University of California

Los Angeles

On-line Arithmetic Algorithms for Efficient Implementation

A dissertation

submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Paul Kang-Guo Tu

1990

The dissertation of Paul Kang-Guo Tu is approved.

Kirby A. Baker

Han K. Huang

David Rennels

Tomás Lang

Miloš D. Ercegovac, *Committee Chair*

University of California, Los Angeles

1990

To Tanping and my parents

Table of Contents

List of Figures

List of Tables

List of Symbols

$x, y, z, \cdots$	floating-point numbers
$X, Y, Z, \cdots$	fixed-point numbers, mantissa of floating-point numbers
e_x	exponent of x
x_j	jth most significant digit of X in on-line form
$X[j]$	value of X after jth iteration
$\widehat{X[i]}$	low precision estimate of $X[i]$
$\overline{X[i] + X[i-1]}$	low precision estimate of $X[i] + X[i-1]$
$x_j^{(i)}$	the ith bit of x_j
X_i	the ith most significant digit of X
m	precision of a computation
$r = 2^k$	radix of a computation
A	result of the recurrence expression evaluation
I	comparison point of digit selection function
K	redundancy factor of a number system
L	lower bound of a selection interval
P	partial residual function
R	scaled partial residual function
U	upper bound of a selection interval
α	smallest value of a digit set

β	largest value of a digit set
δ	on-line delay
ϵ	computation error
ε	scaled error bound
η, ξ, μ, ν	truncation error
λ	number of input terms to the adder
ρ	largest digit value of a symmetric digit set
ϕ_X	number of integer bits of X
ψ_X	number of fractional bits of X
σ_X	total number of bits of X
Δ	total on-line delay of a computation
Ω	overlap between selection intervals
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	arithmetic expressions
$\mathcal{D}$	digit set
$\mathcal{F}$	arithmetic computation
$\mathcal{G}$	transformation function
$\mathcal{S}$	digit selection function

Acknowledgments

I am grateful to my advisor, Professor Miloš D. Ercegovac, for without his guidance, inspiration and support this work would not have been completed. Special thanks to Professor Tomás Lang for his interest and many enlightening discussions. I would like to thank members of my committee, Professors David Rennels, Han K. Huang, and Kirby A. Baker for their participation and comments.

Many people have been helpful throughout the course of this study. In particular, I would like to thank Marc Tremblay, Shih-Lien Lu, Dorab Patel, Tanwei Chung, Bob Tisdale, Leon Alkalaj, Pak Chan, Dan Greening, Jeong-A Lee, Jaime Moreno, T.M. Ravi, Frank Schaffa, Martine Schlag, Miquel Huguet, Raffi Dionysian, John Fernando, John Harding and James Liu for the numerous discussions and their friendship. I would also like to express my appreciation to Verra Morgan and Doris Sublette who have been of great help in many ways that made my life as a graduate student more enjoyable.

This research has been supported in part by the NSF Grant No. MIP-8813340 *Composite Operations Using On-Line Arithmetic for Application-Specific Parallel Architectures: Algorithms, Design, and Experimental Studies* and by the Office of Naval Research Contract N00014-85-K-0159 *On-Line Arithmetic Algorithms and Structures for VLSI*.

Vita

1976	Diploma in Mechanical Engineering Wuhan Institute of Iron and Steel Technology, Wuhan, China
1981	M. S. in Computer Sciences University of Wisconsin, Madison

Publications

Paul K.-G. Tu and Miloš D. Ercegovac, "A Radix-4 On-line Division Algorithm", In *Proceedings of the 8th Symposium on Computer Arithmetic*, pp181–187, 1987

Paul K.-G. Tu and Miloš D. Ercegovac, "Design of On-Line Division Unit", In *Proceedings of the 9th Symposium on Computer Arithmetic*, pp42–49, 1989

Abstract of the Dissertation

On-line Arithmetic Algorithms for Efficient Implementation

by

Paul Kang-Guo Tu

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1990

Professor Miloš D. Ercegovac, *Chair*

On-line arithmetic algorithms introduce parallelism between sequential operations by overlapping these operations in a digit-pipelined fashion. They can reduce the computation time of long sequences of arithmetic operations. Thus on-line arithmetic complements other approaches such as parallel processing and pipelining which exploit parallelism at the numerical algorithms level.

The implementation characteristics of on-line arithmetic algorithms and their applications are investigated. First the model of on-line computation is defined, and a systematic and unified derivation of on-line algorithms is given. Parameters that affect the implementation efficiency of an on-line computation are identified and discussed. Gate array implementation of on-line arithmetic units are presented with estimations of the cost and performance. A case study is conducted and the on-line approach is compared with conventional schemes. The study shows that on-line arithmetic is effective and feasible in the implementation of numerical computations where the critical path contains long sequences of arithmetic operations.

Chapter 1

Introduction

1.1 Motivations and objectives

Advances in very large scale integration (VLSI) have resulted in increased density and reduced cost. Gate array and standard cell technologies have provided means of low cost and short turn around time of designing semi-custom digital devices. New packaging techniques further allow integrated circuits to be interconnected with short delays. On the other hand, developments in many application areas, such as real time signal processing, have resulted in new algorithms with ever increasing demands for computing speed and capacity.

One of the approaches to take advantage of the VLSI technology is the design of special-purpose digital systems as a low cost alternative to supercomputers for many important numerical computations. Special-purpose systems have the advantage of being application oriented so that their design closely matches the specific computations they are designed for to achieve high performance.

VLSI system design has some limitations that make it quite different from previous design methods. Circuits have to be designed such that they can be laid out densely and efficiently, and with minimum effort. As the complexity of the circuit on a single chip grows and the sizes of circuit elements decrease, the verification of the circuit becomes more and more difficult. The increased complexity of the circuits and the limited physical access to them make testing of VLSI circuit very hard. Communication becomes expensive since wires take up the majority of the area of a chip. The amount of circuit that can be put onto a single chip is also limited by the physical size of a chip due to the limited number of input/output pins available.

Extensive research has shown that to take advantage of the technology, the

following properties are desired in VLSI design[1, 2].

- It should be implemented by only a few different types of simple cells.
- It should have simple and regular data and control paths so that the cells can be connected by a network with local and regular interconnections. The interconnection requirement should be minimum to reduce the wiring area.
- It should use pipelining and parallelism. In this way, a large number of cells are active at one time so that the overall computational rate of the simple cells is high.
- For a design to be cost effective, the structure of the design should be modular and extensible, and be general such that a class of algorithms can be realized with minimal modification.
- An effective implementation should have a small pin/logic ratio.

The design of a special-purpose digital system can be viewed as a two level process. At the first level, the system may consists of many processing elements, including arithmetic processors, memory devices, I/O devices, control units, and communication and switching networks. These processing elements may be interconnected as a systolic array, a ring, or some other topology depending on the specific computation being realized. At the second level, each processing element has to be designed in such a way that the functions of that processing element are implemented efficiently.

Arithmetic algorithms are an integral part in the implementation of a numerical computation. After all the high level algorithms and architecture have been decided, each individual component has to perform certain arithmetic operations. The design and implementation of these arithmetic algorithms directly affects the overall performance of the system. It is important for arithmetic algorithms to not only be numerically correct, but also have efficient and fast VLSI implementations. Assuming that in the implementation of a computation, due consideration regarding the VLSI design requirements such as modularity and regularity in interconnection has been taken at the higher level algorithm design, we are interested in the implementation issues at the arithmetic operations level.

In implementing numerical computations, it is often desirable to explore maximum parallelism to meet real-time application requirements. The common techniques for exploring parallelism at the *algorithm level* are parallel computation and pipelining. Parallel processing utilizes multiple processing units to speed up the

computation when there is inherent parallelism in the numerical algorithm, while pipelining effectively increases the throughput for vector processing and multi-instances computation. Both methods require the existence of certain data independencies between the operations that are to be performed in parallel. These techniques are not effective when strong data dependencies exist between consecutive operations, as is the case when a number of operations are to be performed on a single set of data in sequence. Effective use of parallelism and pipelining also requires data structures that allow access to their elements without conflict.

On-line arithmetic complements the parallel processing and pipelining techniques by introducing parallelism at the *arithmetic operations level*. By overlapping sequential operations at the digit level, it can reduce the time for computing long sequencies of arithmetic operations. Interconnection requirements are also reduced because the operands and results are transmitted digit serially.

In this dissertation, we investigate the feasibility of implementing numerical computations with on-line arithmetic algorithms.

1.2 Overview of on-line arithmetic

On-line arithmetic algorithms[3, 4, 5] perform arithmetic operations in digit-serial fashion. Generally, the operands of an operation become available one digit at a time, with the *most significant digit first*. After a small number of input digits has arrived, the most significant digit of the result is computed, and thereafter one more digit of the result is computed in each step.

The *on-line delay* δ of an on-line computation is defined as the number of digits per operand accumulated when the first digit of the result is calculated.

$$\begin{array}{lccccc} & x_1 & x_2 & x_3 & x_4 & \cdots \\ \text{input} & y_1 & y_2 & y_3 & y_4 & \cdots \\ \text{output} & & & z_1 & z_2 & \cdots \end{array}$$
$$\underbrace{\qquad\qquad}_{\delta}$$

To calculate the jth digit of the output, $j+\delta-1$ digits of each of the operands are needed. For the on-line algorithms for the basic arithmetic operations, δ is normally a small integer[6, 7, 8, 9].

A fractional number X is expressed in on-line form as

$$X[j] \equiv X[j-1] + x_{j+\delta-1} r^{-j-\delta+1}$$

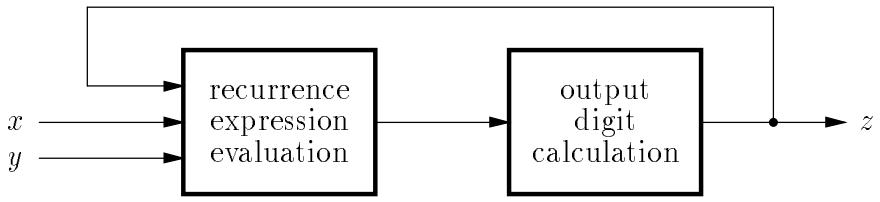


Figure 1.1: Execution of an on-line algorithm

where x_i denotes the ith digit of X, and usually its value is drawn from the symmetrical digit set

$$x_i \in \mathcal{D}_\rho = \{-\rho, \cdots, 0, \cdots, \rho\}, \qquad \frac{r}{2} \leq \rho \leq r-1$$

The redundancy factor K of the digit set $\mathcal{D}_\rho$ is defined as[10],

$$K \equiv \frac{\rho}{r-1}$$

The initial value of X is

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

Each step of an on-line algorithm consists of evaluating a recurrence expression and generating the next output digit by executing a digit selection function based on the result of the recurrence evaluation, as is shown in Figure 1.1. Assume that an on-line algorithm has two operands X and Y, and computes result Z. The recurrence expression is of the form

$$A[j] \leftarrow f(A[j-1], X[j-1], x_{j+\delta-1}, Y[j-1], y_{j+\delta-1}, z_{j-1})$$

For example, the on-line recurrence expression for multiplication is[6]

$$A[j] \leftarrow r(A[j-1] - z_{j-1}) + x_{j+\delta-1}Y[j-1] + y_{j+\delta-1}X[j] \tag{1.1}$$

On-line arithmetic algorithms have the following properties.

1. Parallelism is achieved by overlapping arithmetic operations at the digit level, as is shown in Figure 1.2. A short delay after the first digits of the operands have arrived, the first digit of the result becomes available for the following operations. The execution of the operations which use as input the result of previous operations does not have to wait for the previous ones to finish. Hence on-line arithmetic effectively overlaps arithmetic operations with strong data dependencies.

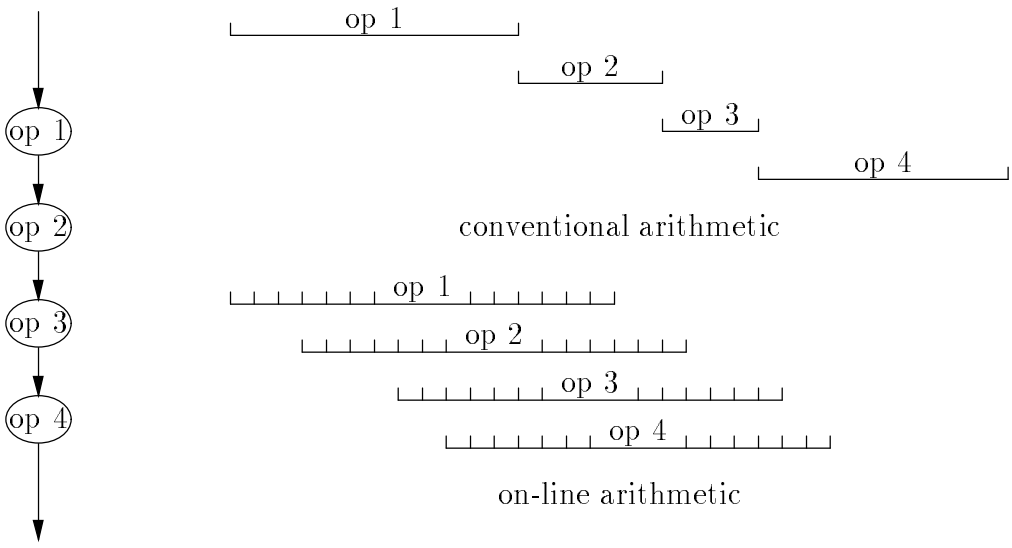


Figure 1.2: Timing of conventional and on-line arithmetic for sequence of operations

2. Since data flow through arithmetic units digit-serially, communication lines are needed for only one digit for each operand and the interconnection requirements are greatly reduced. This not only reduces the wiring area in the circuit, it also makes routing easier, and makes on-line arithmetic very attractive for VLSI implementation.

3. Since on-line algorithms compute output digits based on partial input only, redundancy in the number representation of the output is necessary. Generally, the signed-digit number representation is used, and in order for the result of one on-line computation to be used as the input of another on-line computation, it is usually assumed that the input also has the same redundant number representation system.

1.3 Normalization of floating-point numbers

A floating-point number

$$x = Xr^{-e_x}$$

is said to be *normalized* if

$$\frac{1}{r} \leq X < 1$$

When X is represented in conventional non-redundant form, it is normalized when its most significant digit (MSD) is non-zero. When X is in a redundant form, such

as signed-digit representation, it is possible that X is not normalized even though its MSD is not zero. For example,

$$0.1\bar{9}\bar{9}\bar{9}\bar{9}\bar{9} \quad = \quad 0.000001$$

where $\bar{9} = -9$.

From the definitions given in [5], a fractional number X is said to be

1. *normalized* if

$$r^{-1} \leq \quad |X| \quad < 1$$

2. *quasi-normalized* if

$$r^{-2} \leq \quad |X| \quad < 1$$

3. *pseudo-normalized* if

$$r^{-k} \leq \quad |X| \quad < 1, \qquad 3 \leq k \leq m$$

where m is the precision of X.

The result of an on-line computation may be under-normalized due to the following reasons,

1. The range of an output as determined by the on-line algorithm allows it to be under-normalized.

2. Digit cancellation of the addition/subtraction operation.

When the number representation is not maximally redundant, a number is always quasi-normalized if its MSD is non-zero. In this case, the result of an on-line computation can be quasi-normalized in the same way a conventional non-redundant number is normalized, by removing the leading zeros and adjusting the exponent until the first non-zero mantissa digit is found.

With a maximally redundant number representation, it is not trivial to guarantee that a number is normalized without examining possibly all digits of the number. In this case, a number can be quasi-normalized by the following procedure that examines the 2 most significant digits of the mantissa.

1. Remove leading zeros by left shifting the mantissa and properly adjusting the exponent.

2. If the values of the 2 most significant digits are $1\bar{x}$, where $x \in \{1, \cdots, \rho\}$, they are replaced by $0(r-x)$, where the leading zero is removed by left shifting and the procedure is repeated.

3. If values of the 2 most significant digits are $\bar{1}x$, where $x \in \{1, \cdots, \rho\}$, they are replaced by $0\overline{r-x}$, where the leading zero is removed by left shifting and the procedure is repeated.

4. If the most significant digit is not zero and the tests in (2) and (3) all fail, the result is quasi-normalized and the digits are output.

This procedure may require a maximum on-line delay of m cycles, where m is the precision of the number. It can be implemented as a separate function following the digit selection of an on-line operation, or be combined with the selection function itself.

In this dissertation, we assume that the operands of an on-line algorithm have non-zero most significant digits and are quasi-normalized. We do not include the normalization problem in the discussions of the derivation and implementation of on-line algorithms. In the discussion of application of on-line arithmetic algorithms to numerical computations in Chapter 5, we assume that the output of each on-line operation is also quasi-normalized. This is achieved by incorporating post-normalization step into the on-line algorithms for addition, multiplication and division operations. The on-line result of square root operation is always quasi-normalized. The performance and cost figures are adjusted to include the quasi-normalization of the results of on-line operations.

1.4 Review of related works

On-line algorithms for fixed-point division and multiplication were originally proposed by Trivedi and Ercegovac[6]. Trivedi and Rusnak[11] presented a formal proof of correctness of the on-line division algorithm, and presented two radix-4 on-line division algorithms. Raghavendra and Ercegovac[12] described a simulator for on-line arithmetic operations. Oklobdzija and Ercegovac[8] presented on-line algorithm for the square root operation. Watanuki and Ercegovac[7, 13, 14] presented on-line algorithms and their error analysis for floating-point arithmetic operations. Gorji-Sinaki and Ercegovac[15] gave gate level design of a digit-slice on-line arithmetic unit. Grnarov and Ercegovac[16] developed an on-line multiplicative normalization algorithm. Tullsen and Ercegovac[17] presented VLSI implementation of a radix-2 on-line algorithm computing the function $Y = AX + B$.

Ercegovac and Grnarov[18] gave an analysis of the performance and effectiveness of on-line arithmetic algorithms. Ercegovac and Lang[19] developped an on-line scheme for the computation of rotation factors, and Faris[20] described a VLSI implementation of the algorithm. Ercegovac[9] presented a simple and fast on-line algorithm for the evaluation of polynomials, certain rational functions and arithmetic expressions, solving a class of systems of linear equations, or performing the basic arithmetic operations in a fixed-point number representation system. Ercegovac and Lang[21] described the general procedure of on-line algorithm derivation with some examples.

Other types of digit-serial algorithms have been proposed in the literature. Irwin and Owens[4] presented algorithms based on the signed-digit addition algorithm.

Owens[22] described on-line algorithms based on the continued sums/products algorithms by DeLugish[23]. The algorithms consist of two types of recurrence equations. One type of recurrence computes approximations to the additive or multiplicative inverse of the input variables. The second type of recurrence uses these inverse approximations to generate the output, and are not directly dependent on the inputs. In these algorithms, the computation of the output digits of the desired function is separated from the input variables. It is claimed that these algorithms have the advantage of smaller on-line delay and smaller comparison tables for the digit selection function. But the recurrence equation for the computation of the multiplicative inverse involves variable shift addition, which is difficult to implement. As a result, the on-line delay may be small, but the digit-step time is large.

Denyer and Renshaw[24] presented VLSI design methodology using the bit-serial strategy. Their approach assumes bit-serial transmission of data, with the least-significant-bit(LSB) first. The primitive operators were ADD, MULTIPLY and DELAY. When operations such as division is encountered, the data is transformed into either bit-parallel or most-significant-bit(MSB) first format. If in a computation, division operations occur relatively frequent and are interleaved with other operations such as addition and multiplication, the data format transformation will result in severe time penalty. The application of this class of designs is thus limited to computations consisting mostly of addition and multiplication. In on-line arithmetic, the data transmission format is unified for all arithmetic operations, and the mixture of operations involved in a computation imposes no time penalty.

1.5 Outline of the dissertation

Chapter 2 begins with a formal treatment of the general derivation of on-line algorithms with considerations for implementation efficiency. It then discusses the timing and structure of on-line algorithms. Parameterized algorithms for the basic arithmetic operations are presented. Chapter 3 further generalizes the discussion of Chapter 2 and discusses how to improve the on-line computations of sequences of arithmetic operations. In Chapter 4, the implementation of on-line algorithms are discussed. Gate array designs of on-line algorithms for the basic arithmetic operations are described. Application of on-line arithmetic to the singular value decomposition computation is studied in Chapter 5. The design complexity and performance are compared between the on-line approach and conventional approaches. Chapter 6 is a summary with some suggestions for future research.

Chapter 2

Theoretical background of on-line arithmetic algorithms

Comparing to conventional arithmetic, on-line arithmetic algorithms have two distinct features. The first is that in on-line arithmetic output digits are computed before complete inputs are received. The second is that in on-line arithmetic the results are computed from left to right in all operations, unlike in conventional arithmetic. An on-line algorithm accomplishes these goals by utilizing redundancy in the number representation of the result, and keeping an internal state that reflects input digits received and output digits that have been generated. During each iteration, which we refer to as a *step*, internal computations are performed to update this internal state and calculate the next output digit. From the standpoint of the application of on-line arithmetic, it is most important that the internal computations and the output digit calculation have efficient implementations and short step times. Fortunately, the internal computations of on-line algorithms consist of only simple operations such as short precision addition/subtraction, totally parallel addition, shifting, and multiplying by a single digit. On the other hand, the need for redundancy increases the implementation complexity.

On-line arithmetic effectively overlaps sequential computations at the digit level. It is suited for applications where long sequences of arithmetic operations are performed. In such applications, the throughput of the overall computation is decided by the on-line unit that requires the longest step time, very similar to the situation in a pipelined computation. Consequently, the criteria for efficient implementation of on-line algorithms is to minimize the step time of the computation. The hardware complexity of an on-line unit, on the other hand, will only effect the area used by the particular on-line operation, and there will be no other effects on the overall computation as long as the step time is not increased.

We refer to addition, subtraction, multiplication, division and square root as the *basic arithmetic operations*. On-line algorithms can be classified according to the number of basic arithmetic operations an on-line algorithm realizes and how they interface with each other.

1. An on-line algorithm that realizes a single basic arithmetic operation is referred to as a *single-operation on-line algorithm*.

2. An on-line algorithm that realizes more than one basic arithmetic operation is referred to as a *multi-operation on-line algorithm*.

3. Generally, an on-line algorithm uses the same number representation for its input and output, such that the output of one on-line algorithm can be used directly as input of another. If an on-line computation consists of multiple on-line algorithms, and the on-line algorithms are interconnected through standard input/output interfaces, then the on-line computation is referred to as a *network of on-line algorithm*.

4. When multiple on-line algorithms are interconnected to realize a fixed sequence of operations, it is often possible to integrate these algorithms as a group and design special interfaces that utilize the knowledge of the structure of the computation to achieve better overall design. Such on-line algorithms are referred to as *composite on-line algorithms*.

Table 2.1 is a summary of different classes of on-line algorithms.

Class	number of basic arithmetic operations	number of on-line algorithms
Single-operation on-line algorithm	single	single
Multi-operation on-line algorithm	multiple	single
Network of on-line algorithm	multiple	multiple
Composite on-line algorithm	multiple	multiple (integrated)

Table 2.1: Classes of on-line algorithms

Single-operation on-line algorithms have been discussed in [6, 25, 11, 26, 8, 27, 22, 18, 7, 13, 14, 12, 28]. Applications of on-line algorithms in more complex

computations and composite algorithms are discussed in [29, 19]. A systematic method for deriving on-line algorithms was presented in [21]. Most of these works emphasize the numerical correctness of the algorithms, not their hardware implementations.

In Section 2.1, we define the model of on-line computation and derive the general form of on-line algorithms. We also discuss mathematical conditions for the derivation of on-line algorithms. In Section 2.2 the timing characteristics of on-line computations are discussed. Section 2.3 and 2.4 describe the general structure and functional components of on-line algorithms, and parameters that affect their timing and complexity. In Section 2.5 we discuss multi-operation and composite on-line algorithms. Then in Section 2.6 we discuss on-line algorithms for the basic arithmetic operations.

2.1 The general approach for the derivation of on-line algorithms

We assume arithmetic computations with two operands. The extension of the results to computations with fewer or more operands is straightforward.

Consider an arithmetic computation

$$Z = \mathcal{F}(X, Y) \tag{2.1}$$

where X, Y, Z represent fixed-point fractional numbers. In on-line arithmetic, the operands become available digit serially, with the most significant digit first. The output is computed digit by digit, with the most significant digit first, based on incomplete inputs. Consequently, the output digits must take values from a redundant digit set. Usually the operands and the result are in a redundant number representation, where each digit is assumed to belong to a symmetric digit set $\mathcal{D}_\rho$ defined as

$$\mathcal{D}_\rho \equiv \{-\rho, -\rho+1, \cdots, 0, \cdots, \rho\}$$

where ρ is the maximum absolute value for a single digit. In Chapter 3 the use of arbitrary redundant digit sets is discussed. The *redundancy factor* of $\mathcal{D}_\rho$ of radix r is defined as

$$K \equiv \frac{\rho}{r-1}$$

An important parameter of an on-line computation is the on-line delay.

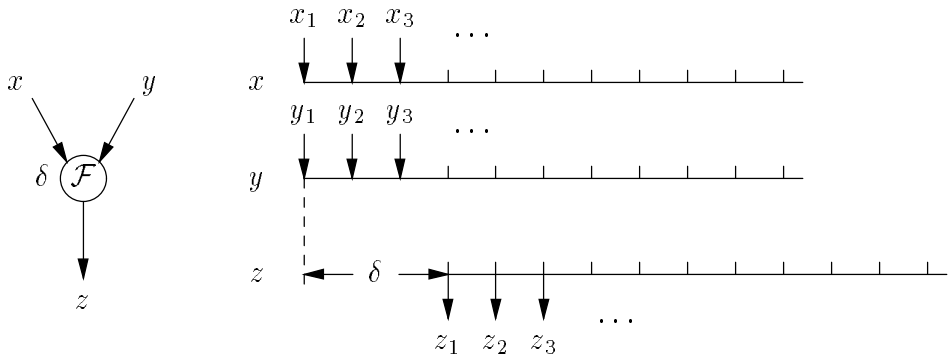


Figure 2.1: On-line delay

Definition 2.1 (On-line delay[1]) *The on-line delay of an on-line computation, denoted by δ, is the number of digits per operand that is accumulated when the first digit of the result is computed. In other words, $j+\delta-1$ digits per operand are required to compute j digits of the output.*

Figure 2.1 illustrates the on-line delay. The operands and the result of an on-line computation are represented in on-line form as

$$\begin{aligned} X[j] &= \sum_{i=1}^{j+\delta-1} x_i r^{-i} = append(X[j-1], x_{j+\delta-1}), \qquad x_{j+\delta-1} \in \mathcal{D}_\rho \\ Y[j] &= \sum_{i=1}^{j+\delta-1} y_i r^{-i} = append(Y[j-1], y_{j+\delta-1}), \qquad y_{j+\delta-1} \in \mathcal{D}_\rho \\ Z[j] &= \sum_{i=1}^{j} z_i r^{-i} = append(Z[j-1], z_j), \qquad z_j \in \mathcal{D}_\rho \end{aligned}$$

After all input digits have been exhausted, zeros are supplied as input when needed.

Definition 2.2 (Model of on-line computation) *The computation (2.1) is realized in on-line arithmetic as*

$$Z[j] = \mathcal{F}(X[j], Y[j]) + \epsilon_j r^{-j} \tag{2.2}$$

where j is the step index and ϵ_j is bounded as

$$|\epsilon_j| < 1$$

Under this definition, after $j+\delta-1$ steps, j digits of the result are available with an error bound of $\pm r^{-j}$. If m is the precision of the computation, then after step

[1]This definition is slightly different than that given in previous works, such that the calculation of on-line delay for a sequence of on-line operations is simpler.

$m + \delta - 1$, since $X[m] = X$ and $Y[m] = Y$,

$$|Z[m] - \mathcal{F}(X, Y)| \quad < \quad r^{-m}$$

and $Z[m]$ is the correct result of $\mathcal{F}(X, Y)$ with m digits of precision.

Definition 2.3 (Partial residual function) *The partial residual function for the on-line computation (2.2) is expressed as*

$$P[j] \quad = \quad \mathcal{G}(Z[j] - \mathcal{F}(X[j], Y[j]))$$

where the transformation function $\mathcal{G}$ has the following properties.

1. *The expression for $P[j]$ contains only add/subtract, multiply and shift operations.*

2. *If*

$$|\mathcal{G}(Z[j] - \mathcal{F}(X[j], Y[j]))| < \left|\mathcal{G}(r^{-j})\right| \tag{2.3}$$

   then

$$|Z[j] - \mathcal{F}(X[j], Y[j])| < r^{-j} \tag{2.4}$$

The importance of the transformation function $\mathcal{G}$ is that we can derive on-line algorithms that can be implemented efficiently, while satisfying the requirements for on-line computation given in Definition 2.2. The existence of such $\mathcal{G}$ for a specific computation is one of the limiting factors in developing on-line algorithms.

The bound $\mathcal{G}(r^{-j})$ normally contains the r^{-j} factor. To facilitate implementation, this factor is removed by scaling.

Definition 2.4 (Scaled partial residual) *The scaled partial residual function for the on-line computation (2.2) is*

$$\begin{aligned} R[j] \quad &\equiv \quad r^j P[j] \\ &= \quad r^j \mathcal{G}(Z[j] - \mathcal{F}(X[j], Y[j])) \end{aligned} \tag{2.5}$$

We now derive the on-line recurrence expression. Expanding (2.5) yields

$$R[j] \quad = \quad rR[j-1] + r^j(P[j] - P[j-1]) \tag{2.6}$$

Since the expressions for $P[j]$ and $P[j-1]$ contain only add/subtract, multiply and shift operations, after substituting $Z[j]$ by

$$Z[j] \;=\; Z[j-1] + z_j r^{-j}$$

we can separate the product terms that contain z_j from the others as

$$R[j] \;=\; rR[j-1] + \mathcal{H}_1[j] + z_j \mathcal{H}_2[j] \tag{2.7}$$

where

$$\mathcal{H}_1[j] + z_j \mathcal{H}_2[j] \;=\; r^j (P[j] - P[j-1])$$

and $\mathcal{H}_1[j]$ is independent of z_j. Let

$$A[j] \;\equiv\; rR[j-1] + \mathcal{H}_1[j] \tag{2.8}$$

then

$$R[j] \;=\; A[j] + z_j \mathcal{H}_2[j] \tag{2.9}$$

and we obtain the general form of an on-line arithmetic algorithm,

$$\begin{cases} A[j] &= r(A[j-1] + z_{j-1}\mathcal{H}_2[j-1]) + \mathcal{H}_1[j] \\ z_j &= \mathcal{S}(A[j]) \end{cases} \tag{2.10}$$

where $\mathcal{S}$ denotes the output digit selection function. To guarantee that the result computed by the on-line recurrence (2.10) converges to the correct solution, we need the following proposition.

Proposition 2.5 (Error condition) *Assume that the digit selection function $\mathcal{S}$ in (2.10) is defined for the range of its argument $A[j]$. The on-line recurrence (2.10) realizes the on-line computation of Definition 2.2 if*

$$|R[j]| \;\leq\; \varepsilon_j \tag{2.11}$$

where ε_j satisfies

$$0 \leq \varepsilon_j < r^j \left| \mathcal{G}(r^{-j}) \right| \tag{2.12}$$

PROOF. From (2.5), (2.11) and (2.12) we have

$$|P[j]| = |R[j]|\, r^{-j} \leq \varepsilon_j r^{-j} < \left| \mathcal{G}(r^{-j}) \right|$$

which is condition (2.3) in Definition 2.3. Hence (2.4) holds and the requirements given in Definition 2.2 are satisfied. ∎

Now we derive the digit selection function of (2.10). First we define the *selection intervals* for $\mathcal{S}$.

Definition 2.6 (Selection interval) *For each output digit value $k \in \mathcal{D}_\rho$, a selection interval, denoted by $[L_k, U_k]$, is defined such that if*

$$L_k \leq A[j] \leq U_k$$

then selecting the output digit as

$$z_j = \mathcal{S}(A[j]) = k$$

satisfies the error condition (2.11).

By this definition, k is a valid choice for the output digit only if $A[j] \in [L_k, U_k]$. From (2.11) and (2.9)

$$|A[j] + z_j \mathcal{H}_2[j]| \quad \leq \quad \varepsilon_j$$

hence

$$-z_j \mathcal{H}_2[j] - \varepsilon_j \leq A[j] \leq -z_j \mathcal{H}_2[j] + \varepsilon_j$$

from which we derive the expressions for L_k, U_k

$$\begin{cases} L_k = -k\mathcal{H}_2[j] - \varepsilon_j \\ U_k = -k\mathcal{H}_2[j] + \varepsilon_j \end{cases} \tag{2.13}$$

The relative positions of the selection intervals depend on whether the value of $\mathcal{H}_2[j]$ is positive or negative, as is shown in Figure 2.2. To make the notations simple, we assume $\mathcal{H}_2[j] < 0$ in the general discussions. Extension of the results for $\mathcal{H}_2[j] > 0$ is trivial.

With Definition 2.6, the selection function $\mathcal{S}$ of (2.10) produces the correct output digit if and only if for all possible values of $A[j]$,

$$A[j] \quad \in \quad [L_k, U_k]$$

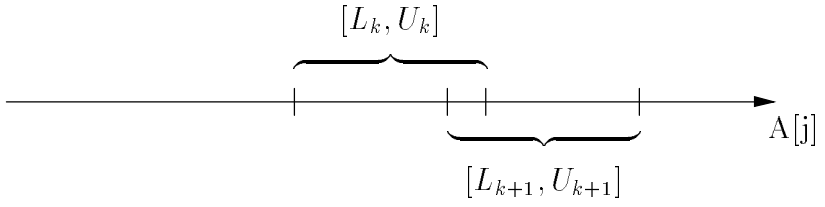
for some $k \in \mathcal{D}_\rho$.

Definition 2.7 (Digit selection function) *Let ψ_A denote the number of fractional bits of $A[j]$. A digit selection function $\mathcal{S}$ consists of a set of comparison points $\{I_k 2^{-\psi_A}\}$, for $-\rho + 1 \leq k \leq \rho$, where I_k denotes an integer, such that*

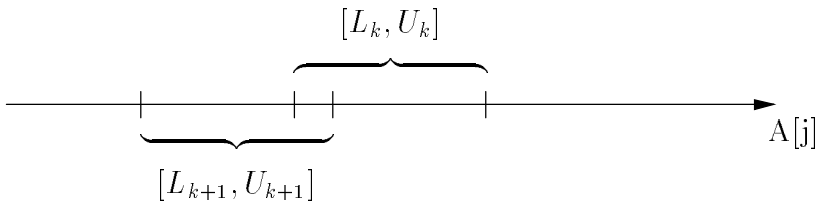
$$\mathcal{S}(A[j]) = \begin{cases} -\rho & \textbf{if } A[j] < I_{-\rho+1} 2^{-\psi_A} \\ k & \textbf{if } I_k 2^{-\psi_A} \leq A[j] < I_{k+1} 2^{-\psi_A} \textbf{ and} \\ & \quad -\rho + 1 \leq k \leq \rho - 1 \\ \rho & \textbf{if } A[j] \geq I_\rho 2^{-\psi_A} \end{cases}$$

and the comparison points must satisfy the following condition,

$$L_k \leq I_k 2^{-\psi_A} \leq U_{k-1} + 2^{-\psi_A} \tag{2.14}$$



Case 1: $\mathcal{H}_2[j] < 0$



Case 2: $\mathcal{H}_2[j] > 0$

Figure 2.2: Relative positions of selection intervals

This definition assumes that the comparison is implemented in non-redundant form. In this case, choosing $\mathcal{S}(A[j]) = k$ for $A[j] = I_k 2^{-\psi_A}$ results in simpler implementations for $\mathcal{S}$ when the comparison points $\{I_k 2^{-\psi_A}\}$ are chosen as low precision values. Figure 2.3 illustrates the definition of the comparison points.

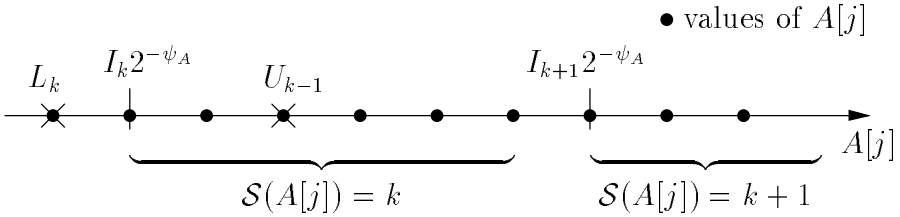


Figure 2.3: Comparison points of the digit selection function

The precision of $A[j]$ depends on the precision m of the input and output variables, the on-line delay δ, and the computation $\mathcal{F}$ being realized. For most computations,

$$\psi_A \quad = \quad m + \delta - 1$$

To meet the requirements of the error condition, the following two conditions must be satisfied.

1. (**Continuity condition**) The region covered by all selection intervals $[L_k, U_k]$ must be *continuous* with respect to the values of $A[j]$. From (2.14)

we get

$$U_{k-1} \geq L_k - 2^{-\psi_A} \tag{2.15}$$

2. (**Containment condition**) The argument $A[j]$ must be *contained* in the region covered by the selection intervals,

$$L_{-\rho} \leq A[j] \leq U_\rho \tag{2.16}$$

With Definition 2.7, it can be deduced that if the continuity condition (2.15) and containment condition (2.16) are satisfied at each step, then the on-line computation (2.10) realizes the on-line computation of Definition 2.2.

From expression (2.8), $A[j]$ is bounded as

$$-r\varepsilon_{j-1} + \min(\mathcal{H}_1[j]) \leq A[j] \leq r\varepsilon_{j-1} + \max(\mathcal{H}_1[j]) \tag{2.17}$$

Then

$$\begin{cases} -r\varepsilon_{j-1} + \min(\mathcal{H}_1[j]) & \geq & \rho\mathcal{H}_2[j] - \varepsilon_j(-\rho) \\ r\varepsilon_{j-1} + \max(\mathcal{H}_1[j]) & \leq & -\rho\mathcal{H}_2[j] + \varepsilon_j(\rho) \end{cases}$$

or

$$\begin{cases} r\varepsilon_{j-1} - \varepsilon_j(-\rho) & \leq & -\rho\mathcal{H}_2[j] + \min(\mathcal{H}_1[j]) \\ r\varepsilon_{j-1} - \varepsilon_j(\rho) & \leq & -\rho\mathcal{H}_2[j] - \max(\mathcal{H}_1[j]) \end{cases} \tag{2.18}$$

If these relations give different solutions for ε_j, the smaller one is used. When the expression for $\mathcal{H}_2[j]$ contains input or output variables, ε_j can be solved by the solution described in Appendix A as

$$\varepsilon_j \quad = \quad \frac{-\rho}{r-1}\mathcal{H}_2[j] + \mathcal{B}r^{-j} + \mathcal{C} \tag{2.19}$$

where $\mathcal{B}$ and $\mathcal{C}$ are derived from $\mathcal{H}_1[j]$ and $\mathcal{H}_2[j]$. Then the expressions for the selection intervals (2.13) become

$$\begin{cases} L_k = -(k-K)\mathcal{H}_2[j] - \mathcal{B}r^{-j} - \mathcal{C} \\ U_k = -(k+K)\mathcal{H}_2[j] + \mathcal{B}r^{-j} + \mathcal{C} \end{cases} \tag{2.20}$$

For efficient implementation of on-line algorithms, it is highly desirable to avoid full precision carry propagation in the calculation of the on-line recurrence and to have low precision inputs for the digit selection function. If the expression for $\mathcal{H}_2[j]$ contain input or output variables, the selection intervals $[L_k, U_k]$ and the comparison points I_k are functions of these variables, and the input of the digit selection function consists of these variables as well as $A[j]$. Rather than using the full precision values, low precision estimates of $A[j]$ and $\mathcal{H}_2[j]$, denoted as $\widehat{A[j]}$ and $\widehat{\mathcal{H}_2[j]}$, can be used to select the next output digit. We give the following theorem.

