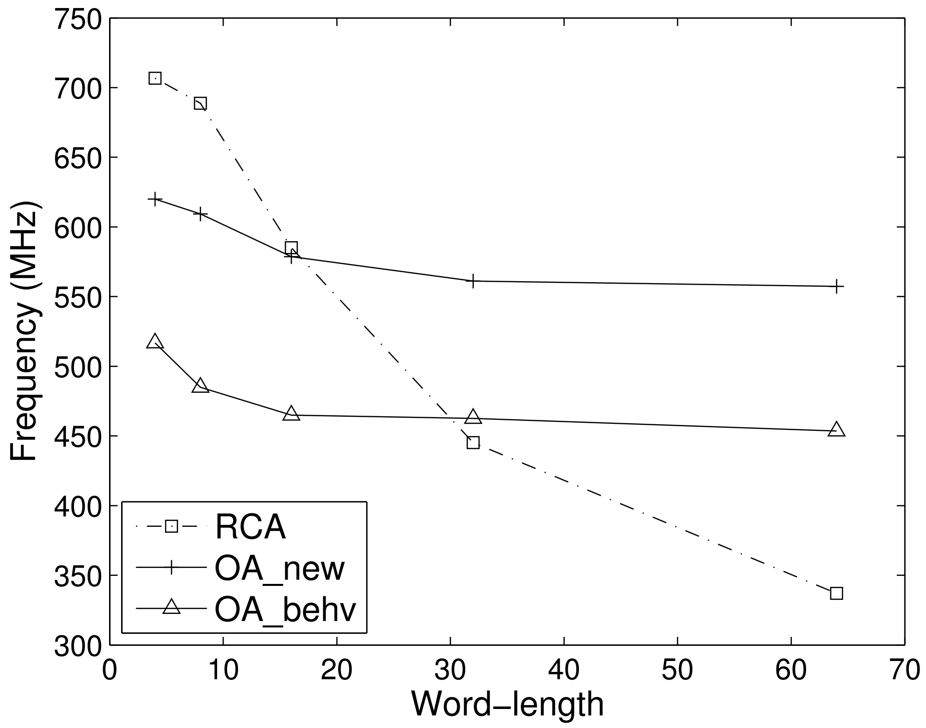


Figure 5.5: Rated frequencies of RCA and online adder with different implementation methods. The results are obtained from post place-and-route timing reports in ISE 14.7.

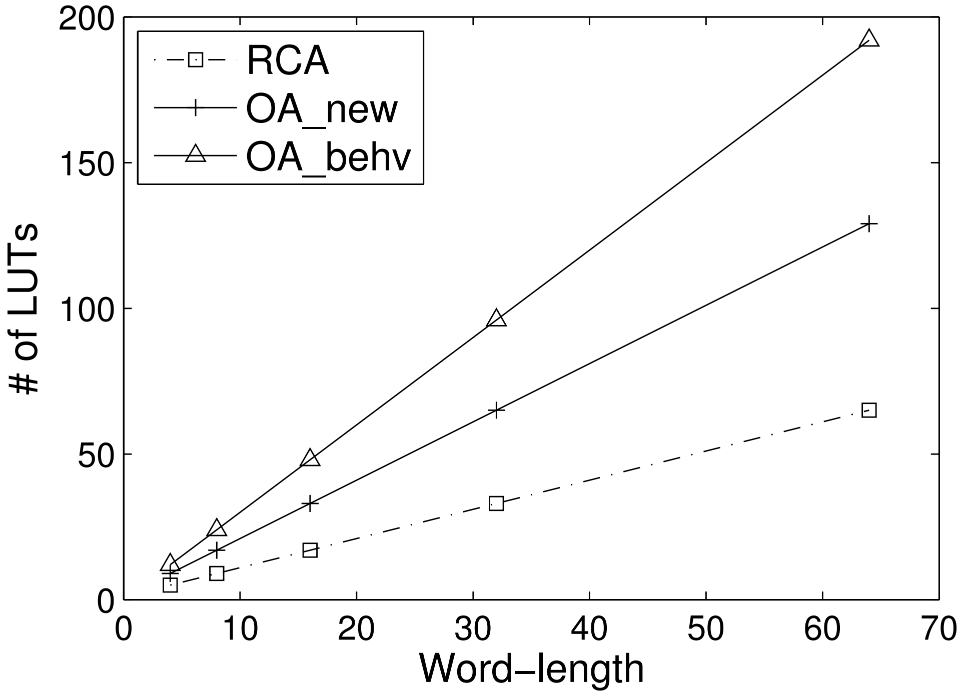
which uses conventional arithmetic. While our experiments target Xilinx Virtex-6 FPGAs (speed grade -1), our methods are applicable to any FPGAs with the same slice architecture such as all the Xilinx 7 series FPGAs. Notice that in order to utilise the carry resources and avoid logic optimisations by the synthesis tool, the Xilinx primitive component "`Carry4`" is used to create `OA_new`.

From timing reports we record the rated frequencies of all designs when varying the operand word-lengths. The results are shown in Figure 5.5. The frequency values are obtained through Xilinx Timing Analyzer after placing and routing the designs in ISE 14.7. It can be seen that both designs with online arithmetic achieve relatively stable operating frequencies across a variety of operand word-lengths. This is as expected, because the critical path delay of the online adder is theoretically independent of the operand precision. In comparison, the rated frequency of the `RCA` drops dramatically for larger word-lengths. It follows that both online adders outperform `RCA` for large word-lengths. More specifically, the frequency of `OA_behv` is faster than `RCA` when $N \geqslant 30$, whereas our new online adder outperforms `RCA` for $N > 16$. As a result, this makes the `OA_new` more attractive for a wider range of applications. Furthermore, the speed-ups of `OA_new` against `OA_behv` are over 1.2×.

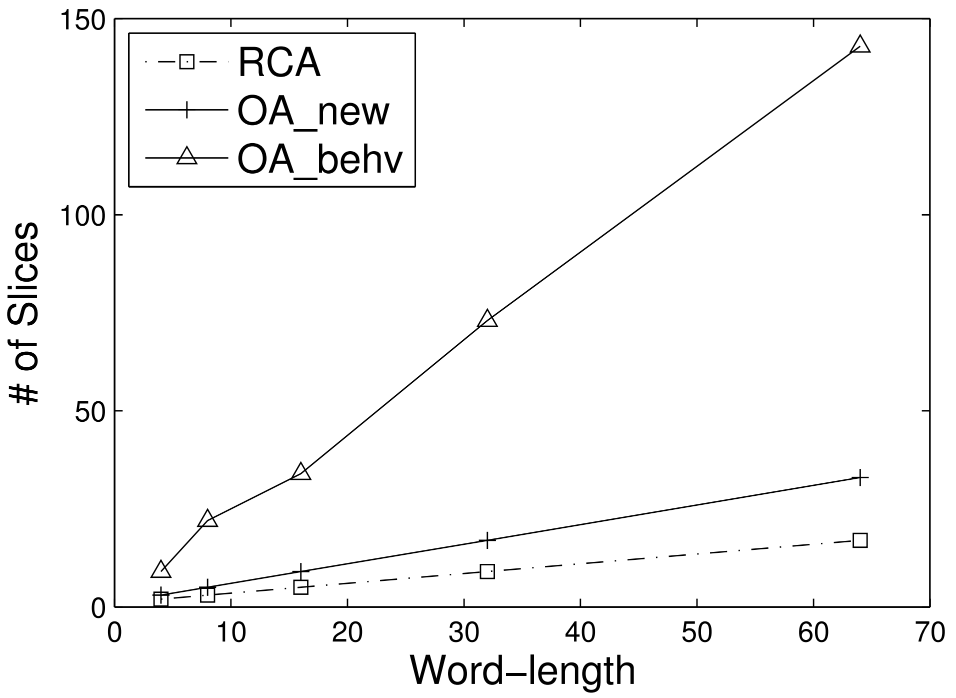
We also compare the resource usage of all three adder designs with respect to a variety of operand word-lengths. The results are presented in Figure 5.6. We can see that in comparison to the original `OA_behv`, our implementation `OA_new` achieves significant area savings from 25% to 33% in LUTs and from 67% to 77% in slices. This is because the dedicated FPGA resources are fully utilised. These area savings reduce the overhead of an `OA` over an `RCA` from 2.95× to 1.98× for LUTs, and from 8.41× to 1.88× for slices. It is unlikely that the overhead can be completely avoided when using an `OA` because of the usage of a redundant number system. Nevertheless, with the reduced overhead, using `OA` to take advantage of the graceful degradation when operating beyond the deterministic clocking region may become more attractive for an FPGA developer.

5.3 Design Trade-off Evaluation for Different Adders

In this section, we expand the previous evaluation of optimal adders in Chapter 3, by adding our newly proposed implementation of online adder in to the comparison. We compare different forms of adders when taking into account various design trade-offs such as accuracy, latency and area.



(a) Usage of LUTs



(b) Usage of slices

Figure 5.6: Area comparisons of different binary adder implementations with respect to a variety of operand word-lengths.

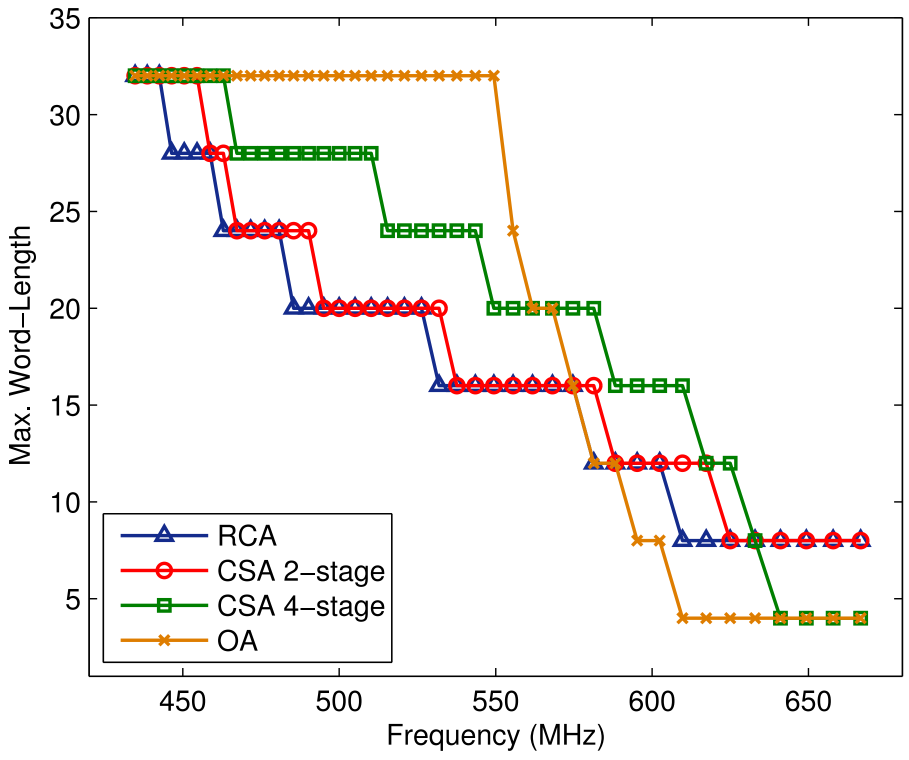


Figure 5.7: The maximum word-lengths of different adder structures with respect to a variety of frequency requirements.

5.3.1 Accuracy, Latency and Area Trade-offs for Adders

Firstly, we demonstrate the benefits of CSA and OA in accuracy and performance, as well as the area overhead in comparison to RCA. As an example, the maximum word-lengths of RCA, CSA and OA with respect to a variety of operating frequencies are illustrated in Figure 5.7. The results are obtained from post place-and-route simulations in ISE 14.7. In this experiment, circuits are operated with timing closure, *i.e.* the conventional design scenario is evaluated for all structures. We consider the operand word-length of adders ranging from 32-digit to 4-digit with a step of 4-digit.

As can be seen in Figure 5.7, for a relatively relaxed frequency requirement (*e.g.* at 470 MHz), both CSA and OA can be implemented with greater word-lengths than RCA. For CSA, this is because the stage parallelism enables a larger word-length, even

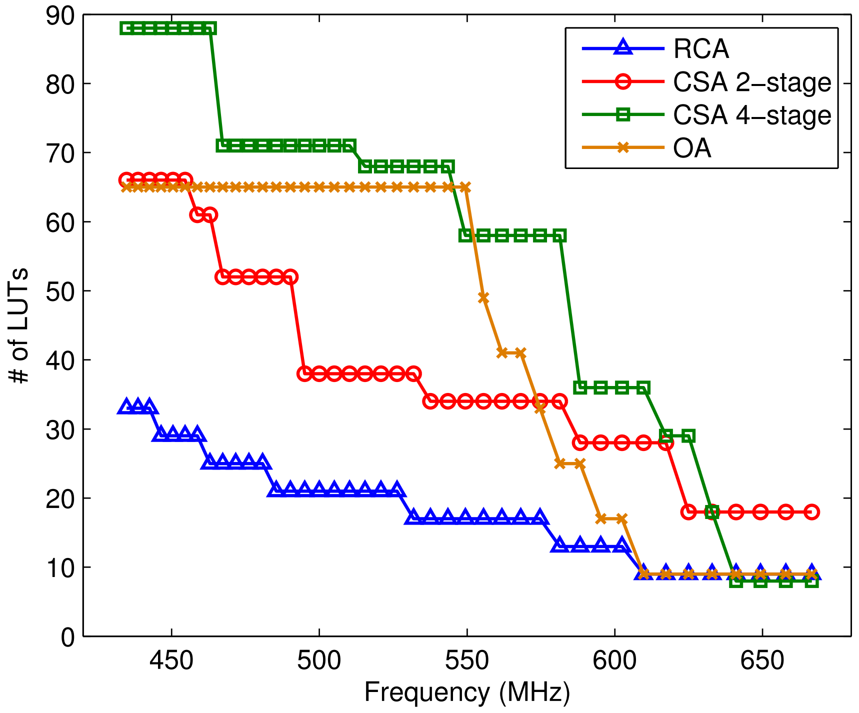


Figure 5.8: Area consumption with respect to a variety of frequencies of different adder structures.

though the multiplexer delay limits the precision of each CSA stage in comparison to RCA. For OA, an even larger gap in word-length can be observed when compared to RCA. Additionally, unlike RCA and CSA of which the maximum word-length drops gradually with the increment of frequency, OA maintains full precision across a large range of frequencies. This is as expected, because the critical path delay of OA is theoretically irrelevant to the operand word-length as discussed in Section 4.2.1. However, we also observe that for a large operating frequency, the word-length of OA drops drastically because OA performs computations in parallel, and if it is operated beyond the maximum frequency, most output digits are likely to be affected. Besides, CSA cannot be implemented with larger precision than RCA either, because in this case the multiplexer delay becomes comparable to the delay of the carry chain for CSA, and it inhibits the benefits of parallelism.

We also record the corresponding area consumption of each adder structure in terms of the usage of LUTs as depicted in Figure 5.8. It can be seen that the CSA with four stages and two stages costs up to $4.5\times$ and $3.1\times$ area than RCA, respectively. OA is also up to $3.8\times$ larger than RCA.

To combine the results seen in Figure 5.7 and Figure 5.8, it is not straightforward to decide which adder structure is optimum for given design specifications on accuracy, performance and area consumption. Therefore, it is necessary to provide comprehensive evaluations of different adder structures to quantify the trade-offs between these design constraints, as will be detailed in the following section.

5.3.2 Evaluation of Optimum Adder Structure

In this section, we evaluate three types of adders: RCA, CSA (with two stages and four stages) and OA. All adders are implemented in the digit parallel form with the original word-length of 32 digits. Similarly to Chapter 3, our evaluation is performed when considering either of the following two situations:

1. Operate the circuit at a given frequency within a specific area budget, while achieving the minimum output error;
2. Operate the circuit at the fastest frequency with a minimum resource usage, under a given error budget.

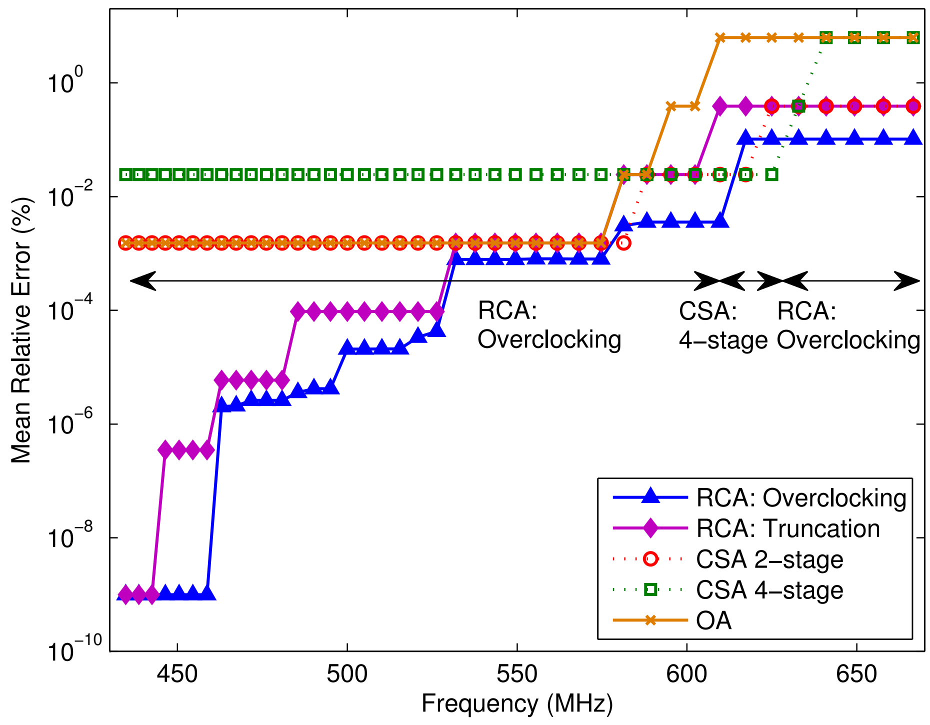
In both cases, decisions must be made on which adder structure achieves this minimum, and which design scenario should be adopted. Again in our experiments, the inputs are set to randomly generated data following uniform distribution.

Optimum Adder with Given Frequency and Area Requirements

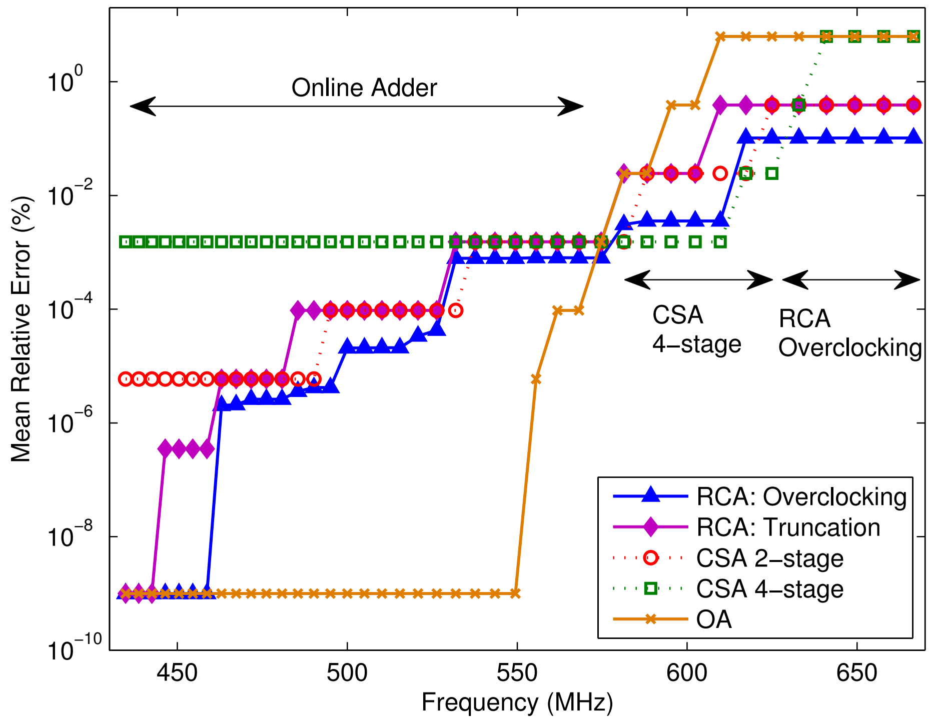
For this type of application, the available area and expected operating frequency are given at the design time. Again we aim to provide the optimum design approach when considering this trade-off, instead of ranking of all possible design approaches. As an example slice through the design space, in Figure 5.9 we record the mean relative error (MRE) with respect to a range of operating frequencies for different design scenarios, when the area budget is set to 35 LUTs and 55 LUTs separately. The results are obtained from post place-and-route simulations in ISE 14.7. MRE can be calculated as given by (3.26) in Chapter 3, and it represents the ratio between the mean value of error and mean value of outputs. Notice that the optimum adder design metric that achieves the minimum MRE is labelled.

In Figure 5.9(a), the area budget is set to 35 LUTs. For all frequency values, the overclocked RCA achieves no larger MRE than the RCA with truncated operand word-lengths. This is in accordance with the analysis in Section 3.2 of Chapter 3. We also notice that both CSA and OA cannot be implemented with full precision due to area limitation, and this leads to large truncation errors. Despite the CSA with four stages is best for some frequencies, in general the overclocked RCA is the optimum design for most frequency requirements. In this case, the actual difference between the overclocked RCA and the second best design ranges from 1.4× to 26.5× in terms of MRE reduction.

However if the area budget is released to 55 LUTs, OA can be implemented with the original word-length. Additionally CSA can also be implemented with larger precision, but still with truncation errors. Therefore as shown in Figure 5.9(b), OA serves as the optimum design with respect to a wide range of frequency requirements. In this case at least 8.4× MRE reduction can be achieved by OA in comparison to the second best



(a) Available LUT=35.



(b) Available LUT=55.

Figure 5.9: Two examples of comparisons between different design scenarios and adder implementations with limited area budget. Design scenario with minimum error is labelled.

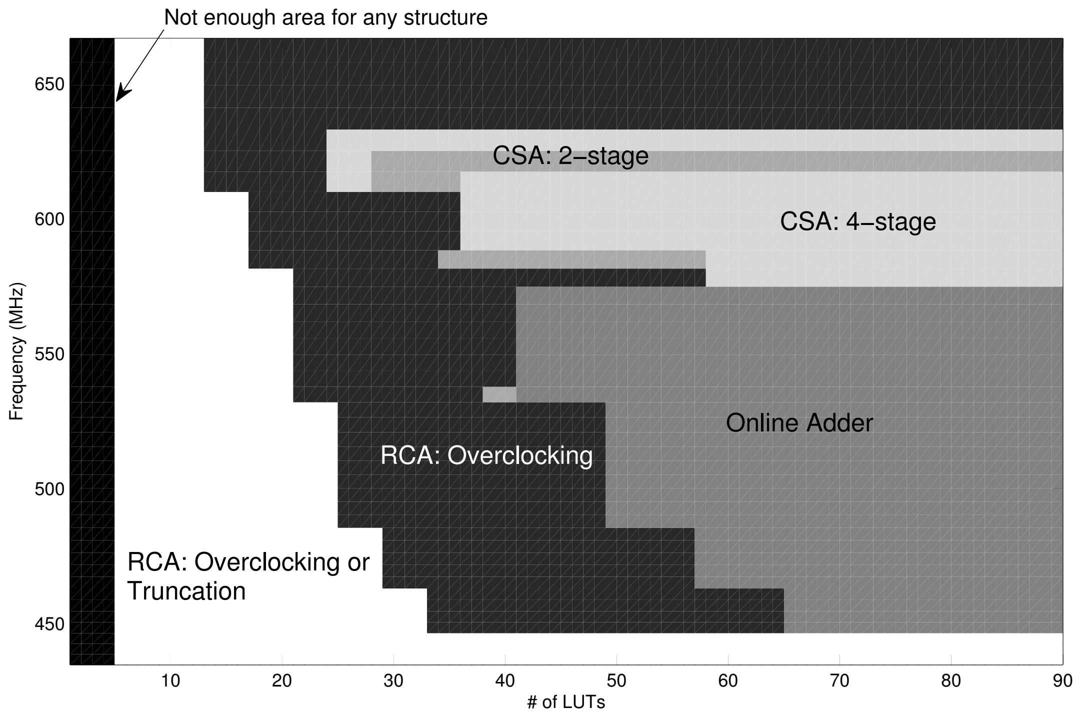


Figure 5.10: Mapping of the optimum design metric of adders, which achieves minimum error with respect to a variety of frequency and area constraints.

design. For higher frequencies, the MRE of OA increases rapidly, and CSA with four stages outperforms both OA and RCA for higher frequencies. Similar to Figure 5.9(a), the overclocked RCA is the optimum design for even higher frequencies, because in this case the CSA can only be implemented with small precisions, and the multiplexer delay limits the benefits of parallelism.

The results shown in Figure 5.9 can be further extended by testing with a variety of area constraints. In this case, we can map the optimum design scenarios into the design space of different operating frequencies and area consumptions, as shown in Figure 5.10. In general this graph can be divided into several regions for in-depth analysis. Firstly, if the frequency requirement is moderate whilst the area budget is large enough to implement an OA in full precision, it will be the optimum design.

This is because long carry chains are eliminated from OA such that it can maintain full precision for a wide range of frequency values. Also this is in accordance with the results in Figure 5.9(b).

Secondly, CSA is a better design choice for high frequency requirements and large area budget, because it is originally designed for high speed operation. Besides, each stage of CSA is implemented in the FPGA with fast carry logic [135], hence it runs even faster than OA. In addition, in this region we notice that the 2-stage CSA is better than the 4-stage CSA for 610 MHz. This can also be observed from Figure 5.9(b) that the results of these two designs are overlapped for this frequency value. In this case, since the 2-stage CSA costs less silicon area, it serves the optimum design.

Thirdly, for a tighter area budget only part of OA and CSA can be implemented, whereas RCA still keeps original precision. In this situation, area becomes the limitation factor. Both OA and CSA generate truncation errors, which are greater than the overclocking error of RCA. Therefore, the overclocked RCA is the optimum design option across almost the entire frequency domain. Furthermore, the precision of RCA is also limited under very stringent area constraints. This causes truncation errors for every design option. Nevertheless, RCA with either overclocking or truncation scenario equally serves to be optimal, because it maintains more precision than OA and CSA for a specific area budget.

Optimum Adder with Given Accuracy and Area Budgets

If the circuit is designed to operate as fast as possible with minimum area, whilst a certain amount of inaccuracy can be tolerated, the optimum adder design metric can also be determined as illustrated in Figure 5.11. In this set of experiments, the error budget is evaluated in terms of MRE, which varies from $10^{-6}\%$ to 10%. For a given

MRE, the optimum adder is selected with maximum operating frequency. In this case, if multiple designs can operate with the same frequency, the optimum design is then selected based on actual area consumption within the area constraint.

Similar to the analysis in Section 5.3.2, for tight accuracy requirement and large area budget, OA is the optimum design choice because it keeps full precision while running at high frequencies. If relaxing the accuracy requirement (*e.g.* 0.01%), CSA gradually outperforms because its precision loss is less than that of OA. Once again, if area budget is tightened, the RCA becomes the best design choice, because the precision of OA and CSA is largely limited. Furthermore, when the accuracy requirement is released (*e.g.* over 1%), RCA can operate with the fastest frequency across most area constraints.