Theorem 2.8 *Suppose that $\widehat{A[j]}$ and $\widehat{\mathcal{H}_2[j]}$ are low precision estimates of $A[j]$ and $\mathcal{H}_2[j]$ of the on-line recurrence (2.10), respectively, such that*

$$A[j] - \eta \le \widehat{A[j]} \le A[j] + \xi \tag{2.21}$$

$$\mathcal{H}_2[j] - \mu \le \widehat{\mathcal{H}_2[j]} \le \mathcal{H}_2[j] + \nu \tag{2.22}$$

where $\eta, \xi, \mu, \nu \ge 0$ and that $\widehat{A[j]}$ has ψ_A fractional bits[2]*. If $\widehat{A[j]}$ and $\widehat{\mathcal{H}_2[j]}$ instead of $A[j]$ and $\mathcal{H}_2[j]$ are used as input, the digit selection function becomes*

$$\mathcal{S}(\widehat{A[j]}) = \begin{cases} -\rho & \textbf{if } \widehat{A[j]} < I_{-\rho+1}2^{-\psi_A} \\ k & \textbf{if } I_k 2^{-\psi_A} \le \widehat{A[j]} < I_{k+1}2^{-\psi_A} \textbf{ and} \\ & \quad -\rho + 1 \le k \le \rho - 1 \\ \rho & \textbf{if } \widehat{A[j]} \ge I_\rho 2^{-\psi_A} \end{cases} \tag{2.23}$$

Let $[\widehat{L_k}, \widehat{U_k}]$ denote the selection intervals for $\mathcal{S}(\widehat{A[j]})$, then the comparison points must satisfy the following condition

$$\widehat{L_k} \le I_k 2^{-\psi_A} \le \widehat{U_{k-1}} + 2^{-\psi_A}, \qquad k \in \{-\rho+1, \cdots, \rho\} \tag{2.24}$$

where the selection intervals are

$$\widehat{L_k} \ge \begin{cases} -(k-K)\widehat{\mathcal{H}_2[j]} + (k-K)\nu + \xi - \mathcal{B}r^{-j} - \mathcal{C}, & k \ge K \\ -(k-K)\widehat{\mathcal{H}_2[j]} - (k-K)\mu + \xi - \mathcal{B}r^{-j} - \mathcal{C}, & k < K \end{cases} \tag{2.25}$$

$$\widehat{U_k} \le \begin{cases} -(k+K)\widehat{\mathcal{H}_2[j]} - (k+K)\mu - \eta + \mathcal{B}r^{-j} + \mathcal{C}, & k > -K \\ -(k+K)\widehat{\mathcal{H}_2[j]} + (k+K)\nu - \eta + \mathcal{B}r^{-j} + \mathcal{C}, & k \le -K \end{cases} \tag{2.26}$$

and the continuity condition requires that

$$U_{k-1} - L_k \ge \begin{cases} (2k-1)(\mu+\nu) + \xi + \eta - 2^{-\psi_A}, & K \le k \le \rho \\ (2K-1)(\mu+\nu) + \xi + \eta - 2^{-\psi_A}, & -K+1 < k < K \\ -(2k-1)(\mu+\nu) + \xi + \eta - 2^{-\psi_A}, & -\rho+1 \le k \le -K+1 \end{cases} \tag{2.27}$$

PROOF. From (2.22) we have

$$\widehat{\mathcal{H}_2[j]} - \nu \le \mathcal{H}_2[j] \le \widehat{\mathcal{H}_2[j]} + \mu$$

Then from (2.20) we have the expressions of L_k and U_k as functions of $\widehat{\mathcal{H}_2[j]}$,

$$\begin{cases} -(k-K)\widehat{\mathcal{H}_2[j]} - (k-K)\mu - \mathcal{B}r^{-j} - \mathcal{C} \\ \qquad \le L_k \le -(k-K)\widehat{\mathcal{H}_2[j]} + (k-K)\nu - \mathcal{B}r^{-j} - \mathcal{C}, & k \ge K \\ -(k-K)\widehat{\mathcal{H}_2[j]} + (k-K)\nu - \mathcal{B}r^{-j} - \mathcal{C} \\ \qquad \le L_k \le -(k-K)\widehat{\mathcal{H}_2[j]} - (k-K)\mu - \mathcal{B}r^{-j} - \mathcal{C}, & k < K \end{cases} \tag{2.28}$$

[2]The number of fractional bits of $\widehat{A[j]}$ should have been denoted as $\psi_{\widehat{A}}$, but to make the notation simple, ψ_A is used in the rest of this dissertation.

$$\begin{cases} -(k+K)\widehat{\mathcal{H}_2[j]} - (k+K)\mu + \mathcal{B}r^{-j} + \mathcal{C} \\ \qquad \le U_k \le -(k+K)\widehat{\mathcal{H}_2[j]} + (k+K)\nu + \mathcal{B}r^{-j} + \mathcal{C}, & k > -K \\ -(k+K)\widehat{\mathcal{H}_2[j]} + (k+K)\nu + \mathcal{B}r^{-j} + \mathcal{C} \\ \qquad \le U_k \le -(k+K)\widehat{\mathcal{H}_2[j]} - (k+K)\mu + \mathcal{B}r^{-j} + \mathcal{C}, & k \le -K \end{cases} \tag{2.29}$$

From (2.21) we have

$$\widehat{A[j]} - \xi \le A[j] \le \widehat{A[j]} + \eta$$

Then by taking the upper bounds of (2.28) and the lower bounds of (2.29) we get the expressions for $\widehat{L_k}$ and $\widehat{U_k}$.

From (2.24) the continuity condition is

$$\widehat{U_{k-1}} \quad \ge \quad \widehat{L_k} - 2^{-\psi_A}, \qquad k \in \{-\rho+1, \cdots, \rho\} \tag{2.30}$$

From the expressions (2.25), (2.26), (2.28) and (2.29) we have

$$\widehat{U_{k-1}} - \widehat{L_k} \le \begin{cases} U_{k-1} - L_k - (2k-1)(\mu+\nu) - \xi - \eta, & K \le k \le \rho \\ U_{k-1} - L_k - (2K-1)(\mu+\nu) - \xi - \eta, & -K+1 < k < K \\ U_{k-1} - L_k + (2k-1)(\mu+\nu) - \xi - \eta, & -\rho+1 \le k \le -K+1 \end{cases}$$

which reduces (2.30) to (2.27). ∎

The efficient implementation of an on-line algorithm may require the internal computations to be realized in a different number representation system than that of the input and output of the algorithm, and the values of η and ξ depend on the number representation of $A[j]$. If $A[j]$ is in non-redundant 2's complement form and $\widehat{A[j]}$ is obtained by truncation, then $\eta = 2^{-\psi_A}$ and $\xi = 0$. If $A[j]$ is in non-redundant signed-magnitude form, and truncation is used to obtain $\widehat{A[j]}$, then $\eta = \xi = 2^{-\psi_A}$. If the internal computation is performed in 2's complement number system and $A[j]$ is in carry-save form, then $\eta = 2^{-\psi_A+1}$ and $\xi = 0$. When $A[j]$ is in signed-digit form and $\widehat{A[j]}$ is obtained by truncation, $\eta = \xi = 2^{-\psi_A}$.

Denote the overlap between adjacent selection intervals as

$$\Omega_k \quad \equiv \quad U_{k-1} - L_k$$

then

$$\Omega_k \quad = \quad 2\varepsilon_j + \mathcal{H}_2[j]$$

and (2.27) becomes

$$\varepsilon_j + \frac{\mathcal{H}_2[j]}{2}$$

$$\geq \begin{cases} (2k-1)\dfrac{\mu+\nu}{2} + \dfrac{\xi+\eta}{2} - 2^{-\psi_A-1}, & K \leq k \leq \rho \\ (2K-1)\dfrac{\mu+\nu}{2} + \dfrac{\xi+\eta}{2} - 2^{-\psi_A-1}, & -K+1 < k < K \\ -(2k-1)\dfrac{\mu+\nu}{2} + \dfrac{\xi+\eta}{2} - 2^{-\psi_A-1}, & -\rho+1 \leq k \leq -K+1 \end{cases} \quad (2.31)$$

For some computations the relation involves the variable j, and there may not be a general solution for all values of j. In such cases a general solution can be found for j within some range, and the other values of j are treated separately.

Generally, larger values of the scaled error bound ε_j result in larger overlaps between adjacent digit selection intervals, which allow lower precision input for the digit selection function.

When the functions $\mathcal{H}_1[j]$ and $\mathcal{H}_2[j]$ are known, the value of ε_j can be obtained by solving (2.18). Then expressions (2.24) and (2.31) can be used to determine the on-line delay δ, the number of fractional bits ψ_A and the comparison points I_k for the digit selection function. The number of integer bits of $\widehat{A[j]}$, denoted as ϕ_A, is determined by the upper and lower bounds of $A[j]$, which can be derived from (2.16). When the internal computation is in 2's complement carry-save form, we have $\eta = 2^{-\psi_A+1}$ and $\xi = 0$. Carry-assimilation is performed on the most significant $\sigma_A = \phi_A + \psi_A$ bit positions of $A[j]$. Expressions (2.21) and (2.31) become

$$\widehat{A[j]} \leq A[j] < \widehat{A[j]} + 2^{-\psi_A+1}$$

$$\varepsilon_j + \frac{\mathcal{H}_2[j]}{2} \geq \begin{cases} (2k-1)\dfrac{\mu+\nu}{2} + 2^{-\psi_A-1}, & K \leq k \leq \rho \\ (2K-1)\dfrac{\mu+\nu}{2} + 2^{-\psi_A-1}, & -K+1 < k < K \\ -(2k-1)\dfrac{\mu+\nu}{2} + 2^{-\psi_A-1}, & -\rho+1 \leq k \leq -K+1 \end{cases}$$

2.2 Timing of on-line computations

Let t denote the time needed for the execution of each step of an on-line algorithm. For a single on-line algorithm, the total time T_1 needed to compute an m digit result is

$$T_1 \quad = \quad (m + \delta - 1) \cdot t$$

In most on-line computations the on-line recurrence (2.10) involves accumulated partial values of the input and output, which can be calculated on-line and stored in accumulated form. As mentioned in Section 2.1, while the number representation

system used for the input and output variables is to facilitate the output digit selection of on-line operations, efficient implementation of the internal computation of on-line algorithms may require a different number system. When the internal computation uses a different number system than that of the input and output, conversion from the external to the internal number system is required. The step time t is determined by the delays of the data accumulation and conversion, the recurrence expression evaluation, and the output digit selection. A tradeoff usually exits between t and the on-line delay δ, and the best solution to improve a particular performance measure often depends on other system parameters.

Suppose for a given precision m, increasing the on-line delay by $\nabla\delta$ results in a decrease in the step time by ∇t. If we want to improve the overall computation time, then we have

$$T_1' \quad = \quad T_1 + \nabla\delta \cdot t - (m + (\delta + \nabla\delta) - 1) \cdot \nabla t$$

and when

$$(m + (\delta + \nabla\delta) - 1) \cdot \nabla t \quad > \quad \nabla\delta \cdot t$$

adopting a larger on-line delay yields shorter overall computation time. Otherwise, a smaller on-line delay with longer step time results in faster computation time.

On the other hand, if the on-line latency, or start-up time, $\delta \cdot t$, is to be minimized, we have

$$(\delta + \nabla\delta) \cdot (t - \nabla t) \quad = \quad \delta \cdot t + \nabla\delta \cdot (t - \nabla t) - \delta \cdot \nabla t$$

and when

$$\delta \cdot \nabla t \quad > \quad \nabla\delta \cdot (t - \nabla t)$$

adopting larger on-line delay improves the latency time.

Now consider an on-line computation that consists of a sequence of arithmetic algorithms. The on-line delay of the computation is

$$\Delta_{seq} \quad = \quad \sum_i \delta_i$$

where δ_i denotes the on-line delay of each individual on-line algorithm in the sequence. Figure 2.4 shows an example of the timing of a sequence of on-line computations. The total computation time is

$$T_{seq} \quad = \quad (\Delta_{seq} + m - 1)\, t_{seq}$$

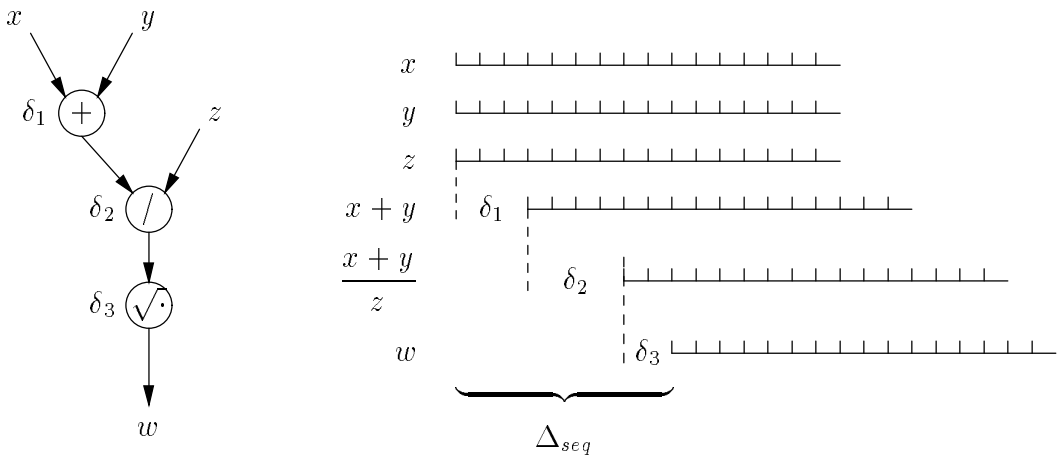


Figure 2.4: Computing $w = \sqrt{\frac{x+y}{z}}$

where m is the number of output digits and t_{seq} is the step time of the computation. Since the overall on-line computation operates in pipelined fashion, t_{seq} is determined by the on-line operation with the largest step time in the sequence. For a complex sequence of arithmetic operations, Δ_{seq} is usually quite large, and minimizing t_{seq} results in the best performance.

Generally speaking, for an on-line computation which may consist of many arithmetic operations, the total computation time is determined by the on-line delay, the step time of the computation, and the precision of the result. The total on-line delay depends on the type of on-line algorithms involved and the way in which they are connected. Within the computation there exists a critical path which is the sequence of operations with the largest on-line delay. The on-line delay of this critical path is the on-line delay of the total computation, and the largest of the step times of all on-line operations within the entire computation determines the step time of the computation.

2.3 General structure of on-line algorithms

An on-line algorithm for floating-point arithmetic computation consists of exponent and mantissa calculation. Figure 2.5 shows the functional components of a floating-point on-line algorithm. The exponent calculation consists of simple operations such as addition/subtraction and shifting. The mantissa calculation has three functional components: operand processing, recurrence expression evaluation, and output digit selection. Generally, the mantissa calculation is the critical

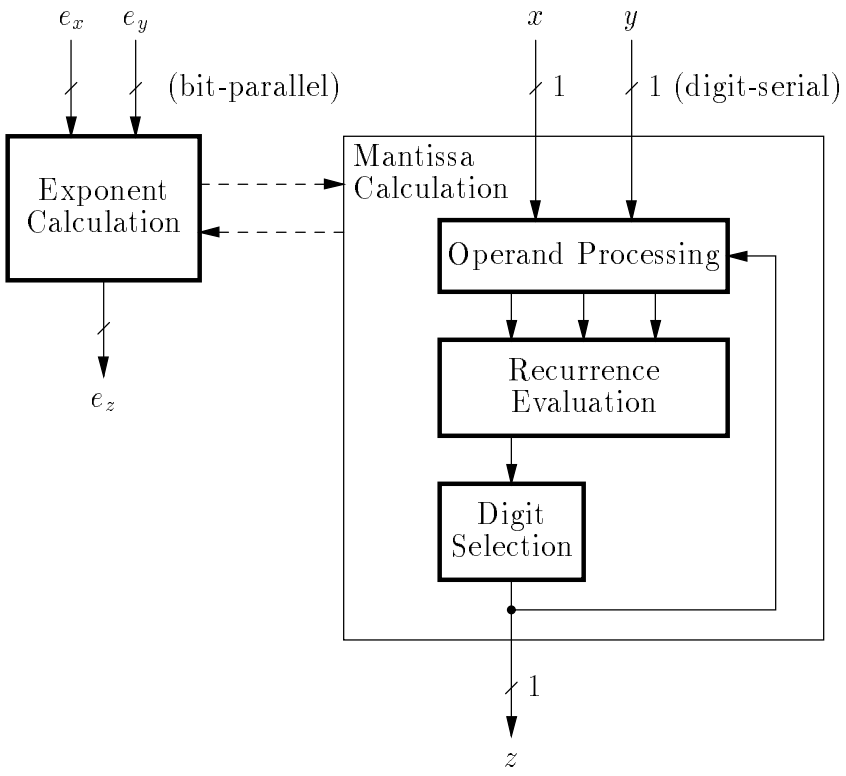


Figure 2.5: Functional components of an on-line algorithm

factor in the timing of an on-line computation. In this dissertation we assume that the exponent calculations are realized by conventional parallel schemes, and most of our discussion will focus on the mantissa calculation.

The internal number representation can have a significant effect on the performance and complexity of the implementation of an on-line computation. One major consideration in choosing an internal number system is the type of adder that can be used to implement the recurrence expression evaluation. Non-carry-propagate adders are essential to achieve short delays. Candidates for the internal number representation include signed-digit number system, and 2's complement number system. Carry-save adders can be used with 2's complement number system, while signed-digit adders can be used for the signed-digit number system. With 2's complement number system, the input data is converted to non-redundant form, and the intermediate result of the recurrence evaluation is in carry-save form. Because of its simple design and short delay, we choose 2's complement number system for the internal computation and use carry-save adders for the recurrence expression evaluation.

2.3.1 Operand processing

The operand processing provides data in the proper format required by the recurrence expression evaluation. Depending on the recurrence expression and its realization, operations performed by operand processing may include mantissa alignment, data accumulation, and conversion from redundant to non-redundant number representation. Among these functions, the data accumulation and conversion from redundant to non-redundant representation function need special attention.

The data accumulation and conversion operation accepts an operand in on-line form, and produces the accumulated values of the input digits in non-redundant form during each step of the computation. In general, the following function is realized

$$X[j] \;=\; X[j-1] + x_j r^{-j}, \qquad x_j \in \mathcal{D}_\rho \tag{2.32}$$

where $X[j]$ and $X[j-1]$ are the accumulative values in 2's complement form and x_j is in redundant form. When the next incoming digit $x_j \geq 0$, expression (2.32) requires simply appending x_j to the right of $X[j-1]$. When $x_j < 0$, the accumulation requires carry propagation, which must be avoided in order to achieve efficient implementation.

Ercegovac and Lang[30] developed a conversion algorithm that converts redundant number representations into conventional representations *on-the-fly*. In order to avoid carry propagation, two conditional forms of the intermediate result are kept, where

$$\begin{aligned} A[j-1] &= X[j-1] \\ B[j-1] &= X[j-1] - r^{-j+1} \end{aligned}$$

and $X[j]$ is obtained as

$$X[j] = A[j] = \begin{cases} A[j-1] + x_j r^{-j} & \textbf{if } x_j \geq 0 \\ B[j-1] + (r - |x_j|) r^{-j} & \textbf{if } x_j < 0 \end{cases}$$

In either case only appending operation is required. Based on the algorithm in [30], here we give an on-line algorithm that converts a number represented in redundant number system into conventional r's complement number system. The input is assumed to be on-line, that is, the input is available one digit at a time, with the most significant digit first. It does not require the input to be normalized. At the end of each step, the accumulated result is available in r's complement form, in parallel. This algorithm is useful for generating partial accumulated values of input

operands required in on-line computations such as multiplication, division and square root operation, and for converting the final result of an on-line computation into conventional non-redundant representation. The functions to be performed during each iteration are quite simple and can be implemented efficiently. For detailed verification of the algorithm, refer to [30].

Algorithm 2.1 (On-the-fly conversion)

step 1. [Initialization]

$A[0] \leftarrow 0$

$B[0] \leftarrow -1$

step 2. [On-line conversion]

for $j = 1, \cdots, m$ **do**

$$A[j] \leftarrow \begin{cases} append(A[j-1], x_j) & \textbf{if } x_j \geq 0 \\ append(B[j-1], r - |x_j|) & \textbf{if } x_j < 0 \end{cases}$$

$$B[j] \leftarrow \begin{cases} append(A[j-1], x_j - 1) & \textbf{if } x_j > 0 \\ append(B[j-1], r - |x_j| - 1) & \textbf{if } x_j \leq 0 \end{cases}$$

end.

2.3.2 The on-line recurrence evaluation

The on-line recurrence expression of (2.10)

$$A[j] \;=\; r(A[j-1] + z_{j-1}\mathcal{H}_2[j-1]) + \mathcal{H}_1[j]$$

contains only multiply, add, and shift operations. For the basic arithmetic operations, the partial residual functions contain product terms of no more than two factors, which can be expanded as

$$X[j]Y[j] \;=\; X[j-1]Y[j-1] + x_j Y[j] r^{-j} + y_j X[j-1] r^{-j}$$

As a consequence, their recurrence expressions consist of only add, shift, and multiplying a number by a single digit operations. The exact terms of $\mathcal{H}_2$ and $\mathcal{H}_1$ depend on which computation is being realized. The intermediate result, $A[j]$, of

the recurrence evaluation is used as the input to the digit selection function which calculates the next output digit.

Under the assumption that the internal computations are performed in 2's complement number representation, the operation of adding multiple terms can be implemented efficiently in carry-save form. Multiplying a fractional number by a single signed-digit can be realized by generating multiple terms using multiplexers, each term being a properly shifted version of the fractional number or its 2's complement.

2.3.3 The digit selection function

The output digit is generated by the computation defined by expression (2.23). It can be implemented by random logic, PLA, or ROM. This is usually the most time consuming operation of an on-line algorithm and its delay and complexity depend on low level design details and technology. However, the number of input bits required by the selection function gives a good indication of its implementation complexity. It is generally true that a selection function requiring a larger number of input bits is more complex and time consuming. Consequently, the criteria for improving the digit selection function of an on-line algorithm is to reduce the number of input bits.

As is discussed in Section 2.1, the output digit can be calculated from low precision estimates of the arguments, $\widehat{A[j]}$ and $\widehat{\mathcal{H}_2[j]}$. Since $\widehat{A[j]}$ is in carry-save form, an adder is used to reduce $\widehat{A[j]}$ to non-redundant form for the digit selection. This operation can be time consuming if the number of input bits from $\widehat{A[j]}$ is large.

Choosing low precision comparison points can also reduce the delay of the digit selection function, since this increases the number of don't care terms for the digit selection logic and simplifies the logic design of the output digit generation circuit.

2.4 System parameters

Here is a brief description of the main parameters that affect the cost and performance of an on-line computation. They are discussed further with the derivation of individual on-line algorithms.

The on-line delay δ is similar to the latency time of a pipelined computation. Small values of δ are desired to achieve low latency times. However, in most applications the main performance measure is the throughput, and it is sometimes preferable to sacrifice the on-line delay if a shorter step time can be achieved. For

each computation, there is a lower limit for δ which is affected by the conditions for the convergence of the algorithm and the use of low precision estimates of the intermediate results to calculate the output digits. This lower bound can be determined by expression (2.27).

The precision σ_A of $\widehat{A[j]}$ required by the digit selection function consists of integer and fractional bit positions of the result of the on-line recurrence evaluation. Since $\widehat{A[j]}$ is reduced from redundant to non-redundant form by a carry-assimilation adder, it is highly desirable to have a small σ_A since this would result in a faster digit selection function, which has a critical effect on the step time of the on-line computation. The value of σ_A is affected by the redundancy of the representation of the output, and the representation and range of the input operands. The precision of $\widehat{\mathcal{H}_2[j]}$ also affects σ_A when it contains input or output variables. Increasing the number of input bits from $\widehat{\mathcal{H}_2[j]}$ may result in reduced value of σ_A and achieve a more efficient design since $\widehat{\mathcal{H}_2[j]}$ is usually in non-redundant form.

Other system parameters that affect the implementation of an on-line algorithm include the radix and redundancy factor of the number representation system for the global computation. When the recurrence expression of an on-line algorithm includes multiplying a fraction by a single digit, a higher radix system may require more levels of carry-save addition. The digit selection function of a higher radix system is likely to have more input bits, and hence longer delay. An implementation technology related factor is that a higher radix number system will increase the load on some of the control signals, which may also affect the delay time. On the other hand, the use of a higher radix results in fewer steps and usually reduced overall computation time than a smaller radix system. A disadvantage of the higher radix system is that larger data transmission bandwidth is required for its operands and results. For a given radix, a larger redundancy factor for the output may result in a simpler and faster output digit selection function.

2.5 On-line algorithms for sequences of arithmetic operations

We now consider an on-line arithmetic system consisting of a network of on-line arithmetic units interconnected to each other. Each on-line unit is the hardware implementation of an on-line algorithm that realizes some arithmetic operations. These on-line algorithms are the basic components of an on-line computation, and they collectively determine the cost (number of units) and performance of the computation (on-line delay and throughput).

When implementing a fixed computation with on-line arithmetic, it is possible and desirable to utilize the knowledge of the structure of the computation to achieve a more efficient design than merely interconnect standard single arithmetic operation units. Here we discuss two types of on-line algorithms that realize multiple arithmetic operations in order to improve the cost and performance of an on-line computation.

The first type is *composite on-line algorithms*. A composite on-line algorithm consists of several on-line algorithms developed as a group with fixed interconnections, as is shown in Figure 2.6. Special interfaces are designed that incorporate

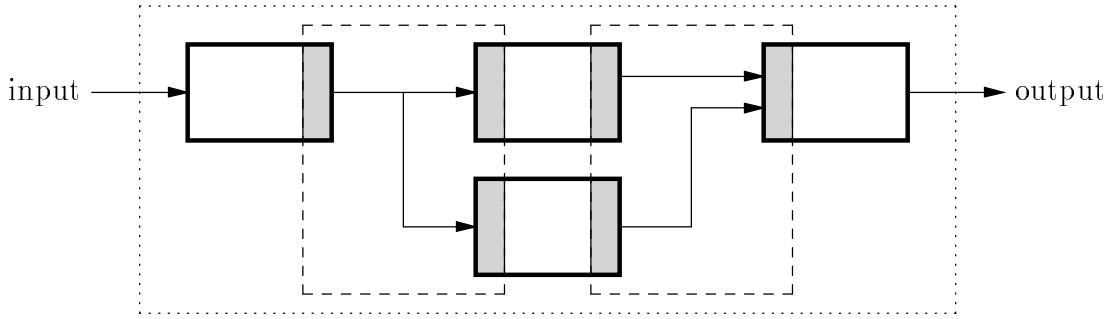


Figure 2.6: Composite on-line algorithm

the knowledge of these fixed interconnections to improve the overall design. These special interfaces may involve unorthodox number representation of data. An example of composite on-line algorithm is the on-line sum of squares and on-line square root algorithms presented in [19].

Another type of on-line algorithm realizing multiple arithmetic operations is *multi-operation on-line algorithms*. A multi-operation on-line algorithm realizes an arithmetic expression consisting of multiple arithmetic operations with a single recurrence expression and a single digit selection function, thus combining several arithmetic opreations into one on-line unit. In an on-line computation that consists of multiple on-line units, only the overall step time affects the performance. Hence even though the step time of a multi-operation on-line algorithm is likely to be larger than the on-line algorithms realizing each individual arithmetic operation, it still may be a better choice either because it does not result in increased step time of the overall computation, or because it results in better overall design according to the specific cost/performance criteria. It is also possible to compensate the increase in step time by a combination of composite and multi-operation on-line algorithm design.

In Chapter 3 we will discuss in detail the development of composite and multi-operation on-line algorithms.

2.6 Radix-r on-line algorithms for the basic arithmetic operations

Addition, subtraction, multiplication, division, and square root are the basic arithmetic operations. In designing digital systems for numerical computations it is essential that these operations be implemented efficiently. Although on-line algorithms for these operations have been published in the literature, most of them emphasize the numerical correctness of the algorithms, not their implementation efficiency. To demonstrate the feasibility of on-line arithmetic in complex numerical computations, we need to show how efficient can on-line arithmetic algorithms for the basic operations be implemented.

In this section we derive parameterized on-line algorithms for the basic arithmetic operations. The implementation of these algorithms are discussed in Chapter 4. We derive on-line algorithms with considerations of their hardware implementation. With each specific computation to be realized, we discuss design parameters that affect the performance and complexity of its implementation. We make the following assumptions.

1. The input and output of each basic operation are in the same number representation system, so that the output of one arithmetic unit can be fed into another without the need for intermediate data transformation.

2. The input variables are quasi-normalized.

3. The on-line recurrence evaluation is performed in 2's complement number system and with carry-save adders.

4. If an arithmetic operation requires more than one operand, all operands' corresponding digits become available at the same time.

2.6.1 On-line floating-point addition algorithm

In floating-point addition, the on-line calculation of the mantissa digits of the sum is quite simple. The complexity of the computation lies in the need to align the mantissas of the operands depending on the values of their exponents. In parallel addition schemes, it is required that the mantissa alignment be completed before the mantissa calculation takes place. This is not the case with on-line arithmetic. Mantissa digits of the sum can be calculated while the mantissas of the operands are still being aligned[19]. In the following, we first describe the algorithm for

operands alignment, then present the algorithm for the mantissa calculation for on-line addition.

2.6.1.1 Exponent calculation and mantissa alignment

The exponent calculation and mantissa alignment of on-line addition performs the following operations.

1. Exponent calculation

$$e_z \;=\; \max(e_x, e_y)$$

2. Mantissa alignment

$$\begin{aligned}
&e_d = e_x - e_y \\
&\textbf{if } e_d \geq 0 \qquad /* \, e_x \geq e_y \, */ \\
&\quad x'_j = x_j \qquad 1 \leq j \leq m \\
&\quad y'_j = \begin{cases} 0, & 1 \leq j \leq e_d \\ y_{j-e_d}, & e_d < j \leq m \end{cases} \\
&\textbf{else} \qquad /* \, e_x < e_y \, */ \\
&\quad x'_j = \begin{cases} 0, & 1 \leq j \leq -e_d \\ x_{j+e_d}, & -e_d < j \leq m \end{cases} \\
&\quad y'_j = y_j \qquad 1 \leq j \leq m
\end{aligned}$$

It is assumed that the difference of the exponents of the input operands is obtained in one clock cycle, so the alignment process has an on-line delay of one clock cycle. The value of the difference does not affect the on-line delay.

2.6.1.2 Mantissa computation

Denote the mantissas of the operands as X and Y, and the sum as Z,

$$Z \;=\; X + Y$$

where $X \in [r^{-2}, 1)$ and $Y \in [r^{-2}, 1)$. Their on-line representations are

$$\begin{aligned}
X[j] \;&\equiv\; \sum_{i=1}^{j+\delta-1} x_i r^{-i} = X[j-1] + x_{j+\delta-1} r^{-j-\delta+1}, \qquad x_{j+\delta-1} \in \mathcal{D}_\rho \\
Y[j] \;&\equiv\; \sum_{i=1}^{j+\delta-1} y_i r^{-i} = Y[j-1] + y_{j+\delta-1} r^{-j-\delta+1}, \qquad y_{j+\delta-1} \in \mathcal{D}_\rho \\
Z[j] \;&\equiv\; \sum_{i=1}^{j} z_i r^{-i} = Z[j-1] + z_j r^{-j}, \qquad z_j \in \mathcal{D}_\rho
\end{aligned}$$

where δ is the on-line delay for addition.

The partial residual function is

$$P[j] \;=\; X[j] + Y[j] - Z[j]$$

and

$$\begin{aligned} \mathcal{H}_1[j] &= (x_{j+\delta-1} + y_{j+\delta-1}) r^{-\delta+1} \\ \mathcal{H}_2[j] &= -1 \end{aligned}$$

The on-line recurrence expression for addition is

$$A[j] \;=\; r(A[j-1] - z_{j-1}) + x_{j+\delta-1} r^{-\delta+1} + y_{j+\delta-1} r^{-\delta+1} \qquad j \geq 1 \quad (2.33)$$

with the initial condition

$$A[0] \;=\; X[0] + Y[0]$$

The selection intervals are

$$\begin{aligned} L_k &= k - \varepsilon \\ U_k &= k + \varepsilon \end{aligned}$$

From the containment condition we have

$$r\varepsilon + 2\rho r^{-\delta+1} \;\leq\; \rho + \varepsilon$$

or

$$\varepsilon \;\leq\; K(1 - 2r^{-\delta+1}) \quad (2.34)$$

From expression (2.31) we have

$$\varepsilon \geq 2^{-1} + 2^{-\psi_A - 1} \quad (2.35)$$

where ψ_A denotes the number of fractional bit positions required for the digit selection function, assuming the intermediate result of the recurrence evaluation is in carry-save form. Expressions (2.34) and (2.35) imply that

$$2^{-\psi_A - 1} + 2Kr^{-\delta+1} \;\leq\; K - \frac{1}{2} \quad (2.36)$$

which can be used to determine the values of ψ_A and the on-line delay δ. The value of ϕ_A is determined by

$$2^{\phi_A - 1} \;\geq\; \rho + \varepsilon$$

r	ρ	K	ϕ_A	ψ_A	$\sigma_A = \phi_A + \psi_A$	δ
2^*	1	1	2	2	4^*	3
4	3	1	4	1	5	3
	2	2/3	3	2(3)	5(6)	4(3)
8	7	1	5	1	6	2
	4	4/7	4	3	7	4
			4	4	8	3
16	15	1	6	1	7	2
	8	8/15	5	4	9	4
			5	5	10	3

*For the binary case, the on-line recurrence is implemented with a 3-bit non-redundant adder. See Section 4.4 for details.

Table 2.2: Parameter values for on-line addition

The comparison points for the digit selection function must satisfy

$$\widehat{L_k} \leq I_k 2^{-\psi_A} \leq \widehat{U_{k-1}} + 2^{-\psi_A}, \qquad k \in \{-\rho+1, \cdots, \rho\} \tag{2.37}$$

where

$$\begin{aligned} \widehat{L_k} &\geq k - K + 2Kr^{-\delta+1} \\ \widehat{U_k} &\leq k + K - 2^{-\psi_A+1} - 2Kr^{-\delta+1} \end{aligned}$$

Algorithm 2.2 is the on-line addition algorithm. The selection function $\mathcal{S}_{add}$ is defined by (2.23).

For on-line addition it is always possible to choose constant values for the comparison points for the digit selection, even for high radices. As a consequence, the output digit selection logic should be relatively simple. Table 2.2 lists the values of various system parameters for on-line addition.

2.6.2 On-line floating-point multiplication algorithm

Denote the operands and their product as X, Y, and Z, respectively,

$$Z = X \cdot Y$$

where $X \in [r^{-2}, 1)$ and $Y \in [r^{-2}, 1)$. Their on-line representations are

$$X[j] \equiv \sum_{i=1}^{j+\delta-1} x_i r^{-i} = X[j-1] + x_{j+\delta-1} r^{-j-\delta+1} \qquad x_{j+\delta-1} \in \mathcal{D}_\rho$$

Algorithm 2.2 (On-line floating-point addition)

step 1. [Initialization and alignment]

$e_d \leftarrow e_x - e_y$

$e_z \leftarrow \max(e_x, e_y)$

$A[-\delta+1] = 0$, $z_0 = 0$

for $j = 1, \cdots, |e_d|$ **do**

  if $e_d \geq 0$ **then**

    $x'_j = x_j$; $y'_j = 0$

  else

    $x'_j = 0$; $y'_j = y_j$

for $j = |e_d|+1, \cdots, m$ **do**

  if $e_d \geq 0$ **then**

    $x'_j = x_j$; $y'_j = y_{j-|e_d|}$

  else

    $x'_j = x_{j-|e_d|}$; $y'_j = y_j$

for $j = -\delta+2, \cdots, 0$ **do**

  $A[j] \leftarrow rA[j-1] + (x'_{j+\delta-1} + y'_{j+\delta-1})r^{-\delta+1}$

step 2. [Output generation]

for $j = 1, \cdots, m$ **do**

  $A[j] \leftarrow r(A[j-1] - z_{j-1}) + (x'_{j+\delta-1} + y'_{j+\delta-1})r^{-\delta+1}$

  $z_j \leftarrow \mathcal{S}_{add}(\widehat{A[j]})$

end.

$$Y[j] \equiv \sum_{i=1}^{j+\delta-1} y_i r^{-i} = Y[j-1] + y_{j+\delta-1} r^{-j-\delta+1} \qquad y_{j+\delta-1} \in \mathcal{D}_\rho$$

$$Z[j] \equiv \sum_{i=1}^{j} z_i r^{-i} = Z[j-1] + z_j r^{-j} \qquad z_j \in \mathcal{D}_\rho$$

where δ is the on-line delay for multiplication.

The partial residual function is

$$P[j] = X[j]Y[j] - Z[j]$$

and

$$\begin{aligned} \mathcal{H}_1[j] &= (x_{j+\delta-1} Y[j-1] + y_{j+\delta-1} X[j]) r^{-\delta+1} \\ \mathcal{H}_2[j] &= -1 \end{aligned}$$

The on-line recurrence expression for multiplication is

$$\begin{aligned} A[j] &= r(A[j-1] - z_{j-1}) \\ &\quad + (x_{j+\delta-1} Y[j-1] + y_{j+\delta-1} X[j]) r^{-\delta+1} \qquad j \geq 1 \end{aligned} \tag{2.38}$$

with the initial condition

$$A[0] = X[0]Y[0]$$

The selection intervals are

$$\begin{aligned} L_k &= k - \varepsilon \\ U_k &= k + \varepsilon \end{aligned}$$

From the containment condition we have

$$r\varepsilon + 2\rho r^{-\delta+1} \leq \rho + \varepsilon$$

or

$$\varepsilon \leq K(1 - 2r^{-\delta+1}) \tag{2.39}$$

From expression (2.31) we have

$$\varepsilon \geq 2^{-1} + 2^{-\psi_A - 1} \tag{2.40}$$

where ψ_A denotes the number of fractional bit positions required for the digit selection function, assuming the intermediate result of the recurrence evaluation is in carry-save form. Expressions (2.39) and (2.40) imply that

$$2^{-\psi_A - 1} + 2Kr^{-\delta+1} \leq K - \frac{1}{2}$$

radix r	no. of inputs	CSA levels
2	4	2
4	6	3
8	8	4
16	10	5
32	12	5

Table 2.3: Number of CSA levels needed for multiplication (K=1)

which can be used to determine the values of ψ_A and the on-line delay δ. The value of ϕ_A is determined by

$$2^{\phi_A - 1} \quad \geq \quad \rho + \varepsilon$$

The comparison points for the digit selection function must satisfy

$$\widehat{L_k} \leq I_k 2^{-\psi_A} \leq \widehat{U_{k-1}} + 2^{-\psi_A}, \qquad k \in \{-\rho + 1, \cdots, \rho\} \tag{2.41}$$

where

$$\begin{aligned} \widehat{L_k} &\geq k - K + 2Kr^{-\delta+1} \\ \widehat{U_k} &\leq k + K - 2^{-\psi_A+1} - 2Kr^{-\delta+1} \end{aligned}$$

Algorithm 2.3 is the on-line algorithm for floating-point multiplication. The digit selection function $\mathcal{S}_{mul}$ is defined by (2.23).

The realization of the on-line recurrence (2.38) reqires two fractional by digit multipliers and a multiple input carry-save adder. The number of input terms to the adder is affected by the radix r and the maximum digit value ρ of the number representation of the operands and result. For maximally redundant number systems,

$$\lambda \quad = \quad 2 + 2\log_2 r, \qquad K = 1$$

where the r is assumed to be some power of 2. Table 2.3 lists the number of levels of carry-save adders needed for different radices with $K = 1$.

For on-line multiplication, it is always possible to choose constant values for the comparison points for the digit selection, even for high radices. As a consequence, the output digit selection logic should be relatively simple. Table 2.4 lists the values of various system parameters for on-line multiplication.

Algorithm 2.3 (On-line floating-point multiplication)

step 1. [Initialization]

$e_z \leftarrow e_x + e_y - 1$
$X[-\delta+1] = 0$
$Y[-\delta+1] = 0$
$A[-\delta+1] = 0$
for $j = -\delta+2, \cdots, 0$ **do**
$X[j] \leftarrow X[j-1] + x_{j+\delta-1} r^{-j-\delta+1}$
$A[j] \leftarrow rA[j-1] + (x_{j+\delta-1} Y[j-1] + y_{j+\delta-1} X[j]) r^{-\delta+1}$
$Y[j] \leftarrow Y[j-1] + y_{j+\delta-1} r^{-j-\delta+1}$
$Z[0] = 0$

step 2. [Product generation]

for $j = 1, \cdots, m$ **do**
$X[j] \leftarrow X[j-1] + x_{j+\delta-1} r^{-j-\delta+1}$
$A[j] \leftarrow r(A[j-1] - z_{j-1}) + (x_{j+\delta-1} Y[j-1] + y_{j+\delta-1} X[j]) r^{-\delta+1}$
$Y[j] \leftarrow Y[j-1] + y_{j+\delta-1} r^{-j-\delta+1}$
$z_j \leftarrow \mathcal{S}_{mul}(\widehat{A[j]})$

end.

2.6.3 On-line floating-point division algorithm

Denote the dividend, divisor and the quotient as N, D, and Q, respectively,

$$Q = \frac{N}{D}$$

where $N \in [r^{-2}, 1)$ and $D \in [r^{-2}, 1)$. Their on-line representations are

$$N[j] \equiv \sum_{i=1}^{j+\delta-1} n_i r^{-i} = N[j-1] + n_{j+\delta-1} r^{-j-\delta+1}, \qquad n_{j+\delta-1} \in \mathcal{D}_\rho$$

$$D[j] \equiv \sum_{i=1}^{j+\delta-1} d_i r^{-i} = D[j-1] + d_{j+\delta-1} r^{-j-\delta+1}, \qquad d_{j+\delta-1} \in \mathcal{D}_\rho$$

$$Q[j] \equiv \sum_{i=1}^{j} q_i r^{-i} = Q[j-1] + q_j r^{-j}, \qquad q_j \in \mathcal{D}_\rho$$

r	ρ	K	ϕ_A	ψ_A	$\sigma_A = \phi_A + \psi_A$	δ
2	1	1	2	1	3	4
4	3	1	3	1	4	3
	2	2/3	3	3	6	3
			3	2	5	4
8	7	1	4	1	5	2
	4	4/7	4	3	7	4
			4	4	8	3
16	15	1	5	1	6	2
	8	8/15	5	4	9	4
			5	5	10	3

Table 2.4: Parameter values for on-line multiplication

where δ is the on-line delay for division.