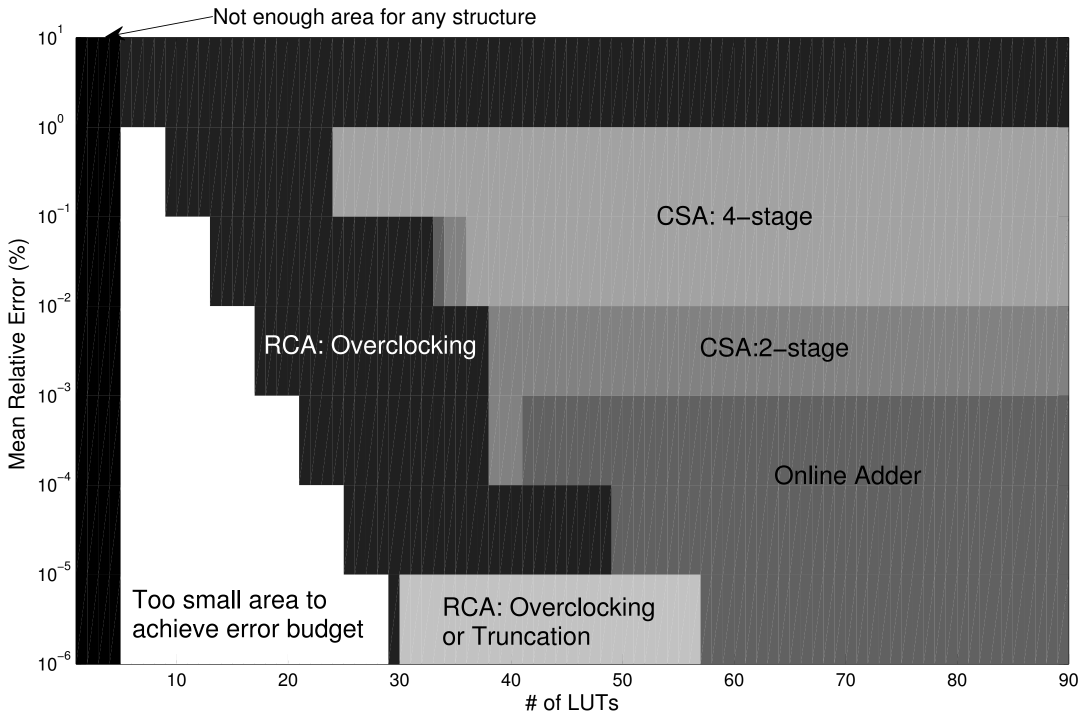


Figure 5.11: Mapping of optimum design metric of adders, which achieves highest frequency with respect to a variety of accuracy and area constraints.

5.4 Digit Parallel Online Multiplier for FPGAs

In this section, we investigate an area-efficient FPGA implementation of digit parallel online multiplier. In addition, we introduce a method to implement correctly rounded online multipliers for a chosen output precision, together with the area-efficient FPGA implementation for this structure.

5.4.1 Digit Parallel Online Multiplication Algorithm

The binary online multiplication algorithm and its digit parallel implementation are discussed in the previous chapter. For ease of discussion, the algorithm is re-listed below, where the selection function is given in (5.3).

Algorithm 3 Online Multiplication

```
1: Initialization: $X_{[-\delta]} = Y_{[-\delta]} = P_{[-\delta]} = 0$
2: for $j = -\delta, \ -\delta+1, \ \cdots, \ 2N-1$ do
3:     $H_{[j]} \leftarrow r^{-\delta}\left(x_{j+\delta+1} \cdot Y_{[j+1]} + y_{j+\delta+1} \cdot X_{[j]}\right)$
4:     $W_{[j]} \leftarrow P_{[j]} + H_{[j]}$
5:     $z_j \leftarrow sel(W_{[j]})$
6:     $P_{[j+1]} \leftarrow r\left(W_{[j]} - Z_{[j]}\right)$
7: end for
```

$$sel(W_{[j]}) = \begin{cases} 1 & \text{if } W_{[j]} \geqslant \frac{1}{2} \\ 0 & \text{if } -\frac{1}{2} \leqslant W_{[j]} < \frac{1}{2} \\ \overline{1} & \text{if } W_{[j]} < -\frac{1}{2} \end{cases} \tag{5.3}$$

This algorithm can be synthesized into a unrolled digit parallel structure, as examined in the previous chapter. We have also demonstrated that the digit-parallel online

Algorithm 4 Digit Parallel Online Multiplication

```
1: Initialization: $P_{[0]} = 0$
2: for $j = 1, 2, \cdots, N$ do
3:     $Xy_j \leftarrow X \cdot y_j$
4:     $W_{[j]} \leftarrow P_{[j-1]} + Xy_j$
5:     $z_j \leftarrow sel(W_{[j]})$
6:     $P_{[j+1]} \leftarrow r\left(W_{[j]} - Z_{[j]}\right)$
7: end for
8: $Z_{[N+1:2N]} \leftarrow frac(W_{[N]})$
```

multiplier (OM) is also more tolerant to timing violations in comparison to the multiplier with conventional arithmetic, because timing errors initially affect the LSDs and this will lead to smaller error magnitude.

However, we have noticed that significant area overhead is required in order to implement this type of OM, generally for three main reasons. First, in comparison to the conventional multiplier, more iteration stages are needed to generate same precision of output. For instance in an N-digit OM, $(2N + \delta)$ iterations are required to generate $2N$ digits outputs, or $(N + \delta)$ iterations are required to generate the most significant N digits. Second, the logic of each stage is complex as it involves operations such as online addition, digit-vector multiplication (to generate $x_{j+\delta+1} \cdot Y_{[j+1]}$ and $y_{j+\delta+1} \cdot X_{[j]}$ in Algorithm 3), shifting and other combinational logic blocks such as the selection function. Also, the word-length of signals $H_{[j]}$, $W_{[j]}$ and $P_{[j]}$ is $(N + 2 + \delta)$, because the selection function needs two integer digits and one fractional digit to generate z_j, as described in Section 4.2 of Chapter 4. Third, this OM architecture is not optimised specifically for FPGA technology, such that the dedicated logic resources within an FPGA such as the fast carry logic are not efficiently utilised.

For the first two design issues, we address them by providing a modified form of online multiplication algorithm. Instead of simply implementing the unrolled version of Algorithm 3, it is optimised specifically for digit parallel operations. In the original online multiplication algorithm, because the input data is available in a digit serial fashion, the online delay δ is used to accumulate enough input digits to generate the MSD of the result. However for digit parallel operation, normally all digits of the inputs will be available simultaneously. In this case the online delay δ is no longer necessary. For the same reason, $H[j]$, which takes one digit of each input per stage as shown in the original algorithm, can be optimised to take one partial product Xy_j or Yx_j per stage. To summarise, the modified algorithm for digit parallel online multiplication is described in Algorithm 4, where N denotes the operand word-length, function $frac(W_{[N]})$ refers to the fractional digits of $W_{[N]}$, and the selection function $sel()$ is unchanged as given in (5.3) .

In comparison to Algorithm 3, the maximum number of stages required to generate results with full precision drops by a factor of two using the proposed new algorithm. In each stage, the word-length of all signals also reduces from $(N + 2 + \delta)$ to $(N + 2)$, as δ is not necessary for digit parallel operations. In addition, the $frac()$ function can be implemented only based on wire connections without using logic resources.

5.4.2 FPGA Implementation

We now address the third aforementioned design issue by presenting an area efficient implementation of Algorithm 4 on modern FPGAs. The general structure of a 4-digit `OM` using the proposed algorithm is shown in Figure 5.12(a), and the structure of a single stage j is shown in Figure 5.12(b) with the word-lengths of all signals labelled. Area optimisations can be performed on all modules. It can be seen that in each stage, an $(N + 2)$ digit online adder is used to derive $W_{[j]}$. Our proposed online adder architecture can be employed. The selection logic takes three input digits (six binary bits) and generates one output digit (two binary bits). Hence it can be implemented using two 6-LUTs for each output bit. Similarly the generation of $P_{[j]}$ can also be

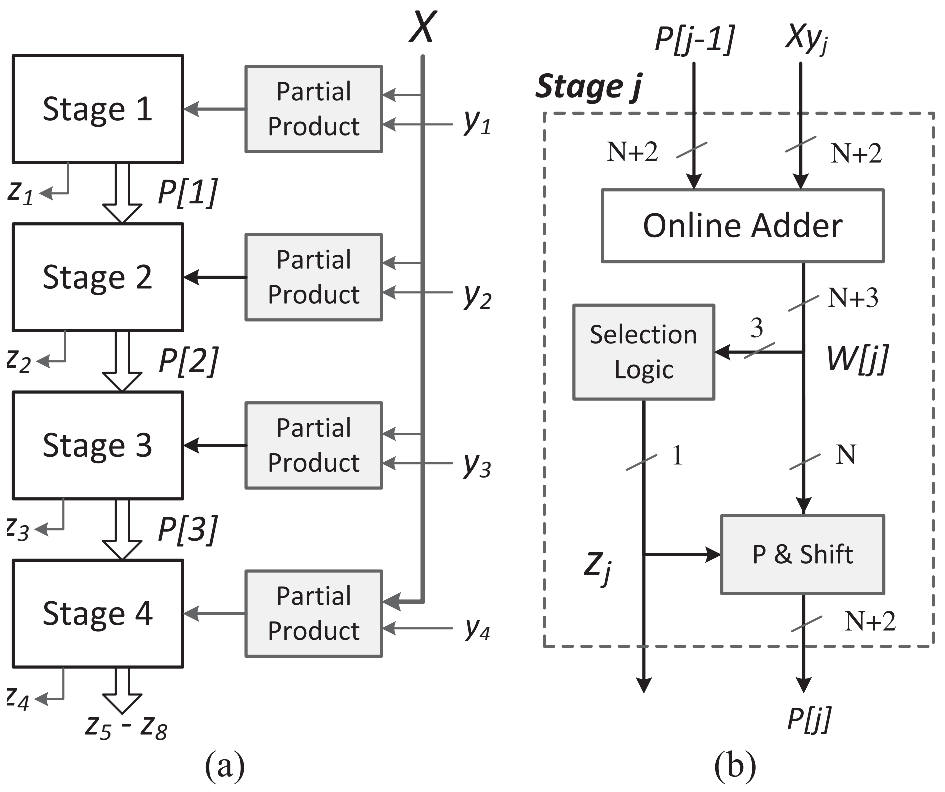


Figure 5.12: (a) Structure of a 4-digit online multiplier using the proposed algorithm. (b) Structure of Stage j. The word-length of all signals are labelled in terms of the number of digits. N denotes the word-length of the input signals.

implemented using one 6-LUT, since only the integer digits of $W_{[j]}$ need to be modified due to the selection of $Z_{[j]}$. In addition, the structure of Stage one in Figure 5.12(a) can be further optimised by removing the online adder, because $P_{[0]} = 0$.

The blocks that generate partial products Xy_j can be incorporated into the online adder for further area reduction. In the online adder structure as presented in Figure 5.4, we notice that only four inputs per 6-LUT are used. This leaves at least two available inputs per LUT. Originally in order to generate one digit of partial product, one digit of inputs X and Y are required respectively, as indicated in the left of Figure 5.13. Alternatively, instead of using extra logic to generate Xy_j, it can be merged into the corresponding LUTs in the online adder by fully utilising all six LUT inputs, as shown on the right of Figure 5.13. Notice that this approach is only available for

FPGAs with 6-LUTs.

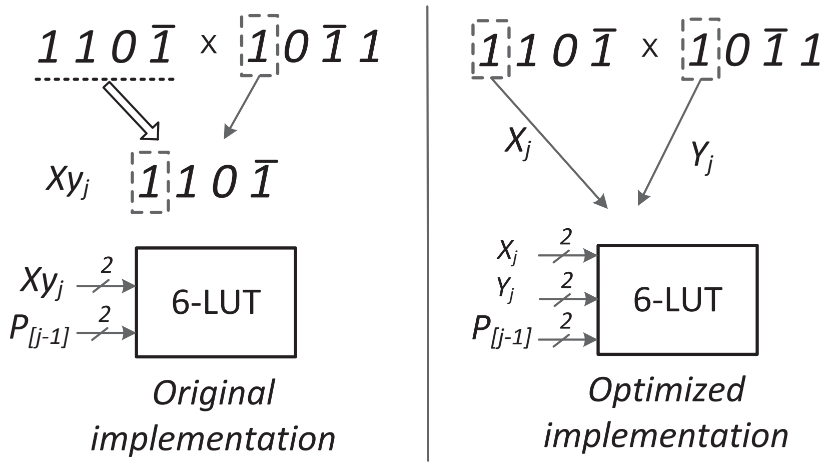


Figure 5.13: Left: Direct implementation with extra logic to generate partial products. Right: Combining the logic blocks that generate partial products into the Online Adder by fully utilizing all the inputs of the 6-LUTs.