The partial residual function is

$$P[j] = N[j] - Q[j]D[j]$$

and

$$\begin{aligned} \mathcal{H}_1[j] &= n_{j+\delta-1} r^{-\delta+1} - d_{j+\delta-1} Q[j-1] r^{-\delta+1} \\ \mathcal{H}_2[j] &= -D[j] \end{aligned}$$

The on-line recurrence expression for division is

$$\begin{aligned} A[j] &= r(A[j-1] - q_{j-1} D[j-1]) + \\ &\quad + n_{j+\delta-1} r^{-\delta+1} - d_{j+\delta-1} Q[j-1] r^{-\delta+1} \qquad j \geq 1 \end{aligned} \tag{2.42}$$

with the initial condition

$$A[0] = N[0]$$

The digit selection intervals are

$$\begin{aligned} L_k &= kD[j] - \varepsilon_j \\ U_k &= kD[j] + \varepsilon_j \end{aligned}$$

From the containment condition we have

$$r\varepsilon_{j-1} + 2\rho r^{-\delta+1} \leq \rho D[j] + \varepsilon_j$$

or

$$r\varepsilon_{j-1} - \varepsilon_j \quad \leq \quad \rho D[j] - 2\rho r^{-\delta+1} \tag{2.43}$$

From the solution given in Appendix A we get

$$\varepsilon_j = K D[j] - \frac{K^2}{r+1} r^{-j-\delta+2} - 2Kr^{-\delta+1} \tag{2.44}$$

Assume that the quotient selection function uses a truncated version of the divisor, denoted as $\widehat{D[j]}$, as input, and that $\widehat{D[j]}$ contains ψ_D fractional bits. Then from expression (2.31), with $\mu = 0$ and $\nu = 2^{-\psi_D}$, we have

$$2\varepsilon_j - D[j] \quad \geq \quad \begin{cases} (2k-1)2^{-\psi_D} + 2^{-\psi_A}, & 1 \leq k \leq \rho \\ -(2k-1)2^{-\psi_D} + 2^{-\psi_A}, & -\rho+1 \leq k \leq 0 \end{cases} \tag{2.45}$$

where ψ_A is the number of fractional bits of $\widehat{A[j]}$. Substituting (2.44) into (2.45) and taking the worst case values of k we get

$$(2\rho-1)2^{-\psi_D-1} + 2^{-\psi_A-1} + (2 + \frac{K}{r+1} r^{-j+1}) K r^{-\delta+1} \quad \leq \quad (K - \frac{1}{2}) D[j] \tag{2.46}$$

which can be used to determine the values of ψ_A, ψ_D and the on-line delay δ. The number of integer bits of $\widehat{A[j]}$, ϕ_A, is determined by

$$2^{\phi_A - 1} \quad \geq \quad \rho \cdot \max(D[j]) + \varepsilon_j \tag{2.47}$$

The comparison points for the digit selection function must satisfy

$$\widehat{L_k} \leq I_k 2^{-\psi_A} \leq \widehat{U_{k-1}} + 2^{-\psi_A}, \qquad k \in \{-\rho+1, \cdots, \rho\} \tag{2.48}$$

where

$$\widehat{L_k} \quad \geq \quad \begin{cases} (k-K)\widehat{D[j]} + (k-K)2^{-\psi_D} + \dfrac{K^2}{r+1} r^{-j-\delta+2} + 2Kr^{-\delta+1}, & k \geq 1 \\ (k-K)\widehat{D[j]} + \dfrac{K^2}{r+1} r^{-j-\delta+2} + 2Kr^{-\delta+1}, & k \leq 0 \end{cases}$$

$$\widehat{U_k} \quad \leq \quad \begin{cases} (k+K)\widehat{D[j]} - 2^{-\psi_A+1} - \dfrac{K^2}{r+1} r^{-j-\delta+2} - 2Kr^{-\delta+1}, & k > 0 \\ (k+K)\widehat{D[j]} + (k+K)2^{-\psi_D} - 2^{-\psi_A+1} & \\ \qquad - \dfrac{K^2}{r+1} r^{-j-\delta+2} - 2Kr^{-\delta+1}, & k \leq -1 \end{cases}$$

Algorithm 2.4 is the on-line algorithm for floating-point division. The digit selection function $\mathcal{S}_{div}$ is defined by (2.23).

Algorithm 2.4 (On-line floating-point division)

step 1. [Initialization]

$e_q \leftarrow e_n - e_d + 1$

$A[0] \leftarrow \sum_{i=1}^{\delta-1} n_i r^{-i}$

$D[0] \leftarrow \sum_{i=1}^{\delta-1} d_i r^{-i}$

$Q[0] \leftarrow 0; \qquad q_0 \leftarrow 0$

step 2. [Quotient generation]

for $j = 1, \cdots, m$ **do**

$A[j] \leftarrow r(A[j-1] - q_{j-1} D[j-1]) + n_{j+\delta-1} r^{-\delta+1} -$
$d_{j+\delta-1} Q[j-1] r^{-\delta+1}$

$D[j] \leftarrow D[j-1] + d_{j+\delta-1} r^{-j-\delta+1}$

$q_j \leftarrow \mathcal{S}_{div}(\widehat{A[j]}, \widehat{D[j]})$

$Q[j] \leftarrow Q[j-1] + q_j r^{-j}$

end.

The recurrence expression evaluation is executed by adding multiple terms in carry-save form. Assume that the shifted single digit term $n_{j+\delta-1} r^{-\delta+1}$ can be combined with another term before entering the adder. Then the the number of input terms to the adder is the same as that for on-line multiplication as is discussed in Section 2.6.2.

The execution time of the quotient digit selection function constitutes a significant portion of the step time of on-line division, mainly due to the carry assimilation needed to convert the estimate $\widehat{A[j]}$ from carry-save form into non-redundant form. A large number of input bits for the carry-assimilation operation and for the digit selection logic will result in longer execution time. Hence minimizing the number of bits of $\widehat{A[j]}$ is very important. The effect of the precision of $\widehat{D[j]}$ is usually less significant because $D[j]$ can be stored in non-redundant form and hence no carry-assimilation is needed.

From equations (2.47) and (2.46) we have the parameter relations for on-line

r	ρ	K	ϕ_A	ψ_A	σ_A	ψ_D	δ
2	1	1	2	4	6	5	6
4	3	1	3	5	8	9	5
	2	2/3	3	7	10	11	5
8	7	1	4	8	12	12	4
	4	4/7	4	10	14	14	5
16	15	1	5	9	14	15	4
	8	8/15	5	13	18	18	5

Table 2.5: Parameter values for $r^{-2} \leq |D| < 1$

division

$$2^{\phi_A - 1} \geq rK \cdot \max(D[j]) - \frac{K^2}{r+1} r^{-j-\delta+2} - 2Kr^{-\delta+1}$$
$$(2\rho - 1)2^{-\psi_D - 1} + 2^{-\psi_A - 1} + (2 + \frac{K}{r+1} r^{-j+1}) Kr^{-\delta+1} \leq (K - \frac{1}{2}) \cdot \min(|D[j]|)$$

If we assume that the operands are quasi-normalized,

$$|D| \quad \in \quad [r^{-2}, 1)$$

then the relations are

$$2^{\phi_A - 1} \geq rK - \frac{K^2}{r+1} r^{-j-\delta+2} - 2Kr^{-\delta+1}$$
$$(2\rho - 1)2^{-\psi_D - 1} + 2^{-\psi_A - 1} + (2 + \frac{K}{r+1} r^{-j+1}) Kr^{-\delta+1} \leq (K - \frac{1}{2}) r^{-2}$$

Table 2.5 lists the values of various system parameters for on-line division with $r^{-2} \leq |D| < 1$.

If we assume that the divisor D is normalized,

$$|D| \quad \in \quad [r^{-1}, 1)$$

then the relations become

$$2^{\phi_A - 1} \geq rK - \frac{K^2}{r+1} r^{-j-\delta+2} - 2Kr^{-\delta+1}$$
$$(2\rho - 1)2^{-\psi_D - 1} + 2^{-\psi_A - 1} + (2 + \frac{K}{r+1} r^{-j+1}) Kr^{-\delta+1} \leq (K - \frac{1}{2}) r^{-1}$$

and Table 2.6 lists the parameter values with $r^{-1} \leq |D| < 1$.

r	ρ	K	ϕ_A	ψ_A	σ_A	ψ_D	δ
2	1	1	2	3	5	4	5
4	3	1	3	3	6	7	4
	2	2/3	3	5	8	9	4
8	7	1	4	5	9	9	3
	4	4/7	4	7	11	11	4
16	15	1	4	5	9	9	3
	8	8/15	5	9	14	1	4

Table 2.6: Parameter values for $r^{-1} \leq |D| < 1$

Tables 2.5 and 2.6 show that the number of input bits for the digit selection function and the on-line delay may be reduced by preprocessing the divisor to reduce the *size* of the range of $|D|$.

If the dividend N is on-line and the divisor D is not on-line, then the term $d_{j+\delta-1}Q_{j-1}r^{-\delta+1}$ does not appear in (2.42). In this situation, the delay time for the recurrence evaluation could be reduced. The expression (2.46) becomes

$$(2 + \frac{K}{2(r+1)}r^{-j+1})Kr^{-\delta+1} + 2^{-\psi_A-1} \quad \leq \quad (K - \frac{1}{2})D$$

which may result in better performance of the digit selection function.

If the dividend N is not on-line and D is on-line, then the term $n_{j+\delta-1}r^{-\delta+1}$ does not appear in the recurrence expression (2.42). Under our assumption that in the general situation this term does not present any separate inputs to the multiple operand addition, the delay time of the recurrence expression evaluation remains the same. The digit selection function is the same as the case when N is on-line and D is not on-line.

The step time of the on-line division algorithm can be reduced by pipelining. The way system parameters affect performance thus depends on the pipeline scheme adopted for a particular implementation. For example, to adopt a two stage pipeline scheme we modify the on-line recurrence (2.42) as

$$\begin{aligned} A[j] \quad &= \quad rA[j-1] + n_{j+\delta-1}r^{-\delta+1} \\ &\qquad -d_{j+\delta-1}Q[j-2]r^{-\delta+1} - rq_{j-1}D[j] \qquad j \geq 1 \end{aligned} \tag{2.49}$$

If we define the intermediate pipeline variable as

$$A'[j] \quad \equiv \quad rA[j-1] + n_{j+\delta-1}r^{-\delta+1} - d_{j+\delta-1}Q[j-2]r^{-\delta+1}$$

then

$$A[j] = A'[j] - rq_{j-1}D[j]$$

and in each step, the following functions are executed.

$$\begin{aligned} D[j+1] &= D[j] + d_{j+\delta}r^{-j-\delta} \\ Q[j-1] &= Q[j-2] + q_{j-1}r^{-j+1} \\ q_j &= \mathcal{S}_{div}(\widehat{A[j]}, \widehat{D[j]}) \\ A'[j+1] &= r(A'[j] - rq_{j-1}D[j]) + n_{j+\delta}r^{-\delta+1} - d_{j+\delta}Q[j-1]r^{-\delta+1} \end{aligned}$$

where $\widehat{A[j]}$ denotes truncated value of $A[j]$. The critical path includes one digit-fraction multiply, one carry-save add, and the digit selection.

2.6.4 On-line floating-point square root algorithm

Denote the mantissas of the operand and its square root as X and Y, respectively,

$$Y = \sqrt{Xr^{-p}}$$

where p is a shift factor used to adjust the range of the output. The on-line representations of the input and output are

$$\begin{aligned} X[j] &\equiv \sum_{i=1}^{j+\delta-1} x_i r^{-i} = X[j-1] + x_{j+\delta-1}r^{-j-\delta+1}, \qquad x_{j+\delta-1} \in \mathcal{D}_\rho \\ Y[j] &\equiv \sum_{i=1}^{j} y_i r^{-i} = Y[j-1] + y_j r^{-j}, \qquad y_{j+\delta-1} \in \mathcal{D}_\rho \end{aligned}$$

It is sometimes necessary to right shift one digit position the mantissa of the operand such that the output exponent is an integer. Then we have

$$Xr^{-p} \in \left(r^{-p-2} - r^{-p-3}, r^{-p}\right) \tag{2.50}$$

$$Y \in \left(r^{-\frac{p}{2}-1}\left(1 - r^{-1}\right)^{\frac{1}{2}}, r^{-\frac{p}{2}}\right) \tag{2.51}$$

The partial residual function is

$$P[j] = X[j]r^{-p} - Y[j]^2$$

and

$$\begin{aligned} \mathcal{H}_1[j] &= x_{j+\delta-1}r^{-\delta+1-p} \\ \mathcal{H}_2[j] &= -(Y[j] + Y[j-1]) \end{aligned}$$

The on-line recurrence expression for square root is

$$A[j] = r(A[j-1] - y_{j-1}(Y[j-1] + Y[j-2])) + x_{j+\delta-1} r^{-\delta+1-p}, \quad j \geq 1 \tag{2.52}$$

with the initial condition

$$A[0] \;=\; X[0] r^{-p}$$

The digit selection intervals are

$$\begin{aligned} L_k &= k(Y[j] + Y[j-1]) - \varepsilon_j \\ U_k &= k(Y[j] + Y[j-1]) + \varepsilon_j \end{aligned}$$

From the containment condition we have

$$r\varepsilon_{j-1} + \rho r^{-\delta+1-p} \;\leq\; \rho(Y[j] + Y[j-1]) + \varepsilon_j \tag{2.53}$$

or

$$r\varepsilon_{j-1} - \varepsilon_j \;\leq\; \rho(Y[j] + Y[j-1]) - \rho r^{-\delta+1-p} \tag{2.54}$$

From the solution given in Appendix A we get

$$\varepsilon_j \;=\; K(Y[j] + Y[j-1]) - K^2 r^{-j+1} - K r^{-\delta+1-p} \tag{2.55}$$

Assume that the sum $(Y[j]+Y[j-1])$ is available in non-redundant form and that a truncated version of it, with ψ_{2Y} fractional bits, is used by the digit selection function. Then from expression (2.31), with $\mu = 0$ and $\nu = 2^{-\psi_{2Y}}$, we have

$$\varepsilon_j - \frac{Y[j] + Y[j-1]}{2} \;\geq\; \begin{cases} (2k-1)2^{-\psi_{2Y}-1} + 2^{-\psi_A - 1}, & 1 \leq k \leq \rho \\ -(2k-1)2^{-\psi_{2Y}-1} + 2^{-\psi_A-1}, & -\rho + 1 \leq k \leq 0 \end{cases} \tag{2.56}$$

where ψ_A is the number of fractional bits of $\widehat{A[j]}$. Substituting (2.55) into (2.56) and taking the worst case values of k we get

$$\begin{aligned} (2\rho-1)2^{-\psi_{2Y}-1} + 2^{-\psi_A-1} + K r^{-\delta+1-p} + K^2 r^{-j+1} & \\ \leq (K - \frac{1}{2})(Y[j] + Y[j-1]) & \end{aligned} \tag{2.57}$$

Since the relation (2.57) may not have a solution for all values of j, we can choose some small integer J, such that (2.57) can be solved for $j \geq J$, and then we have

$$\begin{aligned} (2\rho-1)2^{-\psi_{2Y}-1} + 2^{-\psi_A-1} + K r^{-\delta+1-p} + K^2 r^{-J+1} & \\ \leq (K - \frac{1}{2})(Y[j] + Y[j-1]), \qquad j \geq J & \end{aligned} \tag{2.58}$$

which can be used to determine the values of ψ_A, ψ_{2Y} and the on-line delay δ. For $1 \leq j < J$, the output digit y_j can usually be selected by table lookup. The number of integer bits of $\widehat{A[j]}$, ϕ_A, is determined by

$$2^{\phi_A - 1} \quad \geq \quad 2\rho \cdot \max(Y[j]) + \varepsilon_j \tag{2.59}$$

The comparison points for the digit selection function must satisfy

$$\widehat{L_k} \leq I_k 2^{-\psi_A} \leq \widehat{U_{k-1}} + 2^{-\psi_A}, \qquad k \in \{-\rho + 1, \cdots, \rho\} \tag{2.60}$$

where

$$\widehat{L_k} \quad \geq \quad \begin{cases} (k-K)(\overline{Y[j] + Y[j-1]}) + (k-K)2^{-\psi_{2Y}} \\ \qquad\qquad +K^2 r^{-j+1} + K r^{-\delta+1-p}, & k \geq 1 \\ (k-K)(\overline{Y[j] + Y[j-1]}) + K^2 r^{-j+1} + K r^{-\delta+1-p}, & k \leq 0 \end{cases}$$

$$\widehat{U_k} \quad \leq \quad \begin{cases} (k+K)(\overline{Y[j] + Y[j-1]}) - 2^{-\psi_A+1} - K^2 r^{-j+1} - K r^{-\delta+1-p}, & k \geq 0 \\ (k+K)(\overline{Y[j] + Y[j-1]}) + (k+K)2^{-\psi_{2Y}} - 2^{-\psi_A+1} \\ \qquad\qquad -K^2 r^{-j+1} - K r^{-\delta+1-p}, & k \leq -1 \end{cases}$$

and $\overline{Y[j] + Y[j-1]}$ denotes the truncated value of $Y[j] + Y[j-1]$. Algorithm 2.5 is the on-line algorithm for floating-point square root computation. The digit selection function $\mathcal{S}_{sqrt}$ consists of a lookup table for $1 \leq j \leq J-1$, and expression (2.23) which is used for $j \geq J$.

The recurrence expression evaluation of the square root computation can be performed in two ways. We may use one data conversion unit to accumulate $Y[j]$, and then use two multiplying units to generate the terms of $(y_{j-1}Y[j-2] + y_{j-1}Y[j-1])$, as is illustrated in Figure 2.7. The total number of input terms to the multiple input adder is

$$\lambda_1 \quad = \quad 3 + 2\log_2 r$$

Another approach, shown in Figure 2.8, is to rewrite (2.52) as

$$A[j] \quad = \quad r(A[j-1] - y_{j-1}(2Y[j-2] + y_{j-1}r^{-j+1})) + x_{j+\delta-1}r^{-\delta+1}$$

The calculation of $2Y[j-2] + y_{j-1}r^{-j+1}$ is almost the same as the regular data accumulation, except that the previous accumulated value Y_{j-2} is shifted. So we can use one data conversion unit to accumulate Y_{j-1}, which is not used in the current recurrence step, and use a second data conversion unit to generate $2Y_{j-2} + y_{j-1}r^{-j+1}$. In this case the total number of input terms to the multiple input adder is

$$\lambda_2 \quad = \quad 3 + \log_2 r$$

Algorithm 2.5 (On-line floating-point square root)

step 1. [Initialization and shifting]

if $(e_x + p)$ *odd* **then**

$A[0] \leftarrow \sum_{i=1}^{\delta-p-2} x_i r^{-i-p-1}$

else

$A[0] \leftarrow \sum_{i=1}^{\delta-p-1} x_i r^{-i-p}$

$e_y \leftarrow \lceil (e_x + p)/2 \rceil + 1$

$Y[0] \leftarrow 0; \quad Y[-1] \leftarrow 0$

$y_0 \leftarrow 0$

step 2. [output generation]

for $j = 1, \cdots, m$ **do**

$A[j] \leftarrow r\ (A[j-1] - y_{j-1}(Y[j-2] + Y[j-1])) + x_{next} r^{-\delta+1}$

$y_j \leftarrow \mathcal{S}_{sqrt}(\widehat{A[j]}, \widehat{Y[j-1]})$

$Y_j \leftarrow Y_{j-1} + y_j r^{-j}$

end.

Table 2.7 shows the number of carry-save adder levels required for different radices with $K = 1$.

For the digit selection function, the key issue for efficient implementation is to minimize the number of bit positions of $\widehat{A[j]}$, which is

$$\sigma_A \quad = \quad \phi_A + \psi_A$$

Tables 2.8 and 2.9 show various parameter values for $p = 0$ and $p = 2$, respectively.

2.6.5 Remarks

From the on-line algorithm derivations and discussions we make the following general observations.

1. A larger radix r for the on-line computation results in more input bits for the output digit selection function, possibly more levels of CSA levels for the on-

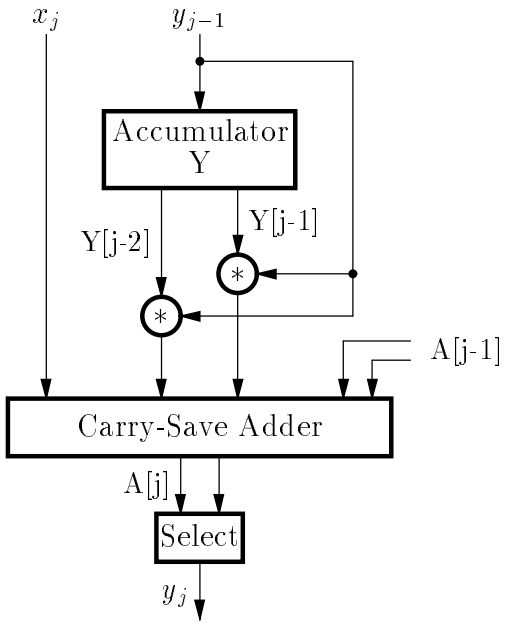


Figure 2.7: On-line square root computation: first approach

line recurrence evaluation, and hence longer step time. On the other hand, a larger radix also significantly reduces the total number of computation steps to achieve the same precision.

2. For a given radix r, a larger redundancy factor K results in reduced number of input bits for the digit selection function and smaller on-line delay. However, a larger K may also increase the number of CSA levels of the recurrence evaluation.

3. For some of the values of r and K, it is possible to reduce the number of input bits of the selection function by increasing the on-line delay.

radix	num. of inputs		CSA levels of	
r	λ_1	λ_2	λ_1	λ_2
2	5	4	3	2
4	7	5	4	3
8	9	6	4	3
16	11	7	5	4
32	13	8	5	4

Table 2.7: Number of CSA levels needed for square root ($K = 1$)

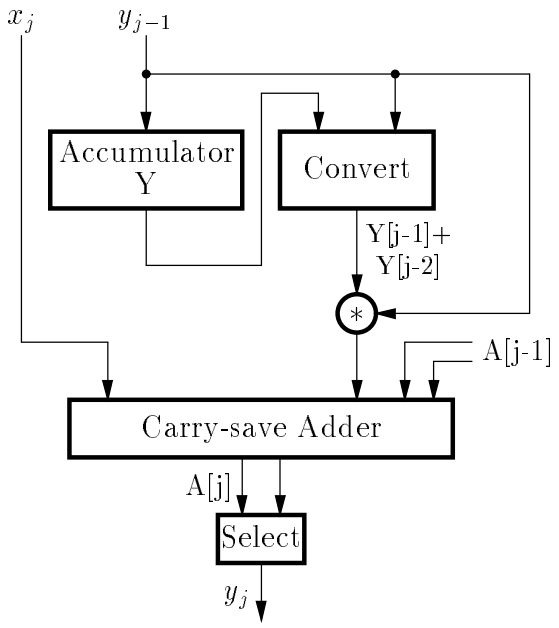


Figure 2.8: On-line square root computation: second approach

r	ρ	K	ϕ_A	ψ_A	σ_A	ψ_{2Y}	δ	J
2	1	1	3	3	6	4	4	4
4	3	1	4	3	7	7	3	3
	2	2/3	4	8	12	12	3	3
8	7	1	5	3	8	9	3	3
	4	4/7	5	8	13	13	3	3
16	15	1	6	4	10	10	3	3
	8	8/15	6	10	16	15	3	3

Table 2.8: Parameter values for $p = 0$

r	ρ	K	ϕ_A	ψ_A	σ_A	ψ_{2Y}	δ	J
2	1	1	2	5	7	7	4	4
4	3	1	2	5	7	9	2	4
	2	2/3	2	10	12	15	2	4
8	7	1	2	6	8	12	2	4
	4	4/7	2	12	16	14	2	4
16	15	1	2	8	10	14	2	4
	8	8/15	2	14	16	19	2	4

Table 2.9: Parameter values for $p = 2$

Chapter 3

On-line algorithms for sequences of arithmetic operations

In Chapter 2 we discussed the fundamental characteristics of on-line arithmetic algorithms, and described the algorithms for the basic arithmetic operations. It was mentioned that on-line arithmetic is especially suitable for computations where long sequences of arithmetic operations are involved, since such operations are effectively overlapped at the digit level. In this chapter we discuss methods to improve the performance and complexity of an on-line computation when sequential operations are being implemented. In particular, we investigate the development of on-line algorithms for realizing computations consisting of multiple arithmetic operations as special purpose digital devices.

Given on-line algorithms for addition, multiplication, division and square root operations, a computation consisting of these operations can be realized using the network of on-line algorithm approach. The step time of such a computation will be the maximum step time of all the on-line algorithms involved. A large number of basic operations usually results in a large number of arithmetic units and a large on-line delay. To improve the implementation efficiency of an on-line computation, we would like to reduce the complexity and on-line delay while keeping the step time to a minimum.

In the design of special purpose arithmetic units that realize fixed computations, the arithmetic operations to be performed and the structure of the computation are fixed and known to the designer. This knowledge can often be used to improve the implementation of the computation. For example, consider the computation

$$z = \sqrt{x^2 + y^2} \tag{3.1}$$

We can realize (3.1) by a network of single-operation on-line algorithms, as is

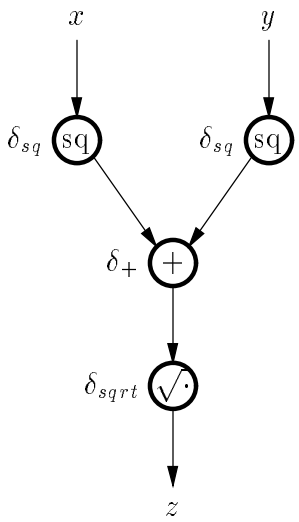


Figure 3.1: Network of on-line algorithms approach for $z = \sqrt{x^2 + y^2}$

illustrated in Figure 3.1. The implementation will consist of 4 on-line units and have an on-line delay of

$$\Delta_{net} \quad = \quad \delta_{sq} + \delta_{add} + \delta_{sqrt}$$

The performance can often be improved by realizing (3.1) as composite on-line algorithms, which adopt special interfaces that take into consideration the fixed interconnection between arithmetic operations to reduce the step time and on-line delay of the computation.

A third approach is to realize the computation as a multi-operation on-line algorithm, as shown in Figure 3.2. The complexity of the computation can often be reduced and it often results in reduced on-line delay as well. The main restriction for this approach is the step time of the overall computation, since a multi-operation algorithm is likely to be more complex and require longer step

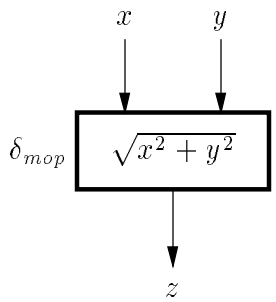


Figure 3.2: Computing $z = \sqrt{x^2 + y^2}$ as a multi-operation on-line algorithm

time than an on-line algorithm realizing any single operation involved in the computation.

In essence, the major goals for combining several arithmetic operations into a single algorithm are the effectiveness and efficiency of the implementation of the algorithm, and for developing composite algorithms, it is to tailor the parameters of the interfaces between algorithms in such a way that the overall performance is improved.

For multi-operation on-line algorithms, we need to address the following issues.

- The exponent calculation of a multi-operation algorithm is usually more complex than that of a single-operation algorithm.
- The on-line recurrence expression of a multi-operation algorithm is also more complex. Certain restrictions must be established on the form of the on-line recurrence, depending on implementation constraints. These restrictions in turn limit the types of arithmetic expressions that can be considered for multi-operation on-line algorithm realization.
- The internal operations to be performed by a multi-operation on-line algorithm is usually more complex and hence more time consuming than single-operation algorithms. In order to develop algorithms that meet performance requirements, we need to understand how system parameters affect the timing of the algorithm.

To develop composite on-line algorithms we need to understand the following.

- How digit set parameters of the input variables affect the performance parameters and implementation of an on-line algorithm.
- How digit set parameters of the output variable affect the performance parameters and implementation of an on-line algorithm.

It is often possible to improve the throughput of an on-line computation by the use of pipeline schemes. Since choosing the proper pipeline scheme requires the knowledge of low level design details, we leave its discussion with the development of individual on-line algorithms, and focus on the effect of design parameters on the implementation of the components of on-line algorithms.

This chapter is organized as the following. Section 3.1 gives a brief description of the exponent calculation of on-line algorithms. Then we discuss specific forms of arithmetic expressions that are suitable to be realized by an on-line algorithm.

Section 3.3 extends the results of Section 2.1 to allow more generalized digit set parameters for the input and output interfaces. In Sections 3.4 and 3.5 we give guidelines for developing multioperation and composite on-line algorithms with examples of such algorithms.

3.1 Exponent calculation of on-line algorithms

The exponent calculations of the single-operation on-line algorithms described in Chapter 2 consist of very few simple operations such as addition, subtraction and shifting. For multi-operation on-line algorithms usually more of these simple operations are performed which may require multiple steps to execute. In order that the exponent calculation meets the performance criteria of a computation, we assume that it is always possible to break the exponent calculation into stages such that each stage can be performed within the required step time, and that within limited range the number of steps can be adjusted by changing the step time of the computation.

The effect of a multi-step exponent calculation on the on-line delay of the overall computation dependents on certain characteristics of the computation being realized by the multi-operation algorithm. Let n_E be the number of steps of the exponent calculation and δ the on-line delay of the mantissa calculation. According to the dependency between the exponent and mantissa calculations and the values of n_E and δ, we have the following situations.

Case 1. The mantissa calculation is performed in parallel with the exponent calculation, and

$$n_E \leq \delta$$

The exponent of the result is ready when the first mantissa digit of the result is generated, and there is no extra delay. Figure 3.3 illustrates this case.

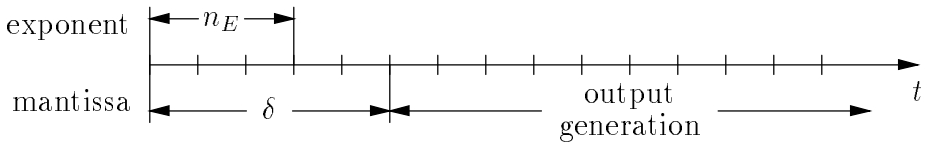


Figure 3.3: Exponent calculation: Case 1

Case 2. The mantissa calculation is performed in parallel with the exponent calculation, but

$$n_E > \delta$$

The exponent calculation does not finish before the first output mantissa digit is generated, and it causes an extra on-line delay of $n_E - \delta$ steps, as is illustrated in Figure 3.4.

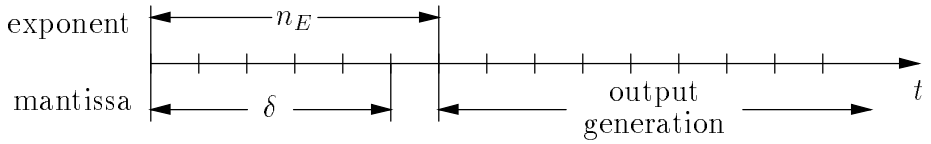


Figure 3.4: Exponent calculation: Case 2

Case 3. The start of the mantissa calculation depends on the result of the exponent calculation. This situation occurs when the computation contains addition and part of the exponent calculation and alignment have to be performed before the mantissa calculation begins. The exponent calculation causes extra on-line delay for the multi-operation on-line algorithm, as is illustrated in Figure 3.5.

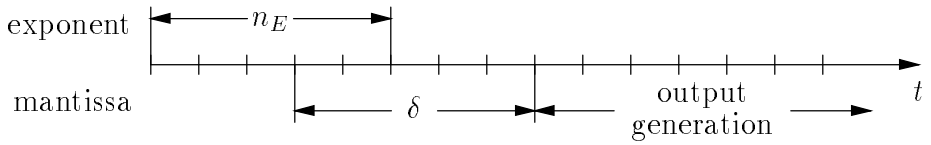


Figure 3.5: Exponent calculation: Case 3

3.2 Considerations for the on-line recurrence expression

From expression (2.10), the general form of the on-line recurrence expression is

$$A[j] \;=\; r(A[j-1] + z_{j-1}\mathcal{H}_2[j-1]) + \mathcal{H}_1[j]$$

where the operations involved in the on-line recurrence is determined by the partial residual function

$$\mathcal{H}_1[j] + z_j\mathcal{H}_2[j] \;=\; r^j(P[j] - P[j-1])$$

Since $P[j]$ is composed of addition, subtraction, multiplication, and shifting operations, it can be expressed as the sum

$$P[j] \;=\; \sum_{i=1}^{p} W^{(i)}[j] \tag{3.2}$$

where each $W^{(i)}[j]$ term is a product. Expression (3.2) can be expressed in incremental form as

$$\begin{aligned} P[j] &= \sum_{i=1}^{p} \left(W^{(i)}[j-1] + w_j^{(i)} r^{-j} \right) \\ &= P[j-1] + \sum_{i=1}^{p} w_j^{(i)} r^{-j} \end{aligned}$$

thus

$$\mathcal{H}_1[j] + z_j \mathcal{H}_2[j] = \sum_{i=1}^{p} w_j^{(i)}$$

If a product term consists of a single factor, then

$$\begin{aligned} W[j] &\equiv X[j] \\ &= X[j-1] + x_{j+\delta-1} r^{-j-\delta+1} \\ &= W[j-1] + w_j r^{-j} \end{aligned} \tag{3.3}$$

where $X[j]$ is a single number and

$$w_j = x_{j+\delta-1} r^{-\delta+1} \tag{3.4}$$

If $W[j]$ contains 2 factors then

$$\begin{aligned} W[j] &\equiv X[j] \cdot Y[j] \\ &= (X[j-1] + x_{j+\delta-1} r^{-j-\delta+1}) \cdot (Y[j-1] + y_{j+\delta-1} r^{-j-\delta+1}) \\ &= W[j-1] + w_j r^{-j} \end{aligned} \tag{3.5}$$

where $X[j]$, $Y[j]$ denote single numbers and

$$w_j = x_{j+\delta-1} Y[j] r^{-\delta+1} + y_{j+\delta-1} X[j-1] r^{-\delta+1} \tag{3.6}$$

Expression (3.5) can be conveniently computed during each step by adding two terms to the accumulated product value from the previous step, each term being the product of a full precision number and a single digit.

It is easy to show that for the on-line computation of a product of 3 or more factors, it is more efficient to compute it as a sequence of 2-input multiplications. Hence in order to achieve short step times, we require that each $W^{(i)}[j]$ term in (3.2) contain no more than 2 factors. Then the partial residual function can be expressed as

$$P[j] = \sum_{i=1}^{p} X^{(i)}[j] \cdot Y^{(i)}[j] \tag{3.7}$$

where $X^{(i)}[j]$ denotes a fractional number, and $Y^{(i)}[j]$ can be either a fractional number or 1, in which case the product term is just a single fractional number. Under this condition, the on-line recurrence expression consists of addition, subtraction, multiply a number by a single digit, and shifting operations. This requirement limits the arithmetic expressions that can be considered for multi-operation on-line algorithm implementation.

Consider the following types of arithmetic expressions.

Type 1. The arithmetic expression contains only addition and multiplication, and no division or square root operation,

$$Z = \sum_{i=1}^{p} X^{(i)} \cdot Y^{(i)} \tag{3.8}$$

where $X^{(i)}$ and $Y^{(i)}$ are single numbers. Its partial residual function is

$$P[j] = \sum_{i=1}^{p} X^{(i)}[j] \cdot Y^{(i)}[j] - Z[j]$$

which is in the form of (3.7).

Type 2. The expression contains division,

$$Z = \frac{\mathcal{A}}{\mathcal{B}} + \mathcal{C} \tag{3.9}$$

where $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are arithmetic expressions. The partial residual function is

$$P[j] = \mathcal{A}[j] - Z[j] \cdot \mathcal{B}[j] + \mathcal{C}[j] \cdot \mathcal{B}[j]$$

For this expression to be in the form of (3.7), the following conditions must hold.

1. $\mathcal{A}$ is of the form

$$\mathcal{A} \equiv \sum_{i=1}^{p} X^{(i)} \cdot Y^{(i)}$$

   where $X^{(i)}$ and $Y^{(i)}$ denote single numbers.

2. $\mathcal{B}$ is of the form

$$\mathcal{B} \equiv \sum_{i=1}^{q} V^{(i)}$$

   where $V^{(i)}$ denotes a single number.

3. $\mathcal{C}$ is of the form

$$\mathcal{C} \equiv \sum_{i=1}^{n} W^{(i)} \tag{3.10}$$

where $W^{(i)}$ denotes a single number.

Hence the expressions $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ may not contain division or square root operations.

From expression (3.6), each product of two variables in a computation results in two number by digit multiplications and two input terms to be added in the on-line recurrence evaluation. Hence the product $\mathcal{C}[j] \cdot \mathcal{B}[j]$ will result in $2qn$ number by digit multiplications and $2qn$ terms to be added, along with complicated mantissa alignment and exponent operations. It is more efficient to compute the expression (3.10) separately and then add the result with the quotient without the extra multiplications. For this reason, we eliminate the term $\mathcal{C}$ from (3.9) and define Type 2 expressions to be

$$Z = \frac{\mathcal{A}}{\mathcal{B}} \tag{3.11}$$

of which the partial residual function becomes

$$P[j] = \mathcal{A}[j] - \mathcal{B}[j] \cdot Z[j]$$

Type 3. The expression contains the square root operation,

$$Z = \sqrt{\mathcal{A}} + \mathcal{C} \tag{3.12}$$

where $\mathcal{A}$ and $\mathcal{C}$ are arithmetic expressions. The partial residual function is

$$\begin{aligned} P[j] &= \mathcal{A}[j] - (Z[j] - \mathcal{C}[j])^2 \\ &= \mathcal{A}[j] - Z[j]^2 - \mathcal{C}[j]^2 + 2 \cdot Z[j] \cdot \mathcal{C}[j] \end{aligned}$$

For this expression to be in the form of (3.7), the following must hold.

1. $\mathcal{A}$ is of the form

$$\mathcal{A} \equiv \sum_{i=1}^{p} X^{(i)} \cdot Y^{(i)}$$

where $X^{(i)}$ and $Y^{(i)}$ denote single numbers.

2. $\mathcal{C}$ is of the form

$$\mathcal{C} \equiv \sum_{i=1}^{q} V^{(i)}$$

where $V^{(i)}$ denotes a single number.

Hence the expressions $\mathcal{A}$ and $\mathcal{C}$ may not contain division or square root operations. For the same argument that was presented for Type 2 expressions, it is better not to combine the square root operation with the addition of $\mathcal{C}$ in a multi-operation on-line algorithm. Then the definition of Type 3 expression becomes

$$Z = \sqrt{\mathcal{A}} \tag{3.13}$$

and the partial residual function is

$$P[j] = \mathcal{A}[j] - Z[j]^2$$

To summarize, under the assumption that the operations allowed in the recurrence expression are add, shift, and multiply a number by a single digit, the arithmetic expressions that can be considered for implementation by a multi-operation on-line algorithm may contain at most one division or square root operation, and must be in the form of (3.8), (3.11), or (3.13).

3.3 Generalization of the output digit selection

In the discussion of Section 2.1, it was assumed that all input and output variables of an on-line algorithm have the same symmetrical digit set. Now we extend these results to allow each input and output variable to have a different, more general digit set defined as

$$\mathcal{D}_x \equiv \{\alpha_x, \alpha_x + 1, \cdots, \beta_x\} \tag{3.14}$$

where α_x, β_x are integers and we use subscripts to denote the variable which the digit set is associated with. The on-line representation of a variable x is

$$X[j] = \sum_{i=1}^{j+\delta-1} x_i r^{-i} = X[j-1] + x_{j+\delta-1} r^{-j-\delta+1}, \qquad x_i \in \mathcal{D}_x$$

The digit selection function for the digit set $\mathcal{D}_z$ is the following.

Definition 3.1 (Modified digit selection function) *Assume that* $\widehat{A[j]}$ *has* ψ_A *fractional bits, and*

$$A[j] - \eta \leq \widehat{A[j]} \leq A[j] + \xi$$

where $\eta, \xi \geq 0$. *The digit selection function* $\mathcal{S}$ *consists of a set of comparison points* $\{I_k 2^{-\psi_A}\}$, *for* $\alpha_z + 1 \leq k \leq \beta_z$, *such that*

$$\mathcal{S}(\widehat{A[j]}) = \begin{cases} \alpha_z & \textbf{if } \widehat{A[j]} < I_{\alpha_z+1} 2^{-\psi_A} \\ k & \textbf{if } I_k 2^{-\psi_A} \leq \widehat{A[j]} < I_{k+1} 2^{-\psi_A}, \alpha_z + 1 \leq k \leq \beta_z - 1 \\ \beta_z & \textbf{if } \widehat{A[j]} \geq I_{\beta_z} 2^{-\psi_A} \end{cases} \tag{3.15}$$

where the comparison points $I_k 2^{-\psi_A}$ *satisfy the condition (2.24) for*

$$k \quad \in \quad \{\alpha_z + 1, \cdots, \beta_z\}$$

The containment condition becomes

$$L_\alpha \leq A[j] \leq U_\beta \tag{3.16}$$

and from (2.13) and (2.17) we have

$$\begin{cases} r\varepsilon_{j-1} - \varepsilon_j(\alpha_z) & \leq & \alpha_z \mathcal{H}_2[j] + \min(\mathcal{H}_1[j]) \\ r\varepsilon_{j-1} - \varepsilon_j(\beta_z) & \leq & -\beta_z \mathcal{H}_2[j] - \max(\mathcal{H}_1[j]) \end{cases} \tag{3.17}$$

Since the expression for $\mathcal{H}_1[j]$ contains only digits and accumulated values of input and output variables, its boundary values can be derived from boundary values of these variables and their digit sets. If solving (3.17) gives different solutions for ε_j, the smaller one is used. In the following discussion we will assume that the solutions are equal.

The relations (3.17) can be solved for ε_j as a function of α_z, β_z, and the parameters of the input digit set. Then from (3.16) and (2.31) we can determine the number of input bits for the digit selection function and the values of other parameters.

In Appendix B, parameter relations are derived for computations of Types 1, 2, and 3. From these derivations, we make the following remarks.

- The choice of digit sets for the input and output variables should make the two upper bounds for ε_j equal. This is because the error bound is defined on the absolute value of the computation error.
- Larger redundancy in the digit set of the *output* variable increases the value of ε_j and hence reduces the complexity and delay of the digit selection function.

- The effect of the *input* digit set parameters depends on the expression for $\mathcal{H}_1[j]$. Larger number of terms and larger redundancy in the digit sets of the variables in $\mathcal{H}_1[j]$ reduces the value of ε_j and hence increases the required precision for the digit selection function. Within limited range, increasing the on-line delays of the variables may result in larger values of ε_j.

- For a given number of input bits for the digit selection function, larger redundancy in the output digit set may reduce the on-line delay.

- Under the assumption that digits of the input and output variables belong to the symmetrical digit set $\mathcal{D}_\rho$, a larger redundancy factor K results in a larger value of ε_j and shorter delay for the digit selection function.

- To a limited extent, a tradeoff exists between the on-line delay and the required precision of the digit selection function.

- For Type 2 expressions, the range of the denominator $\mathcal{B}[j]$ affects the digit selection function. The input to the digit selection may include most-significant bits from both $\mathcal{B}[j]$ and $A[j]$. In general, a smaller range for $\mathcal{B}[j]$ reduces the complexity and delay of the digit selection function.

- For Type 3 expressions, the range of the output $Z[j]$ affects the digit selection function. The input to the digit selection may include most-significant bits from both $Z[j]$ and $A[j]$. In general, a smaller range for $Z[j]$ reduces the complexity and delay of the digit selection function.

3.4 Development of multi-operation on-line algorithms

A multi-operation on-line algorithm realizes multiple arithmetic operations with a single on-line recurrence expression and a single output digit selection function. Comparing with an implementation that utilizes only single-operation on-line algorithms, the multi-operation algorithm approach has the potential of reducing hardware complexity and on-line delay.

Whether a specific arithmetic expression $\mathcal{F}$ can be realized by a multi-operation on-line algorithm conforming to the requirements specified in Section 2.1 is contingent upon finding the proper transformation function $\mathcal{G}$ satisfying the conditions of Definition 2.3. Further implementation criteria are established in Section 3.2, and the types of arithmetic expressions that can be considered for multi-operation algorithms realization based on these criteria are discussed.

Since a multi-operation on-line algorithm realizes more arithmetic operations than a single-operation algorithm, it is very likely that it requires more computations to be performed during each step and longer delay. Hence our main concern for the implementation of a multi-operation on-line algorithm is to keep the step time short. The step time of an on-line algorithm consists of the delays of the data accumulation and conversion, the on-line recurrence expression evaluation, and the output digit selection.

The data accumulation and conversion function is relatively simple, and we assume that the variation in time needed to perform its operations for different types of on-line algorithms is negligible.

Functions of the on-line recurrence evaluation consist of number by digit multiplication and multiple operand addition. The number by digit multiplication is realized by multiplexers which generate $\log_2 r$ binary multiples of the multiplicand, and feeding these terms into the multiple operand adder. We assume that the delay of this operation is constant. The delay of the multiple operand addition is affected by the number of terms to be added, the number system in which the addition is performed, and the type of adders used. More input terms to the adder usually means longer delay.

The digit selection time is assumed to be determined by the number of input bits, which may include bits from the input and output variables in addition to those from the result of the on-line recurrence evaluation. From the discussion in Sections 2.1 and 3.3, larger redundancy for the output digit set, smaller redundancy for the input digit sets, and larger on-line delay for the inputs may result in larger values of the scaled error bound ε_j and hence smaller number of input bits for the digit selection function. For Type 2 and Type 3 computations, the range of the variables in the $\mathcal{H}_2[j]$ expression also affects the selection function significantly. Smaller ranges of these variables results is reduced number of input bits for the selection.

In summary, the exponent and alignment operations of a multi-operation on-line algorithm may be more complex than those of single-operation on-line algorithms, but since these functions are performed only at the start of an on-line operation, not throughout all steps of the on-line recurrence, we assume that in the worst case the on-line delay is increased to accommodate these operations and the step time of the computation is not affected. The step time of a multi-operation on-line algorithm is mainly affected by the on-line recurrence expression and the digit selection function, and the key issues are the number of carry-save adder levels needed by the recurrence expression evaluation and the number of input bits required by the digit selection function.

Example of multi-operation on-line algorithm

Denote the mantissas of the operands and result as X, Y, and Z, respectively, and suppose we want to compute the expression

$$Z = \sqrt{(X^2 + Y^2)r^{-p}} \tag{3.18}$$

where p is a shift factor used to adjust the range of the output. The on-line representations of the input and output are

$$\begin{aligned}
X[j] &\equiv \sum_{i=1}^{j+\delta-1} x_i r^{-i} = X[j-1] + x_{j+\delta-1} r^{-j-\delta+1}, \qquad x_{j+\delta-1} \in \mathcal{D}_\rho \\
Y[j] &\equiv \sum_{i=1}^{j+\delta-1} y_i r^{-i} = Y[j-1] + y_{j+\delta-1} r^{-j-\delta+1}, \qquad y_{j+\delta-1} \in \mathcal{D}_\rho \\
Z[j] &\equiv \sum_{i=1}^{j} z_i r^{-i} = Z[j-1] + z_j r^{-j}, \qquad z_j \in \mathcal{D}_\rho
\end{aligned}$$

Assuming X and Y are quasi-normalized, positive numbers,

$$\begin{aligned}
X &\in (r^{-2}, 1) \\
Y &\in (r^{-2}, 1)
\end{aligned}$$

then

$$\begin{aligned}
\left(X^2 + Y^2\right) r^{-p} &\in \left[r^{-4-p}, 2r^{-p}\right) \\
Z &\in \left[r^{-2-\frac{p}{2}}, \sqrt{2} r^{-\frac{p}{2}}\right)
\end{aligned} \tag{3.19}$$

The partial residual function is

$$\begin{aligned}
P[j] &= (X[j]^2 + Y[j]^2)r^{-p} - Z[j]^2 \\
&= P[j-1] - z_j(Z[j] + Z[j-1]) + x_{j+\delta-1}(X[j] + X[j-1])r^{-\delta+1-p} \\
&\quad + y_{j+\delta-1}(Y[j] + Y[j-1])r^{-\delta+1-p}
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{H}_1[j] &= x_{j+\delta-1}(X[j] + X[j-1])r^{-\delta+1-p} + y_{j+\delta-1}(Y[j] + Y[j-1])r^{-\delta+1-p} \\
\mathcal{H}_2[j] &= -(Z[j] + Z[j-1])
\end{aligned}$$

The on-line recurrence expression is

$$\begin{aligned}
A[j] &= r(A[j-1] - z_{j-1}(Z[j-1] + Z[j-2])) \\
&\quad + x_{j+\delta-1}(X[j-1] + X[j])r^{-\delta+1-p} \\
&\quad + y_{j+\delta-1}(Y[j-1] + Y[j])r^{-\delta+1-p}, \qquad j \geq 1
\end{aligned}$$

r	ρ	K	ϕ_A	ψ_A	σ_A	ψ_{2Z}	δ	J
2	1	1	3	5	8	5	6	5
4	3	1	3	7	10	10	4	5
	2	2/3	2	10	12	12	4	5
8	7	1	3	10	13	15	3	5
	4	4/7	2	12	14	19	4	5

Table 3.1: Parameter values for $p = 2$ and $Z \in [r^{-3}, \sqrt{2}r^{-1})$

with the initial condition

$$A[0] \quad = \quad (X[0]^2 + Y[0]^2)r^{-p}$$

Since

$$\max(\mathcal{H}_1[j]) = -\min(\mathcal{H}_1[j]) = 4\rho r^{-\delta+1-p}$$

we get the scaled error bound as

$$\varepsilon_j \quad = \quad K(Z[j] + Z[j-1]) - K^2 r^{-j+1} - 4Kr^{-\delta+1-p}$$

Assume that $A[j]$ is in carry-save form, $(Z[j] + Z[j-1])$ is in non-redundant form, and their estimates are obtained by truncation, then $\eta = 2^{-\psi_A+1}$, $\xi = 0$, $\mu = 2^{-\psi_{2Z}}$ and $\nu = 0$, and we have the parameter relations

$$\begin{aligned} (2\rho - 1)2^{-\psi_{2Z}-1} + 2^{-\psi_A-1} + 4Kr^{-\delta+1-p} + K^2 r^{-j+1} & \\ \leq (K - \frac{1}{2})(Z[j] + Z[j-1]) & \qquad (3.20) \\ 2^{\phi_A-1} \geq 2rKZ[j] - K^2(r-2)r^{-j+1} - 4Kr^{-\delta+1-p} & \end{aligned}$$

Since relation (3.20) does not have a general solution for all values of j, it can be solved for $j \geq J$, where J is some small integer value, while the selection function for $j < J$ is defined separately. Then we have

$$(2\rho - 1)2^{-\psi_{2Z}-1} + 2^{-\psi_A-1} + 4Kr^{-\delta+1-p} + K^2 r^{-J+1}$$

The implementation of the recurrence expression consists of $2 + 3\log_2 r$ levels of carry-save adders. Table 3.1 shows parameter values for the on-line computation of expression (3.18) for $p = 2$.

Table 3.2 lists the parameter values for the computation (3.18) with a network of 2 multiplication, 1 addition, and 1 square root algorithms. The parameters σ_A,

r	K	σ_A	ψ_{2Z}	J	Δ_{net}
2	1	6	4	4	11
4	1	7	7	3	9
	2/3	12	12	3	9
8	1	8	9	3	7
	4/7	13	13	3	9

Table 3.2: Parameter values for the network approach

ψ_{2Z} and J are the values for the square root algorithm, since it has the longest delay. The total on-line delay Δ_{net} is the calculated from the values given in Tables 2.2, 2.4 and 2.8. The multi-operation approach will have a longer step time but shorter on-line delay than the network approach.

3.5 Development of composite on-line algorithms

Composite on-line algorithms improve the performance of an on-line computation by integrating successive arithmetic operations with special interfaces that take into consideration the characteristics of the specific operations involved. Usually both the on-line delay and step time of the computation can be reduced.

The interface between two on-line algorithms consists of the digits generated by one algorithm which are sent to the other algorithm as input. This interface is characterized by the digit set that the digits belong to.

When developing composite on-line algorithms, the objective is to reduce the global step time of the computation, which is determined by the largest step time of a single on-line algorithm within the computation. Usually the step times of different on-line algorithms are not the same, so the algorithm with the largest step time becomes critical in the sense that if this step time is reduced, then the overall computation is improved.

There are two possible ways to reduce the step time of the critical algorithm.

1. Changing the digit set parameters of the input variables such that the step time of the algorithm is reduced. This may cause the step times of the previous algorithms to increase.

2. Changing the digit set parameters of the output variable such that the step

	Network				Composite			
	ρ_{in}	ρ_{out}	δ	σ_A	ρ_{in}	ρ_{out}	δ	σ_A
Mult	2	2	4 (3)	5 (6)	2	3	2 (3)	2 (1)
Add	2	2	5 (4)	5 (6)	3	2	5 (4)	5 (7)

Table 3.3: Parameter values for $XY + WV$ (r=4)

time of the algorithm is reduced. This may cause the step time of the following algorithm of the computation to increase.

Usually the output interface has a greater effect than the input interface on the delay of the digit selection function, which is most time consuming component of the step time.

Example of composite on-line algorithm

As an example, consider the expression

$$Z = XY + WV$$

where we assume

$$\begin{aligned} X &\in (r^{-2}, 1) \\ Y &\in (r^{-2}, 1) \\ V &\in (r^{-2}, 1) \\ W &\in (r^{-2}, 1) \end{aligned}$$

Table 3.3 lists the parameters for the network and the composite on-line algorithm schemes, with $r = 4$ and $\rho = 2$ for the input and output. For the network scheme, the input and output of the addition and multiplication all have $\rho = 2$, and the total on-line delay is

$$\Delta_{net} = 4 + 5 = 9$$

For the composite approach, the output of the multiplication operation and the input of addition have $\rho = 3$. The calculation of the difference of the exponents in on-line addition overlaps can be performed before the the mantissa of the inputs to the adder is generated, hence

$$\Delta_{comp} = 2 + 5 - 1 = 6$$

Chapter 4

Gate array implementation of on-line algorithms

In this chapter we present gate array designs of on-line arithmetic units for radix-2 floating-point addition, multiplication, division and square root operations. Performance and complexity characteristics of the implementations of on-line arithmetic units are discussed. A signed radix-2 digit is referred to as a *sbit*. It is assumed that the operands and results of the computations each has an 8-bit exponent and a 24-sbit mantissa. The mantissa of an operand or result is transmitted on-line, while the exponent is transmitted in parallel form simultaneously with the first sbit of the mantissa. The exponent of the result is computed in parallel form, while the mantissa is computed on-line. We choose radix-2 because it can serve as a basic measuring stick for higher radix on-line arithmetic designs. To estimate the implementation characteristics, the designs are captured and simulated with the WORKVIEW[1] CAD tools[31] and the LSI[2] Design Kit[32] by Viewlogic Systems, inc.. The gate level complexity of the designs are measured in terms of *equivalent gate*, which is equivalent to two n-channel and two p-channel transistors. Under these assumptions, the timing and complexity measurements are based on the data given for the LSI LL9000 series gate array libraries[3].

We first discuss the general structure of on-line arithmetic units, and the design of components common to the implementation of some of the arithmetic operations. Then the gate array design of radix-2 on-line division unit is presented including the derivation of binary level algorithms and design parameters. We discuss in

[1]WORKVIEW is a trademark of View*logic* Systems, Inc..

[2]LSI is a trademark of LSI Logic Corporation.

[3]The currently available LSI LCA10000 series is much faster than the LL9000 series used in this work.

detail the performance and complexity of the design. Then designs of on-line units for addition, multiplication and square root operations are presented. It is assumed that the recurrence computations are implemented in redundant form.

4.1 Organization of an on-line arithmetic unit

The design of an on-line arithmetic unit is organized as a linear array of modules as is shown in Figure 4.1. The first module M_1 contains components for exponent calculation and output digit selection, and digit slices for the most significant portion of the mantissa computation. The modules $M_2, \cdots, M_s$ each consists of a number of digit slices for the mantissa computation. Each digit slice contains circuit components realizing data accumulation and conversion, fraction digit multiplier, and carry-save adder. Each module is connected to its immediate neighbors only. The precision of the computation can be increased by simply adding more modules to the array, while the step time and on-line delay remain the same.

The number of digit slices required in an on-line computation for a given precision is usually less than that required for a parallel implementation. It depends on the precision of the digit selection function and the number of output digits to be computed[3]. Denote m as the number of digits to be computed, ϕ_A and ψ_A the number of integer and fractional bits of $\widehat{A[j]}$ required by the digit selection function where $\widehat{A[j]}$ is the low precision estimate of the on-line recurrence evaluation result, and δ the on-line delay. When all operands of the computation are on-line, the total number of digit slices n is

$$n = \left\lceil \frac{m + \psi_A + \delta}{2} \right\rceil + \phi_A \tag{4.1}$$

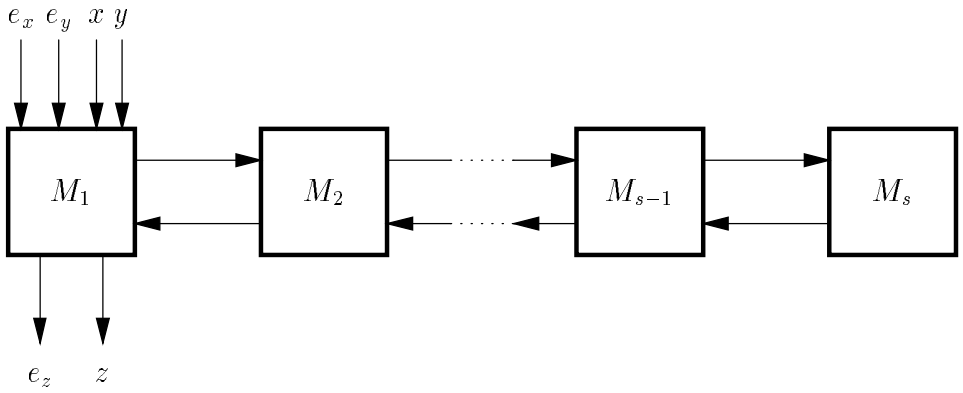


Figure 4.1: Modular structure of an on-line arithmetic unit

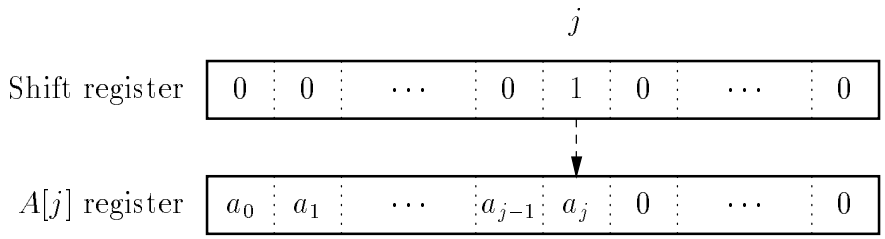


Figure 4.2: Control scheme for the next digit loading position

4.2 Design of functional components

A floating-point on-line arithmetic computation consists of exponent and mantissa calculation. Since the exponent calculation is performed in parallel form, the main component of its implementation is a parallel adder. Functional components of the mantissa calculation include the data accumulation and conversion, fraction by digit multiplication, carry-save adder, low precision carry assimilation adder, and output digit generation logic.

4.2.1 Data accumulation and conversion unit

In Algorithm 2.3.1 of Section 2.3, two variables, $A[j-1]$ and $B[j-1]$, are needed during step j. These variables can be kept in two registers. If only the final accumulative value of the input digits is needed, we can use the scheme suggested in [30], in which the partial accumulative values are shifted during each step, and the incoming digits are always inserted at the least significant digit position. In most on-line algorithms, the accumulative values of the input are used in the on-line recurrence. In order to avoid the variable shift and add operations, the values of $A[j]$ are stored in alignment with respect to the recurrence evaluation adder. This requires that the digit position where the incoming digits are loaded to move from left to right one position per recurrence. The control can be implemented with a shift register which has one of the bits set to 1 to indicate the loading position and the rest of the bits set to 0, as is illustrated in Figure 4.2.

Our design of the data accumulation and conversion component is based on the scheme described by Tullsen[33], which implements a radix-2 conversion algorithm that is functionally equivalent to and derived from the algorithm in [30], but results in slightly simpler design.

In this algorithm, $B[j]$ is not explicitly formed during each step. Instead, each bit position of $A[j]$, denoted as $A_i[j]$, has a flag C_i, which indicates whether the current bit is *confirmed(c)* or *unconfirmed(u)*. An *init* signal is applied to all bit

init/ld	previous bit·flag	input 0	1	−1
0/0	0c	0c	0c	0c
	0u	0u	0c	1c
	1c	1c	1c	1c
	1u	1u	1c	0c
1/∗		0c	0c	0c
0/1		0u	1u	1u

Table 4.1: Conversion transition table

positions at the beginning of the computation, and a *ld* signal, generated by a shift register, is applied to each bit position in succession(i.e., the *ld* signal is set to 1 for bit i when the ith input sbit becomes available). For each bit, its value and flag are set according to the transition table given in Table 4.1.

Let $C[j]$ denote the vector of flags associated with $A[j]$, and $Z[j]$ the shift register, where

$$Z_l[j] \quad = \quad \begin{cases} 1 & \textbf{if } l = j \\ 0 & \textbf{if } l \neq j \end{cases}$$

Algorithm 4.1 is the binary level data accumulation and conversion algorithm.

The gate array design of the bitslice of the radix-2 on-the-fly conversion unit is shown in Figure 4.3.

4.2.2 Fraction by digit multiplier

The fraction by digit multiplier realizes the computation of

$$Z \quad = \quad x_j \cdot Y, \qquad x_j \in \mathcal{D}_\rho$$

where x_j is a single signed digit, and Y and Z are fractional numbers. Let l denote the largest possible number of 1s in the representation of x_j. This multiplication can be implemented by using multiplexers to generate l binary multiples of Y. For radix-r system, we have

$$l \quad \leq \quad \lceil \log_2 r \rceil$$

The product to be computed is equal to the sum of these binary multiples. The summation is combined with the summation for the on-line recurrence expression evaluation and performed by a multiple input adder.

Algorithm 4.1 (Radix-2 on-the-fly conversion algorithm)

step 1. [Initialization]

$A[0] \leftarrow \langle 000 \cdots 0 \rangle$

$C[0] \leftarrow \langle 111 \cdots 1 \rangle$

$Z[0] \leftarrow \langle 100 \cdots 0 \rangle$

step 2. [on-line recurrence]

for $j = 1, 2, \cdots, m$ **do**

$$A_i[j] \leftarrow \begin{cases} 0 & \textbf{if } Z_i[j-1] = 1 \textbf{ and } x_j = 0 \\ 1 & \textbf{if } Z_i[j-1] = 1 \textbf{ and } x_j \neq 0 \\ A_i[j-1] & \textbf{if } Z_i[j-1] = 0 \\ & \quad \textbf{and } (C_i[j-1] = 1 \textbf{ or } x_j \geq 0) \\ \overline{A_i[j-1]} & \textbf{if } Z_i[j-1] = 0 \textbf{ and} \\ & \quad C_i[j-1] = 0 \textbf{ and } x_j < 0 \end{cases}$$

$$C_i[j] \leftarrow \begin{cases} 0 & \textbf{if } (Z_i[j-1] = 1) \textbf{ or } (Z_i[j-1] = 0 \\ & \quad \textbf{and } C_i[j-1] = 0 \textbf{ and } x_j = 0) \\ 1 & \textbf{if } Z_i[j-1] = 0 \textbf{ and} \\ & \quad (C_i[j-1] = 1 \textbf{ or } x_j \neq 0) \end{cases}$$

$$Z_i[j] \leftarrow \begin{cases} 0 & i = 1 \\ Z_{i-1}[j-1] & i \neq 1 \end{cases}$$

end.

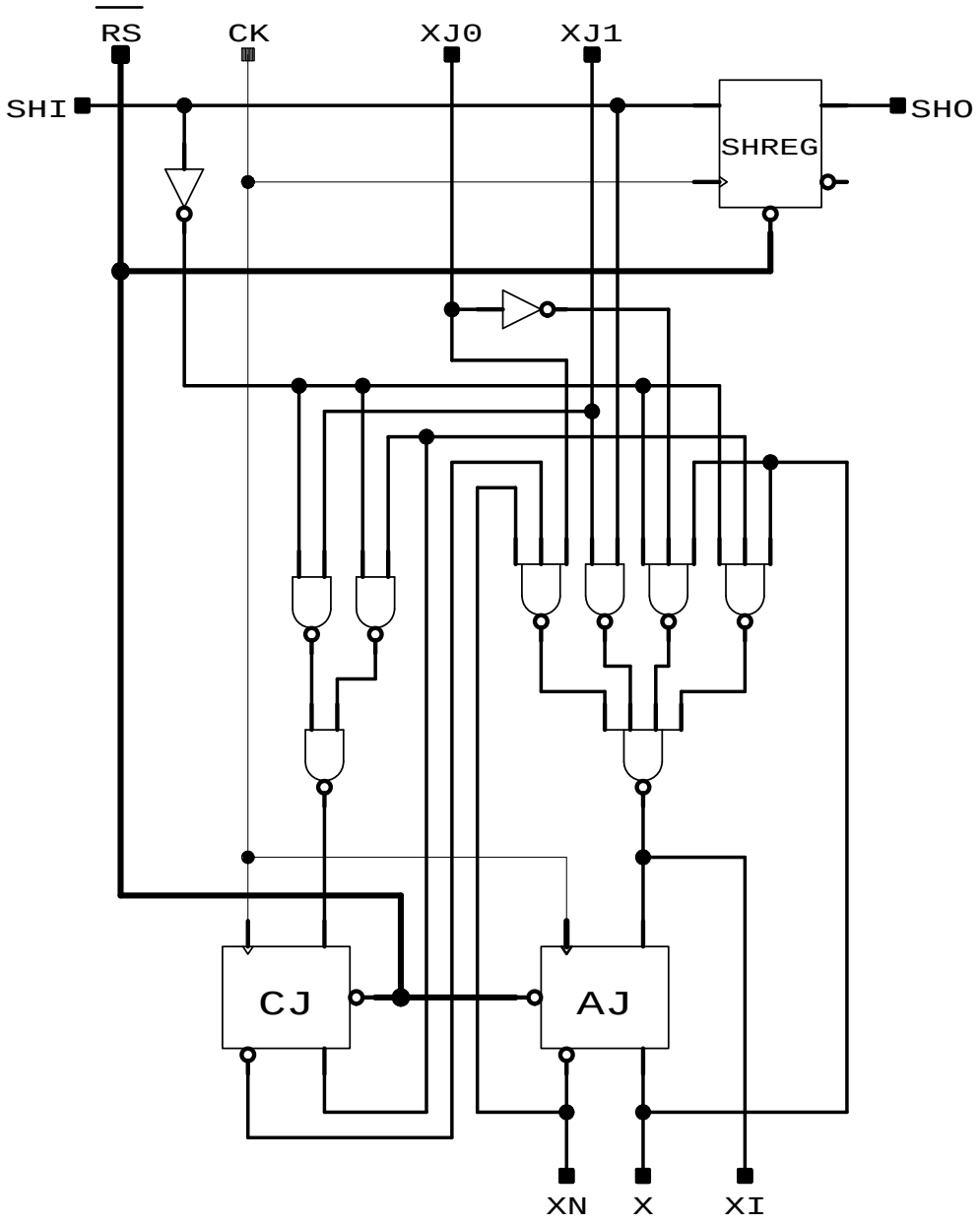


Figure 4.3: Radix-2 on-the-fly conversion bitslice

4.2.3 Multiple input adder

A multiple operand adder is needed for the on-line recurrence expression evaluation. We choose to implement the addition in carry-save form. The design of the multiple operand adder is constructed with multiple levels of 3-to-2 carry-save adders(CSA), where the number of CSA levels is determined by the total number of binary terms to be added. Each CSA is a linear array of full adders(FA). The delay of the adder depends on the delay of a single CSA and the number of CSA levels.

4.2.4 Carry assimilation adder

To simplify the digit selection function, the result of the on-line recurrence which is in carry-save form is first reduced to non-redundant form by a limited precision carry assimilation adder. Carry-look-ahead schemes are used to reduce the delay. Since the number of bit positions to be carry-assimilated varies depending on the particular computation being implemented, the actual size and delay of this adder also differs. The carry-assimilation adder is in the critical path of the on-line recurrence and its time delay is a significant part of the step time. Minimizing the number of input bits of the carry assimilation is highly desirable to achieve small step time of the on-line computation.

4.2.5 Variable delay component

Some arithmetic operations, such as addition, require the alignment of two operands before computing the output mantissa. The alignment requires right-shifting the mantissa of one operand. In on-line arithmetic, operand alignment is realized by delaying the incoming digits of the mantissa that is to be shifted, and supplying 0s to the on-line recurrence. Since the amount of shifting usually depends on the values of the operands themselves, variable delay of an operand may be necessary. Note that in on-line operations, the recurrence computation does not have to wait for the alignment to finish, i.e., the on-line computation goes on as one of its input operands is being delayed. It is important that the operand alignment does not increase the step time of the computation.

When an operand is being delayed, its incoming digits have to be buffered so that they can be used in the on-line recurrence later. To keep the digits in order, a shift register is used as the delay buffer. The incoming digits enter the buffer from the left end, and shift to the right one digit position during each cycle.

At the beginning of the computation, input digits enter the buffer while zeros are supplied to the on-line recurrence. When the desired amount of delay is reached, x_1 appears at the output, and after that subsequent digits are supplied at the output in their original order, one per cycle. Figure 4.4 is a diagram of the variable delay component. A Johnson counter is used to keep track of the position of x_1 in the delay buffer and control the outputs of the buffer. When the delay control signal B is 0, the output of the unit is 0. When B is 1, the mantissa digits are output.

If at the beginning of the computation the amount delay is known, then an alternative approach is to use a decoder to generate the control signals for the shift register.

4.2.6 Parallel adder

The exponent calculation of an on-line computation may include the addition operation. An 8-bit carry-lookahead adder is used for this function.

4.3 Design of radix-2 floating-point on-line division unit

4.3.1 The radix-2 on-line division algorithm

Denote the dividend, divisor and quotient as N, D and Q, respectively. For radix-2 on-line division, we have $\rho = 1$ and $K = 1$. From Section 2.6.3 we have the on-line recurrence expression

$$A[j]=2(A[j-1]-q_{j-1}D[j-1])+n_{j+\delta-1}2^{-\delta+1}-d_{j+\delta-1}Q[j-1]2^{-\delta+1}, \quad j \geq 1$$

In order to achieve a more efficient design, it is advantageous to reduce the size of the range of the divisor $D[j]$. This can be accomplished by preprocessing the divisor[27, 34, 35, 36]. In our design the divisor is conditionally shifted before computing the quotient digits. Let D' denote the original input divisor, and D be the shifted divisor which is used in the on-line recurrence. The shifting takes place during step $\delta - 1$. Assuming that D' is quasi-normalized[5], we have

$$D' \in [2^{-2}, 1)$$

Let H be a constant used to decide whether or not to shift the divisor, then we have

$$D'[0] = \sum_{i=1}^{\delta-1} d_i 2^{-i}$$

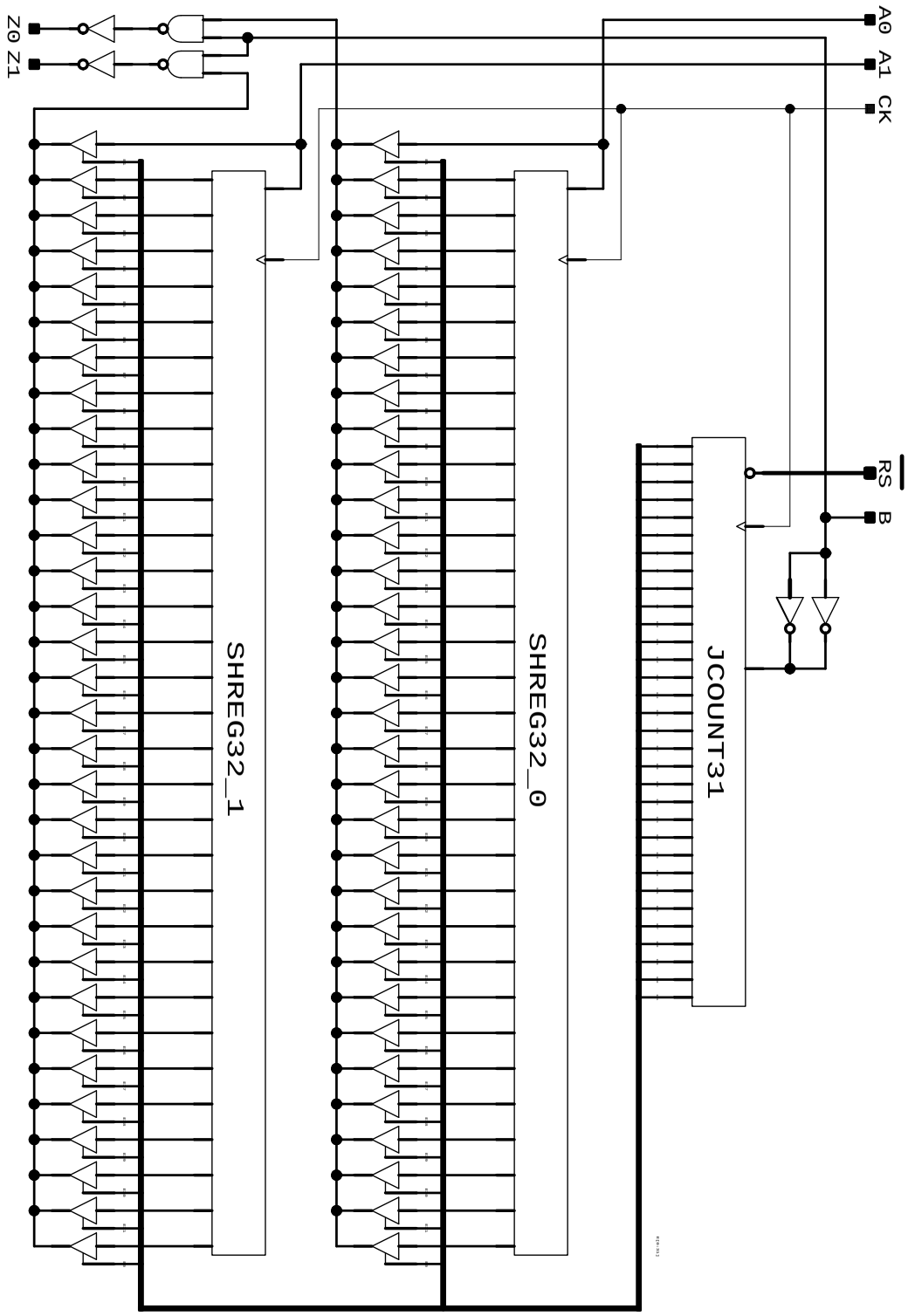


Figure 4.4: Variable delay unit

$$D[0] \;=\; \begin{cases} D'[0] & \textbf{if } D'[0] \geq H \\ 2 \cdot D'[0] + d_\delta 2^{-\delta+1} & \textbf{if } D'[0] < H \end{cases}$$

If the divisor is shifted, an extra input digit is loaded and the first quotient digit is computed in step $\delta + 1$. After shifting, the range of the divisor is

$$D[j] \;\in\; (H - 2^{-\delta+1}, 2H), \qquad j \geq 0$$

To simplify the comparison, let $H = 2^{-1}$, then

$$D[j] \;\in\; (2^{-1} - 2^{-\delta+1}, 1), \qquad j \geq 0$$

The scaled error bound is

$$\varepsilon_j = D[j] - \frac{2}{3}2^{-j-\delta+1} - 2^{-\delta+2}$$

and from (2.46) we have

$$(2 + \frac{2^{-j+1}}{3}) \cdot 2^{-\delta+1} + 2^{-\psi_A-1} \leq \frac{D[j]}{2}$$

Substituting the lower bound of $D[j]$,

$$(5 + \frac{2^{-j+2}}{3}) \cdot 2^{-\delta} + 2^{-\psi_A-1} \;\leq\; 2^{-2}$$

Since δ and ψ_A are non-negative integers, we choose $\delta = 5$ and $\psi_A \geq 3$. From (2.48) the conditions for the comparison points of the selection function are

$$kD[j] + 2^{-3} + \frac{2^{-j-3}}{3} \leq I_k 2^{-\psi_A} \leq (k+1)D[j] - 2^{-3} - 2^{-\psi_A} - \frac{2^{-j-3}}{3}$$

In the worst case, we have $j = 1$ and $D[j] > 2^{-1} - 2^{-4}$, hence

$$-2^{-2} - 2^{-5} \leq I_{-1}2^{-\psi_A} \leq 2^{-5}$$
$$2^{-3} + 2^{-4} - 2^{-5} \leq I_0 \leq 2^{-3} + 2^{-4} + 2^{-5}$$

and we get $k = 4$, $I_{-1}2^{-4} = 2^{-2}$ and $I_0 2^{-4} = 2^{-3} + 2^{-4}$.

The maximum value of $A[j]$ is

$$|A[j]| \;<\; 2D[j-1] - 2^{-3} - \frac{2}{3} \cdot 2^{-j-3}$$

Since $A[j] < 2$, the input to the digit selection function will have 1 sign bit, 1 integer bit, and 4 fractional bits, a total of 6 bits.

In Algorithm 4.2 the subscript *next* denotes the subscript of the next incoming digit. In step j, $next = j + 4$ if the divisor was not shifted during initialization. Otherwise, $next = j + 5$. The dividend mantissa is shifted right one bit position to avoid possible overflow of the result.

Algorithm 4.2 (Radix-2 on-line division)

step 1. [Initialization and shifting]

$e_q \leftarrow e_n - e_d + 1$

$A[0] \leftarrow \sum_{i=1}^{3} n_i 2^{-i-1}$

$D[0] \leftarrow \sum_{i=1}^{4} d_i 2^{-i}$

if $D[0] < 2^{-1}$ **then**

$D[0] \leftarrow 2 \cdot D[0] + d_{next} 2^{-4}$; $\quad e_q \leftarrow e_q + 1$

$Q[0] \leftarrow 0$; $\quad q_0 \leftarrow 0$

step 2. [quotient generation]

for $j = 1, \cdots, m$ **do**

$D[j] \leftarrow D[j-1] + d_{next} 2^{-j-4}$

$A[j] \leftarrow 2(A[j-1] - q_{j-1} D[j]) + n_{next} 2^{-4} - d_{next} Q[j-2] 2^{-4}$

$$q_j = \begin{cases} -1 & \textbf{if } \widehat{A[j]} < -\frac{1}{4} \\ 0 & \textbf{if } -\frac{1}{4} \le \widehat{A[j]} < \frac{3}{16} \\ 1 & \textbf{if } \widehat{A[j]} \ge \frac{3}{16} \end{cases}$$

$Q[j] \leftarrow Q[j-1] + q_j 2^{-j}$

end.

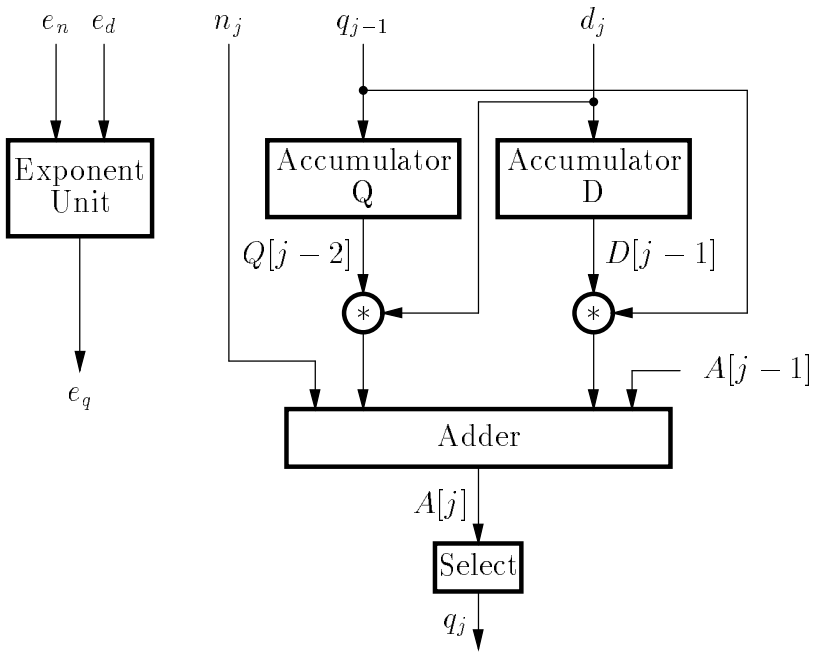


Figure 4.5: Functional components for on-line division

4.3.2 Functional components of on-line division

The scheme implements exponent and mantissa calculation. Exponent calculation for division includes the following functions, which are performed during step 1 of the Algorithm 4.2,

$$\begin{aligned} &1. \quad e_q \leftarrow e_n - e_d + 1 \\ &2. \quad \textbf{if } D'_0 < \frac{1}{2} \textbf{ then } \quad e_q \leftarrow e_q + 1 \end{aligned}$$

The exponent calculation is implemented by a conventional parallel adder.

The mantissa calculation is performed on-line. During each time step of the mantissa calculation, the following recurrence expression is evaluated to obtain intermediate result $A[j]$,

$$A[j] \leftarrow 2(A[j-1] - q_{j-1}D[j] + n_{next}2^{-5} - d_{next}Q_{j-2}2^{-5}) \tag{4.2}$$

The next quotient digit is calculated as a function of $A[j]$. Functional components required for the mantissa computation include 2 on-the-fly conversion units for $D[j]$ and $Q[j]$, 2 fraction by digit multipliers, a 4-to-2 carry-save adder, a 6-bit carry-assimilation adder, and the quotient digit generation logic. Figure 4.5 illustrates the functional components of on-line division.

4.3.3 Binary level algorithms

We now specify binary level algorithms for each functional component of on-line division. To simplify the control mechanism we modify the mantissa computation specified in Algorithm 4.2 such that the initialization step is incorporated as part of the quotient calculation step with simple control requirements. Then operations of the functional components are discussed in detail.

The $n_{next}2^{-5}$ term in (4.2) represents a shifted single sbit, and it is desirable not to let this single sbit cause extra delay to the overall computation. Since

$$\begin{aligned} |Q[j-1]| &< 1 \\ |d_j| &\leq 1 \end{aligned}$$

for all values of j, we have

$$|-d_{next}Q[j-1]| < 1$$

In 2's complement representation, the terms n_j2^{-5}, $P = -d_{next}Q[j-1]2^{-5}$, and their sum are

$$\begin{array}{rcccccccccccc} n_j2^{-5} & \equiv & n_0 & . & n_0 & n_0 & n_0 & n_0 & n_1 & 0 & 0 & \cdots & 0 \\ P = -d_{next}Q[j-1]2^{-5} & \equiv & p_s & . & p_s & p_s & p_s & p_s & p_s & p_1 & p_2 & \cdots & p_{j-1} \\ \hline n_j2^{-5} - d_{next}Q[j-1]2^{-5} & \equiv & x & . & x & x & x & x & y & p_1 & p_2 & \cdots & p_{j-1} \end{array}$$

where x and y are simple switching functions of n_0, n_1 and p_s,

$$x = n_0 + \overline{n}_1 p_s \tag{4.3}$$

$$y = n_1 \oplus p_s \tag{4.4}$$

The terms $n_{next}2^{-5}$ and $-d_{next}Q[j-1]2^{-5}$ can be combined as one term for the multiple input adder.