The resource usage of the proposed `OM` can be calculated as follows. In an N-digit `OM`, in total $(N-1)$ online adders are needed. According to Algorithm 4, the word-length of each online adder is $(N+2)$-digit (two integer digits and N fractional digits). Therefore based on (5.1) and (5.2), the number of LUTs and the number of slices used by the `OM` is given in (5.4) and (5.5), respectively. Note that L_{S1} and S_{S1} denote the number of LUTs and slices used by the first stage, which is built without online adders.

$$OM\_LUT = 2(N+2)(N-1) + L_{S1} \tag{5.4}$$

$$OM\_slice = \left(1 + \left\lceil \frac{N}{2} \right\rceil\right)(N-1) + S_{S1} \tag{5.5}$$

5.4.3 Structure Optimisation for Half Precision Results

In practical applications, normally the multiplier is connected with other arithmetic operators. If the outputs of a multiplier is utilised for subsequent operations, and a

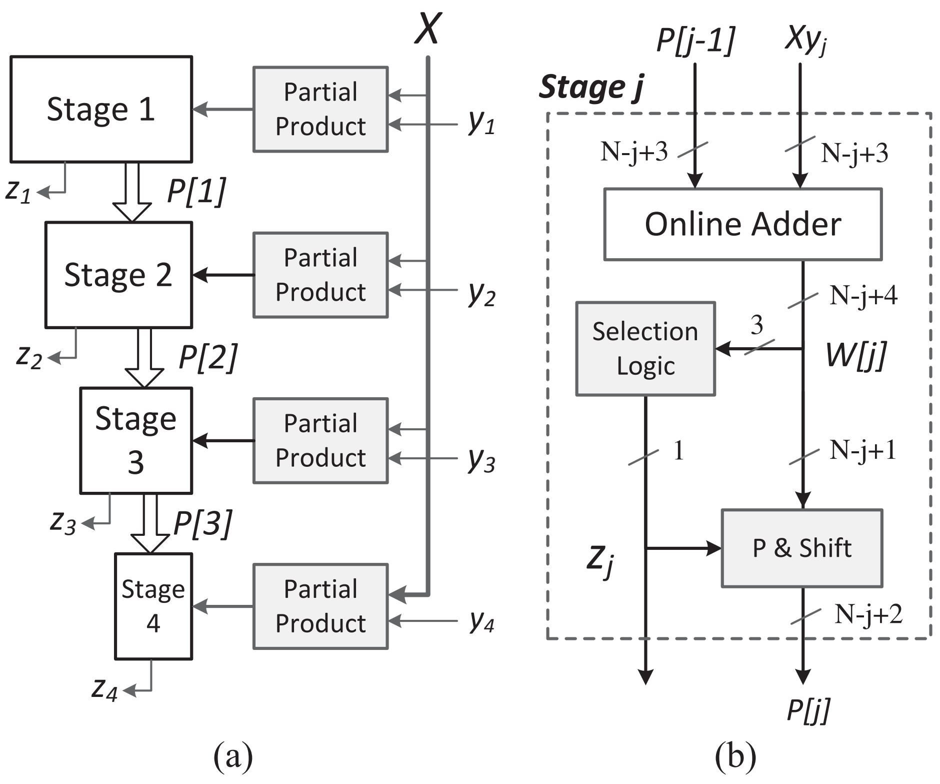


Figure 5.14: (a) Modified structure of a 4-digit online multiplier which only generates the most significant 4-digit result. (b) Structure of Stage j, with the word-lengths of all internal signals labelled.

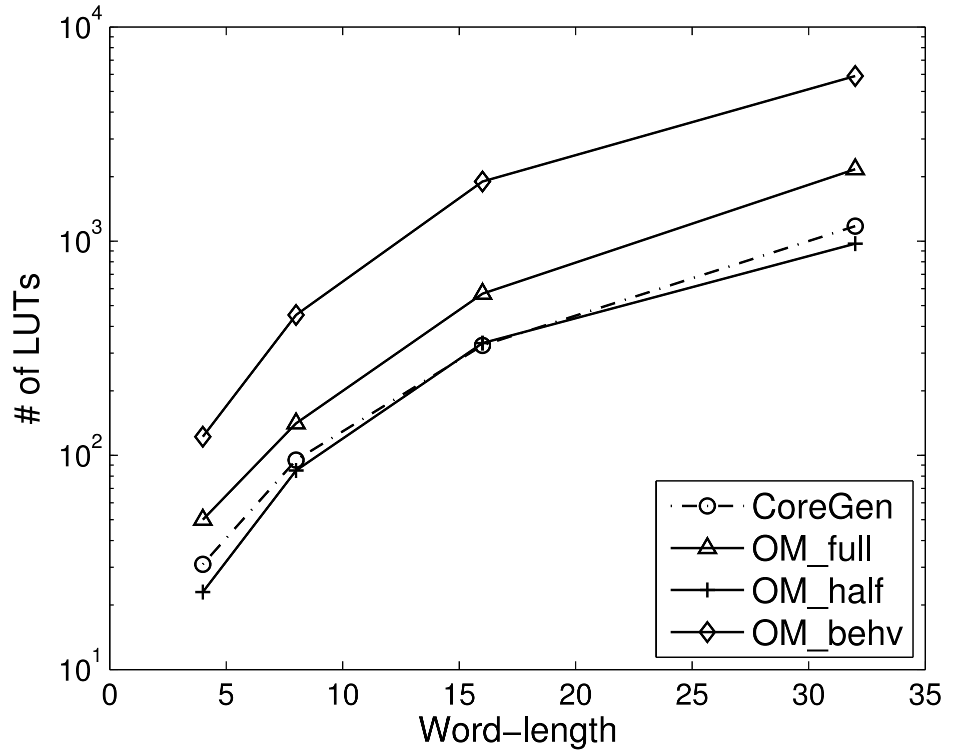
consistent word-length is used throughout the system, then only the most significant half of the product is required. In a conventional multiplier with standard binary arithmetic, this is achieved by either truncating or rounding the least significant half of the products. However, both the computation time and the structure remains unchanged, because the results are generated from LSDs using conventional arithmetic.

In comparison, the `OM` offers the flexibility to simplify the structure corresponding to the required precision. This is possible because in an `OM`, the product digits are generated initially from the MSD, and there is no carry propagation from the LSD to the MSD with the employment of the redundant number system. This will potentially lead to a more area efficient design. For example, the modified structure of a 4-digit `OM` is illustrated in Figure 5.14(a). The word-length of signals within each stage can be correspondingly reduced, as shown in Figure 5.14(b). It can be seen that instead of keeping identical word-length throughout the stages (in Figure 5.12), in the modified structure the signal word-lengths depend on the stage number j and fewer bits of signals are needed for LSD stages.

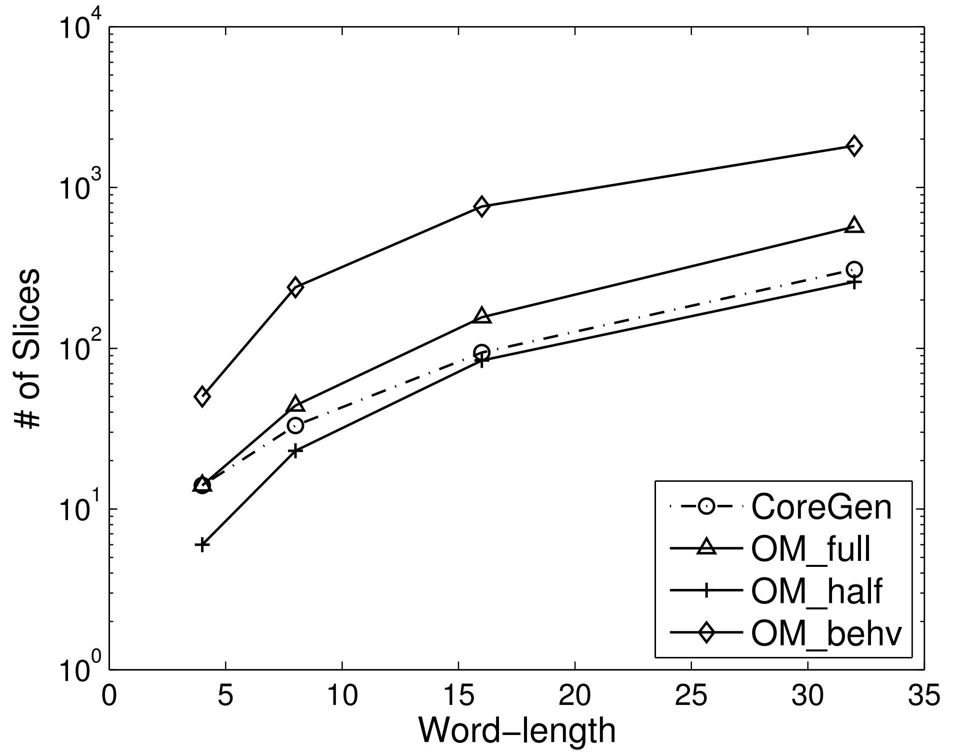
5.4.4 Performance Analysis

Similar to the experiments as described in Section 5.2.3, we compare the behavioural RTL implementation of online multiplier (`OM_behv`) against the proposed two types of multiplier, which are implemented based on Algorithm 4 to generate products with full precision (`OM_full`) and half precision (`OM_half`), respectively. In addition, we also compare `OM` with multiplier implemented with conventional binary arithmetic. The conventional multiplier is created using the Xilinx Core Generator with speed optimisation (`CoreGen`), and it is implemented based on LUTs without DSPs for a fairer comparison.

The comparison of area in terms of the cost of LUTs and slices with respect to different operand word-lengths are shown in Figure 5.15. The results are obtained from ISE 14.7 after mapping the design to the Virtex-6 FPGA. In comparison to `OM_behv`, significant area savings have been obtained by the optimised design `OM_full`. Specifically, the reductions of LUTs and slice utilisations vary from 59.0% to 70.1% and from 68.6% to 81.7%for different operand word-lengths, respectively. In comparison to the `CoreGen` multiplier, the area overhead of online multipliers drops significantly using the proposed new structure. In order to generate full precision product, the area overhead of `OM_behv` against `CoreGen` are 3.94× to 5.82× for LUTs, and 3.58× to 8.11× for slices. Using the proposed structure `OM_full` the area overheads drop to 1.48× to 1.83× for LUTs and up to 1.84× for slices. Furthermore, a traditional multiplier is unable to compute this correctly rounded output using less silicon area. In



(a) Usage of LUTs



(b) Usage of slices

Figure 5.15: Area comparisons of different types of binary multipliers.

contrast, using our approach we can trade area for precision and tune output precision to be the minimum desired. As an example, if only half of the output precision is required, `OM_half` can save silicon area over the traditional `CoreGen` multiplier. The area saving varies from 40% to 55% for LUTs, and 46% to 56% for slices.

We also check the rated frequencies of all multipliers with respect to a variety of operand word-lengths. The frequency values are plotted in Figure 5.16, which are still obtained from the post place-and-route timing reports. In Figure 5.16 we see that in comparison to `OM_behv`, our proposed architectures achieve frequency speed-ups varying from 1.49× to 1.98×.

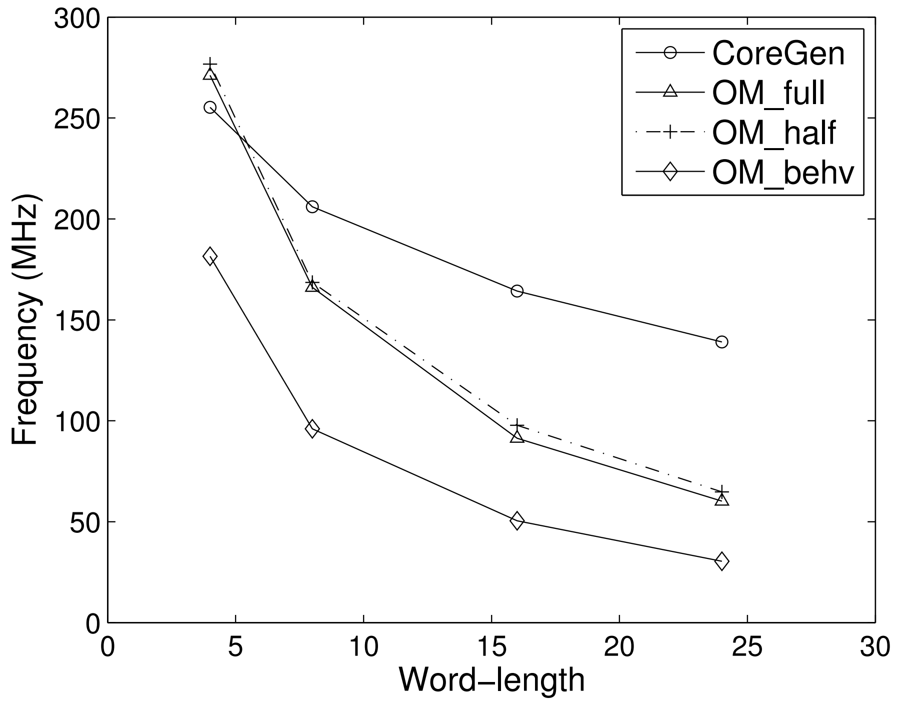


Figure 5.16: Rated frequencies of different types of binary multipliers for a variety of operand word-lengths.

We comment that there remains a large gap in terms of frequency between the proposed online multipliers and the `CoreGen` multiplier, especially for large operand word-lengths. This is because in the online multiplier, even with the carry resources in the FPGA being used efficiently, each 4-digit carry logic within a slice is divided into two parts, as shown previously in Section 5.2. Therefore the critical path delay of an `OM` is determined by both the carry logic delays, and the net delay among different stages, which is significant for large operand word-lengths. This also explains why `OM_full` and `OM_half` run at similar frequencies, because the critical path of both designs are identical in theory.

However, we can take advantage of the "overclocking friendly" feature of the online arithmetic, and push the previous tests one step further by operating the circuits beyond their deterministic region while allowing timing violations to happen. We adopt the same evaluation metric as used in previous chapters and examine the overclocking behaviour of all three types of multipliers in terms of the mean relative error (`MRE`) as given in (3.26).

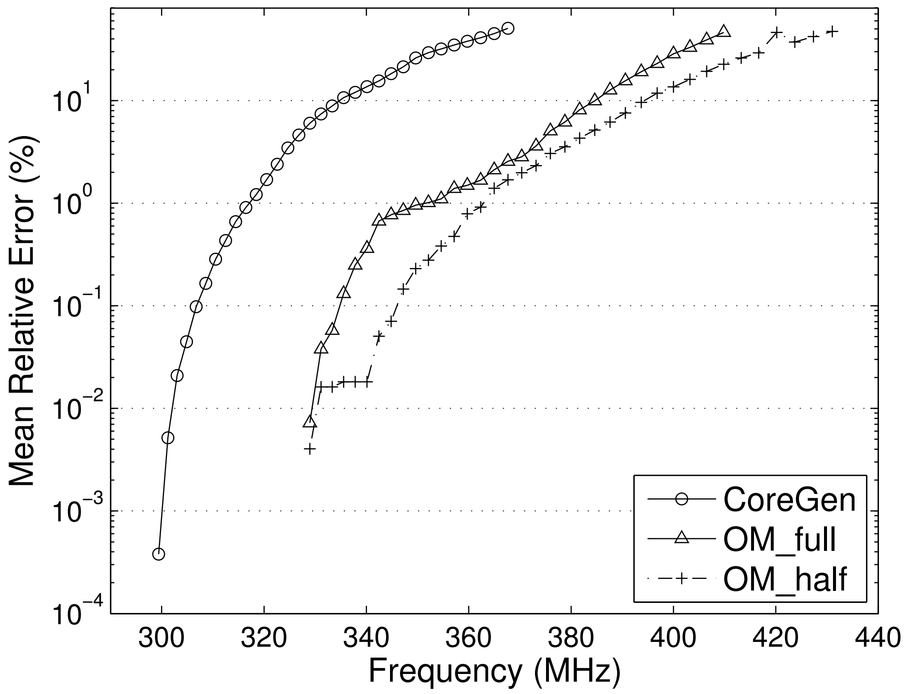


Figure 5.17: Mean relative errors seen at the outputs of different types of binary multipliers, when clocked with faster-than-rated frequencies.