The initialization step of Algorithm 4.2 requires accumulating incoming sbits of N and D. In order to achieve an efficient design, we want to minimize the difference between computations performed during initialization and quotient generation. The expression for $D[0]$ is the same as that for $D[j]$ in the quotient generation step. By keeping $q_j = 0$ in the recurrence expression (4.2), the expression for computing $A[j]$ during quotient digit generation becomes the same as that for computing $A[0]$ during initialization. This simplifies the control mechanism for the recurrence evaluation. The modified mantissa algorithm (higher level) is given in Algorithm 4.3.

The on-the-fly conversion unit design discussed in Section 4.2.1 is used for the accumulation and conversion of $Q[j]$. Simple modification is made for the

Algorithm 4.3 (Modified radix-2 on-line division algorithm)

Initialization

$A[-4] = 0, \quad D[-4] = 0, \quad Q[-5] = 0$

$q_{-4} = 0, \quad j = -3$

begin [recurrence]

$D[j] \leftarrow D[j-1] + d_{next} 2^{-j-4}$

$A[j] \leftarrow 2(A[j-1] - q_{j-1} D[j] + n_{next} 2^{-5} - d_{next} Q[j-2] 2^{-5})$

$$q_j \leftarrow \begin{cases} 0 & \textbf{if } j \leq 0 \\ select(\widehat{A[j]}) & \textbf{if } j > 0 \end{cases}$$

$Q[j-1] \leftarrow Q[j-2] + q_{j-1} 2^{-j+1}$

if $j = 0$ **and** $D[0] < 2^{-1}$ **then**

$D[0] \leftarrow D[0] \cdot 2$

else if $j \geq m$ **then** **stop**

else $j \leftarrow j + 1$

end [recurrence]

end.

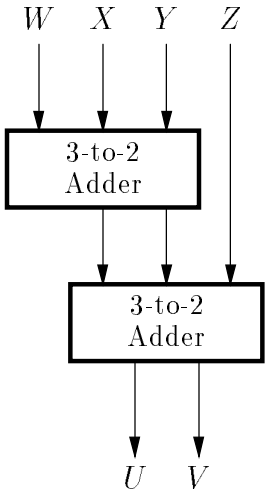


Figure 4.6: Structure of 4-to-2 adder

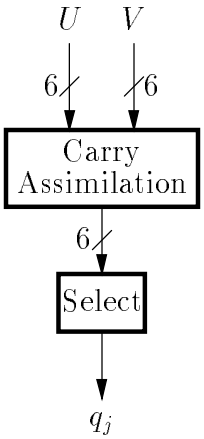


Figure 4.7: Quotient digit selection

accumulation for $D[j]$ to perform the shifting required in the initialization step. It will be shown later that the timing of the conversion is not critical to the performance due to pipelining.

There are 4 input terms to the recurrence evaluation, hence two levels of 3-to-2 reductions are used to realize the required 4-to-2 carry-save adder as shown in Figure 4.6.

The result $A[j]$ of the recurrence expression evaluation is in carry-save form. The quotient digit selection function is composed of two parts, as shown in Figure 4.7. A 6-bit carry-assimilation adder converts the most significant portion of $A[j]$ into non-redundant form, and then the digit selection logic generates the next quotient digit. The following are the logic expressions for the carry-assimilation

adder.

$$
\begin{aligned}
P_i &\leftarrow \begin{cases} U_i \oplus V_i & i = 5 \\ U_i \oplus V_i \oplus C_i & i = 0,1,2,3,4 \end{cases} \\
C_4 &= U_5 V_5 \\
C_3 &= U_4 V_4 + (U_4 + V_4) U_5 V_5 \\
C_2 &= U_3 V_3 + (U_3 + V_3) U_4 V_4 + (U_3 + V_3)(U_4 + V_4) U_5 V_5 \\
C_1 &= U_2 V_2 + (U_2 + V_2) C_2 \\
C_0 &= U_1 V_1 + (U_1 + V_1) U_2 V_2 + (U_1 + V_1)(U_2 + V_2) C_2
\end{aligned}
$$

The quotient digit q_j is generated by the following function,

$$
\begin{aligned}
q_j^{(0)} &\leftarrow P_0(\overline{P_1} + \overline{P_2} + \overline{P_3} + \overline{P_4} \cdot \overline{P_5}) \\
q_j^{(1)} &\leftarrow P_0(\overline{P_1} + \overline{P_2} + \overline{P_3} + \overline{P_4} \cdot \overline{P_5}) + \overline{P_0}(P_1 + P_2 + P_3)
\end{aligned}
$$

4.3.4 Pipelining

To minimize the step time of the computation, we use a two-stage pipeline scheme, as shown in Figure 4.8, where the shaded boxes indicate the pipeline buffers.

In stage 1, the following calculations are performed.

$$
\begin{aligned}
D[j] &\leftarrow D[j-1] + d_{next} 2^{-j-4} \\
A'[j] &\leftarrow 2(A'[j-1] - q_{j-2} D[j-1]) + n_{next} 2^{-5} - d_{next} Q[j-2] 2^{-5} \\
Q[j-1] &\leftarrow Q[j-2] + q_{j-1} 2^{-j+1}
\end{aligned}
$$

And in stage 2, q_j is generated.

$$
q_j \leftarrow select(2(A'[j] - q_{j-1} D[j]))
$$

Figure 4.9 illustrates the timing of the mantissa computation.

The most time consuming part in the computation is the quotient digit selection function, which includes the 6-bit carry assimilation adder and quotient digit generation logic. The critical path, which is shown in thick lines in Figure 4.8, is that from the q_j buffer output to the signed digit multiplier to *CSA2* to *Select* to the q_j buffer input. The data accumulation and conversion components are not in the critical path. The pipeline scheme given here allows extra time in stage 1 for the distribution of q_j and d_j.

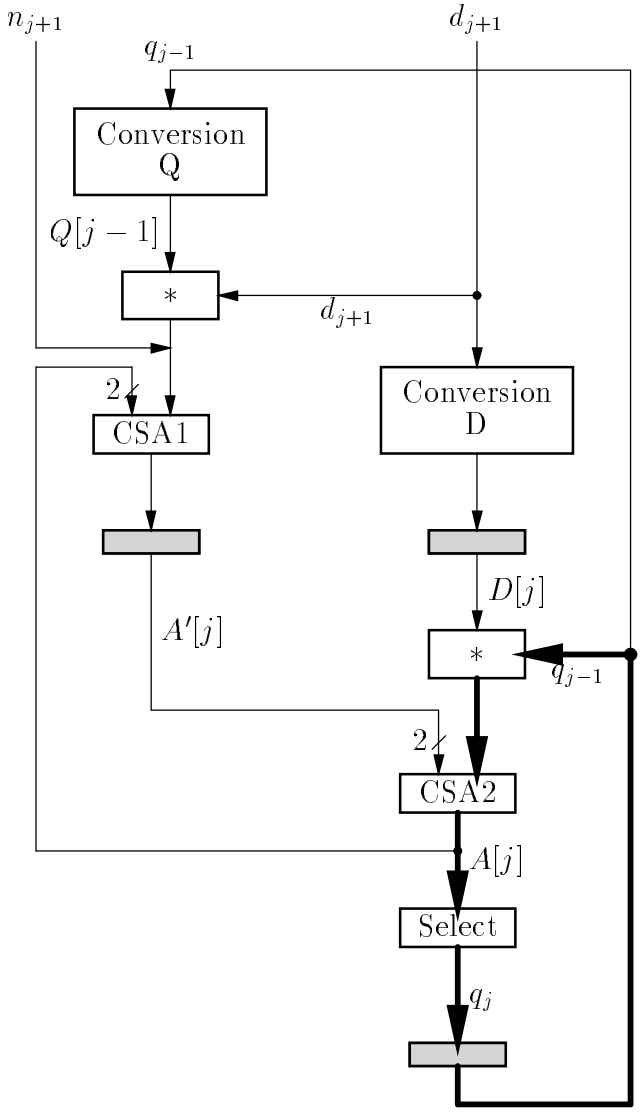


Figure 4.8: Pipeline scheme for on-line division

Step	$\cdots$	$j-2$	$j-1$	j	$j+1$	$\cdots$
Input	$\cdots$	d_{j-1}	d_j	d_{j+1}	d_{j+2}	$\cdots$
	$\cdots$	n_{j-1}	n_j	n_{j+1}	n_{j+2}	$\cdots$
Values	$\cdots$	$D[j-1]$	$D[j]$	$D[j+1]$	$D[j+2]$	$\cdots$
Calculated	$\cdots$	$Q[j-3]$	$Q[j-2]$	$Q[j-1]$	$Q[j]$	$\cdots$
	$\cdots$	$A'[j-1]$	$A'[j]$	$A'[j+1]$	$A'[j+2]$	$\cdots$
	$\cdots$	q_{j-2}	q_{j-1}	q_j	q_{j+1}	$\cdots$

Figure 4.9: Timing for on-line division mantissa computation

4.3.5 Organization and design of on-line division unit

The on-line division unit is organized as a linear array of modules as is shown in Figure 4.1. The first module M_1, shown in Figure 4.10, contains components for generating control signals ($CTRL$), exponent calculation (EU), quotient digit selection($DIVSEL$), and bit-slices for the most significant portion of the recurrence calculation, which includes the sign, integer bit positions, and 5 fractional bit positions. The modules $M_2, \cdots, M_q$ are identical, each containing a number of bit-slices of the recurrence expression evaluation.

From Algorithm 4.2, the number of fractional bit positions involved in the on-line recurrence computation is $j+4$ at step j. To generate the jth quotient digit, only the 4 most significant fractional bit positions plus the sign and integer parts are needed, so it is not necessary to carry out the recurrence expression evaluation in full precision. The quotient digit selection uses 4 fractional bit positions and 2 integer bit positions of $A[j]$, and the on-line delay of the divisor $D[j]$ is 5, hence from expression (4.1) we have the number of bit slices needed to compute an m sbit quotient

$$p = \left\lceil \frac{m+4+5}{2} \right\rceil + 1 \tag{4.5}$$

For on-line division of 24-sbit precision, 18 bit-slices are needed.

The main component of the exponent unit EU is an 8-bit parallel adder with carry lookahead. Figure 4.11 shows the design of the exponent unit.

Each bit-slice includes functional components for data conversion, signed digit multiplication, and carry-save addition. Figure 4.12 shows the bit-slice design. Two slightly different designs of the bit-slices are used in module M_1 to incorporate the $n_{next}2^{-5}$ term in the recurrence expression. As was discussed in Section 4.3.3, the most significant bit positions in the recurrence expression evaluation need to generate the values x and y, which are defined in expressions (4.3) and (4.4). Figures 4.13 and 4.14 shows the logic diagrams for generating x and y. Figure 4.15 shows the design of the quotient digit selection component.

4.3.6 Design characteristics

The total gate count for the on-line division design is

$$G_{div} = G_{M1} + G_{bit\text{-}slice} \times n'$$

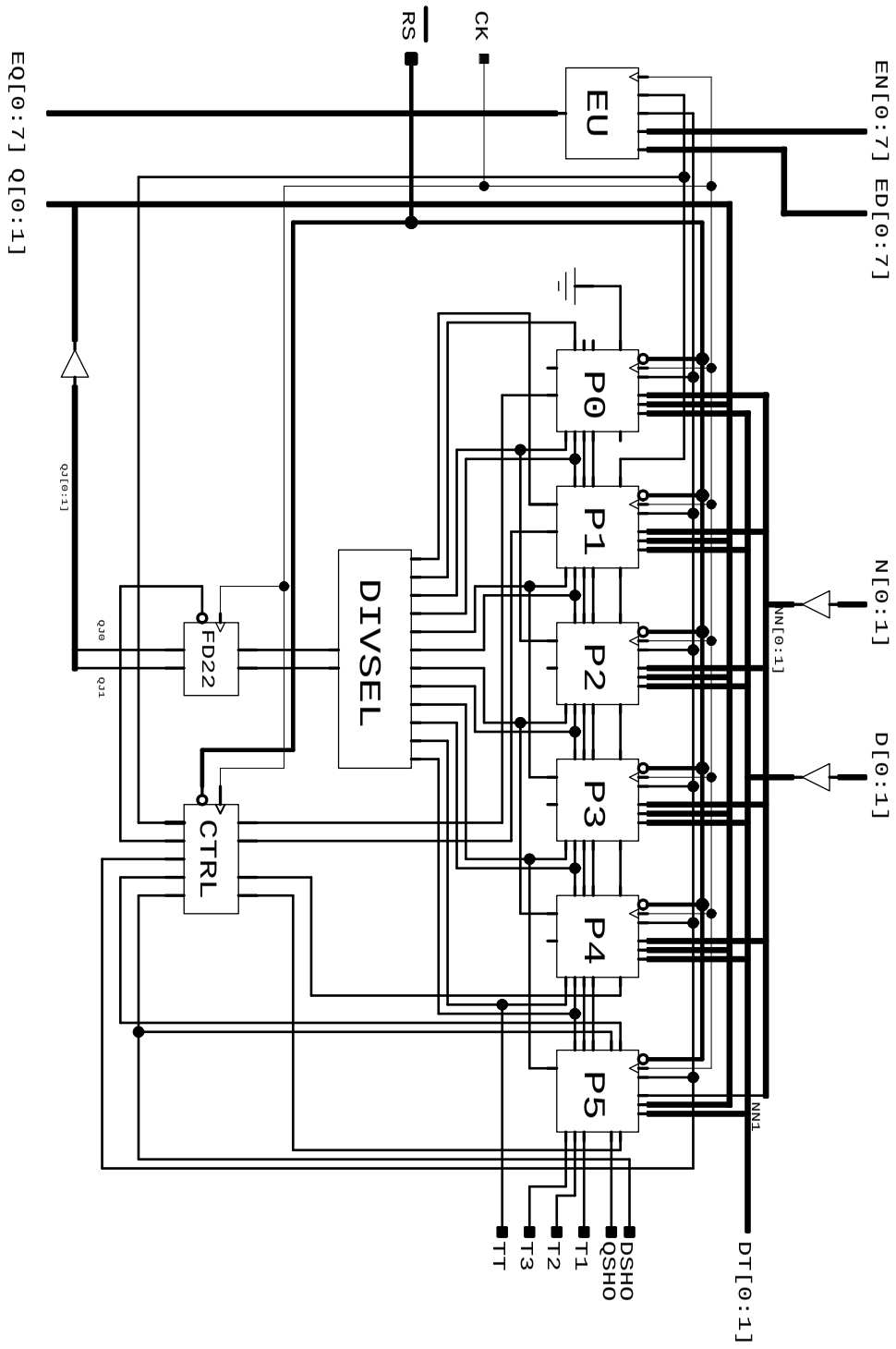


Figure 4.10: Module M1 of on-line division

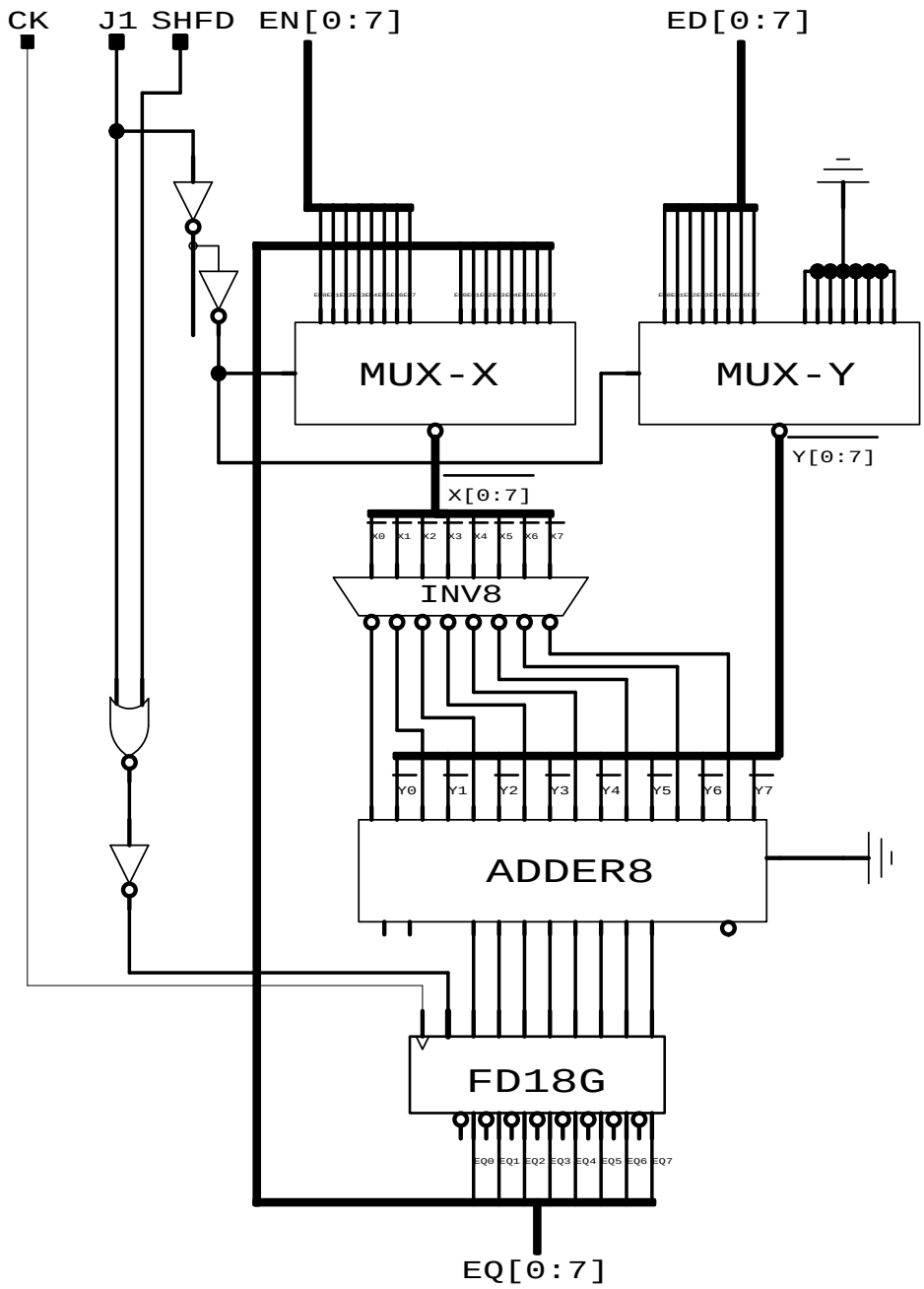


Figure 4.11: Division exponent unit

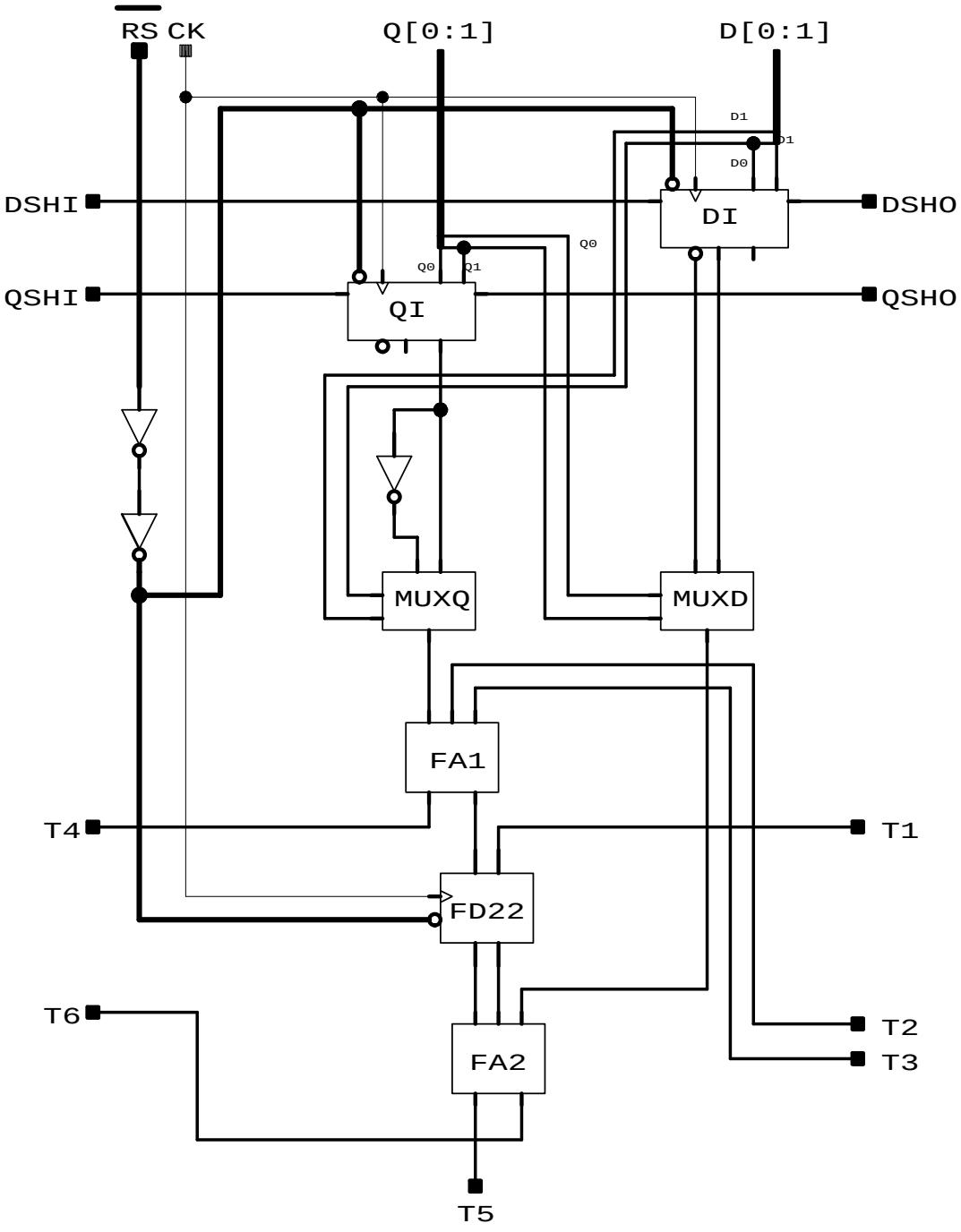


Figure 4.12: Bit-slice for on-line division

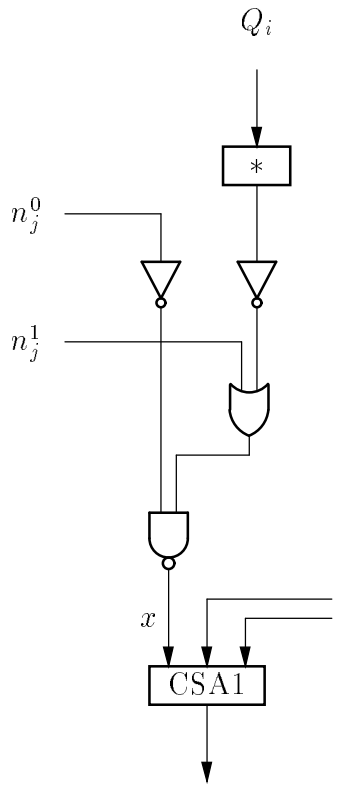


Figure 4.13: Implementation of x logic

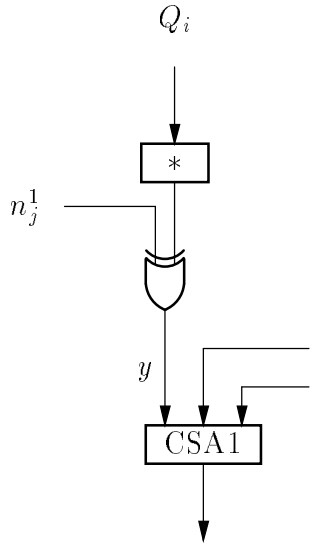


Figure 4.14: Implementation of y logic

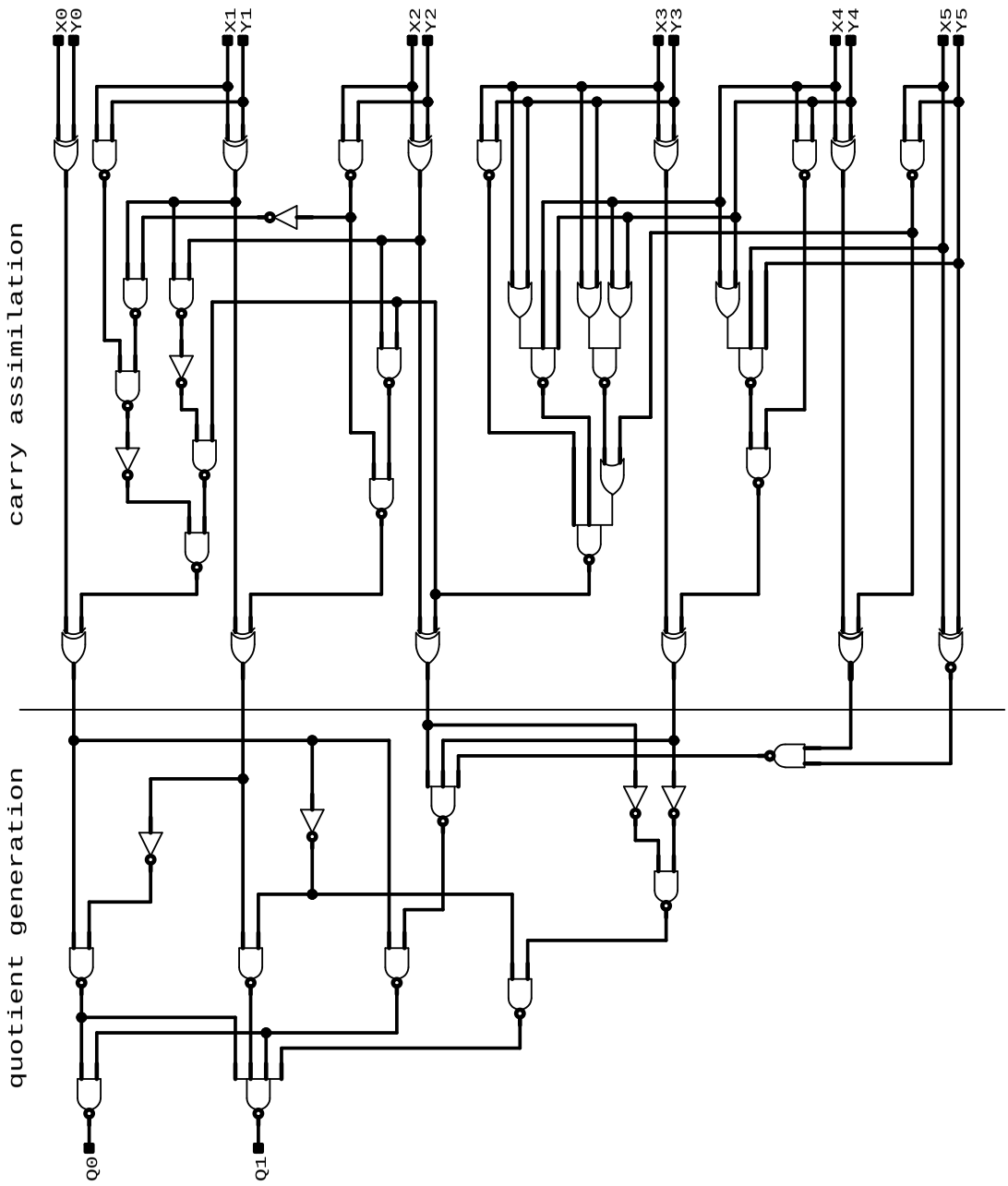


Figure 4.15: Quotient digit selection

component	equiv. gate
EU	248
P0, P1, ..., P5	$6 \times 119 = 714$
DIVSEL	72
Control, Buffers	69
M_1 Total	1103

Table 4.2: Gate count of module M_1 of on-line division

component	equiv. gate
conversion	$2 \times 32 = 64$
sbit multiply	$2 \times 6 = 12$
FA	$2 \times 10 = 20$
Buffers and drivers	15
bit-slice total	111

Table 4.3: Gate count of bit-slice of on-line division

where n' is the number of bit-slices in modules M_2, $\cdots$, M_s. Since module M_1 contains the most significant 6 bit-slices, from (4.5) we have

$$n' = \left\lceil \frac{m+9}{2} \right\rceil + 1 - 6$$

Table 4.2 shows the equivalent gate counts of the components of module M_1, which consists of components necessary for the generation of quotient mantissa digits and exponent, and has a fixed gate count regardless of the precision of the computation. Table 4.3 shows the component gate count of a bit-slice. An on-line division unit for single precision floating-point operands with an 8-bit exponent and 24-sbit mantissa has a total gate count of 2497, including the input/output drivers for the chip.

Figure 4.16 shows the data dependencies between components in the cycle period of the exponent and mantissa calculation. The time delays of the major components are shown in the timing diagram in Figure 4.17. The figures are estimates based on the load driven by each component, design parameters of the LSI LL9000 series gate array components, and average wire lengths without performing the actual routing and layout design. In Figure 4.17, *FD* denotes the transition of D flipflops, and *setup* is their setup time. *CONVERT* is the time for converting from signed digit form to 2's complement form. n_j *I/O* and d_j *I/O* are the input

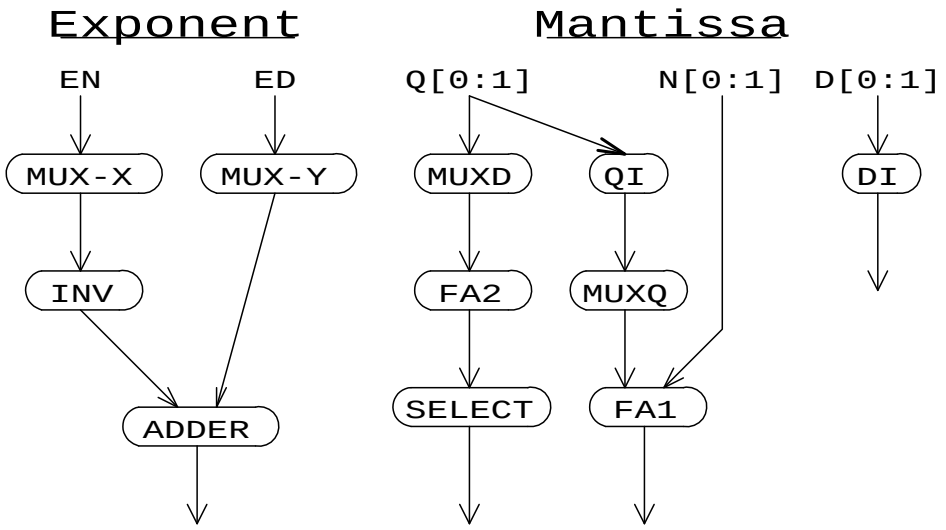


Figure 4.16: Dependency graph for on-line division step

pad buffer delays for the dividend and divisor. Dashed lines in the diagram denote components that are connected in sequence but not in the critical path. The step time for the design is determined by the delays of the multiplier, the full adder, and the quotient digit selection unit, as is indicated by the critical path in Figure 4.8. The minimum step time, with the LSI LL9000 series components, is 24ns. This figure includes the input pad buffers for the dividend and divisor, but does not include the output pin drivers required for the quotient. The delay of the output drivers depends on the load of the output pins for the quotient.

4.4 Design of radix-2 floating-point on-line addition unit

4.4.1 The radix-2 on-line addition algorithm

Denote the operands and result as X, Y, and Z, respectively, where

$$\begin{aligned} X &\in [2^{-2}, 1) \\ Y &\in [2^{-2}, 1) \end{aligned}$$

For radix-2 on-line addition, we have $\rho = 1$ and $K = 1$. From Section 2.6.1 we have the on-line recurrence expression

$$A[j] = 2(A[j-1] - z_{j-1}) + x_{j+\delta-1}2^{-\delta+1} + y_{j+\delta-1}2^{-\delta+1}, \qquad j \geq 1$$

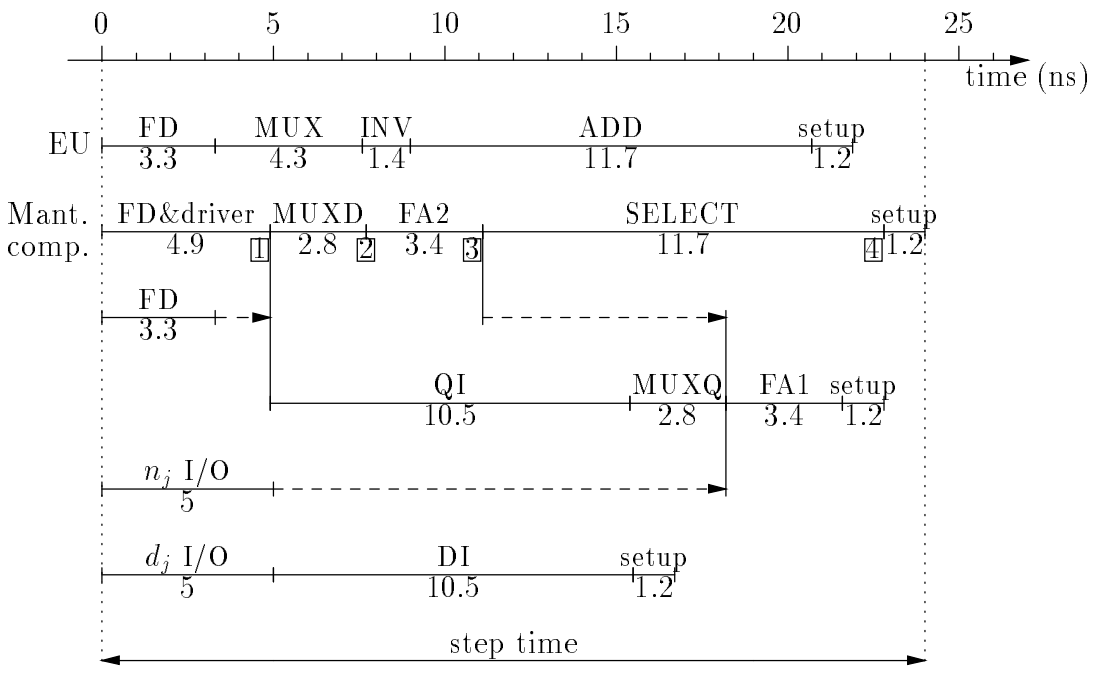


Figure 4.17: Timing of on-line division

For the radix-2 computation, $A[j]$ is of very limited precision, the on-line recurrence evaluation is performed in full precision. The scaled error bound is

$$2^{-1} \leq \varepsilon \leq 1 - 2^{-\delta+2}$$

So we can choose

$$\begin{aligned} \varepsilon &= 2^{-1} \\ \delta &= 3 \end{aligned}$$

and the comparison points are

$$\begin{cases} I_{-1} &= -2^{-1} \\ I_0 &= 2^{-1} \end{cases}$$

The range of $A[j]$ satisfies

$$|A[j]| < 1 + 2^{-1}$$

hence the input to the digit selection function will have 1 sign bit, 1 integer bit, and 1 fractional bit, a total of 3 bits. Algorithm 4.4 is the radix-2 algorithm for on-line addition.

Algorithm 4.4 (Radix-2 on-line addition)

step 1. [Initialization and alignment]

$e_d \leftarrow e_x - e_y$

$e_z \leftarrow \max(e_x, e_y) + 1$

$A[-2] = 0, \quad z_0 = 0$

for $j = 1, \cdots, |e_d|$ **do**

  if $e_d \geq 0$ **then**

    $x'_j = x_j, \quad y'_j = 0$

  else

    $x'_j = 0, \quad y'_j = y_j$

for $j = |e_d| + 1, \cdots, m$ **do**

  if $e_d \geq 0$ **then**

    $x'_j = x_j, \quad y'_j = y_{j-|e_d|}$

  else

    $x'_j = x_{j-|e_d|}, \quad y'_j = y_j$

for $j = -1, \cdots, 0$ **do**

  $A[j] \leftarrow 2A[j-1] + (x'_{j+2} + y'_{j+2})2^{-2}$

step 2. [result generation]

for $j = 1, \cdots, m$ **do**

  $A[j] \leftarrow 2(A[j-1] - z_{j-1}) + (x'_{j+2} + y'_{j+2})2^{-2}$

$$z_j \leftarrow \begin{cases} -1 & \textbf{if } A[j] < -2^{-1} \\ 0 & \textbf{if } -2^{-1} \leq A[j] < 2^{-1} \\ 1 & \textbf{if } A[j] \geq 2^{-1} \end{cases}$$

end.

4.4.2 Functional components of on-line addition

The on-line addition scheme consists of exponent and mantissa calculation. Exponent calculation includes the following functions during the first step of the computation,

- Calculate the exponent of the result

$$e_z = \max(e_x, e_y) + 1$$

- Calculate the difference of the exponents, denoted by e_d, for mantissa alignment,

$$e_d = e_x - e_y$$

For mantissa calculation, the following functions are performed during each recurrence,

- Mantissa alignment
- On-line recurrence evaluation,

$$A[j] \leftarrow 2(A[j-1] - z_j) + (x'_{j+2} + y'_{j+2})2^{-2} \tag{4.6}$$

- Output digit selection.

Figure 4.18 is a functional diagram for on-line addition.

4.4.3 Binary level algorithms

The on-line recurrence evaluation (4.6) requires adding 3 terms. This is realized in non-redundant form, in two stages as is shown in Figure 4.19. The *XIYI* component is a single sbit adder that calculates the sum of the inputs x_j and y_j by executing the following algorithm.

$$\begin{aligned} W_0 &= x_j^0\overline{y_j^1} + \overline{x_j^1}y_j^0 + x_j^0 y_j^0 \\ W_1 &= x_j^0\overline{y_j^1} + \overline{x_j^1}y_j^0 + x_j^0 y_j^0 + \overline{x_j^0}x_j^1\overline{y_j^0}y_j^1 \\ W_2 &= x_j^1 \oplus y_j^1 \end{aligned}$$

The component *ADD3* is a 3 bit adder, which implements the following algorithm.

$$\begin{aligned} V_0 &= A_0[j-1] \oplus W_0 \oplus (A_1[j-1]W_0 + (A_1[j-1] + W_0)A_2[j-1]W_1) \\ V_1 &= A_1[j-1] \oplus W_0 \oplus (A_2[j-1]W_1) \\ V_2 &= A_2[j-1] \oplus W_1 \end{aligned}$$

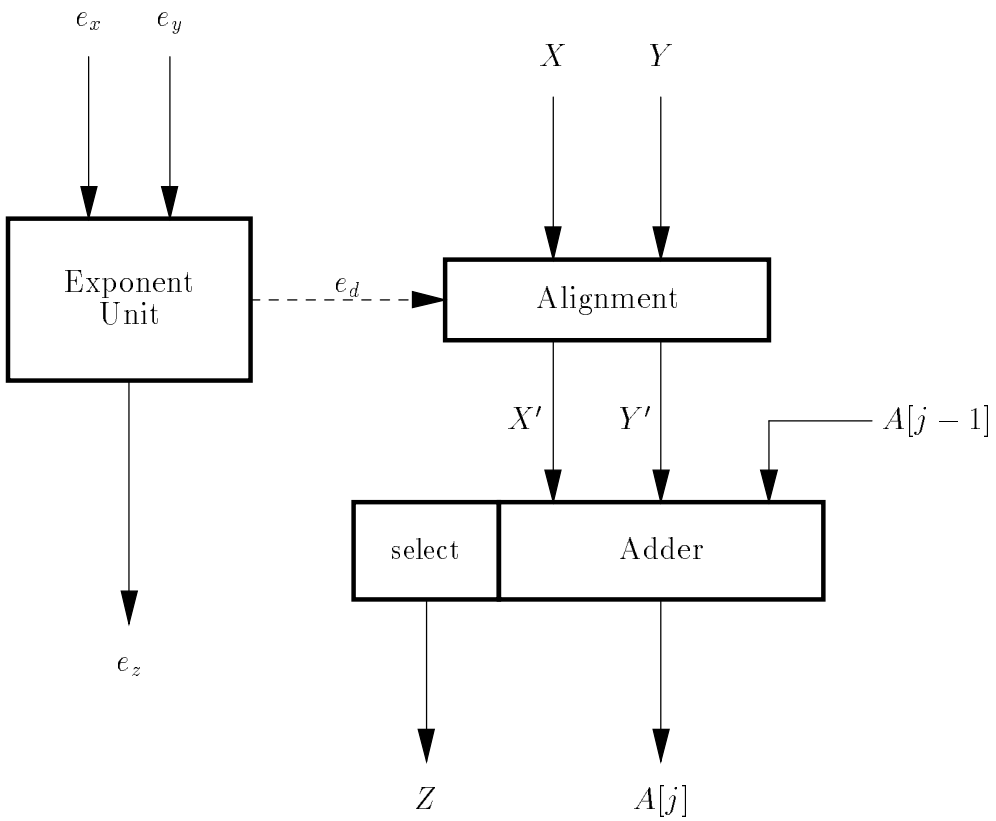


Figure 4.18: Functional components of on-line addition

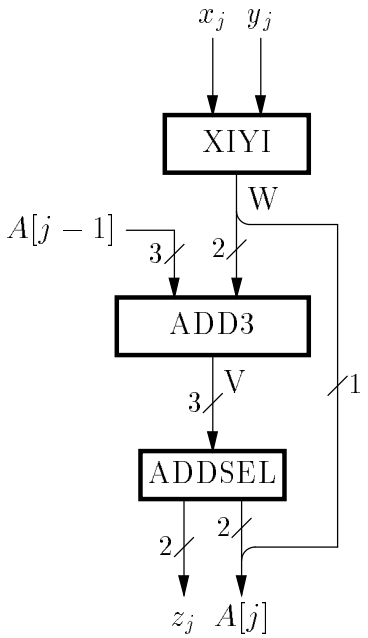


Figure 4.19: Implementation of on-line recurrence

The binary level selection algorithm is the following.

$$\begin{aligned} z_j^0 &= V_0\overline{V_1} + V_0\overline{V_2} \\ z_j^1 &= \overline{V_0}V_1 + \overline{V_1}V_2 + V_0\overline{V_2} \\ A_0[j] &= V_0V_2 + V_0\overline{V_1} + \overline{V_1}V_2 \\ A_1[j] &= V_2 \end{aligned}$$

4.4.4 Design of the on-line addition unit

Figure 4.20 shows the design of the on-line unit for radix-2 floating-point addition. In this scheme, the calculation of e_d and e_z are performed in conventional non-redundant form, using parallel adders. The component *ADD1* increments the input value by 1.