For instance, Figure 5.17 demonstrates the `MRE` values of the 8-digit multipliers. The results are from post place-and-route simulations targeting Xilinx Virtex 6 FPGA, and the input stimulus are randomly sampled from a uniform distribution of 8-digit numbers. We see that although the timing analysis tool predicts slower frequencies for the `OM`s, they actually operate faster than `CoreGen` without the occurrence of timing errors. This is in accordance with the results seen in the previous chapter.

5.5 Results from FPGA Measurements

In this section, we evaluate the performance of online arithmetic operators by performing experiments using the FPGA test platform, as described in Section 3.7 of Chapter 3.

5.5.1 DSP Benchmark Circuits

We first implement the DSP benchmark circuits as listed in Table 3.3 of Chapter 3 using online arithmetic operators such as adder and constant coefficient multiplier. The rated operating frequencies of designs with online arithmetic are recorded according to the post place-and-route timing reports from Xilinx ISE 14.7. Table 5.1 summarises the rated frequencies of designs with these two types of computer arithmetic. It can be seen that designs with online arithmetic achieve constantly higher rated frequencies in comparison to design implemented with traditional arithmetic. The speed-ups in terms of rated frequency range from 1.45$\times$ to 2.83$\times$.

Similar to previous chapters, we also perform experiments on FPGA platform with overclocking to quantify both the reduction of `MRE` and the frequency speed-ups with online arithmetic, for given frequency specifications and given error budgets that can be tolerated, respectively. Again we use two types of input data: the uniform independent data (UI) and real data from several image benchmarks.

Table 5.1: Rated Frequencies of Example Designs with Traditional Arithmetic and Online Arithmetic.

Design	**Description**	**Frequency (MHz)** Trad. Arith.	Online Arith.	**Speed-up**
FIR Filter	5th order	126	356	2.83×
Sobel Edge	3×3 Detector	197	457	2.32×
IIR Filter	7th order	140	277	1.98×
Butterworth Filter	9th order	117	289	2.47×
DCT	4-point	177	255	1.45×

If given frequency specifications are given, the reduction of MRE with online arithmetic can be calculated by (5.6) where MRE_{trad} and MRE_{online} denote the MRE of designs with traditional arithmetic and online arithmetic, respectively.

$$\frac{MRE_{trad} - MRE_{online}}{MRE_{trad}} \times 100\% \tag{5.6}$$

For each design, the relative reduction of MRE obtained from FPGA measurements are summarised from Table 5.2 to Table 5.6. In each table, we use the normalised frequency with respect to the maximum error-free frequency for the corresponding design when the input signal is 8-digit. Notice that the N/A in these tables refers to the situations where a certain operating frequency cannot be achieved using traditional arithmetic. In addition, the geometric mean values of the reduction of MRE for all designs with respect to different types of input data are summarised in Table 5.7. From the experimental results it can be seen that a significant reduction of MRE can be obtained using online arithmetic. For uniform input data, the geometric mean reduction varies from 73% to 92%. For real image data, at least 80% of MRE reduction can be achieved.

Table 5.2: Relative Reduction of MRE of FIR Filter with Online Arithmetic.

Frequency (MHz)	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
1.20	100%	100%	100%	100%	100%
1.25	100%	100%	100%	100%	100%
1.30	100%	100%	100%	100%	100%
1.35	100%	100%	100%	100%	100%
1.40	100%	100%	100%	100%	100%
1.45	99.83%	100%	100%	99.98%	100%
1.50	54.61%	98.70%	95.75%	96.48%	100%

Table 5.3: Relative Reduction of MRE of Sobel Edge Detector with Online Arithmetic.

Frequency (MHz)	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
1.20	100%	100%	100%	100%	100%
1.25	100%	100%	100%	100%	100%
1.30	100%	100%	100%	100%	100%
1.35	88.12%	100%	100%	100%	100%
1.40	90.57%	95.29%	100%	63.37%	100%
1.45	81.28%	67.57%	52.78%	66.92%	58.50%
1.50	N/A	N/A	N/A	N/A	N/A

Table 5.4: Relative Reduction of MRE of IIR Filter with Online Arithmetic.

Frequency (MHz)	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
1.20	29.06%	100%	100%	100%	100%
1.25	74.82%	34.20%	71.24%	71.86%	42.58%
1.30	62.06%	64.80%	68.94%	67.57%	60.22%
1.35	56.57%	56.06%	61.98%	58.14%	56.67%
1.40	N/A	59.55%	60.84%	60.52%	61.01%
1.45	N/A	N/A	N/A	N/A	N/A
1.50	N/A	N/A	N/A	N/A	N/A

Table 5.5: Relative Reduction of MRE of Butterworth Filter with Online Arithmetic.

Frequency (MHz)	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
1.20	100%	100%	100%	100%	100%
1.25	100%	100%	100%	100%	100%
1.30	100%	100%	100%	100%	100%
1.35	88.43%	100%	100%	100%	100%
1.40	88.54%	56.78%	87.70%	100%	90.55%
1.45	86.53%	78.41%	76.61%	76.85%	80.23%
1.50	100%	68.07%	100%	74.84%	68.84%

Table 5.6: Relative Reduction of MRE of DCT with Online Arithmetic.

Frequency (MHz)	UI	Lena	Pepper	Sailboat	Tiffany
1.20	74.14%	100%	100%	100%	100%
1.25	33.63%	100%	100%	100%	100%
1.30	59.48%	100%	100%	100%	100%
1.35	78.20%	100%	100%	100%	100%
1.40	90.66%	100%	100%	100%	100%
1.45	80.19%	100%	99.56%	99.11%	100%
1.50	71.21%	98.98%	98.98%	96.93%	97.95%

Table 5.7: Geometric Mean of Relative Reduction of MRE of All Designs with Online Arithmetic.

Frequency (MHz)	UI	Lena	Pepper	Sailboat	Tiffany
1.20	73.57%	100%	100%	100%	100%
1.25	75.89%	80.69%	93.44%	93.61%	84.30%
1.30	78.60%	91.69%	92.83%	92.46%	90.35%
1.35	78.90%	89.07%	90.88%	89.72%	89.26%
1.40	92.34%	79.73%	88.19%	82.56%	79.78%
1.45	91.23%	85.32%	79.65%	87.12%	77.79%
1.50	72.99%	87.29%	98.23%	88.79%	87.69%

For the second design target, certain error budget can be tolerated. In this case, we set the error budget to vary from 0.05% to 25%. The corresponding frequency speed-ups for each design with online arithmetic are recorded from in Table 5.8 to Table 5.12. In addition, the geometric mean of frequency speed-ups for all designs with online arithmetic are presented in Table 5.13. We can see that designs with online arithmetic achieve frequency speed-ups for all input types with respect to all error budgets, in comparison to designs implemented with traditional arithmetic. Besides, higher frequency speed-up can be obtained using real image inputs than uniform inputs for most error budgets. Specifically, the geometric mean of frequency speed-ups varying from 8.7% to 22.1% can be achieved by using uniform data, while from 4.5% to 25.0% when using real image data.

Table 5.8: Frequency Speed-ups of FIR Filter with Online Arithmetic.

Error Budget	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
0.05%	38.64%	23.53%	21.57%	21.57%	25.49%
0.5%	24.00%	23.53%	21.57%	21.57%	27.45%
1%	24.00%	20.37%	14.81%	14.81%	20.37%
5%	10.71%	11.86%	10.17%	8.47%	12.20%
10%	10.71%	11.86%	11.86%	8.47%	12.54%
25%	12.50%	13.56%	12.71%	10.17%	12.88%

Table 5.9: Frequency Speed-ups of Sobel Edge Detector with Online Arithmetic.

Error Budget	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
0.05%	17.02%	25.53%	29.79%	25.53%	25.53%
0.5%	10.00%	25.63%	29.79%	25.53%	25.53%
1%	12.00%	13.46%	19.23%	13.46%	13.46%
5%	15.38%	17.31%	21.15%	13.46%	13.46%
10%	17.31%	17.31%	21.15%	45.38%	13.46%
25%	10.91%	5.00%	5.83%	5.00%	8.33%

Table 5.10: Frequency Speed-ups of IIR Filter with Online Arithmetic.

Error Budget	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
0.05%	15.63%	18.38%	21.50%	18.38%	18.38%
0.5%	15.63%	8.57%	11.43%	8.57%	8.57%
1%	5.71%	2.70%	5.41%	2.70%	2.70%
5%	2.78%	2.43%	5.12%	2.43%	2.43%
10%	1.37%	2.15%	4.84%	2.15%	2.15%
25%	5.00%	1.27%	1.27%	0.25%	0.76%

Table 5.11: Frequency Speed-ups of Butterworth Filter with Online Arithmetic.

Error Budget	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**
0.05%	31.03%	21.88%	22.50%	21.25%	22.81%
0.5%	18.75%	21.88%	22.50%	23.13%	22.81%
1%	18.75%	21.88%	23.75%	23.13%	23.44%
5%	18.75%	21.50%	23.36%	22.74%	23.05%
10%	11.76%	8.33%	10.00%	10.56%	9.72%
25%	5.00%	4.76%	5.95%	4.76%	2.38%

Table 5.12: Frequency Speed-ups of DCT with Online Arithmetic.

Error Budget	UI	Lena	Pepper	Sailboat	Tiffany
0.05%	16.67%	47.27%	44.89%	45.37%	47.27%
0.5%	16.67%	31.91%	31.91%	32.34%	32.71%
1%	16.67%	38.30%	36.17%	32.34%	32.71%
5%	13.33%	44.68%	45.74%	46.38%	44.68%
10%	17.02%	44.68%	45.74%	46.38%	44.68%
25%	23.53%	28.30%	29.25%	29.81%	28.87%

Table 5.13: Geometric Mean of Frequency Speed-ups of All Designs with Online Arithmetic.

Error Budget	UI	Lena	Pepper	Sailboat	Tiffany
0.05%	22.14%	25.79%	26.70%	25.09%	26.21%
0.5%	16.36%	20.47%	21.98%	20.12%	21.11%
1%	13.88%	14.41%	16.49%	13.03%	13.89%
5%	10.27%	13.68%	16.03%	12.16%	13.08%
10%	8.74%	11.05%	13.52%	10.09%	10.62%
25%	9.57%	6.50%	6.95%	4.44%	6.43%

5.5.2 Taylor Expansion

In the previous experiments, the benchmark circuits are implemented using adders and constant coefficient multipliers. In order to evaluate the performance of the general purpose online multiplier, we also implement a Taylor expansion of $\log(x)$ using online adders and general purpose online multipliers. The general form of the Taylor expansion of function $f(x)$ at the point x_0 is given in (5.7) where $f^{(n)}(x_0)$ denotes the n^{th} derivative of $f(x)$ at x_0 and $n!$ denotes the factorial of n.

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{5.7}$$

According to (5.7), as an example the Taylor expansion of $log(x)$ for $n = 2$ at $x_0 = 1$ is given by (5.8) where $x \in (1, 2)$ and constant e denotes the Euler's number.

$$\log(x) = \log(e) \cdot (x - 1) - \frac{1}{2} \log(e) \cdot (x - 1)^2 \tag{5.8}$$

It can be seen that overall three online multipliers and two online adders are required to implement (5.8). In our experiments, we evaluate two common design situations, where computation results are required to either keep full precision or only deliver the most significant half precision. For both situations, we compare the designs implemented with conventional arithmetic and online arithmetic. Again the multiplier with conventional arithmetic is created with Xilinx Core Generator with speed optimisation, and it is implemented with logic resources such as LUTs instead of DSPs.

Design with Full Precision Results

For design requiring full precision outputs, intermediate computations results are not truncated for both implementations with traditional arithmetic and online arithmetic. According to the timing reports after implementation, the rated frequency is 127MHz for traditional arithmetic design, and 69MHz for online arithmetic design. In this case we can notice the conventional design runs almost twice as fast as the online design in terms of rated frequency.

We also perform overclocking experiments on both designs using our FPGA test platform. Again we use uniform independent data as well as image data as inputs to our circuits, and the quality of results is evaluated using `MRE`. The recorded results are demonstrated in Figure 5.18. In this figure, for image data we only plot the results from "Sailboat" benchmark as an example. It can be seen that the design with online arithmetic can actually operate at higher frequencies without timing errors in comparison to the design with conventional arithmetic. In addition, if a certain amount of timing errors can be tolerated, the online design can achieve better performance than conventional design. Correspondingly we record the frequency speed-ups of the online design with respect to a variety of error budgets in Table 5.14. We can see that the geometric mean of frequency speed-ups of the online design vary from 2.6% to 34%

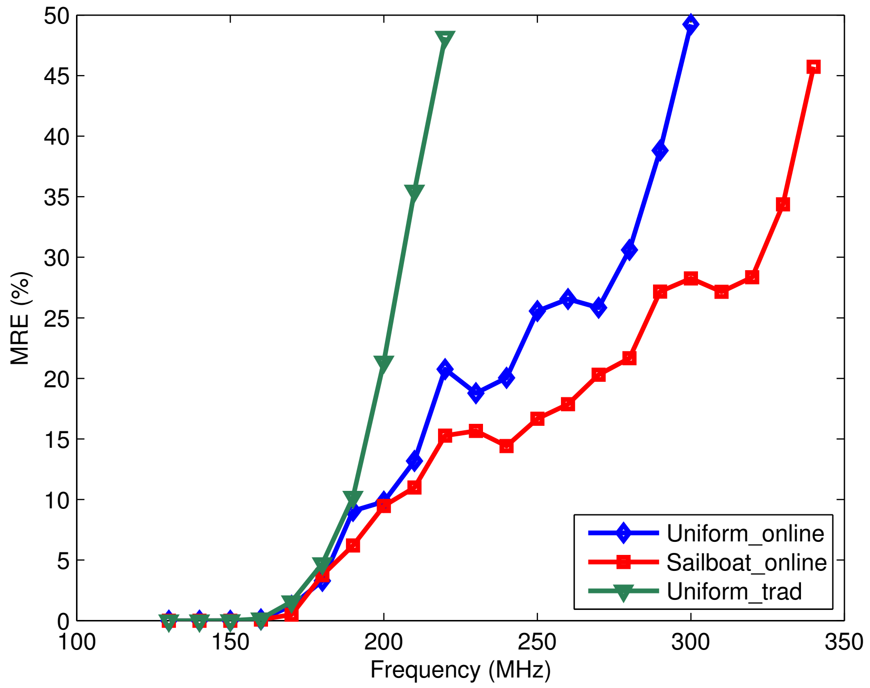


Figure 5.18: Mean relative errors of Taylor expansion of $\log(x)$ implemented with conventional arithmetic and online arithmetic with full precision outputs under overclocking.

with different levels of error tolerance. Especially when a large error budget can be tolerated, using online arithmetic is more beneficial in performance, because timing errors only affect LSDs of computation results, and correspondingly results in small error magnitude. This can also been observed in Figure 5.18 that the slope of results of the online design is much smaller than that of the conventional design. This is in accordance with the results in Section 4.4 of Chapter 4.