During the first step of the computation, the difference of the exponents of the operands is calculated,

$$e_d = e_x - e_y$$

The sign of e_d is used to control the component $XYSW$, which conditionally switches the inputs such that the mantissa digits of the operand with the smaller exponent is sent to the variable delay component while digits of the operand with

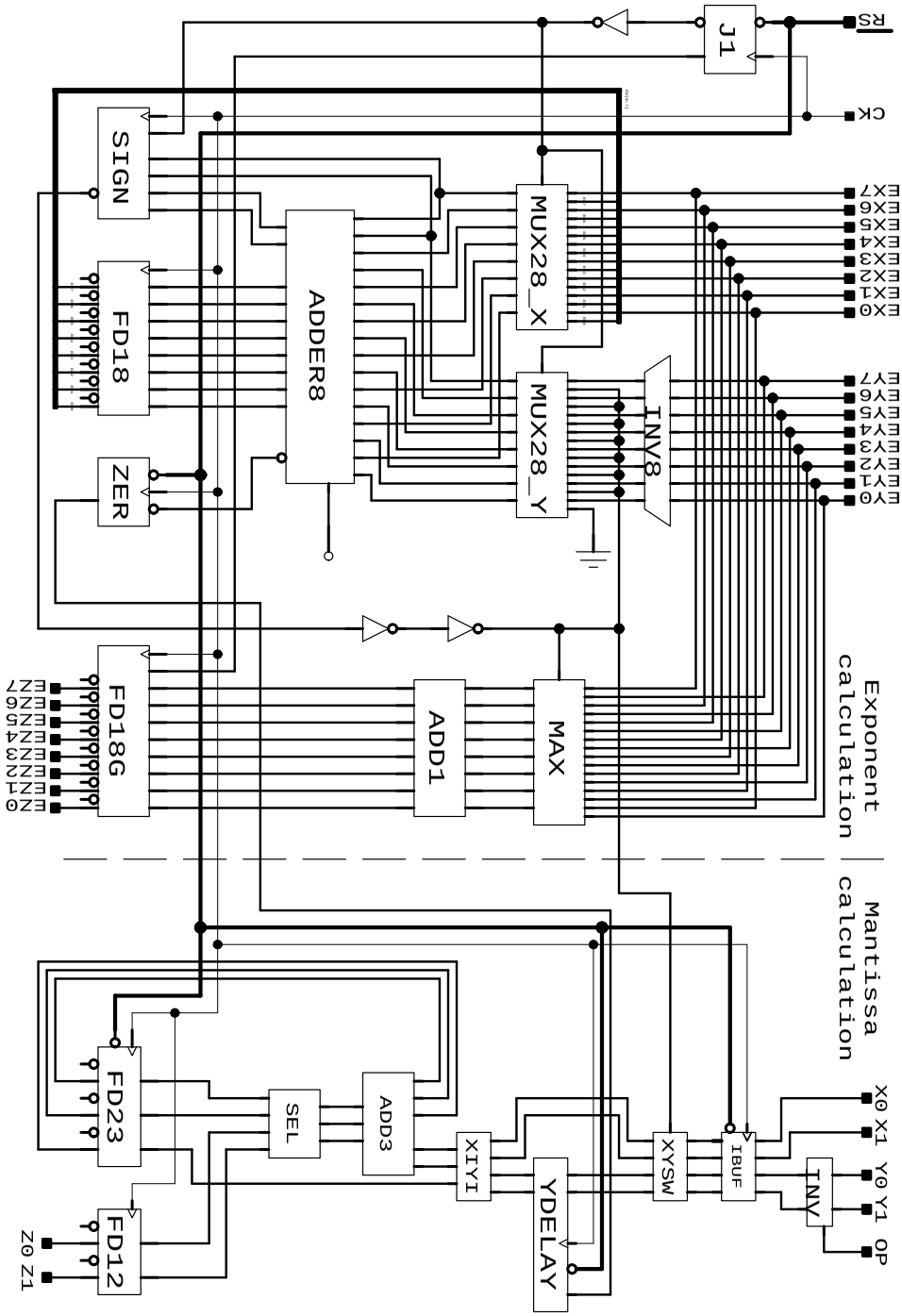


Figure 4.20: Design of on-line addition unit

the larger exponent go directly to the adder. After the first step, the 8-bit adder *ADDER8* functions as a down counter, and $|e_d|$ is decremented by 1 during each cycle. When the value of $|e_d|$ reaches zero, a signal is sent to the variable delay component and digits of the delayed mantissa are supplied to the adder in order. The *SIGN* component detects and remembers the sign of e_d during the first cycle of the computation, while the *ZER* component check if the output of *ADDER8* is equal to zero. Figure 4.21 shows the design of the output digit selection function.

Assuming the operands and result each has an 8-bit exponent and 24-sbit mantissa, the total gate count for the on-line addition unit is 1337.

The dependencies between components of the on-line addition unit are shown in Figure 4.22. Figure 4.23 shows the time delays of the major components. Similar to the case of on-line division, the cycle time of the on-line addition unit design can be reduced by using a 2-stage pipeline scheme in the mantissa computation.

4.5 Design of radix-2 floating-point on-line multiplication unit

4.5.1 The radix-2 multiplication algorithm

Denote the operands and result as X, Y, and Z, respectively, where

$$\begin{aligned} X &\in [2^{-2}, 1) \\ Y &\in [2^{-2}, 1) \end{aligned}$$

For radix-2 on-line multiplication, we have $\rho = 1$ and $K = 1$. From Section 2.6.2 we have the on-line recurrence expression

$$\begin{aligned} A[j] &= 2(A[j-1] - z_{j-1}) \\ &\quad + (x_{j+\delta-1}Y[j-1] + y_{j+\delta-1}X[j])2^{-\delta+1} \qquad j \geq 1 \end{aligned}$$

and expressions for the scaled error bound

$$\begin{aligned} 2^{-1} + 2^{-\psi_A - 1} &\leq \varepsilon \leq 1 - 2^{-\delta+2} \\ 2^{-\psi_A - 1} + 2^{-\delta+2} &\leq \frac{1}{2} \end{aligned}$$

We choose $\delta = 4$, $\psi_A = 1$, and

$$\begin{aligned} \varepsilon &= 2^{-1} + 2^{-2} \\ I_0 &= -2^{-1} \\ I_1 &= 2^{-1} \end{aligned}$$

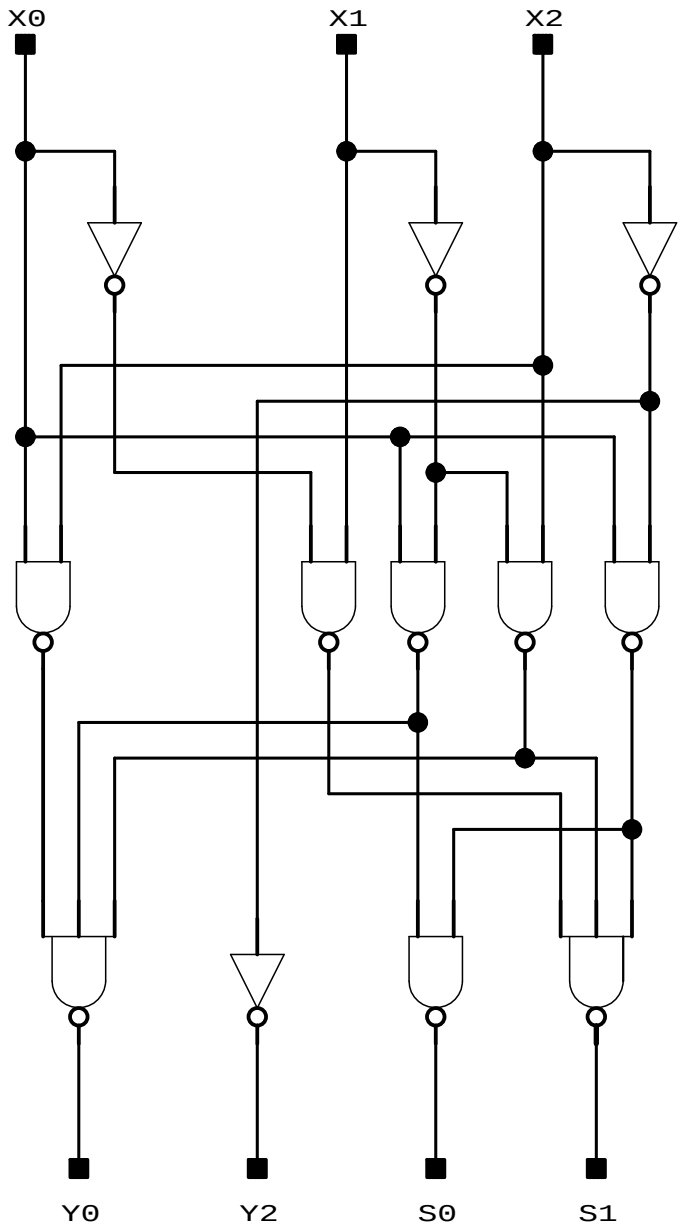


Figure 4.21: Design of digit selection function of on-line addition

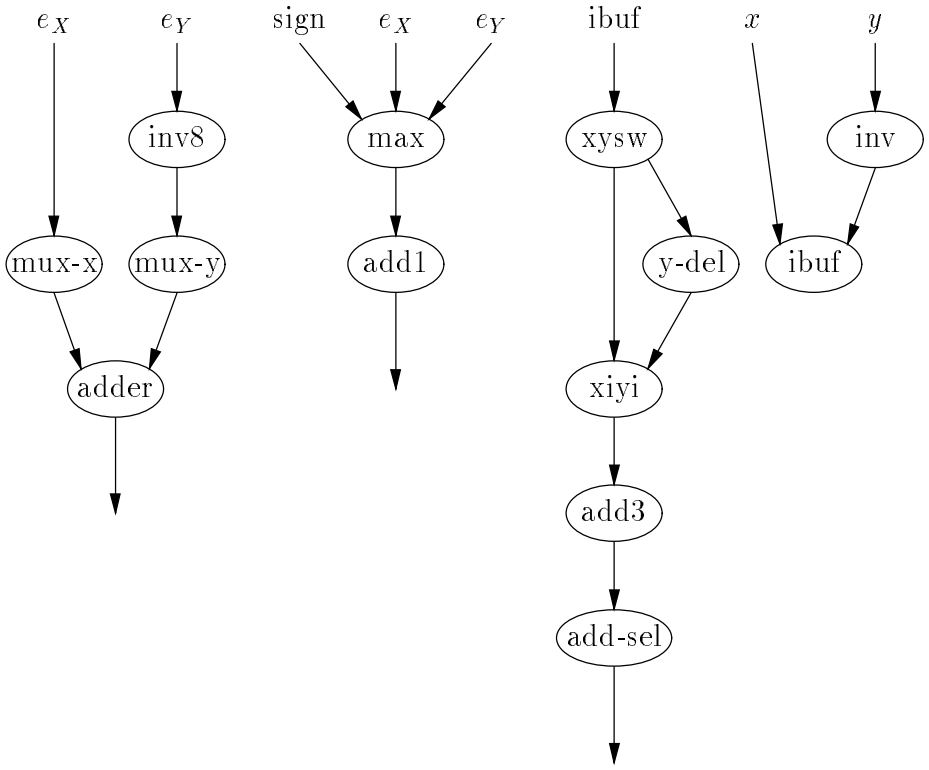


Figure 4.22: Data dependencies of on-line addition

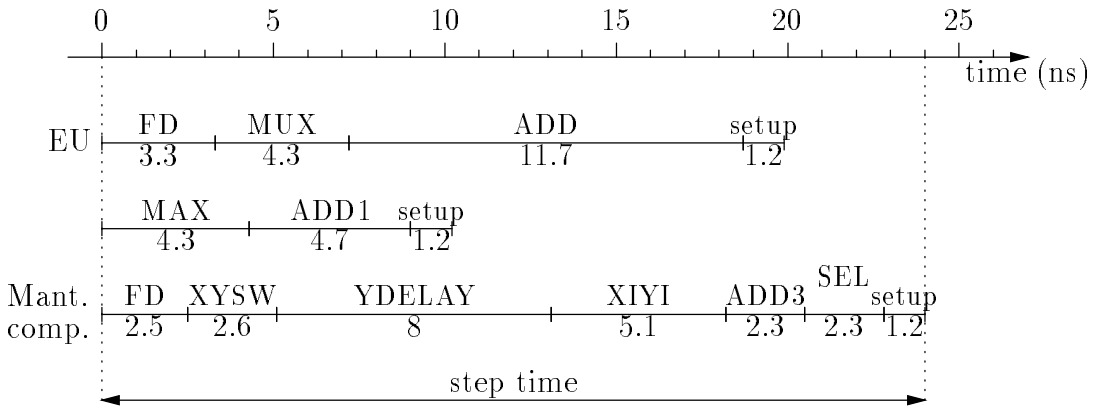


Figure 4.23: Component delays of on-line addition

Algorithm 4.5 (Radix-2 on-line multiplication)

step 1. [Initialization]

$e_z \leftarrow e_x + e_y$

$X[-3] = 0,\ Y[-3] = 0,\ A[-3] = 0,\ z_0 = 0$

for $j = -2, \cdots, 0$ **do**

$X[j] \leftarrow X[j-1] + x_{j+3}2^{-j-3}$

$A[j] \leftarrow 2A[j-1] + (x_{j+3}Y[j-1] + y_{j+3}X[j])2^{-3}$

$Y[j] \leftarrow Y[j-1] + y_{j+3}2^{-j-3}$

step 2. [product generation]

for $j = 1, \cdots, m$ **do**

$X[j] \leftarrow X[j-1] + x_{j+3}2^{-j-3}$

$A[j] \leftarrow 2(A[j-1] - z_{j-1}) + (x_{j+3}Y[j-1] + y_{j+3}X[j])2^{-3}$

$Y[j] \leftarrow Y[j-1] + y_{j+3}2^{-j-3}$

$$z_j \leftarrow \begin{cases} -1 & \textbf{if } \widehat{A[j]} < -2^{-1} \\ 0 & \textbf{if } -2^{-1} \le \widehat{A[j]} < 2^{-1} \\ 1 & \textbf{if } \widehat{A[j]} \ge 2^{-1} \end{cases}$$

end.

The range of $A[j]$ satisfies

$$|A[j]| \ < \ 1 + 2^{-1} + 2^{-2}$$

Since $A[j] < 2$, the input to the digit selection function will have 1 sign bit, 1 integer bit, and 1 fractional bit, a total of 3 bits. Algorithm 4.5 is the on-line algorithm for radix-2 multiplication.

The scaled error bound of the algorithm after step m is

$$R[m] \ \le \ 2^{-1} + 2^{-2}$$

Figure 4.24 is a functional component diagram of the on-line multiplication unit.

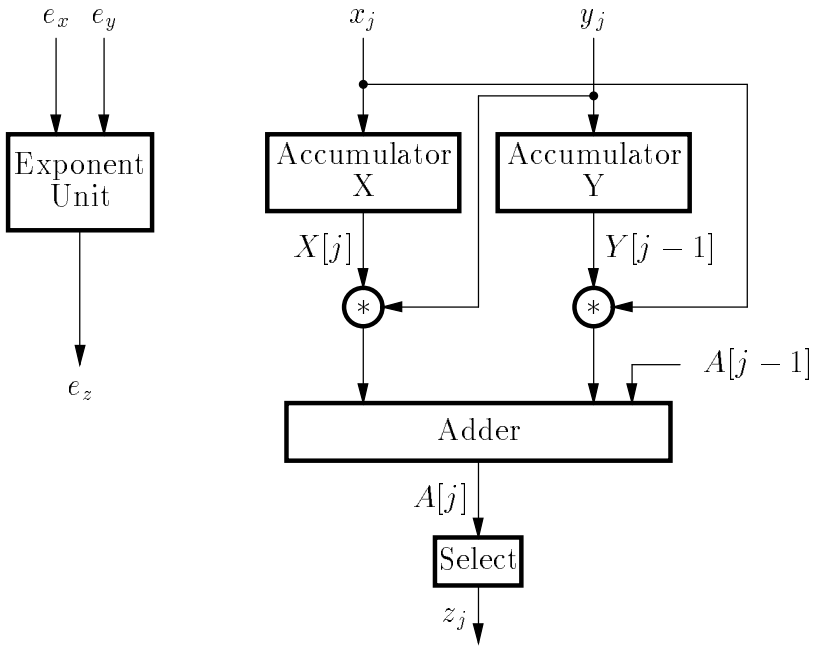


Figure 4.24: Functional components of on-line multiplication

4.5.2 Design of on-line multiplication unit

The on-line multiplication unit is organized as a linear array of modules as is shown in Figure 4.1. The total number of bit-slices is

$$n = \left\lceil \frac{m+2+4}{2} \right\rceil + 1$$

For on-line multiplication of 24-sbit precision, 16 bit-slices are needed.

The First module, shown in Figure 4.25, contains components for generating control signals (*CNTL*), exponent calculation (*EXP*), output digit selection (*MULTSEL*), and bit-slices for the most significant portion of the recurrence calculation. Figure 4.26 shows the design of a bit-slice.

The LSI equivalent gate counts for the on-line multiplication unit and its components are listed in Table 4.4, assuming the operands and result each has an 8-bit exponent and 24-sbit mantissa. The total gate count for the unit is 2329.

The dependencies between components of the on-line addition unit is shown in Figure 4.27. Figure 4.28 shows the time delays of the major components. The design requires a cycle time of 20ns.

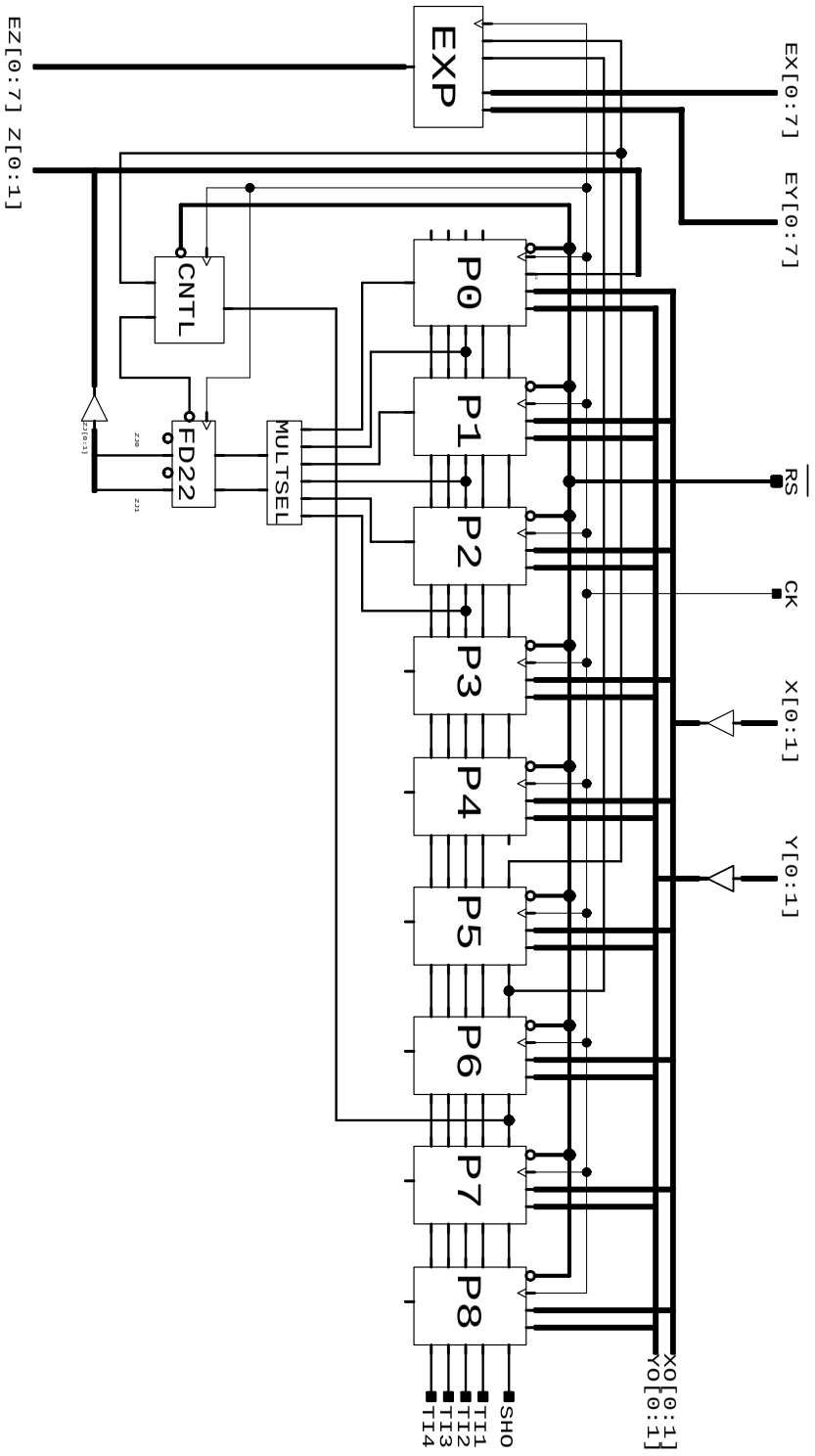


Figure 4.25: Module M1 of on-line multiplication

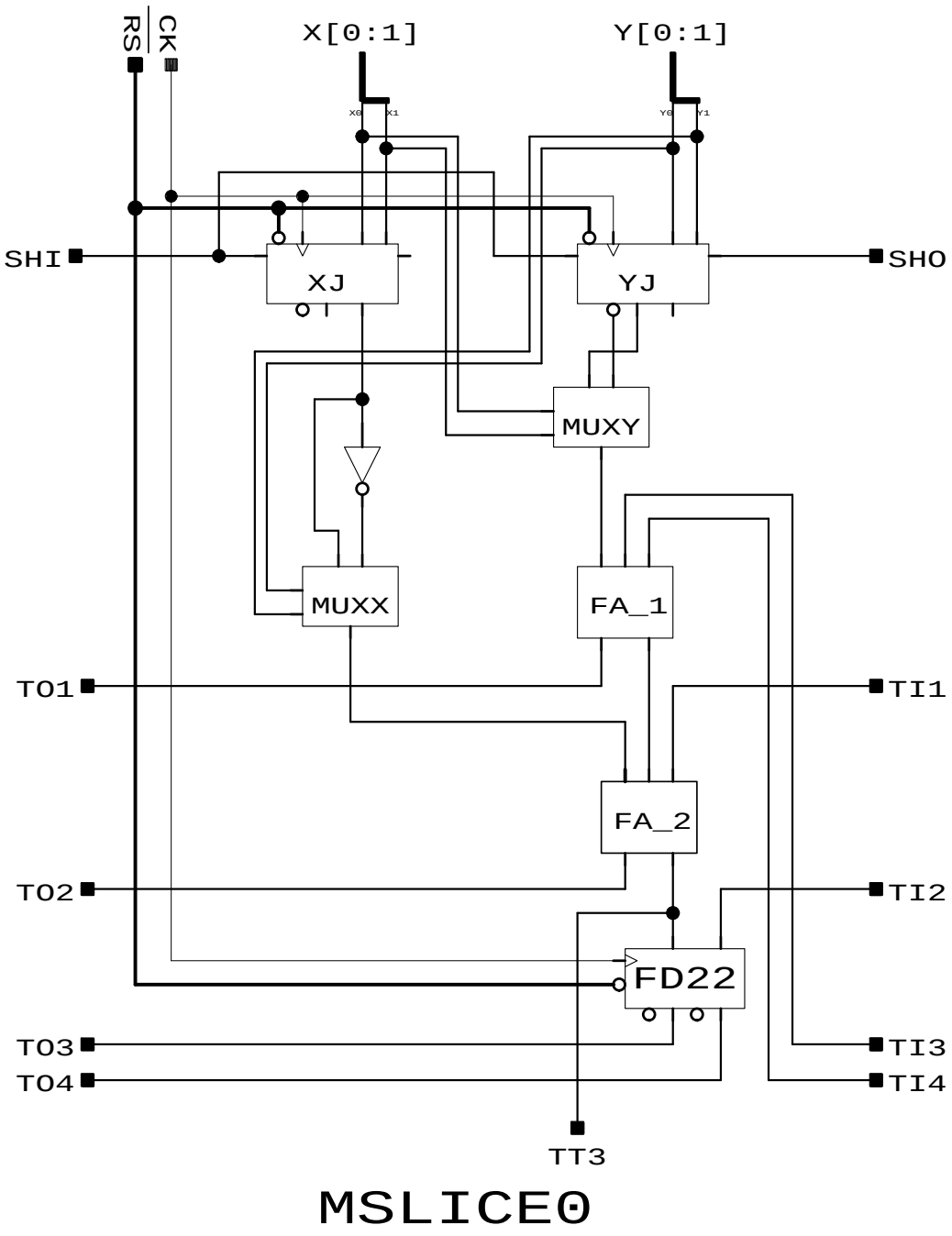


Figure 4.26: Bit-slice for on-line multiplication

component	equiv. gate
EXP	242
bit-slice	112
MULTSEL	30
M_1 Total	1294

Table 4.4: Gate count of on-line multiplication design

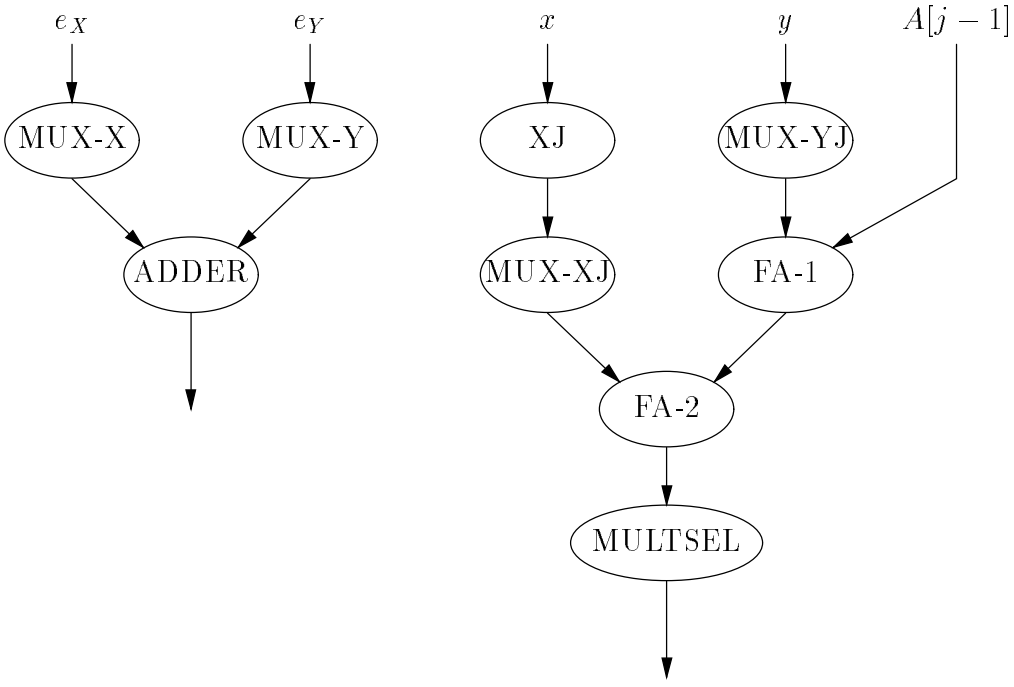


Figure 4.27: Data dependencies of on-line multiplication

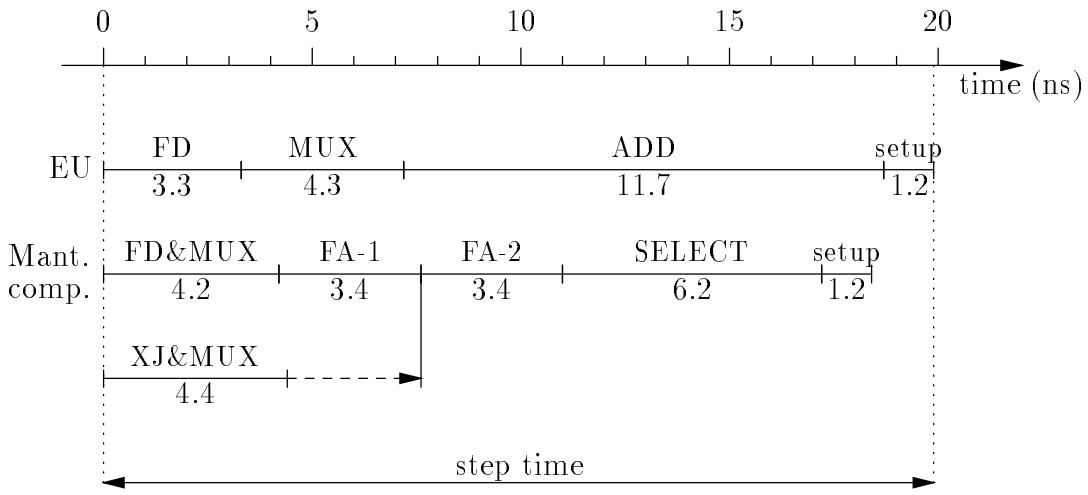


Figure 4.28: Component delays of on-line multiplication

4.6 Design of radix-2 floating-point on-line square root unit

4.6.1 The radix-2 square root algorithm

Denote the operand and result as X and Y, respectively. For the radix-2 square root computation, the output digit can be calculated without input bits from Y. We have

$$
\begin{aligned}
& 2^{-p-3} < X \cdot 2^{-p} < 2^{-p} \\
& 2^{-\frac{p+3}{2}} < Y < 2^{-\frac{p}{2}} \\
& \varepsilon_j = Y[j] + Y[j-1] - 2^{-j+1} - 2^{-\delta+1-p} \\
& 2^{-\psi_A-1} + 2^{-\delta+1-p} + 2^{-j+1} \leq \frac{Y[j] + Y[j-1]}{2}
\end{aligned}
$$

Let $p = 1$, we have

$$
\begin{aligned}
& 2^{-4} < X \cdot 2^{-1} < 2^{-1} \\
& 2^{-2} < Y < 2^{-\frac{1}{2}}
\end{aligned}
$$

and for $j \geq J$, where J is some small integer,

$$
2^{-\psi_A-1} + 2^{-\delta} + 2^{-J+1} \quad \leq \quad 2^{-2}
$$

We choose $J = 4$, $\delta = 4$, and $\psi_A = 4$. The scaled error bound is

$$
\varepsilon_j \quad = \quad Y[j] + Y[j-1] - 2^{-j+1} - 2^{-4}, \qquad j \geq 4
$$

The selection intervals for $j \geq 4$ are

$$\begin{aligned}
\widehat{L_k} &\geq \begin{cases} 2^{-3} + 2^{-4}, & k = 1 \\ (k-1)(Y[j] + Y[j-1]) + 2^{-3} + 2^{-4}, & k = \{-1, 0\} \end{cases} \\
\widehat{U_k} &\leq \begin{cases} (k+1)(Y[j] + Y[j-1]) - 2^{-2} - 2^{-4}, & k = \{0, 1\} \\ -2^{-2} - 2^{-4}, & k = -1 \end{cases}
\end{aligned}$$

and the comparison points for $j \geq 4$ are

$$\begin{aligned}
I_1 \cdot 2^{-4} &= 2^{-3} + 2^{-4} \\
I_0 \cdot 2^{-4} &= -2^{-2} - 2^{-4}
\end{aligned}$$

Since the range of $A[j]$ is

$$|A[j]| < 2^{\frac{3}{2}} - 2^{-4}$$

we have $\phi_A = 3$. The input to the digit selection function will have 1 sign bit, 2 integer bits, and 4 fractional bits of $A[j]$.

For $j = 1, 2, 3$, the output digit selection is a function of j, and each value of j is treated separately. The derivation of the selection function for $j = 1, 2, 3$ is given in Appendix C. Then the complete selection function is defined as

$$\begin{aligned}
I_1 2^{-4} &= \begin{cases} 2^{-2}, & j = 1 \\ 2^{-3} + 2^{-4}, & j = 2 \\ 2^{-2} + 2^{-4}, & j = 3 \\ 2^{-3} + 2^{-4}, & j \geq 4 \end{cases} \\
I_0 2^{-4} &= \begin{cases} -2^{-2} - 2^{-3} - 2^{-4} & j = 3 \\ -2^{-2} - 2^{-4} & j \geq 4 \end{cases}
\end{aligned}$$

Algorithm 4.6 is the on-line algorithm for radix-2 square root computation. Figure 4.29 is the block diagram of on-line square root unit.

4.6.2 Design of on-line square root unit

The on-line square root unit is organized as a linear array of modules as is shown in Figure 4.1. The total number of bit-slices is

$$p = \left\lceil \frac{m + 4 + 4}{2} \right\rceil + 2$$

For on-line square root operation of 24-sbit precision, 18 bit-slices are needed.

Algorithm 4.6 (Radix-2 on-line square root algorithm)

step 1. [Initialization and shifting]

if e_x *odd* **then**

$A[0] \leftarrow \sum_{i=1}^{2} x_i 2^{-i-1}$

else

$A[0] \leftarrow x_1 2^{-3}$

$e_y \leftarrow \lfloor e_x/2 \rfloor + 1$

step 2. [output generation]

for $j = 1, \cdots, m$ **do**

$A[j] \leftarrow 2 \cdot (A[j-1] - y_{j-1}(Y[j-2] + Y[j-1])) + x_{next} 2^{-4}$

$y_j \leftarrow \mathcal{S}_{sqrt}(\widehat{A[j]})$

$Y[j] \leftarrow Y[j-1] + y_j 2^{-j}$

end.

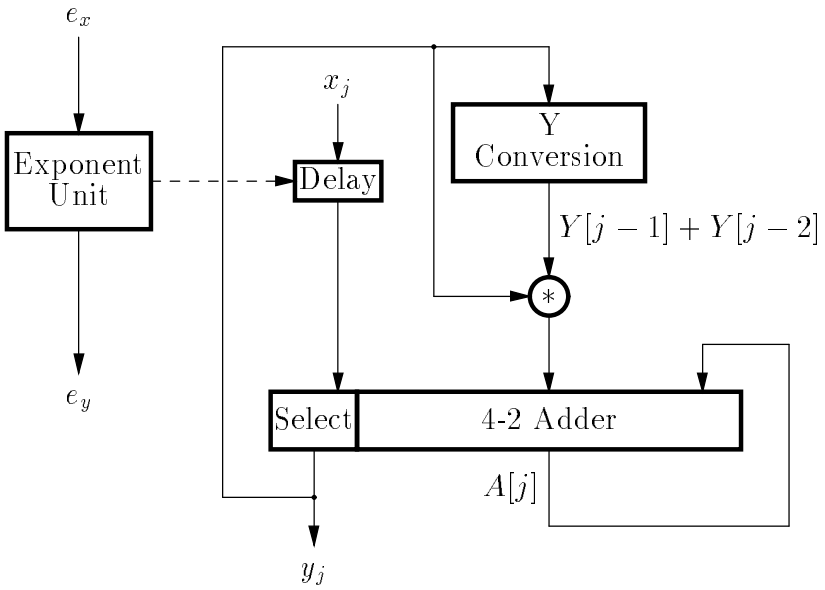


Figure 4.29: Block diagram of on-line square root unit

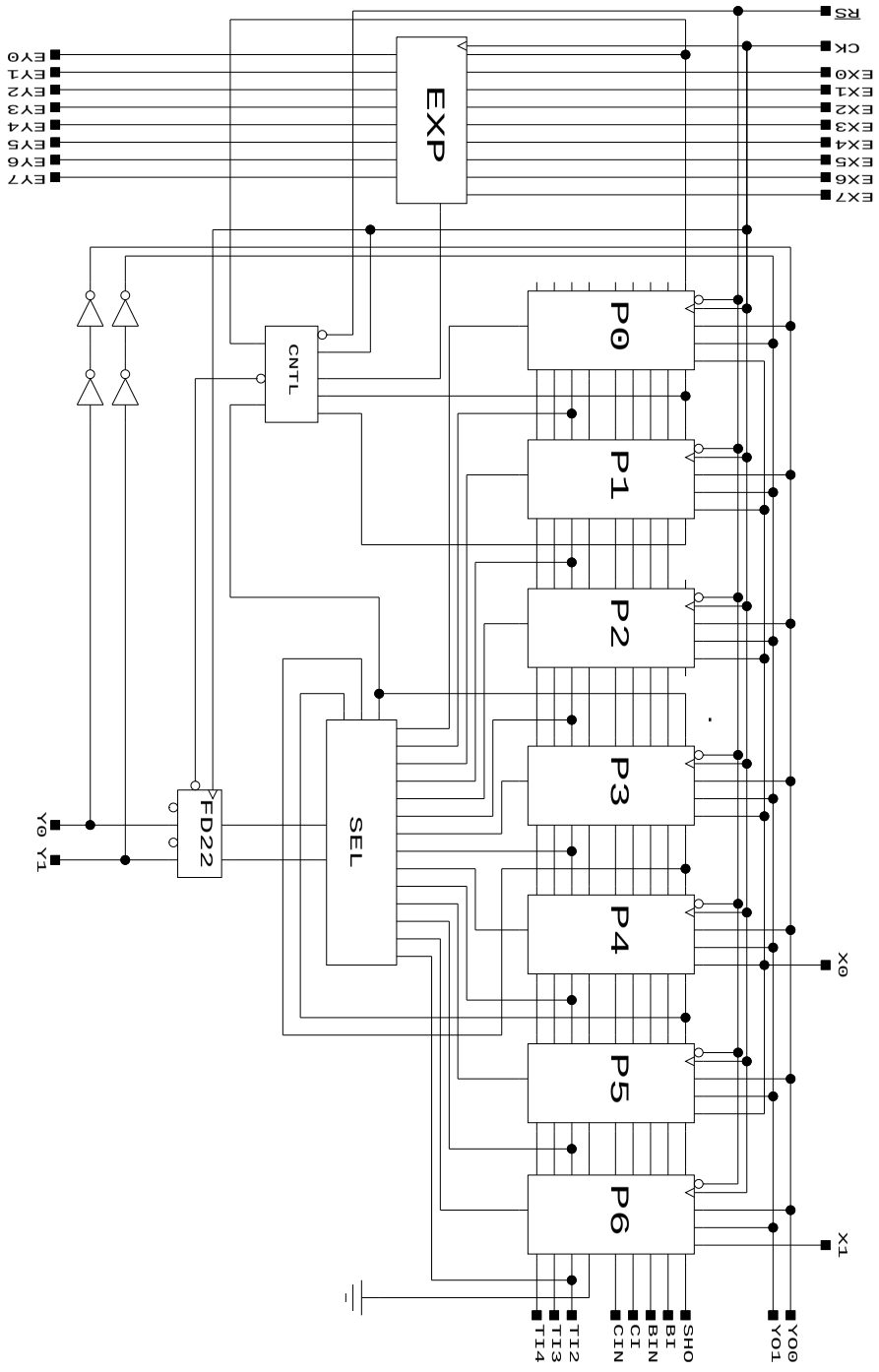


Figure 4.30: Module M1 of on-line square root unit

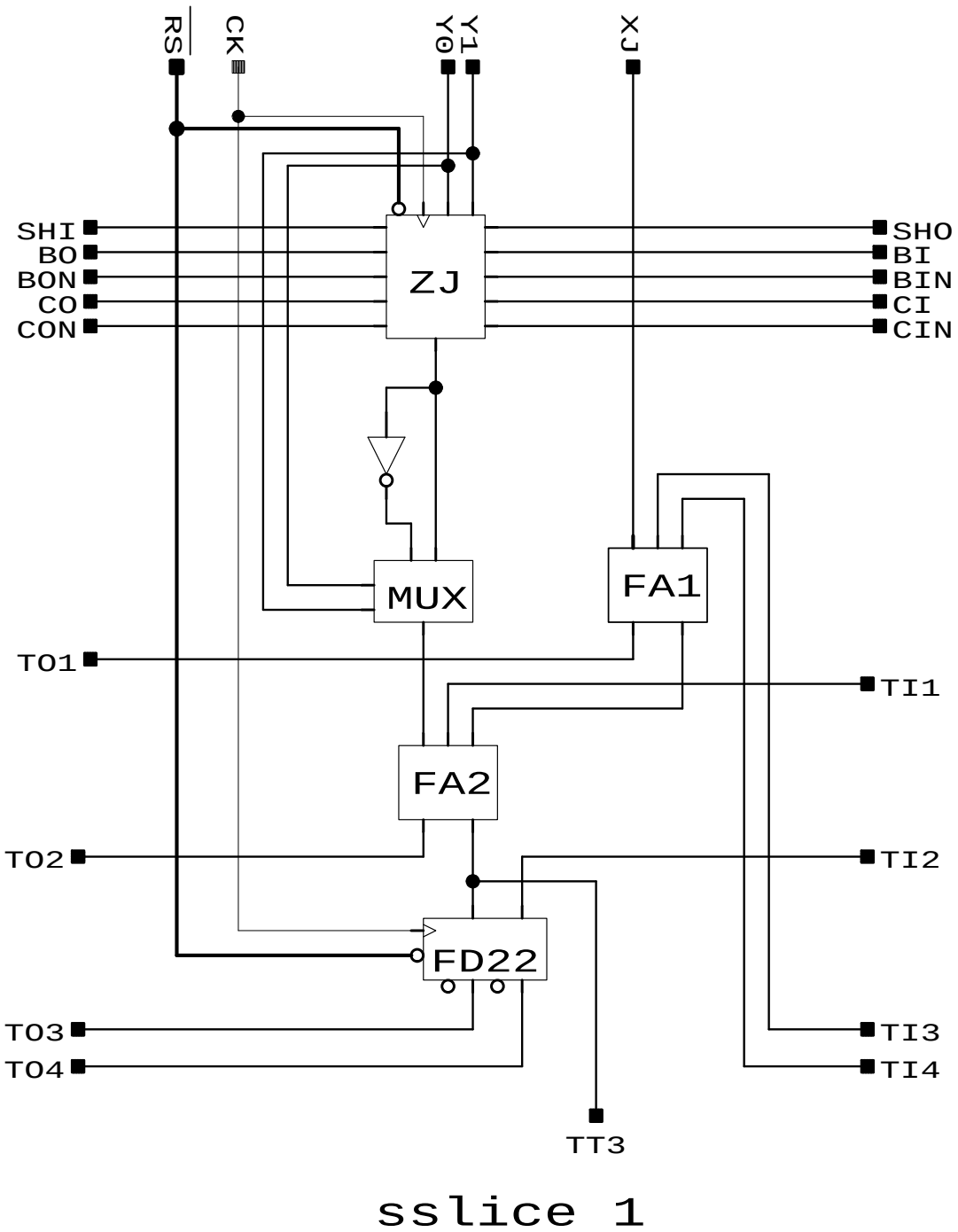


Figure 4.31: Bit-slice of square root unit

component	equiv. gate
EXP	111
conversion	45
bit-slice	84
SEL	67
M_1 Total	805

Table 4.5: Gate count of on-line square root unit design

The design of the first module is shown in Figure 4.30. It contains components for generating control signals (*CNTL*), exponent calculation (*EXP*), output digit selection (*SEL*), and bit-slices for the most significant portion of the recurrence calculation. Figure 4.31 is the bit-slice design. Design of the digit selection component is shown in Figure 4.32, and Figure 4.33 shows the on-the-fly conversion unit, which generates the sum of $Y[j-2] + Y[j-1]$.