Table 5.14: Frequency Speed-ups of Design with Full Precision Results.

Error Budget	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**	**Geo. Mean**
0.05%	0%	1.33%	6.67%	0%	6.67%	3.9%
0.5%	1.25%	2.50%	0%	6.25%	9.38%	3.68%
1%	2.50%	3.75%	8.13%	9.38%	7.50%	5.57%
5%	1.11%	4.44%	5.56%	2.78%	1.67%	2.63%
10%	6.38%	6.38%	17.02%	11.70%	14.36%	10.31%
25%	20.00%	45.00%	40.00%	35.00%	37.50%	34.27%

Design with Half Precision Results

We also examine another common situation where the datapath outputs are required to maintain the same precision of inputs. In this case, it is not necessary for adders and multipliers to deliver results with full precision throughout the datapath. Specifically for multipliers, under this design requirement we can only keep the MSD half of the product. However, for the conventional design, although the LSD half of multiplication results is not required, it still has to be generated before truncation due to carry propagation from LSD to MSD in conventional arithmetic. For instance, in order to deliver the MSD N digits result, $2N$ digits results are generated and truncated to N digits for conventional multiplier. In comparison, online arithmetic performs computations in MSD first manner, therefore we can only implement the most significant N stages of the online multiplier at design time, as discussed in Section 5.4.3.

According to the timing reports after implementation, the rated frequencies are 136MHz and 218MHz for the conventional design and online design, respectively. In comparison to designs with full precision as discussed in the previous section, the rated frequencies for both designs are increased because of the reduction of precision. We can also notice that the rated frequency of the online design is 1.6$\times$ higher than the conventional design in this situation. This is because the structure of the online multiplier can be simplified more significantly, as discussed in Section 5.4.3. For the online multiplier itself, a significant increase of 3.2$\times$ of rated frequency can be achieved in this situation, whereas this value is only 1.1$\times$ for design with conventional arithmetic.

We also perform overclocking experiments on both designs. The FPGA measurements of the `MRE` with respect to a variety of operating frequencies for both designs are demonstrated in Figure 5.19. Again we use two types of input data for online design. From Figure 5.19 it can be clearly observed that the design with online arithmetic can be operated at much higher frequencies than the rated value without timing

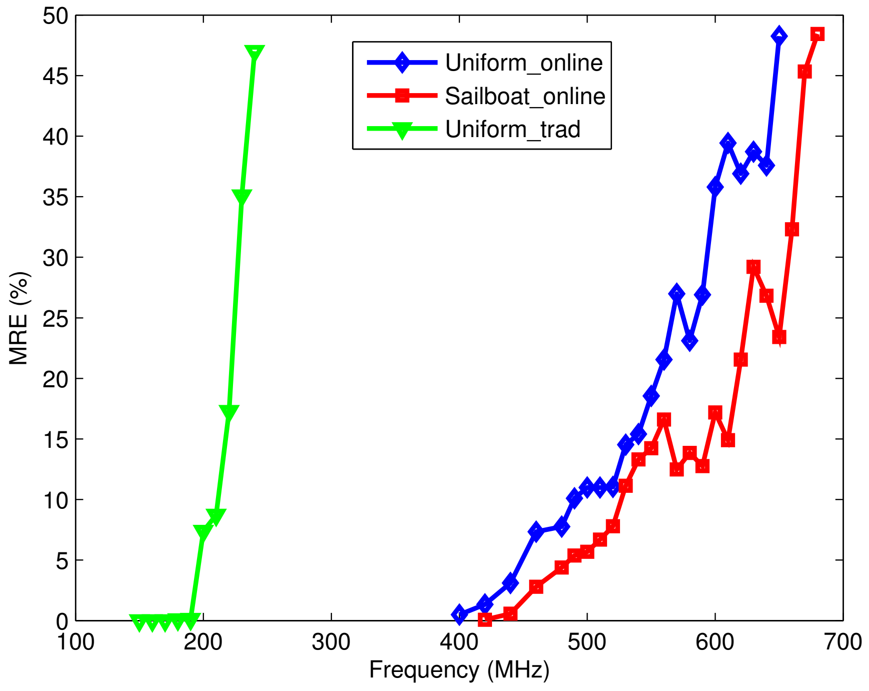


Figure 5.19: Mean relative errors of design implemented with conventional arithmetic and online arithmetic with half precision outputs under overclocking.

errors. Similar to previous results, the design with real image data as inputs can run at higher frequencies than using uniform inputs. In addition, there is a large gap of frequency between the online design and the conventional design. According to our experiments, the maximum error-free frequency of the online design is 2.7× more than the conventional design.

Table 5.15: Frequency Speed-ups of Design with Half Precision Results.

Error Budget	**UI**	**Lena**	**Pepper**	**Sailboat**	**Tiffany**	**Geo. Mean**
0.05%	2.62×	2.73×	2.80×	2.77×	2.87×	2.76×
0.5%	2.11×	2.16×	2.24×	2.31×	2.26×	2.21×
1%	2.16×	2.21×	2.26×	2.37×	2.69×	2.25×
5%	2.29×	2.34×	2.42×	2.60×	2.45×	2.42×
10%	2.29×	2.57×	2.48×	2.52×	2.50×	2.47×
25%	2.55×	2.77×	2.91×	2.82×	2.64×	2.73×

The frequency speed-ups for the online design are recorded in Table 5.15 for a variety of error budgets. It can be seen that the geometric mean of frequency speed-up varies from 2.2× to 2.8× for the online design. In addition, using image data as inputs can achieve consistently higher frequency speed-ups than using UI data, because worst case happens even more rarely with real data.

5.6 Conclusion

In this chapter, we initially describe a novel methodology that can efficiently map digit parallel online adders onto FPGAs. The proposed mapping method targets modern FPGAs, which have 6-input LUTs and four LUTs in a slice, and fully utilises the built-in carry resources. We demonstrate that the proposed architecture is both area and performance efficient in comparison to the original design. In addition, the evaluation of the optimum adder structure as discussed in Chapter 3 is expanded by the considering online adder as another design alternative.

We then propose an area efficient FPGA implementation of the online multiplier, which is specifically optimised for digit parallel operations. We have shown that this also obtains significant area reduction and frequency speed-ups. Furthermore, a correctly rounded online multiplier with reduced output precision can be created that is smaller than a traditional multiplier producing results with same precision. We show that this design can lead to silicon area savings of up to 56%.

The optimised FPGA implementations of online arithmetic operators are evaluated based on FPGA measurements using the same set of DSP benchmark circuits as presented in Chapter 3. We demonstrate that in comparison to design with conventional arithmetic, implementations with online arithmetic can be greatly beneficial in performance and accuracy.

Chapter 6

Conclusion

This thesis has examined methods to design approximate datapath through overclocking. The main focus of our approach is on arithmetic operations, and we seek to provide analytical evaluations together with experimental demonstrations to deliver solutions in designing approximate datapath composed of different forms of arithmetic operators, when considering a range of performance, accuracy and silicon area specifications. In this chapter, we firstly discuss the main contributions and discoveries by answering the questions as proposed in Chapter 1. We then discuss the potential avenue for future research based on the work of this thesis in Section 6.2, before drawing the final remarks in Section 6.3.

6.1 Summary of Contributions

In Chapter 1 we have introduced three key research questions that we would like to address in this thesis. We now summarise our answers to the corresponding question as below:

- **How can we quantify the occurrence and impact of timing errors on**

a datapath? This question highlights the importance of using quantitative approaches for the evaluation of timing errors when designing approximate datapath. In this thesis, generally timing errors are introduced by using our proposed overclocking approach to design approximate datapath. We propose probabilistic models of overclocking errors for basic arithmetic operators, such as ripple carry adder and constant coefficient multiplier. Our model employs two metrics for quantitative evaluation: one is the probability of error occurrence, and the other is the magnitude of overclocking error. These two metrics are then combined into mean absolute value of overclocking error. In addition, we also model the errors generated from the traditional design scenario where a target latency requirement is met by limiting the choice of precision in the datapath. In this case, the impact of both overclocking error and truncation error can be quantified. Through our models we observe that it is beneficial to move away from the traditional approach of creating a conservative design that is guaranteed to avoid timing violations. Alternatively, allowing timing violations to happen under the knowledge that they only happens rarely with certain input patterns can bring significant benefits in performance and accuracy. The proposed methodology can potentially be applied on any hardware platforms with specified datapath structure. In this thesis, our observation is verified with experimental data from FPGAs using a range of benchmark circuits and applications. In summary, our approach can achieve a geometric mean reduction of errors varies from 67.9% to 98.8%, or a frequency speed-up ranging from 3.1% to 27.6% when compared to the traditional approach.

- **Is it possible to minimise the impact of timing errors by carefully selecting the appropriate computer arithmetic and data representations?** This question highlights the importance of utilising proper types of computer arithmetic when designing approximate datapath. In comparison to the traditional design scenario where a given timing requirement is met by truncating datapath precision, we demonstrate that performance benefits can be achieved in designs that timing violations may occur, under the knowledge that they are unlikely to occur frequently because they require specific input patterns to generate errors. On the other hand, as we quantify the impact of timing error in terms of both its probability to happen and its magnitude, the occurrence of timing violations will introduce large errors with standard arithmetic because timing violation initially affects the most significant end of the results. However, we highlight that the impact of timing errors can be mitigated by utilising an alternative form of computer arithmetic known as "online arithmetic", which employs a redundant form of data representation. Because online arithmetic performs computations in a most significant digit first manner, upon the occurrence of timing violations they will only affect the least significant end of the results, which in tern leads to substantial performance benefits compared to the design method using conventional arithmetic. We demonstrate analytically and experimentally that digit parallel online operators can fail more gracefully when operating beyond the deterministic clocking region. In summary, we show an error reduction varying from 89.2% to 98.2% and an improvement in SNR from $21.4dB$ to $45.9dB$ using the proposed technique.

- **How can we efficiently implement the selected "overclocking friendly" computer arithmetic on existing hardware platforms such as modern FPGAs?** This question highlights the importance of efficient implementation of computer arithmetic on hardware when attempting to apply our proposed approach in practice. A major hurdle that limits the usage of the "overclocking friendly" online arithmetic is that its implementation on existing hardware platforms such as FPGAs normally requires a large area overhead in comparison to conventional arithmetic. This is because online arithmetic employs a redundant number system which intrinsically cost extra area than conventional arithmetic. In addition, both main commercial FPGA vendors have introduced dedicated resources within FPGAs to enable efficient mapping of conventional arithmetic onto FPGAs instead of online arithmetic. We tackle this problem by proposing a novel method of efficiently implementing online arithmetic on modern FPGAs by taking advantage of the built-in arithmetic resources. Specifically, our approach targets FPGAs with 6-input LUTs, and ensures the available logic resources within a slice are fully utilised when implementing digit parallel online arithmetic operators such as adder and multiplier. Our approach has the additional advantage of being able to create a correctly rounded multiplier with a reduced output precision using less hardware than a full multiplier. This is because online multipliers compute the output from the MSD first, therefore hardware can be saved by not generating the LSDs in the first place. We demonstrate that the proposed architectures achieve significant area savings of 67% and 69%, and speed-ups of over 1.2× and 1.5× for adders and multipliers, respectively, when compared to the direct HDL implementations.

6.2 Future Work

We now identify possible directions for future research that can be built upon the work of this thesis, as summarised below:

- **Enhancing the Proposed Models.** As discussed in Section 3.9 of Chapter 3, there are numerous theoretical limits of the proposed models, such as the carry propagation delay μ is unlikely to be a constant number in a practical implementation. The potential means of addressing these limitations can be taking into account the actual timing models of the targeting hardware platform (*e.g.*

FPGA or ASIC). Another research direction is to develop a system-level modelling method by considering the architecture of the practical design that the models are targeting on to a further extent.

- **Automatic Reasoning About Overclocking Errors.** There are various aspects in Chapter 3 and Chapter 4 could be automated. For example, a tool for automatic modelling of overclocking errors for a range of arithmetic operators could be developed following the similar procedures as discussed in this thesis.

- **Automatic Selection of the Optimum Arithmetic Operator.** In Chapter 3 we examine the optimum adder design methodology when considering a range of frequency, accuracy and area specifications. In Chapter 5 we expand this evaluation by adding online adder into comparison. Actually this evaluation can be further extended to other arithmetic operators such as multipliers. Furthermore, a tool can be developed for automatically selecting the optimum arithmetic operators for a given specification on frequency, accuracy and silicon area, when performing a system level design.