Table 4.5 lists the equivalent gate counts for the square root unit assuming the operand and result each has an 8-bit exponent and 24-sbit mantissa. The total gate count for the design is 1587.

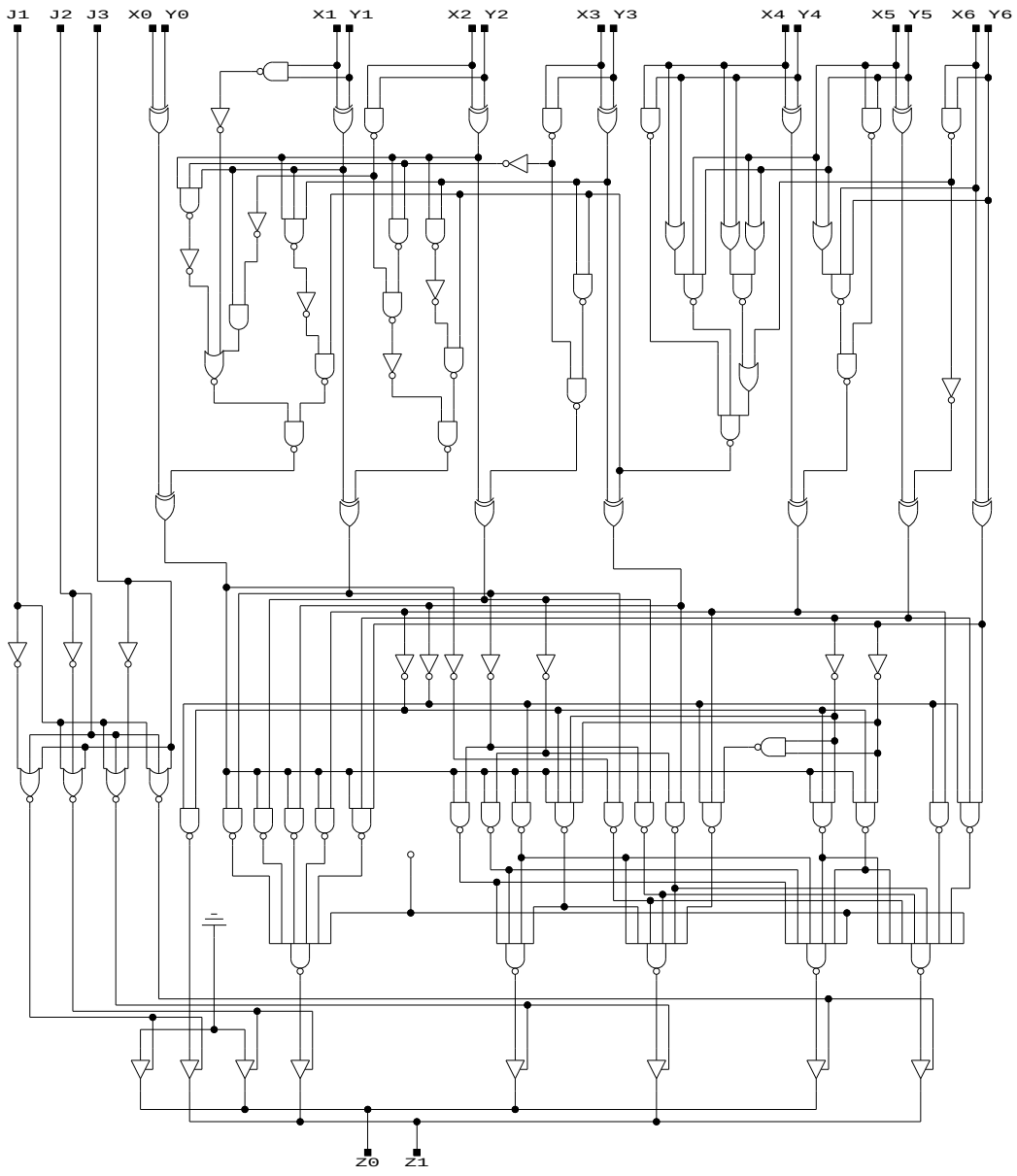


Figure 4.32: Digit selection design for square root unit

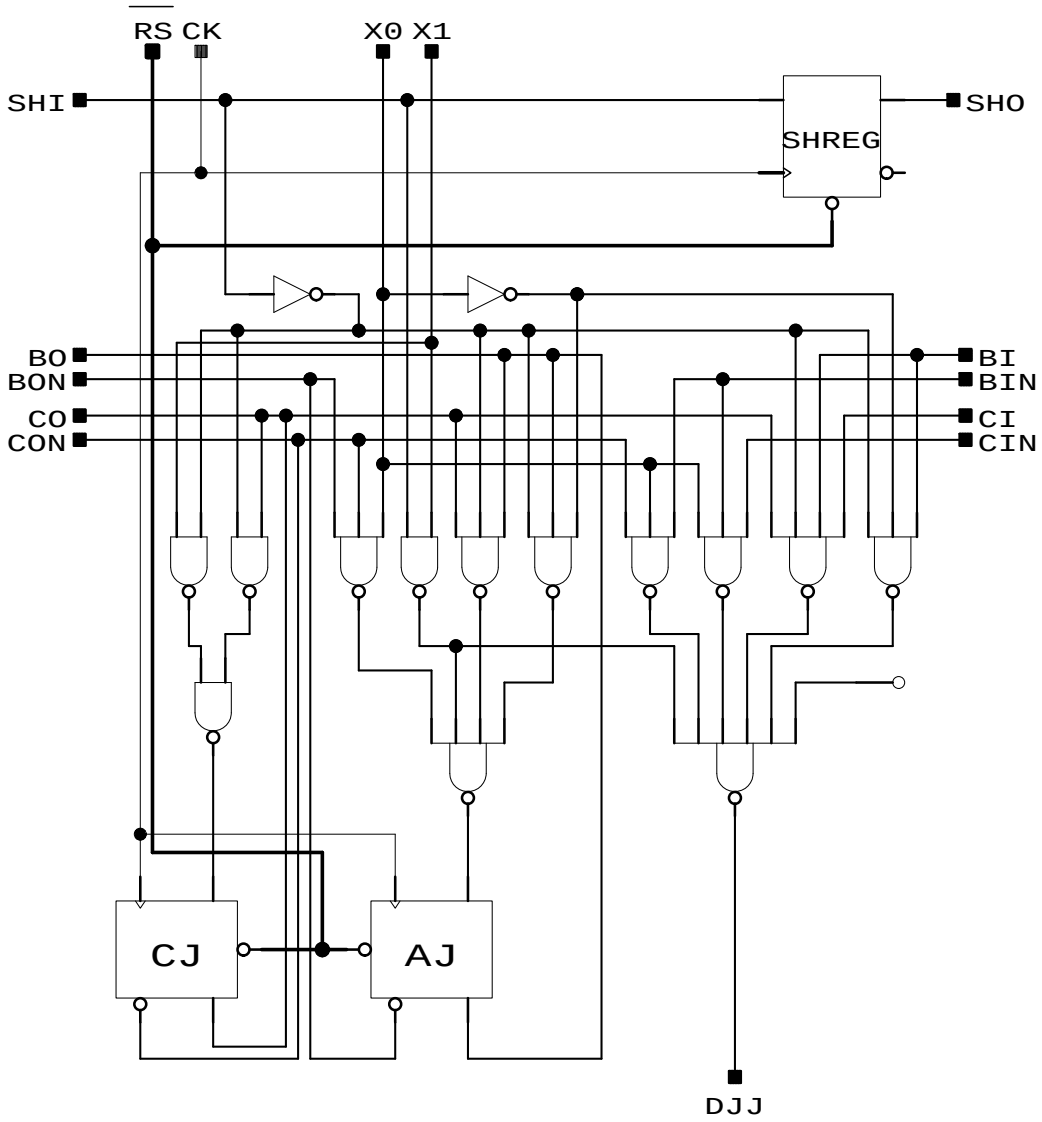


Figure 4.33: Data conversion for on-line square root operation

Chapter 5

Application of on-line arithmetic algorithms

This chapter is to explore the effectiveness of implementing numerical computations with on-line arithmetic algorithms. Of interest are the characteristics of computations that make them suitable for on-line implementation and the cost and performance features of such on-line computations. Since on-line algorithms operate in a digit-pipelined fashion, it is essential that the critical path of its application contain long sequences of arithmetic operations. One such application is the singular value decomposition (SVD)[37], which is used as an example for the presentation.

In order to conduct some preliminary comparison with conventional approaches, we give estimates of the complexity and performance parameters of the on-line approach based on the gate array designs of Chapter 4. Complexity figures are calculated directly from the data given by the designs, and delays are measured from library components using the data given in Appendix D.

Section 5.1 gives an overview of the SVD algorithm. In Section 5.2 the on-line scheme for the SVD computation is presented. It is assumed that standardized single-operation on-line units are used for the implementation. The on-line scheme is compared with the conventional approach in Section 5.3.

5.1 The singular value decomposition algorithm

The singular value decomposition (SVD) is one of the basic matrix operations required for many important modern signal processing computations[37]. The

SVD of a matrix $\mathbf{A}$ is defined as the factorization

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

where $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, and $\mathbf{\Sigma}$ is a diagonal matrix with non-negative diagonal elements. For many real-time signal processing applications, numerically stable SVD algorithms that are suitable for implementation on parallel computer architectures are desirable.

We adopt the algorithm presented by Brent, Luk and van Loan[38] which computes the SVD of an $n \times n$ matrix $\mathbf{A}$ on a mesh-connected processor array. It has been shown by Park [39] that this algorithm converges when n is odd, and does not always converge when n is even. In our discussion on-line arithmetic is used as the low level realization of arithmetic operations of a numerical computation, and it is assumed that for the on-line computation to generate the correct result, the high level algorithm must be numerically correct when realized by conventional parallel arithmetic algorithms. Our emphasis is on the implementation efficiency and performance, not the convergence of the computation. For the sake of clarity in our illustration, we assume that $\mathbf{A}$ is real and n is even. The same conclusions apply when n is odd.

In this scheme, the array consists of $n/2 \times n/2$ processors, where each processor operates on a 2×2 submatrix of $\mathbf{A}$ as is illustrated in Figure 5.1. A series of 2×2

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	a_{58}
a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}	a_{28}
a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}	a_{78}
a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}	a_{87}	a_{88}

Figure 5.1: Mapping a matrix onto a processor array ($n = 8$)

SVDs are performed on submatrices along the main diagonal of $\mathbf{A}$, indicated by the shaded areas in Figure 5.1. Each 2×2 SVD is realized by a 2-sided rotation

that diagonalizes the submatrix. The rotation angles are computed by Algorithm 5.1, which is the FHSVD algorithm given in [38]. Then the rotation is computed as

$$\begin{bmatrix} a'_{i,i} & 0 \\ 0 & a'_{i+1,i+1} \end{bmatrix} = \begin{bmatrix} c_i^L & s_i^L \\ -s_i^L & c_i^L \end{bmatrix}^T \begin{bmatrix} a_{i,i} & a_{i,i+1} \\ a_{i+1,i} & a_{i+1,i+1} \end{bmatrix} \begin{bmatrix} c_i^R & s_i^R \\ -s_i^R & c_i^R \end{bmatrix}$$

where c_i^L, s_i^L, c_i^R and s_i^R are the cosine and sine values of the left and right rotation angles, respectively. Figure 5.2 is the computation graph for the rotation angle calculation.

During each iteration, $n/2$ 2×2 SVDs are performed in parallel. Values of the left rotation angles are passed to processors of the corresponding rows while values of the right rotation angles are passed along the corresponding columns, where 2×2 rotations are performed on the submatrix in each processor,

$$\begin{bmatrix} a'_{i,j} & a'_{i,j+1} \\ a'_{i+1,j} & a'_{i+1,j+1} \end{bmatrix} = \begin{bmatrix} c_i^L & s_i^L \\ -s_i^L & c_i^L \end{bmatrix}^T \begin{bmatrix} a_{i,j} & a_{i,j+1} \\ a_{i+1,j} & a_{i+1,j+1} \end{bmatrix} \begin{bmatrix} c_j^R & s_j^R \\ -s_j^R & c_j^R \end{bmatrix}$$

The 2-sided 2×2 rotation is computed as

$$\begin{aligned} a'_{i,j} &= c_j^R(c_i^L a_{i,j} - s_i^L a_{i+1,j}) - s_j^R(c_i^L a_{i,j+1} - s_i^L a_{i+1,j+1}) \\ a'_{i,j+1} &= s_j^R(c_i^L a_{i,j} - s_i^L a_{i+1,j}) + c_j^R(c_i^L a_{i,j+1} - s_i^L a_{i+1,j+1}) \\ a'_{i+1,j} &= c_j^R(s_i^L a_{i,j} + c_i^L a_{i+1,j}) - s_j^R(s_i^L a_{i,j+1} + c_i^L a_{i+1,j+1}) \\ a'_{i+1,j+1} &= s_j^R(s_i^L a_{i,j} + c_i^L a_{i+1,j}) + c_j^R(s_i^L a_{i,j+1} + c_i^L a_{i+1,j+1}) \end{aligned}$$

The computation graph for the rotation is shown in Figure 5.3.

After each iteration, rows and columns of the matrix are exchanged between adjacent processors according to the *parallel ordering* which is described in [40]. Under this ordering, for each i and j, such that $1 \leq i < j \leq n$, columns i and j of $\mathbf{A}$ are paired and rotated in the same column of processors exactly once in every $n-1$ consecutive iterations which are referred to as a *sweep*. Likewise, for each $1 \leq i < j \leq n$, rows i and j of $\mathbf{A}$ are paired and rotated in the same row of processors exactly once in every sweep. To simplify control, a fixed number of S sweeps is performed to diagonalize $\mathbf{A}$. It is shown in [38] that $S < 9$ for $n \leq 50$.

Assuming that there is no data broadcasting among processors, Figure 5.4 shows the processor array by Brent, Luk and van Loan. The dashed arrow lines represent the transmission of the rotation angles and the solid arrow lines represent the transmission of matrix elements. Processors on the main diagonal compute the rotation angles and the main diagonal elements of the matrix, while those not on the main diagonal perform 2×2 rotations. Figures 5.5 and 5.6 show

Algorithm 5.1 (FHSVD)

$\mu_1 = a_{i+1,i+1} - a_{i,i}$
$\nu_1 = a_{i+1,i} + a_{i,i+1}$
if $|\nu_1| \leq \epsilon|\mu_1|$ **then**
  $\chi_1 = 1$
  $\sigma_1 = 0$
else
  $\rho_1 = \dfrac{\mu_1}{\nu_1}$
  $\tau_1 = \dfrac{\mathrm{sign}(\rho_1)}{|\rho_1| + \sqrt{1+\rho_1^2}}$
  $\chi_1 = \dfrac{1}{\sqrt{1+\tau_1^2}}$
  $\sigma_1 = \chi_1\tau_1$

$\mu_2 = a_{i+1,i+1} + a_{i,i}$
$\nu_2 = a_{i+1,i} - a_{i,i+1}$
if $|\nu_2| \leq \epsilon|\mu_2|$ **then**
  $\chi_2 = 1$
  $\sigma_2 = 0$
else
  $\rho_2 = \dfrac{\mu_2}{\nu_2}$
  $\tau_2 = \dfrac{\mathrm{sign}(\rho_2)}{|\rho_2| + \sqrt{1+\rho_2^2}}$
  $\chi_2 = \dfrac{1}{\sqrt{1+\tau_2^2}}$
  $\sigma_2 = \chi_2\tau_2$

$$
\begin{aligned}
c_i^L &= \chi_1\chi_2 + \sigma_1\sigma_2 \\
s_i^L &= \sigma_1\chi_2 - \chi_1\sigma_2 \\
c_i^R &= \chi_1\chi_2 - \sigma_1\sigma_2 \\
s_i^R &= \sigma_1\chi_2 + \chi_1\sigma_2
\end{aligned}
$$

end.

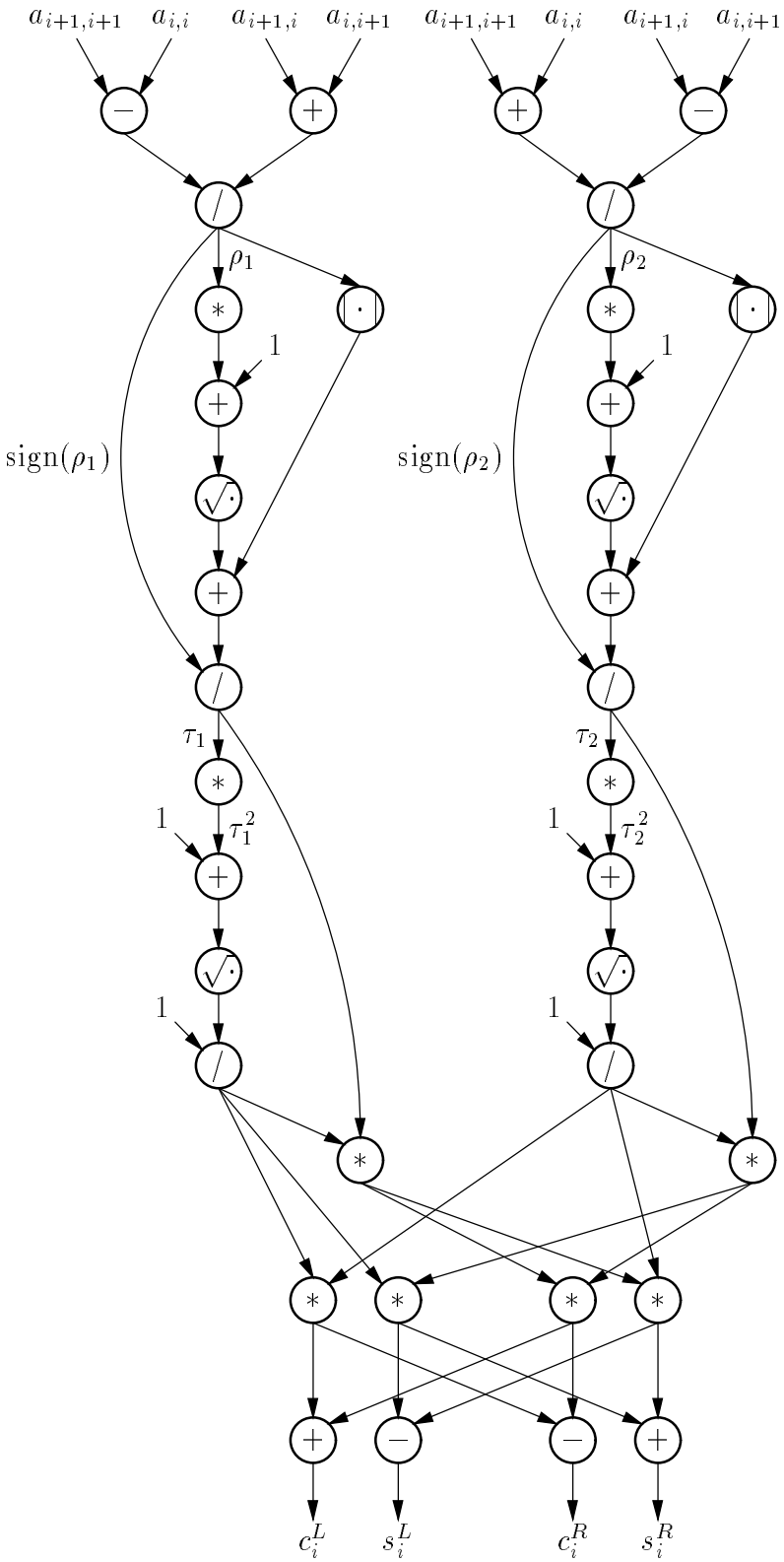


Figure 5.2: Computation graph for algorithm FHSVD

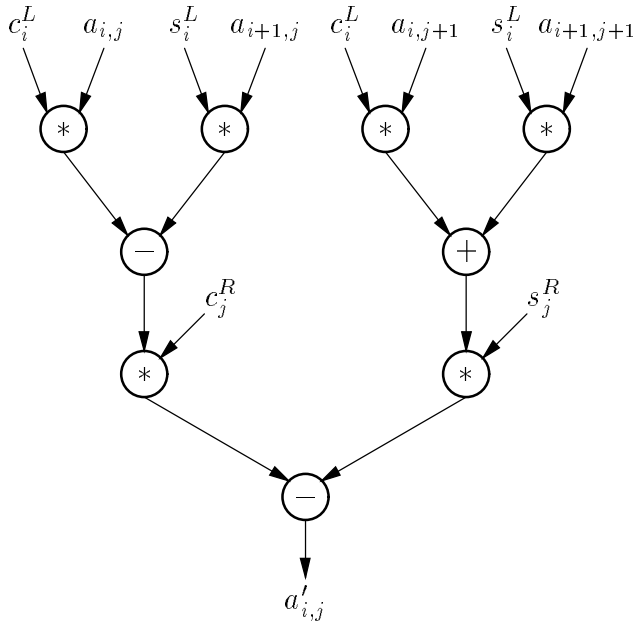


Figure 5.3: Computation graph for rotation calculation

the computations performed by the main diagonal and off-diagonal processors, respectively.

The performance bottleneck of the SVD is clearly the calculation of the rotation angle, which contains long sequences of sequential operations and cannot be effectively sped up by conventional techniques.

5.2 On-line scheme for computing the SVD: the network approach

In the network approach of on-line computation, each arithmetic operation in the computation is realized by a single-operation on-line unit. All on-line units have the same interfaces, so successive units can be interconnected directly with no intermediate data conversion. We assume that the inter-processor transmission of the angles and the matrix elements are on-line.

For each on-line operation, it is assumed that the input operands are quasi-normalized, which is defined as

$$|x| \in [\frac{1}{4}, 1)$$

The output is also quasi-normalized. This is achieved by incorporating post-normalization step into the on-line algorithms for addition, multiplication and

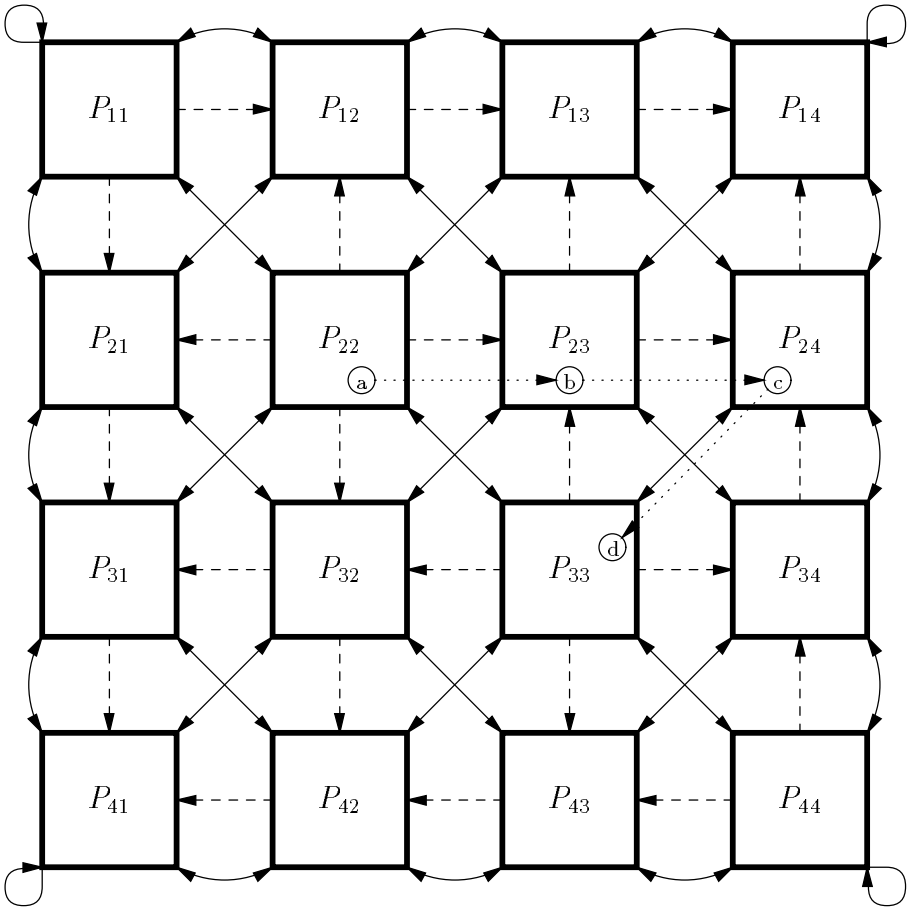


Figure 5.4: Systolic array for the SVD by Brent, Luk and van Loan ($n = 8$)

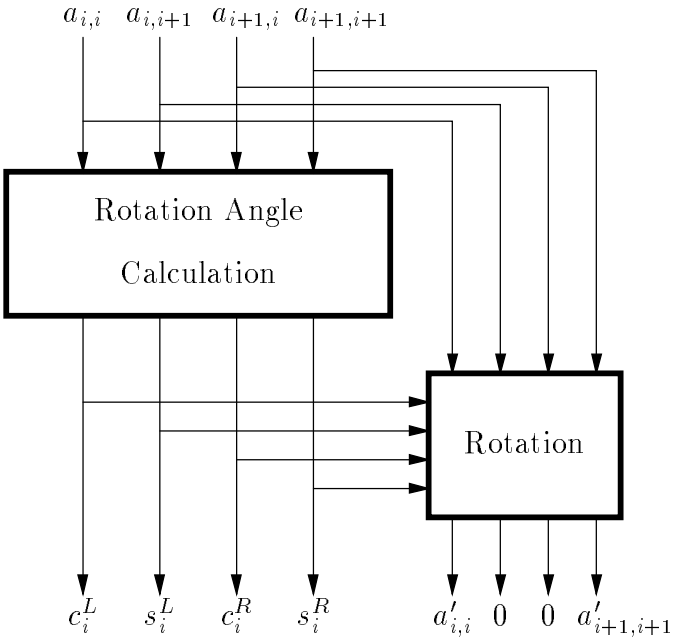


Figure 5.5: Block diagram of a main diagonal processor

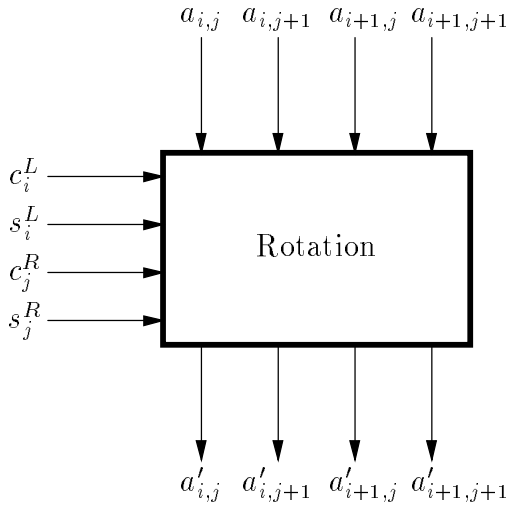


Figure 5.6: Block diagram of an off-diagonal processor

division operations. The on-line result of square root operation is always quasi-normalized. The most-significant bit of the mantissa is always non-zero thus making the sign determination simple. This also holds in the case of cancellation of the most significant digits, since no output is delivered before the result is quasi-normalized. While the on-line delays of multiplication, division and square root given in Table 5.3 are bounded by 7, 7 and 4, respectively, the on-line delay of addition can be as large as 56 due to cancellation of the most significant digits. This is a serious problem that has not been included in the performance analysis in this dissertation. However, the design includes necessary on-line buffers to synchronize inputs arriving out of phase due to cancellation.

5.2.1 Complexity of the on-line SVD computation

Figure 5.7 shows the on-line network for the FHSVD algorithm. The ϵ tests in the FHSVD algorithm are performed concurrently with the computation of ρ_i, τ_i, χ_i and σ_i. If a test succeeds, the corresponding variables are forced to 1 and 0, respectively. Two types of delay buffers are used in the implementation. Fixed delay buffers (FDB) are used to synchronize the arrival of operands for some of the on-line units over the paths with a fixed difference in on-line delays. The out-of-phase inputs, caused by cancellation, are synchronized through variable delay buffers (VDB). These buffers use a design similar to the mantissa alignment and have an estimated cost of 1,150 gates/buffer. The sign of a number can be obtained directly from the sign bit of the first mantissa digit. Constants can be stored in registers and transmitted to the on-line units when needed. The control of on-line units is simple and it is not included in this study.

The on-line network for the rotation computation of main diagonal processors is shown in Figure 5.8. We assume that six inputs to the upper and lower level multipliers are synchronized using 6 variable delay buffers each. Figure 5.9 shows the on-line network for the rotation computation of off-diagonal processors. The number of each type of arithmetic units and estimates of their gate count for each processor are given in Table 5.1 for radix-2 computation, based on the designs described in Chapter 4. The cost of postnormalization is included. Each input and output data requires 10 pins, 8 for the exponent and 2 for the mantissa. Let G_{diag} and G_{off} denote the gate count for each main diagonal and off diagonal processor, respectively. The total cost of an $n/2 \times n/2$ processor array, which computes the SVD of an $n \times n$ matrix, is

$$G_{\frac{n}{2}\times\frac{n}{2}} = G_{diag} \times \frac{n}{2} + G_{off} \times \left(\left(\frac{n}{2} \right)^2 - \frac{n}{2} \right) \tag{5.1}$$

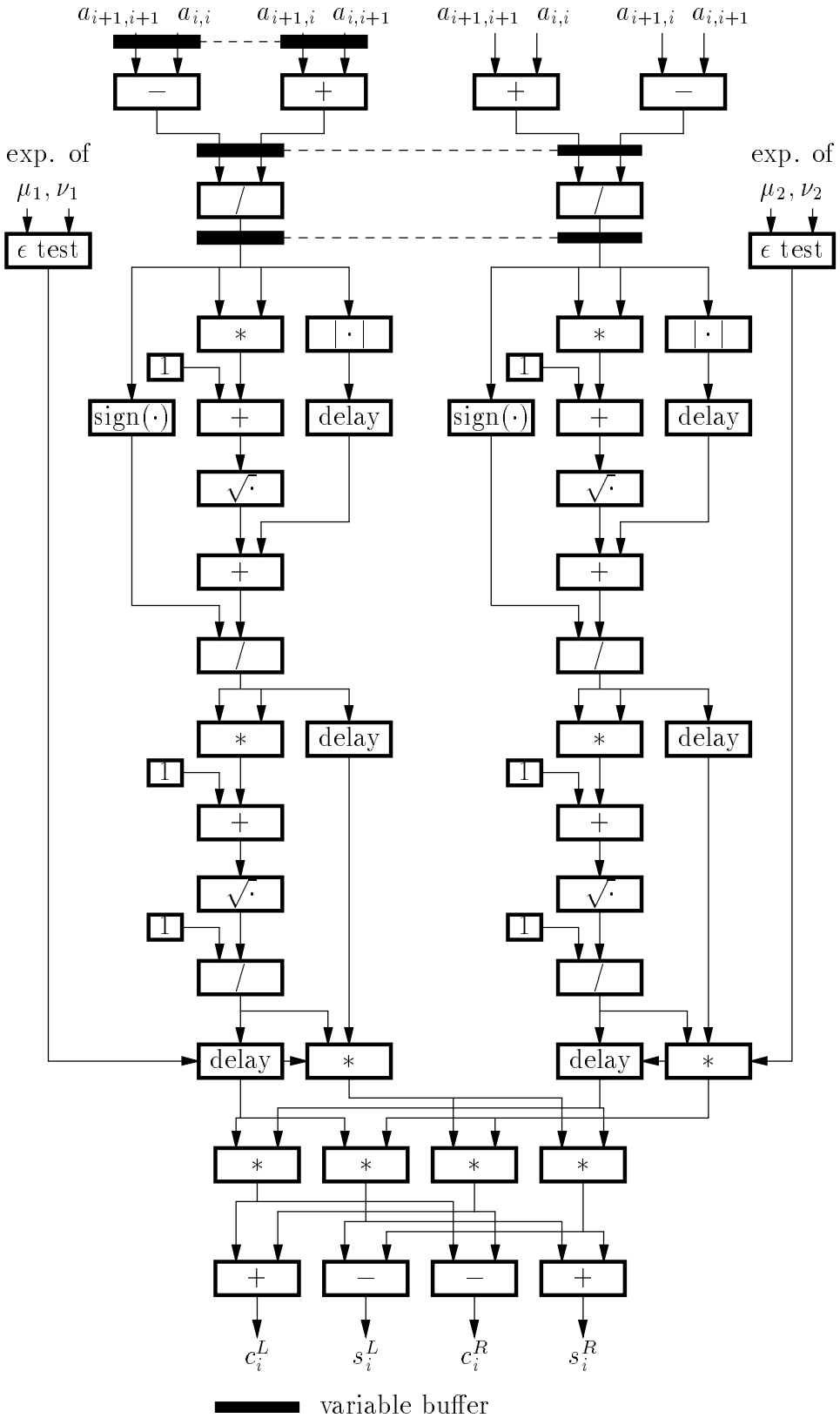


Figure 5.7: On-line scheme for algorithm FHSVD

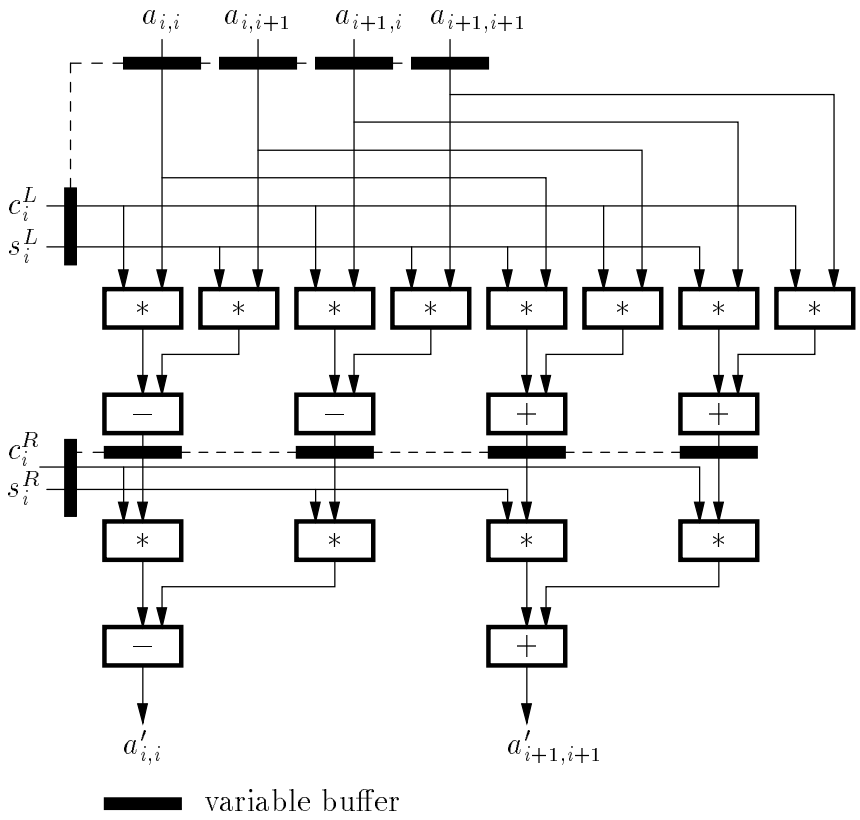


Figure 5.8: On-line scheme for rotation of a diagonal processor

Arith. unit	Gate count	Data pins	Main diag. proc.				Off diag. proc.	
			ang. units	rot. units	total units	cost	rot. units	cost
Add	1337	30	14	6	20	26700	8	10700
Mult	3941	30	10	12	22	86700	16	63100
Div	4261	30	6	0	6	25600	0	0
Sqrt	4261	20	4	0	4	17000	0	0
VD Buffer	1150		5	12	17	19950	12	13800
FD Buffer	100		6	0	6	600	0	0
ϵ Tests	200		2	0	2	400	0	0
Total cost			~ 176950				~ 87600	

Table 5.1: Processor cost of on-line SVD

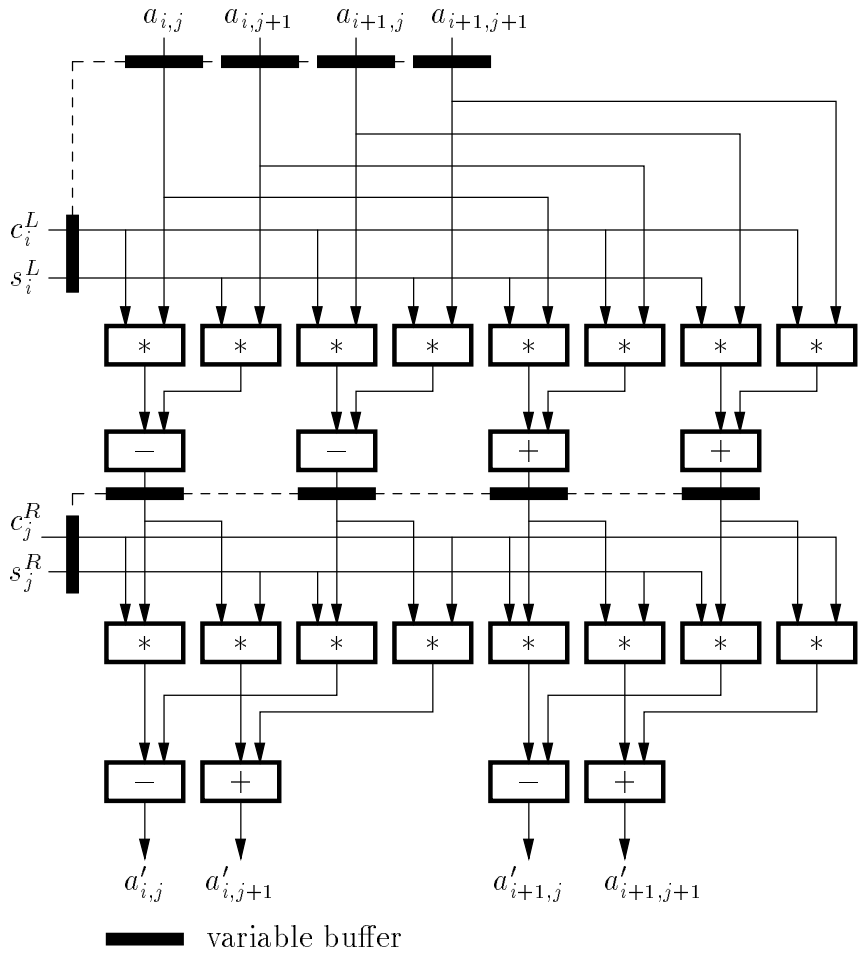


Figure 5.9: On-line scheme for rotation of an off-diagonal processor

5.2.2 Performance of the on-line SVD

The SVD computation is composed of the rotation angle calculation and the rotation calculation. The operations in the critical paths of these calculations are listed in Table 5.2. Let T_{cyc} denote the *iteration cycle time*, defined as the number of clock cycles between starts of consecutive iterations, and T_{ite} the *complete iteration time*, which is the number of clock cycles for a complete iteration. Assume that it takes one clock cycle for data to pass through a processor, and the exchange of matrix elements between processors takes no extra time. Then the *complete sweep time*, defined as the number of clock cycles to compute a sweep, which consists of $n-1$ iterations, is

$$T_{swe} = (n-2) \cdot T_{cyc} + T_{ite} \tag{5.2}$$

The total number of clock cycles for the SVD consisting of S sweeps is

$$T_{svd} = (S \cdot (n-1) - 1) \cdot T_{cyc} + T_{ite} \tag{5.3}$$

When post-normalization is performed in an on-line operation, it costs an extra cycle to the on-line delay since the second digit of the result is examined before determining whether the first digit can be output. When the result contains leading zeros, the zeros are discarded and more digits are examined. In the optimal case, there is no leading zero generated in the mantissa, while in the worst case (assuming no leading digit cancellation for addition), up to 1 (addition), 2 (division) or 3 (multiplication) leading zeros may be present. Based on the designs presented in Chapter 4, the required minimum step-time and the on-line delay δ of the radix-2 implementation of the basic operations are listed in Table 5.3. The step-time is measured in units of t_g, which is roughly the delay of a nand gate with no more than 4 inputs. In the following discussion, we define *cycles* as the maximum of the minimum required step-time for all individual operations. We also assume δ_{max} for each on-line unit. The delays of the components used in the designs are listed in Table D.8 of Appendix D. The on-line delays for the rotation angle calculation

Computation	$+/-$	$\times$	$\div$	$\sqrt{\ }$
Angle calc.	5	4	3	2
Rotation	2	2		

Table 5.2: Critical path operations for the SVD

Operation	minimum step-time	δ_{min}	δ_{max}
Add*	14 t_g	5 *cycles*	5 *cycles*
Mult	15 t_g	5 *cycles*	7 *cycles*
Div	17 t_g	6 *cycles*	7 *cycles*
Sqrt	17 t_g	4 *cycles*	4 *cycles*

* no cancellation of leading digits

Table 5.3: On-line delay and step-time for radix-2 on-line units

and the rotation calculation, denoted as Δ_{ang} and Δ_{rot}, respectively, are

$$\begin{aligned} \Delta_{ang} &= 82 \ cycles \\ \Delta_{rot} &= 24 \ cycles \end{aligned}$$

The iteration cycle time of the on-line SVD is limited by two factors.