- **Verification Challenges of Approximate Datapath Design.** To the best of our knowledge, verification for approximate computing is a rarely investigated field according to existing literature. This issue greatly limits the widespread adoption of approximate computing in industry. For example, quantitative guarantees have to be provided to ensure that approximate techniques will not affect error sensitive parts of the design, such as control logic. Unfortunately existing approaches to address this problem are either *ad-hoc* or empirical. That is, the distinction between error resilience parts and error sensitive parts of a given design is normally identified by circuit designers. This process should be automated and generalised to fit a variety of applications. In addition, given the error models of arithmetic operators proposed in this thesis, how to verify whether the overall computation quality meets the design specification on accuracy, performance and area still requires further examination.

- **A Library of Online Arithmetic Operators.** In Chapter 5 we discuss efficient implementations of online adder and online multiplier on FPGAs for area savings. Although there is research developing libraries of online arithmetic operators, they are not optimised for modern FPGA architectures. The general idea in our work that fully utilising the built-in logic resources within an FPGA can be directly applied to implement other types of online arithmetic operations, such as division, multiply-accumulation and square root. Eventually this will form a novel library of online arithmetic operators targeting mainstream FPGAs. In addition, it can be incorporated into existing tools for FPGA development.

- **High Level Synthesis with Online Arithmetic.** The utilisation of online arithmetic potentially offers great optimisation opportunities in high level synthesis problems. For instance, consider the pseudo code as given in Algorithm 5. It can be seen that the output *Op* is dependent upon the correctness of operation $a > b$. If Computation1 and Computation2 are calculated using conventional arithmetic, $a > b$ is a timing critical operation and cannot be overclocked because large computation latency is required to wait for the update of MSDs of both a and b before the comparison can be performed. In contrast, if online arithmetic is used to calculate a and b, the computing delay can be greatly reduced. Because both a and b are generated from MSD first, it is not necessary to generate full precision for a and b. Instead, computations can be terminated dynamically as long as the result of $a > b$ is obtained. A tool can be developed based on this example to automatically address similar problems using online arithmetic in the process of high level synthesis.

- **Floating Point Operation.** The work presented in this thesis only focuses

Algorithm 5 Example Program

```
1: $a$ := Computation1, $b$ := Computation2
2: **if** $a > b$ **then**
3:     $Op = 2a^5 + b^3$
4: **else**
5:     $Op = \sqrt{a} - 4b$
6: **end if** **return** $Op$
```

on fixed point numbers together with their operations. However, applying approximate computing technique or online arithmetic, or a combination of two, using floating point number representations remains a rarely touched area and requires extensive research.

6.3 Final Remarks

Over the course of this thesis, we have explored the methodology of datapath design through the usage of approximate computing techniques and different forms of computer arithmetic. This thesis has introduced several alternative approaches to create datapaths with higher performance than existing methods. Furthermore, this thesis has laid significant steps in combining approximate computing techniques and online arithmetic for datapath design with the consideration of a range of performance, accuracy and area specifications. The contributions of this thesis should open new avenues of research in both areas.

Bibliography

[1] A. Alaghi, C. Li, and J. P. Hayes. Stochastic circuits for real-time image-processing applications. In *Proc. Design Automation Conference*, page 136, 2013.

[2] G. Alefeld and J. Herzberger. *Introduction to interval computations.* Academic press, 1983.

[3] Altera Co. Cyclone device handbook. `https://www.altera.com/en_US/pdfs/literature/hb/cyc/cyc_c5v1.pdf`, 2008.

[4] Y. Andreopoulos. Error tolerant multimedia stream processing: There's plenty of room at the top (of the system stack). *IEEE Trans. on Multimedia*, 15(2):291–303, 2013.

[5] T. Austin. DIVA: a reliable substrate for deep submicron microarchitecture design. In *Proc. Int. Symp. on Microarchitecture*, pages 196–207, 1999.

[6] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge. Opportunities and challenges for better than worst-case design. In *Proc. Asia and South Pacific Design Automation Conference*, pages 2–7, 2005.

[7] T. Austin, D. Blaauw, T. Mudge, and K. Flautner. Making typical silicon matter with Razor. *IEEE Trans. on Computer*, 37(3):57–65, 2004.

[8] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Trans. on Electronic Computers*, (3):389–400, 1961.

[9] J.-C. Bajard, J. Duprat, S. Kla, and J.-M. Muller. Some operators for on-line radix-2 computations. *Journal of Parallel and Distributed Computing*, 22(2):336–345, 1994.

[10] R. Beguenane, S. Simard, and A. Tisserand. Function evaluation on FPGAs using on-line arithmetic polynomial approximation. In *Proc. Int. Northeast Workshop on Circuits and Systems*, pages 21–24, 2006.

[11] D. Boland and G. Constantinides. A scalable approach for automated precision analysis. In *Proc. Int. Symp. on Field-Programmable Gate Arrays*, pages 185–194, 2012.

[12] R. H. Brackert Jr, M. D. Ercegovac, and A. N. Willson Jr. Design of an on-line multiply-add module for recursive digital filters. In *Proc. Symp. on Computer Arithmetic*, pages 34–41, 1989.

[13] P. Brisk, A. Verma, P. Ienne, and H. Parandeh-Afshar. Enhancing FPGA performance for arithmetic circuits. In *Proc. Design Automation Conference*, pages 334–337, June 2007.

[14] B. D. Brown and H. C. Card. Stochastic neural computation. I. Computational elements. *IEEE Transactions on Computers*, 50(9):891–905, 2001.

[15] G.-C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano. On the use of signed digit arithmetic for the new 6-inputs LUT based FPGAs. In *Proc. Int. Conf. on Electronics, Circuits and Systems*, pages 602–605, 2008.

[16] S. Chatterjee, C. Weaver, and T. Austin. Efficient checker processor design. In *Proc. Int. Symp. on Microarchitecture*, pages 87–97, 2000.

[17] 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. Int. Conf. on Solid State Devices and Materials*, pages 402–403, 2004.

[18] S. Cheemalavagu, P. Korkmaz, K. V. Palem, B. E. Akgul, and L. N. Chakrapani. A probabilistic CMOS switch and its realization by exploiting noise. In *IFIP Int. Conf. on VLSI*, pages 535–541, 2005.

[19] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, and N. Sun. Dadiannao: A machine-learning supercomputer. In *Proc. Int. Symp. on Microarchitecture*, pages 609–622, 2014.

[20] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proc. Design Automation Conference*, page 113, 2013.

[21] V. K. Chippa, S. Venkataramani, K. Roy, and A. Raghunathan. StoRM: a stochastic recognition and mining processor. In *Proc Int. Symp. on Low Power Electronics and Design*, pages 39–44, 2014.

[22] G. Chow, K. Kwok, W. Luk, and P. Leong. Mixed precision processing in reconfigurable systems. In *Proc. Int. Symp. on Field-Programmable Custom Computing Machines*, pages 17–24, 2011.

[23] B. Colwell. We may need a new box. *IEEE Trans. on Computer*, 37(3):40–41, 2004.

[24] J. Cong and K. Minkovich. Logic synthesis for better than worst-case designs. In *Proc. Int. Symp. on VLSI Design, Automation and Test*, pages 166–169, 2009.

[25] G. A. Constantinides, N. Nicolici, and A. B. Kinsman. Numerical data representations for FPGA-based scientific computing. *IEEE Design Test of Computers*, 28(4):8–17, 2011.

[26] M. Dimmler, A. Tisserand, U. Holmbeg, and R. Longchamp. On-line arithmetic for real-time control of microsystems. *IEEE/ASME Trans. on Mechatronics*, 4(2):213–217, Jun 1999.

[27] T. Drane, T. Rose, and G. A. Constantinides. On the systematic creation of faithfully rounded truncated multipliers and arrays. *IEEE Trans. on Computers*, 63(10):2513–2525, 2013.

[28] M. Ercegovac and T. Lang. On-the-fly conversion of redundant into conventional representations. *IEEE Trans. on Computers*, C-36(7):895–897, 1987.

[29] M. D. Ercegovac. A general hardware-oriented method for evaluation of functions and computations in a digital computer. *IEEE Trans. on Computers*, 100(7):667–680, 1977.

[30] M. D. Ercegovac. On-line arithmetic: An overview. In *Proc. Annual Technical Symp. on Real time signal processing VII*, pages 86–93, 1984.

[31] M. D. Ercegovac and T. Lang. Most-significant-digit-first and on-line arithmetic approaches for the design of recursive filters. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, volume 1, pages 7–11, 1989.

[32] M. D. Ercegovac and T. Lang. Fast multiplication without carry-propagate addition. *IEEE Trans. on Computers*, 39(11):1385–1390, 1990.

[33] M. D. Ercegovac and T. Lang. Simple radix-4 division with operands scaling. *IEEE Trans. on Computers*, 39(9):1204–1208, 1990.

[34] M. D. Ercegovac and T. Lang. *Digital arithmetic.* Morgan Kaufmann, 2003.

[35] D. Ernst, N. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proc. Int. Symp. on Microarchitecture*, pages 7–18, 2003.

[36] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proc. Int. Symp. on Computer Architecture*, pages 365–376, 2011.

[37] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *Proc. Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 301–312, 2012.

[38] J. S. Fernando and M. D. Ercegovac. On-line arithmetic modules for recursive digital filters. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, pages 681–685, 1992.

[39] J. Fowers, G. Brown, P. Cooke, and G. Stitt. A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. In *Proc. Int. Symp. on Field-Programmable Gate Arrays*, pages 47–56, 2012.

[40] M. Frederick and A. Somani. Multi-bit carry chains for high-performance reconfigurable fabrics. In *Proc. Int. Conf. on Field-Programmable Logic and Applications*, pages 1–6, Aug 2006.

[41] R. Galli and A. Tenca. A design methodology for networks of online modules and its application to the Levinson-Durbin algorithm. *IEEE Trans. on VLSI*, 12(1):52–66, 2004.

[42] R. Galli and A. F. Tenca. Design and evaluation of online arithmetic for signal processing applications on FPGAs. In *Proc. SPIE Advanced Signal Processing Algorithms, Architectures and Implementations*, pages 134–144, 2001.

[43] C. Garcia-Vega, S. Gonzalez-Navarro, J. Villalba, and E. L. Zapata. Decimal online multioperand addition. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, pages 350–354, 2012.

[44] B. Gojman, S. Nalmela, N. Mehta, N. Howarth, and A. DeHon. GROK-LAB: generating real on-chip knowledge for intra-cluster delays using timing extraction. In *Proc. Int. Symp. on Field-Programmable Gate Array*, pages 81–90, 2013.

[45] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, et al. Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):8–23, 2013.

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

[47] R. Gutierrez, J. Valls, and A. Perez-Pascual. FPGA-implementation of time-multiplexed multiple constant multiplication based on carry-save arithmetic. In *Proc. Int. Conf. on Field-Programmable Logic and Applications*, pages 609–612, Aug 2009.

[48] R. Hamill, J. V. McCanny, and R. L. Walke. Online CORDIC algorithm and VLSI architecture for implementing QR-array processors. *IEEE Trans. on Signal Processing,*, 48(2):592–598, 2000.

[49] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi. A high-speed multiplier using a redundant binary adder tree. *IEEE Jourl. of Solid-State Circuits*, 22(1):28–34, 1987.

[50] S. Hauck, M. Hosler, and T. Fry. High-performance carry chains for FPGA's. *IEEE Trans. VLSI*, 8(2):138–147, April 2000.

[51] Y. He and C.-H. Chang. A power-delay efficient hybrid carry-lookahead/carry-select based redundant binary to two's complement converter. *IEEE Trans. on Circuits and Systems I: Regular Papers*, 55(1):336–346, 2008.

[52] Y. He and C.-H. Chang. A new redundant binary Booth encoding for fast-bit multiplier design. *IEEE Trans. on Circuits and Systems I: Regular Papers*, 56(6):1192–1201, 2009.

[53] Y. He, C.-H. Chang, J. Gu, and A. Fahmy. A novel covalent redundant binary Booth encoder. In *Proc. Int. Symp. on Circuits and Systems*, pages 69–72, 2005.

[54] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012.

[55] J. Hormigo, M. Ortiz, F. Quiles, F. J. Jaime, J. Villalba, and E. L. Zapata. Efficient implementation of carry-save adders in FPGAs. In *Proc. Int. Conf. on Application-specific Systems, Architectures and Processors*, pages 207–210, 2009.

[56] J. Hormigo, J. Villalba, and E. L. Zapata. Multioperand redundant adders on FPGAs. *IEEE Trans. on Computers*, 62(10):2013–2025, 2013.

[57] B. Jose and D. Radhakrishnan. Delay optimized redundant binary adders. In *Proc. Int. Conf. on Electronics, Circuits and Systems*, pages 514–517, 2006.

[58] B. Jose and D. Radhakrishnan. Fast redundant binary partial product generators for Booth multiplication. In *Proc. Midwest Symp. on Circuits and Systems*, pages 297–300, 2007.

[59] A. Kahng, S. Kang, R. Kumar, and J. Sartori. Designing a processor from the ground up to allow voltage/reliability tradeoffs. In *Proc. Int. Symp. on High Performance Computer Architecture*, pages 1–11, Jan 2010.

[60] A. Kahng, S. Kang, R. Kumar, and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *Proc. Asia and South Pacific Design Automation Conference*, pages 825–831, Jan 2010.

[61] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proc. Design Automation Conference*, pages 820–825, 2012.

[62] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proc. Design Automation Conference*, pages 820–825, 2012.

[63] W. Kamp, A. Bainbridge-Smith, and M. Hayes. Efficient implementation of fast redundant number adders for long word-lengths in FPGAs. In *Proc. Int. Conf. on Field-Programmable Technology*, pages 239–246, Dec 2009.

[64] J.-Y. Kang and J.-L. Gaudiot. A simple high-speed multiplier design. *IEEE Trans. on Computers*, 55(10):1253–1258, 2006.

[65] Z. M. Kedem, V. J. Mooney, K. K. Muntimadugu, and K. V. Palem. An approach to energy-error tradeoffs in approximate ripple carry adders. In *Proc. Int. Symp. on Low Power Electronics and Design*, pages 211–216, 2011.

[66] K. Keutzer and M. Orshansky. From blind certainty to informed uncertainty. In *Proc. Int. workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 37–41, 2002.

[67] Y. Kim, B.-S. Song, J. Grosspietsch, and S. F. Gillig. A carry-free 54b$\times$ 54b multiplier using equivalent bit conversion algorithm. *IEEE Journal of Solid-State Circuits*, 36(10):1538–1545, 2001.

[68] A. Kinsman and N. Nicolici. Finite precision bit-width allocation using SAT-modulo theory. In *Proc. of Conf. on Design, Automation and Test in Europe*, pages 1106–1111, 2009.

[69] A. Kinsman and N. Nicolici. Bit-width allocation for hardware accelerators for scientific computing using SAT-modulo theory. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(3):405–413, 2010.

[70] S. D. Krueger and P.-M. Seidel. Design of an on-line IEEE floating-point addition unit for FPGAs. In *Proc. Symp. on Field-Programmable Custom Computing Machines*, pages 239–246, 2004.

[71] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *Proc. Int. Conf. on VLSI Design*, pages 346–351, 2011.

[72] K. Kurbjun and C. Ribbing. Xilinx application note: MMCM dynamic reconfiguration. `http://www.xilinx.com/support/documentation/application_notes/xapp878.pdf`, 2010.

[73] D. Lau, A. Schneider, M. D. Ercegovac, and J. Villasenor. FPGA-based structures for on-line FFT and DCT. In *Proc. Int. Symp. on Field-Programmable Custom Computing Machines*, pages 310–311, 1999.

[74] D. Lau, A. Schneider, M. D. Ercegovac, and J. Villasenor. A FPGA-based library for on-line signal processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 28(1-2):129–143, 2001.

[75] D. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides. Accuracy-guaranteed bit-width optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 25(10):1990–2000, 2006.

[76] J.-G. Lee, J.-A. Lee, B.-S. Lee, and M. Ercegovac. A design method for heterogeneous adders. In Y.-H. Lee, H.-N. Kim, J. Kim, Y. Park, L. Yang, and S. Kim, editors, *Embedded Software and Systems*, volume 4523 of *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin Heidelberg, 2007.

[77] A. Lingamneni, A. Basu, C. Enz, K. V. Palem, and C. Piguet. Improving energy gains of inexact DSP hardware through reciprocative error compensation. In *Proc. Design Automation Conference*, pages 1–8, 2013.

[78] A. Lingamneni, K. K. Muntimadugu, C. Enz, R. M. Karp, K. V. Palem, and C. Piguet. Algorithmic methodologies for ultra-efficient inexact architectures for sustaining technology scaling. In *Proc. Conf. on Computing Frontiers*, pages 3–12, 2012.

[79] G. Liu, Y. Tao, M. Tan, and Z. Zhang. CASA: Correlation-aware speculative adders. In *Proc. Int. Symp. on Low Power Electronics and Design*, pages 189–194, 2014.

[80] S. L. Lu. Speeding up processing with approximation circuits. *IEEE Trans. on Computer*, 37(3):67–73, 2004.

[81] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko. An 8.8-ns 54×54-bit multiplier with high speed redundant binary architecture. *IEEE Jourl. of Solid-State Circuits*, 31(6):773–783, 1996.

[82] T. Matsunaga, S. Kimura, and Y. Matsunaga. Multi-operand adder synthesis on FPGAs using generalized parallel counters. In *Proc. Asia and South Pacific Design Automation Conference*, pages 337–342, 2010.

[83] R. McIlhenny and M. D. Ercegovac. On the design of an on-line FFT network for FPGA's. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, volume 2, pages 1484–1488, 1999.

[84] K. Minkovich and J. Cong. Mapping for better than worst-case delays in LUT-based FPGA designs. In *Proc. Int. Symp. on Field-Programmable Gate Arrays*, pages 56–64, 2008.

[85] P. Montuschi and L. Ciminiera. Algorithm and architectures for radix-4 division with over-redundant digit set and simple digit selection hardware. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, pages 418–422, 1991.

[86] R. Moore. *Interval analysis.* Prentice-Hall Englewood Cliffs, NJ, 1966.

[87] J. V. Moreno, T. Lang, and J. Hormigo. Radix-2 multioperand and multiformat streaming online addition. *IEEE Trans. on Computers*, 61(6):790–803, 2012.

[88] E. Mosanya and E. Sanchez. A FPGA-based hardware implementation of generalized profile search using online arithmetic. In *Proc. Int. Symp. on Field-Programmable Gate Arrays*, pages 101–111, 1999.

[89] C. Nagendra, M. J. Irwin, and R. M. Owens. Area-time-power tradeoffs in parallel adders. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, 43(10):689–702, 1996.

[90] W. G. Natter and B. Nowrouzian. A novel algorithm for signed-digit online multiply-accumulate operation and its purely signed-binary hardware implementation. In *Proc. Int. Symp. on Circuits and Systems*, volume 5, pages 329–332, 2000.

[91] W. G. Natter and B. Nowrouzian. Digit-serial online arithmetic for high-speed digital signal processing applications. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, volume 1, pages 171–176, 2001.

[92] K. Olukotun and L. Hammond. The future of microprocessors. *Communications of the ACM*, 3(7):26–29, 2009.

[93] K. Palem and A. Lingamneni. What to do about the end of Moore's law, probably! In *Proc. Design Automation Conference*, pages 924–929, 2012.

[94] K. Palem, A. Lingamneni, C. Enz, and C. Piguet. Why design reliable chips when faulty ones are even better. In *Proc. European Solid-State Circuits Conference*, pages 255–258, 2013.

[95] K. V. Palem, L. N. Chakrapani, Z. M. Kedem, A. Lingamneni, and K. K. Muntimadugu. Sustaining Moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects. In *Proc. Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 1–10, 2009.

[96] K. V. Palem, P. Korkmaz, and K. Kong. Probabilistic CMOS (PCMOS) logic for nanoscale circuit design. In *Int. Solid State Circuits Conference: Advanced Solid-State Circuits Forum*, 2009.

[97] H. Parandeh-Afshar, A. Verma, P. Brisk, and P. Ienne. Improving FPGA performance for carry-save arithmetic. *IEEE Trans. on VLSI*, 18(4):578–590, April 2010.

[98] M. J. Pelgrom, A. C. Duinmaijer, A. P. Welbers, et al. Matching properties of MOS transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1439, 1989.

[99] T. Preusser and R. Spallek. Enhancing FPGA device capabilities by the automatic logic mapping to additive carry chains. In *Proc. Int. Conf. on Field-Programmable Logic and Applications*, pages 318–325, Aug 2010.

[100] W. Qian and M. D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *Proc. Design Automation Conference*, pages 648–653, 2008.

[101] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic. *Digital integrated circuits: a design perspective (2nd edition).* Prentice-Hall, 2003.

[102] S. Rajagopal and J. Cavallaro. Truncated online arithmetic with applications to communication systems. *IEEE Trans. on Computers*, 55(10):1240–12529, Oct 2006.

[103] S. Rajagopal and J. R. Cavallaro. On-line arithmetic for detection in digital communication receivers. In *Proc. Symp. on Computer Arithmetic*, pages 257–265, 2001.

[104] S. Ramprassad, N. Shanbha, and I. Hajj. Analytical estimation of signal transition activity from word-level statistics. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(7):718–733, 1997.

[105] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proc. ACM SIGPLAN Notices*, volume 46, pages 164–174, 2011.

[106] M. Schaffner, F. K. Gürkaynak, A. Smolic, H. Kaeslin, and L. Benini. An approximate computing technique for reducing the complexity of a direct-solver for sparse linear systems in real-time video processing. In *Proc. Design Automation Conference*, pages 1–6, 2014.

[107] Semiconductor Industry Association. International technology roadmap for semiconductors (ITRS), 2007 edition. `http://www.itrs.net/`, 2007.

[108] Semiconductor Industry Association. International technology roadmap for semiconductors (ITRS), 2013 edition. `http://www.itrs.net/`, 2013.

[109] K. Shi, D. Boland, and G. A. Constantinides. Accuracy-performance tradeoffs on an FPGA through overclocking. In *Proc. Int. Symp. on Field-Programmable Custom Computing Machines*, pages 29–36, 2013.

[110] K. Shi, D. Boland, and G. A. Constantinides. Overclocking datapath for latency-error tradeoff. In *Proc. Int. Symp. on Circuits and Systems*, pages 2537–2540, May 2013.

[111] K. Shi, D. Boland, and G. A. Constantinides. Efficient FPGA implementation of digit parallel online arithmetic operators. In *Proc. Int. Conf. on Field-Programmable Technology*, pages 115–122, 2014.

[112] K. Shi, D. Boland, and G. A. Constantinides. Imprecise datapath design: An overclocking approach. *ACM Trans. on Reconfigurable Technology and Systems*, 8(2):6:1–6:23, Mar. 2015.

[113] K. Shi, D. Boland, E. Stott, S. Bayliss, and G. A. Constantinides. Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs. In *Proc. Design Automation Conference*, pages 1–6, 2014.

[114] K. Shi and G. A. Constantinides. Evaluation of design trade-offs of adders in approximate datapath. In *HiPEAC Workshop on Approximate Computing*, 2015.

[115] J. Stol and L. De Figueiredo. Self-validated numerical methods and applications. In *Monograph for 21st Brazilian Mathematics Colloquium*, 1997.

[116] N. Takagi, T. Asada, and S. Yajima. Redundant CORDIC methods with a constant scale factor for sine and cosine computation. *IEEE Trans. on Computers*, 40(9):989–995, 1991.

[117] N. Takagi, H. Yasuura, and S. Yajima. High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Trans. on Computers*, 100(9):789–796, 1985.

[118] J. Tatsukawa. MMCM and PLL dynamic reconfiguration. `http://www.xilinx.com/support/documentation/application_notes/xapp888_7Series_DynamicRecon.pdf`, 2014.

[119] A. F. Tenca, M. D. Ercegovac, and M. E. Louie. Fast on-line multiplication units using LSA organization. In *Proc. SPIE's Int. Symp. on Optical Science, Engineering, and Instrumentation*, pages 74–83, 1999.

[120] A. F. Tenca and S. U. Hussaini. A design of radix-2 on-line division using LSA organization. In *Proc. Symp. on Computer Arithmetic*, pages 266–273, 2001.

[121] D. Thomas, L. Howes, and W. Luk. A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation. In *Proc. Int. Symp. on Field-Programmable Gate Arrays*, pages 63–72, 2009.

[122] A. Tisserand, P. Marchal, and C. Piguet. An on-line arithmetic based FPGA for low-power custom computing. In *Proc. Int. Conf. on Field-Programmable Logic and Applications*, pages 264–273, 1999.

[123] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung. Reconfigurable computing: Architectures and design methods. *IEE Proc. Computers and Digital Techniques*, 152(2):193–207, Mar 2005.

[124] K. S. Trivedi and M. Ercegovac. On-line algorithms for division and multiplication. *IEEE Trans. on Computers*, 26:161–167, 1975.

[125] A. K. Uht. Going beyond worst-case specs with TEAtime. *IEEE Trans. on Computer*, 37(3):51–56, 2004.

[126] K. Underwood. FPGAs vs. CPUs: trends in peak floating-point performance. In *Proc. Int. Symp. on Field-Programmable Gate Arrays*, pages 171–180, 2004.

[127] C. G. Vega, S. G. Navarro, J. V. Moreno, and E. L. Zapata. On-line decimal adder with RBCD representation. In *Proc. Int. Conf. on Application-Specific Systems, Architectures and Processors*, pages 53–60, 2012.

[128] J. Villalba, J. Hormigo, and T. Lang. Improving the throughput of on-line addition for data streams. In *Proc. Int. Conf. on Application-specific Systems, Architectures and Processors*, pages 272–277, 2007.

[129] J. Villalba, J. Hormigo, J. M. Prades, and E. L. Zapata. On-line multioperand addition based on on-line full adders. In *Proc. Int. Conf. on Application-Specific Systems, Architecture Processors*, pages 322–327, 2005.

[130] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. In *Proc. Int. Conf. on Dependable Systems and Networks*, pages 411–420, 2001.

[131] N. Weaver, Y. Markovskiy, Y. Patel, and J. Wawrzynek. Post-placement C-slow retiming for the Xilinx Virtex FPGA. In *Proc. Int. Symp. on Field-Programmable Gate Arrays*, pages 185–194, 2003.

[132] H. Y. Wong, L. Cheng, Y. Lin, and L. He. FPGA device and architecture evaluation considering process variations. In *Proc. Int. Conf. on Computer-Aided Design*, pages 19–24, 2005.

[133] Xilinx Inc. Spartan-3 generation FPGA user guide. `http://www.xilinx.com/support/documentation/user_guides/ug331.pdf`, Feb. 2008.

[134] Xilinx Inc. Mixed-mode clock manager (MMCM) module. `http://www.xilinx.com/support/documentation/ip_documentation/mmcm_module.pdf`, 2009.

[135] Xilinx Inc. Virtex-6 FPGA configurable logic block user guide. `http://www.xilinx.com/support/documentation/user_guides/ug364.pdf`, 2009.

[136] Xilinx Inc. LogiCORE IP multiplier v11.2. `http://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf`, 2011.

[137] Xilinx Inc. Virtex-6 FPGA clocking resources user guide. `http://www.xilinx.com/support/documentation/user_guides/ug362.pdf`, 2011.

[138] Xilinx Inc. 7 series FPGAs configurable logic block user guide. `http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf`, 2014.

[139] A. Yazdanbakhsh, B. Thwaites, J. Park, and H. Esmaeilzadeh. Methodical approximate hardware design and reuse. `http://act-lab.org/doc/paper/2014-wacas-hdl.pdf`, 2014.

[140] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On reconfiguration-oriented approximate adder design and its application. In *Proc. Int. Conf. on Computer-Aided Design*, pages 48–54, 2013.

[141] N. Zhu, W. L. Goh, and K. S. Yeo. An enhanced low-power high-speed adder for error-tolerant application. In *Proc. Int. Symp. on Integrated Circuits*, pages 69–72, 2009.