1. The availability of input. In the SVD computation, the rotation angle calculation of each iteration requires the result of the rotation from the previous iteration. The time between the start of the rotation angle calculation and the arrival of data for the next iteration, denoted as Δ_{ite}, includes the on-line delays of the rotation angle calculation, of transmitting the computed angles through one processor, and of the rotation calculation on an off-diagonal processor, as is indicated in Figure 5.4 by the dotted path $a \rightarrow b \rightarrow c \rightarrow d$. For radix-2 computation, we have

$$\begin{aligned} \Delta_{ite} &= \Delta_{ang} + \delta_{pas} + \Delta_{rot} \\ &= 82 + 1 + 24 = 107 \ cycles \end{aligned}$$

2. The availability of the arithmetic units. For each on-line unit, the next iteration can not start until it has finished the computation of the previous iteration. Since in on-line computations all units work in pipelined fashion, the cycle time between iterations must be no less than the largest busy cycle time of all on-line units in the critical path of the computation. Assuming $m + 1$ output digits are computed, the busy cycle an on-line unit is

$$t_{busy} = \delta + m$$

where δ is the on-line delay of the on-line unit. For radix-2 computation, division has the largest busy cycle with an on-line delay of 7. For double precision computation of 8 exponent bits and 56 mantissa bits, we have

$$t_{busy} = 63 \ cycles$$

Both factors mentioned above present lower bounds on T_{cyc}. Assuming that the matrix size $n \geq 4$, we have

$$T_{ite} = \Delta_{ang} + (\frac{n}{2} - 2) \cdot \delta_{pas} + \Delta_{rot} + m$$

then for radix-2 double precision floating-point computation, we have

$$\begin{aligned} T_{cyc} &= \Delta_{ite} = 107 \ cycles \\ T_{ite} &= 160 + \frac{n}{2} \ cycles \\ T_{swe} &= (107 + \frac{1}{2}) \cdot n - 54 \ cycles \\ T_{svd} &= S \cdot (n-1) \cdot 107 + \frac{n}{2} + 53 \ cycles \end{aligned}$$

The hardware utilization is measured in terms of $unit \cdot cycle$, which is defined as an arithmetic unit resource for one clock cycle during the computation. The total amount of resource available for the SVD computation is

$$\text{total resource} = (\text{total number of units}) \times T_{svd}$$

and the amount of resources utilized during the computation is the summation of busy cycles of all units

$$T_{busy} \equiv \sum_{\text{all units}} t_{busy}$$

The utilization factor of the computation is defined as the ratio of T_{busy} and the total resource,

$$U \equiv \frac{T_{busy}}{\text{total resource}}$$

For the on-line SVD computation, each main diagonal processor has 20 addition units, 22 multiplication units, 6 division units and 4 square root units. Each off-diagonal processor has 8 addition units and 16 multiplication units. Assuming the worst case on-line delay (δ_{max}) for each on-line operation, and no leading digit calcellation in addition, the busy cycle for each addition, multiplication, division and square root unit is 61, 63, 63 and 60, respectively. Then the utilization factor for the on-line SVD is

$$U_{OL} = \frac{\left(3224 \times \frac{n}{2} + 1496 \times \left(\left(\frac{n}{2}\right)^2 - \frac{n}{2}\right)\right) \times (n-1) \times S}{\left(52 \times \frac{n}{2} + 24 \times \left(\left(\frac{n}{2}\right)^2 - \frac{n}{2}\right)\right) \times T_{svd}}$$

From this expression it can be shown that for $S \geq 10$ and $n \geq 8$, the unitlization factor for the on-line SVD computation is

$$U_{OL} \geq 0.57$$

Figure 5.10 is a timing diagram that shows the busy and idle periods of each of the main diagonal and off-diagonal processors.

5.3 Comparison with conventional schemes

We now compare the on-line implementation with conventional special purpose designs for the SVD computation. The delays are measured in terms of *gate delays*, denoted as t_g, which is approximately the delay time of a nand gate with no more than 4 inputs. The effect of fanout is ignored in the comparison since it depends on low level design details, and we assume that it affects designs of both approaches similarly. Costs of the designs are measured in terms of LSI *equivalent gates*, which is defined in [41]. The data format of input and output of each arithmetic operation is double precision floating-point with 8-bit exponent and 56-bit mantissa.

The designs of the on-line and conventional implementations are based on components available from the LSI HCMOS gate array libraries[41]. The cost of the on-line implementation is from the designs presented in Chapter 4. The cost of the conventional implementation is based on schemes discussed in Appendix D. The cost and delay figures of the components for both approaches are given in Table D.8 in Appendix D.

From Table 5.3, the step-time of the on-line implementation is 16 t_g. The step-time of the conventional approach is assumed to be that of the 56-bit parallel adder, which is 21 t_g. The difference of step-time is taken into consideration when comparing the schemes.

Case 1: Maximum parallelism

In this case we assume for the conventional approach that there are as many arithmetic units available as needed by the computation whenever data is ready. For the processors on the main diagonal of the array, there are 4 adders, 8 multipliers, 2 dividers and 2 square root units. During each iteration, each adder is used 5 times, each multiplier 3 times, each divider 3 times and each square root unit 2 times. For the off-diagonal processors, there are 4 adders and 8 multipliers. Each adder and multiplier is used 2 times per iteration. Table 5.4 lists the multiplexers

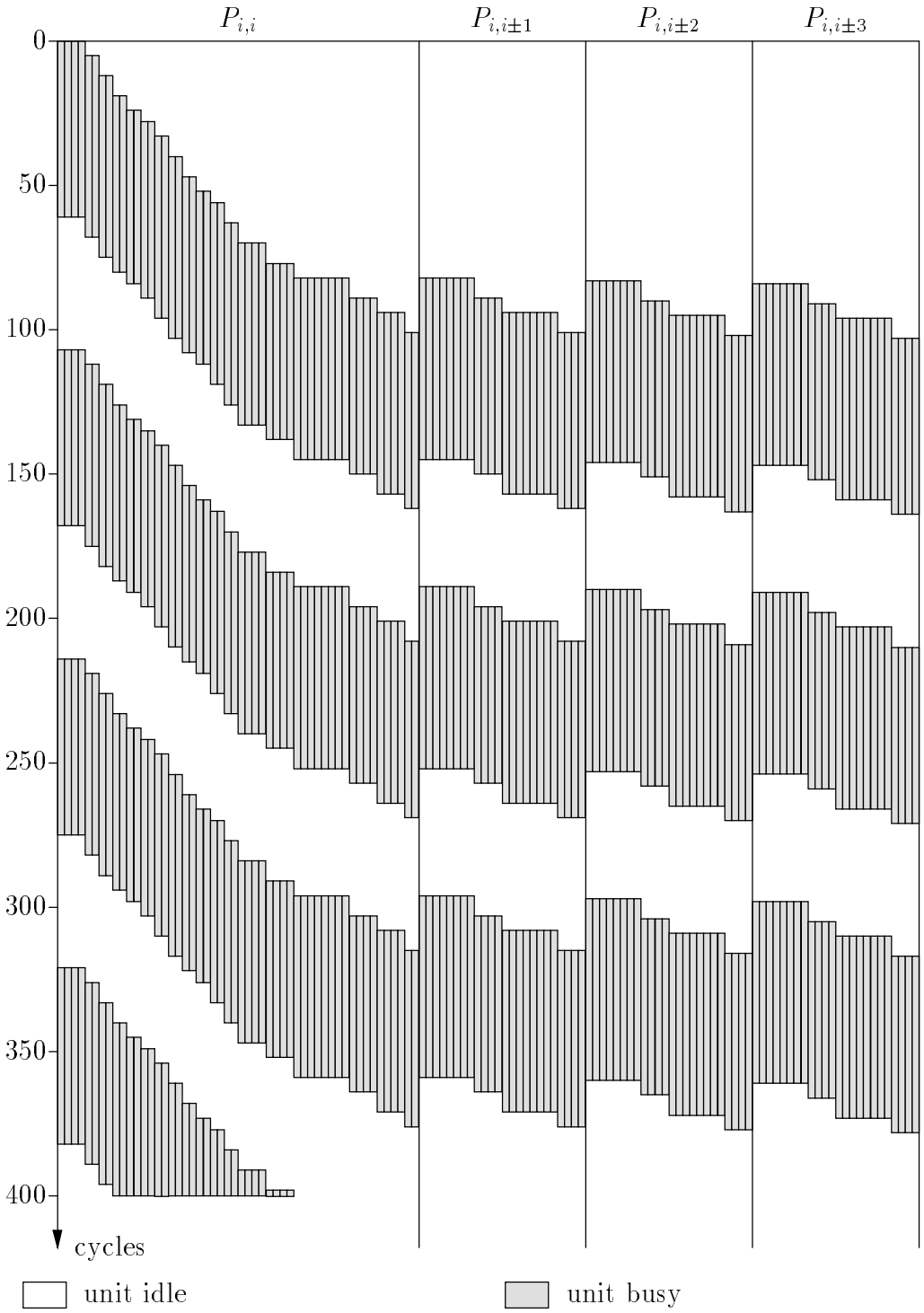


Figure 5.10: Timing diagram for on-line SVD ($n = 8$)

Arithmetic unit	Main diag. proc.		Off diag. proc.	
	Component	num.	Component	num.
Adder	5-input MUX	8	2-input MUX	8
Multiplier	3-input MUX	12	2-input MUX	16
	2-input MUX	4		
Divider	3-input MUX	4		
Sqrt unit	2-input MUX	4		
Total	2-input MUX	8	2-input MUX	24
	3-input MUX	16		
	5-input MUX	8		

Table 5.4: Multiplexers for arithmetic units (Case 1)

that are needed for each processor. The total cost of each main diagonal processor is

$$\begin{aligned} G_{diag} &= 4 \cdot g_{add} + 8 \cdot g_{mult} + 2 \cdot g_{div} + 2 \cdot g_{sqrt} + 8 \cdot g_{mux2} + 16 \cdot g_{mux3} + 8 \cdot g_{mux5} \\ &\approx 149,400 \end{aligned}$$

and the cost of each off diagonal processor is

$$\begin{aligned} G_{off} &= 4 \cdot g_{add} + 8 \cdot g_{mult} + 24 \cdot g_{mux2} \\ &\approx 120,600 \end{aligned}$$

The number of cycles for each arithmetic operation, from Appendix D, are

$$\begin{aligned} t_{add} &= 2 \ cycles \\ t_{mult} &= 6 \ cycles \\ t_{div} &= 30 \ cycles \\ t_{sqrt} &= 29 \ cycles \end{aligned}$$

From the critical path operations given in Table 5.2, the number of cycles for the rotation angle calculation and the rotation calculation are

$$\begin{aligned} T_{ang} &= 5 \cdot t_{add} + 4 \cdot t_{mult} + 3 \cdot t_{div} + 2 \cdot t_{sqrt} = 182 \ cycles \\ T_{rot} &= 2 \cdot t_{add} + 2 \cdot t_{mult} = 16 \ cycles \end{aligned}$$

Then we have the iteration cycle time and the complete iteration time for $n \geq 4$,

$$\begin{aligned} T_{cyc} &= T_{ang} + \delta_{pas} + T_{rot} = 199 \ cycles \\ T_{ite} &= T_{ang} + (\frac{n}{2} - 2) \cdot \delta_{pas} + T_{rot} \\ &= 196 + \frac{n}{2} \ cycles \end{aligned}$$

Arithmetic unit	Main diag. proc.		Off diag. proc.	
	Component	num.	Component	num.
Adder	5-input MUX	8	4-input MUX	4
Multiplier	3-input MUX	12	4-input MUX	8
	2-input MUX	4		
Divider	3-input MUX	4		
Sqrt unit	2-input MUX	4		
Total	2-input MUX	8	4-input MUX	12
	3-input MUX	16		
	5-input MUX	8		

Table 5.5: Multiplexers for arithmetic units (Case 2)

Case 2: Reduced complexity

Assume that each conventional processor on the main diagonal of the array still has 4 adders, 8 multipliers, 2 dividers and 2 square root units, but let the off diagonal processors have 2 adders and 4 multipliers each. Each off diagonal processor performs the rotation computation twice per iteration, each time computing the rotations of 2 of the matrix elements associated with the processor. If a matrix element is going to be transmitted to a main diagonal processor, it is always computed in the first run. Each adder and multiplier of an off diagonal processor is used 4 times per iteration. Table 5.5 shows the number of multiplexers used in each processor. The cost for each off diagonal processor is

$$\begin{aligned} G_{\text{off}} &= 2 \cdot g_{add} + 4 \cdot g_{mult} + 12 \cdot g_{mux4} \\ &\approx 62800 \end{aligned}$$

Because of the long computation delay of the rotation angle calculation performed by the main diagonal processors, T_{cyc} remains the same as in case 1. The complete iteration time becomes

$$\begin{aligned} T_{ite} &= T_{ang} + (\frac{n}{2} - 2) \cdot \delta_{pas} + 2 \cdot T_{rot} \\ &= 212 + \frac{n}{2}\ cycles \end{aligned}$$

Figure 5.11 is a timing diagram showing the busy periods of the arithmetic units

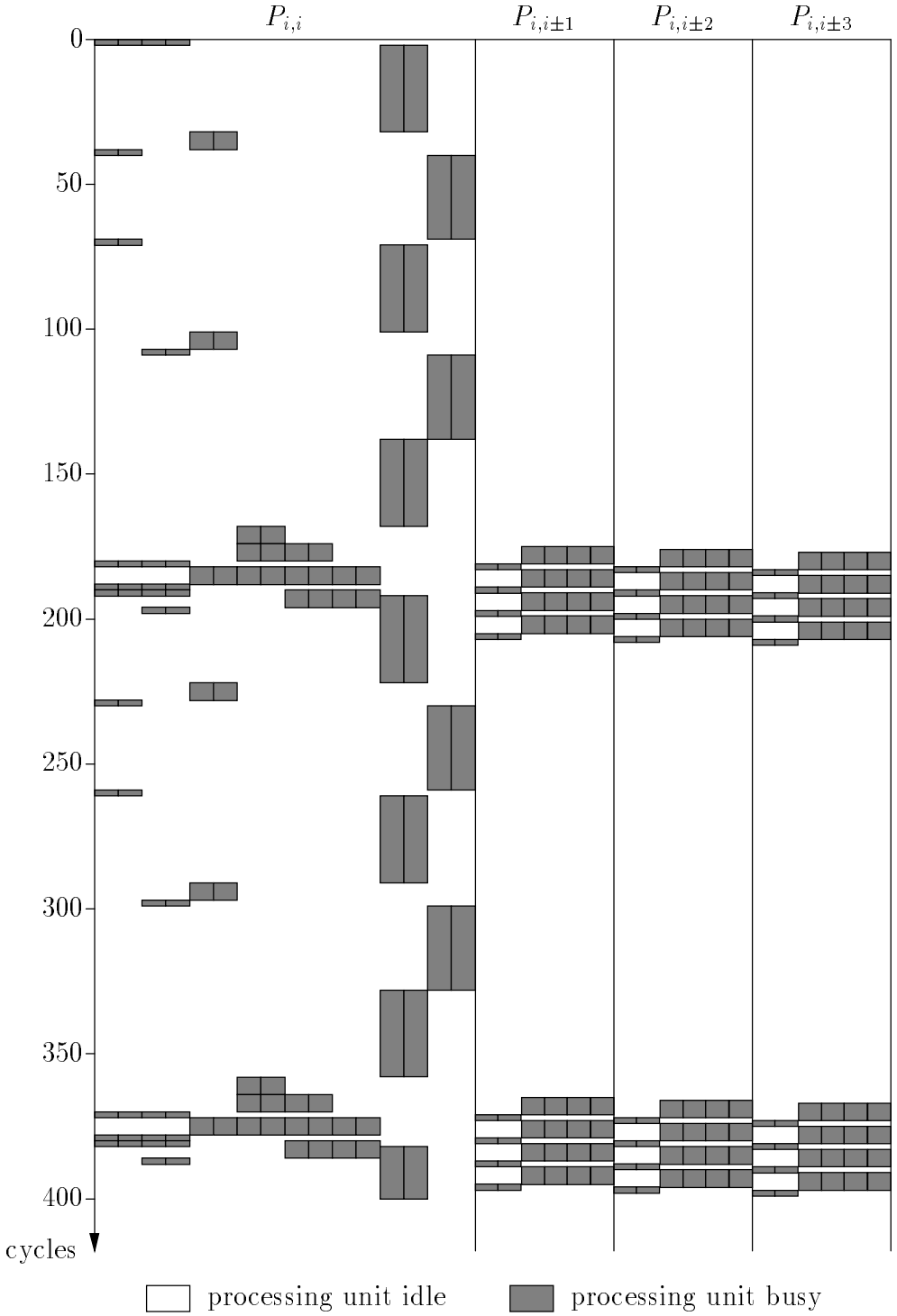


Figure 5.11: Timing diagram for conventional SVD ($n = 8$)

of the conventional design. The utilication factor for this scheme is

$$U_{C2} = \frac{\left(468 \times \frac{n}{2} + 112 \times \left(\left(\frac{n}{2}\right)^2 - \frac{n}{2}\right)\right) \times (n-1) \times S}{\left(16 \times \frac{n}{2} + 6 \times \left(\left(\frac{n}{2}\right)^2 - \frac{n}{2}\right)\right) \times T_{svd}}$$

and for $n = 8$, $S = 10$ we get

$$U_{C2} = 0.124$$

Case 3: Maximum utilization

Assume that each main diagonal processor has one each of adder, multiplier, divider and square root unit, and each off-diagonal processor has one adder and one multiplier. For each iteration, each diagonal processor performs 20 additions, 22 multiplications, 6 divisions, 4 square root operations, and each off diagonal processor performs 8 additions and 16 multiplications. Each processor needs about 12 registers for buffering input, output and intermediate results. The complexity of the processors are

$$\begin{aligned} G_{diag} &= g_{add} + g_{mult} + g_{div} + g_{sqrt} \approx 26800 \\ G_{off} &= g_{add} + g_{mult} \approx 16800 \end{aligned}$$

Assume that the order of performing the arithmetic operations is such that those operations in the critical path always proceed when ready. The performance figures are

$$\begin{aligned} T_{ang} &= 235 \ cycles \\ \Delta_{ang} &= 221 \ cycles \\ T_{rot} &= \begin{cases} 74 \ cycles & \text{diagonal processor} \\ 98 \ cycles & \text{off diagonal processor} \end{cases} \\ \Delta_{rot} &= 38 \ cycles \\ T_{cyc} &= 269 \ cycles \\ T_{ite} &= \Delta_{ang} + (\frac{n}{2} - 2)\delta_{pass} + T_{rot} = 317 + \frac{n}{2} \ cycles \\ T_{swe} &= (n-2)T_{cyc} + T_{ite} = 269.5n - 221 \ cycles \\ T_{svd} &= (S(n-1) - 1)T_{cyc} + T_{ite} \\ &= 269S(n-1) + \frac{n}{2} - 48 \ cycles \end{aligned}$$

Parameter	on-line (radix-2)	conventional		
		Case 1	Case 2	Case 3
T_{cyc}	1	2.45	2.45	3.28
T_{ite}	1	1.60	1.73	2.56
T_{swe}	1	2.27	2.30	3.15
T_{svd}	1	2.42	2.43	3.21
G_{diag}	1	0.84	0.84	0.15
G_{off}	1	1.38	0.77	0.19
Total cost	1	1.16	0.77	0.18

Table 5.6: Performance and cost ratio against on-line scheme for $n = 8$ and $S = 10$

The utilization factor is

$$U_{C3} = \frac{\left(468 \times \frac{n}{2} + 112 \times \left(\left(\frac{n}{2}\right)^2 - \frac{n}{2}\right)\right) \times (n-1) \times S}{\left(4 \times \frac{n}{2} + 2 \times \left(\left(\frac{n}{2}\right)^2 - \frac{n}{2}\right)\right) \times T_{svd}}$$

For $n = 8$ and $S = 10$ we have

$$U_{C3} = 0.3$$

Figure 5.12 is a timing diagram for this case.

Table 5.6 shows the performance and cost ratio for the conventional and on-line SVD with $n = 8$ and $S = 10$, assuming no calcellation of leading digits. The figures also reflect the ratio between the step-times of the schemes.

The comparison shows that for $n = 8$, the on-line scheme achieves a speedup of about 2.4 over Case 2 of the conventional implementation and about 3.2 over Case 3, and the gate count ranging from factor 1.3 to 5.5 of the conventional approach. For each processor, the on-line scheme requires 80 data connections, while the conventional scheme needs 512 data connections.

5.4 Concluding remarks

Through the example given in this chapter, we have demostrated the following.

1. In an on-line implementation, the interconnection bandwidth requirements are reduced due to the digit-serial nature of the algorithms. This will result in reduced wiring area and easier routing in VLSI design.

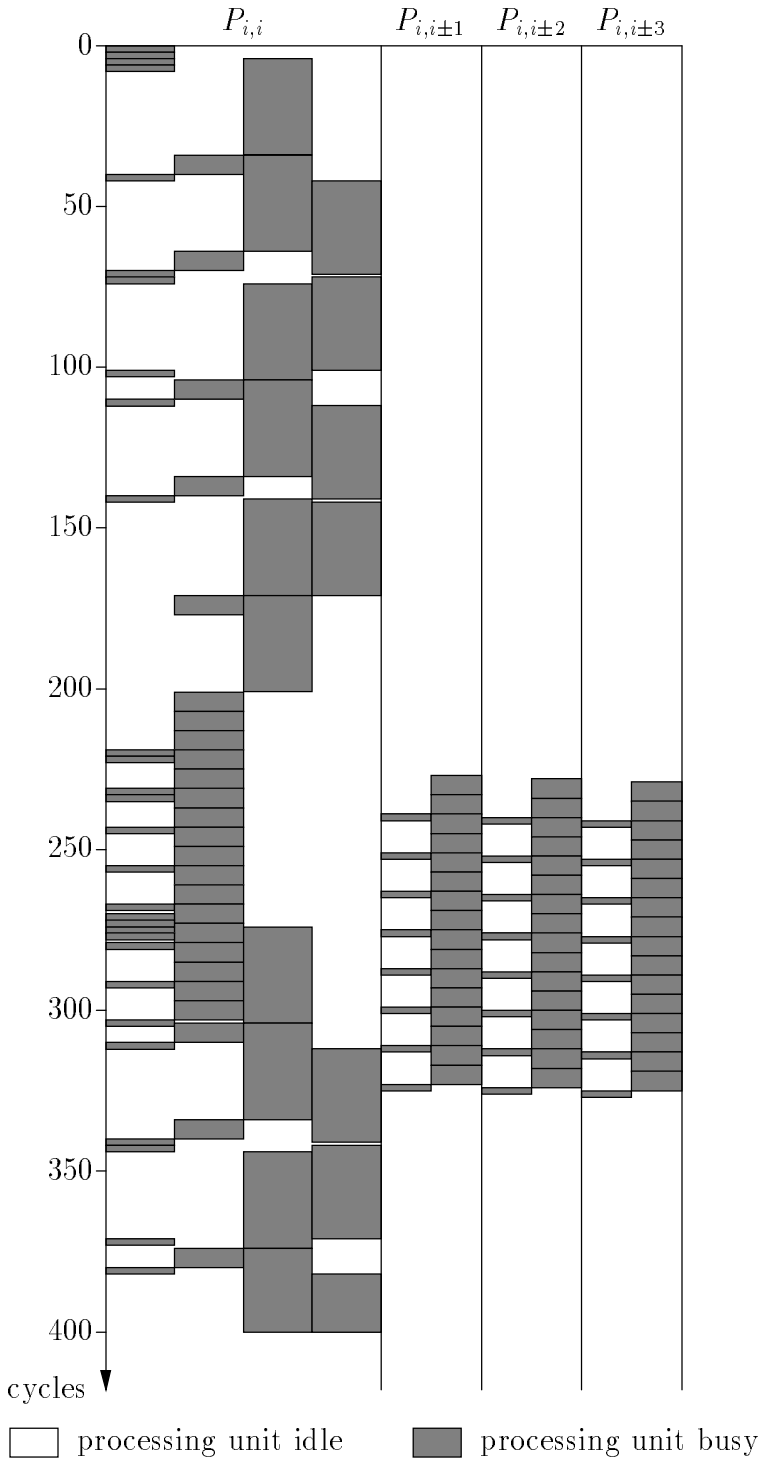


Figure 5.12: Timing diagram for conventional SVD ($n = 8$)

Operation	On-line	Conventional
Add	1337	4773
Multiply	3941	12063
Divide	4261	4993
Square root	4261	4993

Table 5.7: Cost comparison of on-line and conventional arithmetic unit design

2. For a computation, the performance of the on-line approach relies on the *on-line delay* of the overall cirtical path being much smaller than the complete computation time for the same path for the conventional approach. The on-line approach can usually outperform the conventional approach when the critical path of the computation contains long sequences of arithmetic operations.

3. For double precision floating point computations, the designs of the on-line arithmetic units have less complexity than the conventional designs. Table 5.7 compares the gate counts of the designs. For the SVD computation, even though the on-line scheme has a much larger number of arithmetic units than the conventional scheme, the overall complexity of the on-line scheme is still competitive when performance is maximized.

4. Because of the overlapping of the on-line operations, the hardware utilization of the on-line scheme is much higher than the conventional designs.

Chapter 6

Summary and suggestions for future research

6.1 Summary

On-line arithmetic is attractive for VLSI implementation of high performance numerical computation because:

1. Sequential operations are overlapped at the digit level, so for long sequences of operations the computation time can be reduced,

2. Data flow through arithmetic units digit serially, most significant digit first, hence the wiring requirement between arithmetic units is reduced and low pin/logic ratio can be achieved.

We explored the implementation characteristics of on-line arithmetic algorithms. Formal derivation of on-line algorithms was given in Chapter 2 with implementation related parameters that affect the performance of an on-line computation. Parameterized algorithms were derived for the basic arithmetic operations with discussions about the effects of varying the values of the design parameters. These results were further extended in Chapter 3 for the development of multi-operation and composite on-line algorithms that improve the performance and complexity of an on-line computation.

In Chapter 4 we gave gate array implementation for the radix-2 floating-point on-line algorithms for the basic arithmetic operations. Detailed discussion and analysis were given for the on-line division unit since it is most complex among the basic operations.

Chapter 5 studied the application of on-line algorithms to the singular value decomposition computation. Comparison with conventional schemes shows that the on-line approach has the advantage of speed up of $2.4 \sim 3.2$ for the SVD and high utilization of hardware due to overlap between arithmetic operations. The cost of the on-line scheme is also competitive compared with performance oriented conventional schemes.

We conclude that on-line arithmetic is effective in improving the performance of long sequences of arithmetic operations, and it is suitable as a viable alternative in special-purpose system implementing numerical computations.

6.2 Suggestions for future research

We suggest the following subjects for consideration of future research.

- Optimal primitive on-line operators for implementing real time computations in modern signal processing applications.
- Fault tolerance in on-line computation.
- Reuse of on-line network for multiple precision computation.
- Error control and rounding. Since on-line arithmetic is suitable for long sequences of arithmetic operations, the accumulative error should be kept under control.

Bibliography

[1] Philip C. Treleaven. VLSI processor architectures. *IEEE Computer*, pages 33–45, June 1982.

[2] H.T. Kung. Why systolic architectures? *IEEE Computer*, pages 37–46, January 1982.

[3] Miloš D. Ercegovac. On-line arithmetic: an overview. In *Proceedings of the SPIE, Real Time Signal Processing VII*, volume 495, pages 86–93, San Diego, CA, August 1984.

[4] Mary Jane Irwin and Robert Michael Owens. Digit-pipelined arithmetic as illustrated by the Paste-Up system: A tutorial. *IEEE Computer*, pages 61–73, April 1987.

[5] Osaaki Watanuki. Floating-point on-line arithmetic for highly concurrent digit-serial computation: application to mesh problems. Technical Report CSD810529, Computer Science Department, UCLA, May 1981.

[6] Kishor S. Trivedi and Miloš D. Ercegovac. On-line algorithms for division and multiplication. *IEEE Transactions on Computers*, C-26(7):681–687, July 1977.

[7] Osaaki Watanuki and Miloš D. Ercegovac. Floating-point on-line arithmetic: algorithms. In *Proceedings of the 5th Symposium on Computer Arithmetic*, pages 81–86, 1981.

[8] V.G. Oklobdzija and Miloš D. Ercegovac. An on-line square root algorithm. *IEEE Transactions on Computers*, C-31(1):70–75, Jan 1982.

[9] Miloš D. Ercegovac. A general hardware-oriented method for evaluation of functions and computations in a digital computer. *IEEE Transactions on Computers*, C-26(7):667–680, July 1977.

[10] Algirdas Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, EC-10:389–400, September 1961.

[11] Kishor S. Trivedi and Joseph G. Rusnak. Higher radix on-line division. In *Proceedings of the Fourth Symposium on Computer Arithmetic*, pages 164–174, Santa Monica, CA, Oct 1978.

[12] C.S. Raghavendra and Miloš D. Ercegovac. A simulator for on-line arithmetic. In *Proceedings of 5th Symposium on Computer Arithmetic*, pages 92–98, May 1981.

[13] Osaaki Watanuki and Miloš D. Ercegovac. Floating-point on-line arithmetic: error analysis. In *Proceedings of the 5th Symposium on Computer Arithmetic*, pages 87–91, 1981.

[14] Osaaki Watanuki and Miloš D. Ercegovac. Error analysis of certain floating-point on-line algorithms. *IEEE Transactions on Computers*, C-32(4):352–358, April 1983.

[15] Abdolali Gorji-Sinaki and Miloš D. Ercegovac. Design of a digit-slice on-line arithmetic unit. In *Proceedings of 5th Symposium on Computer Arithmetic*, pages 72–80, May 1981.

[16] Aksenti L. Grnarov and Miloš D. Ercegovac. On-line multiplicative normalization. In *Proceedings of IEEE 1983 Sixth Symposium on Computer Arithmetic*, pages 151–155, 1983.

[17] Dean Michael Tullsen and Miloš D. Ercegovac. Design and VLSI implementation of an on-line algorithm. In *Proceedings of the SPIE, vol. 698, Real Time Signal Processing IX*, pages 92–99, San Diego, CA, August 1986.

[18] Miloš D. Ercegovac and A.L. Grnarov. On the performance of on-line arithmetic. In *Proceedings of the 1980 International Conference on Parallel Processing*, pages 55–62. Ohio State University and IEEE Computer Society, August 1980.

[19] Miloš D. Ercegovac and Tomás Lang. On-line scheme for computing rotation factors. *Journal of Parallel and Distributed Computing*, 5(3):209–227, 1988.

[20] Stephen G. Faris. A very large scale integration design of an online algorithm for the computation of rotation factors. Master's thesis, University of California, Los Angeles, 1989.

[21] Miloš D. Ercegovac and Tomás Lang. On-line arithmetic: A design methodology and applications in digital signal processing. *IEEE Acoustics, Speech, and Signal Processing Society Workshop on VLSI Signal Processing*, pages 252–263, November 1988.

[22] Robert Michael Owens. Compound algorithms for digit online arithmetic. In *Proceedings of IEEE 5th Symposium on Computer Arithmetic*, 1981.

[23] B.G. DeLugish. *A Class of Algorithms for Automatic Evaluation of Certain Elementary Functions in a Binary Computer*. PhD thesis, University of Illinois at Urbana-Champaign, june 1970.

[24] Peter Denyer and David Renshaw. *VLSI Signal Processing: A Bit-Serial Approach*. Addison-Wesley Publishing Company, 1985.

[25] Mary Jane Irwin. *An Arithmetic Unit for On-Line Arithmetic Computation*. PhD thesis, University of Illinoins, 1977.

[26] Miloš D. Ercegovac. An on-line square rooting algorithm. In *Proceedings of the 4th Symposium on Computer Arithmetic*, pages 183–189, 1978.

[27] Paul K.-G. Tu and Miloš D. Ercegovac. A radix-4 on-line division algorithm. In *Proceedings of the 8th Symposium on Computer Arithmetic*, pages 181–187, 1987.

[28] Abdolali Gorji-Sinaki. *Error-coded algorithms for on-line arithmetic*. PhD thesis, University of California, Los Angeles, 1981.

[29] Miloš D. Ercegovac. *A General Method for Evaluation of Functions and Computations in a Digital Computer*. PhD thesis, University of Illinois at Urbana-Champaign, 1975.

[30] Miloš D. Ercegovac and Tomás Lang. On-the-fly conversion of redundant into conventional representations. *IEEE Transactions on Computers*, C-36(7):895–897, July 1987.

[31] VIEWlogic Systems, Inc., 313 Boston Post Road West, Marlboro, Massachusetts 01752. *WORKVIEW Manual Set, Release 3.0*, June 1988.

[32] VIEWlogic Systems, Inc., 313 Boston Post Road West, Marlboro, Massachusetts 01752. *LSI Design Kit, Release 1.0*, September 1987.

[33] Dean Michael Tullsen. A very large scale integration implementation of an on-line arithmetic unit. Master's thesis, University of California, Los Angeles, 1986.

[34] Miloš D. Ercegovac. A higher-radix division with simple selection of quotient digits. In *Proceedings of IEEE 1983 Sixth Symposium on Computer Arithmetic*, pages 94–98, 1983.

[35] Miloš D. Ercegovac and Tomás Lang. A division algorithm with prediction of quotient digits. In *Proceedings of the Seventh Symposium on Computer Arithmetic*, Urbana, Illinois, 1985.

[36] Miloš D. Ercegovac and Tomás Lang. Simple radix-4 division with divisor scaling. Technical Report CSD 870015, UCLA Computer Science Department, March 1987.

[37] J. M. Speiser and H. J. Whitehouse. A review of signal processing with systolic arrays. In *Proceedings of the SPIE, vol. 431, Real Time Signal Processing VI*, pages 2–6, August 1983.

[38] Franklin T. Luk Richard P. Brent and Charles Van Loan. Computation of the singular value decomposition using mesh-connected processors. *Journal of VLSI and Computer Systems*, 1(3):242–270, 1985.

[39] Haesun Park. *On the equivalence and convergence of parallel Jacobi SVD algorithms*. PhD thesis, Cornell University, August 1987.

[40] Richard P. Brent and Franklin T. Luk. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM Journal on Scientific and Statistical Computing*, 6(1):69–84, January 1985.

[41] LSI Logic Corporation, 1551 McCarthy Blvd, Milpitas CA 95035. *Databook and Design Manual for HCMOS Macrocells and Macrofunctions*, October 1986.

[42] George S. Taylor. Radix 16 SRT dividers with overlapped quotient selection stages. In *Proceedings of the 7th Symposium on Computer Arithmetic*, pages 64–71, 1985.

Appendix A

Derivation of the scaled error bound

In deriving on-line algorithms, the scaled error bound is often the solution of the difference inequality-equation,

$$r\varepsilon_{j-1} - \varepsilon_j \quad \leq \quad \alpha X[j] + \beta r^{-j} + \gamma$$

where α, β and γ are constants. Assuming that the solution is of the form

$$\varepsilon_j \quad = \quad AX[j] + Br^{-j} + C$$

we have

$$\begin{aligned} r\varepsilon_{j-1} &= rAX[j-1] + Br^{-j+2} + rC \\ r\varepsilon_{j-1} - \varepsilon_j &= AX[j](r-1) + (B(r^2-1) - Ax_{j+\delta-1}r^{-\delta+2})r^{-j} + (r-1)C \end{aligned}$$

which leads to

$$\begin{aligned} AX[j](r-1) + (B(r^2-1) - Ax_{j+\delta-1}r^{-\delta+2})r^{-j} + (r-1)C \leq \\ \alpha X[j] + \beta r^{-j} + \gamma \end{aligned}$$

Hence

$$\begin{aligned} A &\leq \frac{\alpha}{r-1} \\ B &\leq \frac{\beta}{r^2-1} + \frac{\alpha x_{j+\delta-1} r^{-\delta+2}}{(r-1)^2(r+1)} \\ C &\leq \frac{\gamma}{r-1} \end{aligned}$$

The expressions for A and C contain constants only, so we may take the equal sign. For the value of B, δ is the on-line delay of the variable x and $x_{j+\delta-1}$ is the next digit value of x at step j. When x is the output of the on-line computation, we usually have $\delta = 1$. If $x_{j+\delta-1}$ is not given then its worst case value should be assumed to satisfy the inequality. The scaled error bound is given as

$$\varepsilon_j = \frac{\alpha}{r-1}X[j] + \left(\frac{\beta}{r^2-1} + \frac{\alpha x_{j+\delta-1} r^{-\delta+2}}{(r-1)^2(r+1)}\right) r^{-j} + \frac{\gamma}{r-1} \qquad \text{(A.1)}$$

Appendix B

Parameter relations for multi-operation on-line algorithms

We now derive the relations between the input/output digits set parameters and the performance parameters of multi-operation on-line algorithms for different types of arithmetic expressions.

B.1 Type 1 computations

For Type 1 computations we have

$$Z = \sum_{i=1}^{p} X^{(i)} \cdot Y^{(i)}$$

where $X^{(i)}$ and $Y^{(i)}$ are single numbers. The partial residual function and on-line recurrence expression are

$$\begin{aligned} P[j] &= P[j-1] - z_j r^{-j} \\ &\quad + \sum_{i=1}^{p} \left(x^{(i)}_{j+\delta_x-1} Y^{(i)}[j] r^{-j-\delta_x+1} + y^{(i)}_{j+\delta_y-1} X^{(i)}[j-1] r^{-j-\delta_y+1} \right) \\ A[j] &= r(A[j-1] - z_{j-1}) \\ &\quad + \sum_{i=1}^{p} \left(x^{(i)}_{j+\delta_x-1} Y^{(i)}[j] r^{-\delta_x+1} + y^{(i)}_{j+\delta_y-1} X^{(i)}[j-1] r^{-\delta_y+1} \right) \end{aligned}$$

with

$$\mathcal{H}_1[j] = \sum_{i=1}^{p} \left(x^{(i)}_{j+\delta_x-1} Y^{(i)}[j] r^{-\delta_x+1} + y^{(i)}_{j+\delta_y-1} X^{(i)}[j-1] r^{-\delta_y+1} \right)$$

$$\mathcal{H}_2[j] \;=\; -1$$

Then from (3.17) we get

$$\begin{cases} \varepsilon_j \le \dfrac{-\alpha_z}{r-1} + \dfrac{\min(\mathcal{H}_1[j])}{r-1} \\ \varepsilon_j \le \dfrac{\beta_z}{r-1} - \dfrac{\max(\mathcal{H}_1[j])}{r-1} \end{cases} \tag{B.1}$$

Assuming the two relations in (B.1) give the same upper bound which is taken as the value for ε_j, from expressions (2.25) and (2.26) we get the digit selection intervals for Type 1 computations as

$$\begin{cases} \widehat{L_k} \ge k - \dfrac{\beta_z}{r-1} + \dfrac{\max(\mathcal{H}_1[j])}{r-1} + \xi \\ \widehat{U_k} \le k + \dfrac{\beta_z}{r-1} - \dfrac{\max(\mathcal{H}_1[j])}{r-1} - \eta \end{cases} \qquad k \in \mathcal{D}_z$$

Since $\mathcal{H}_2[j]$ is constant, expression (2.31) becomes

$$\frac{\beta_z}{r-1} - \frac{\max(\mathcal{H}_1[j])}{r-1} \;\ge\; 2^{-1} + \frac{\xi+\eta}{2} - 2^{-\psi_A-1} \tag{B.2}$$

If the on-line recurrence is implemented in carry-save form, then $\eta = 2^{-\psi_A+1}$ and $\xi = 0$, and

$$\frac{\beta_z}{r-1} - \frac{\max(\mathcal{H}_1[j])}{r-1} \;\ge\; 2^{-1} + 2^{-\psi_A-1} \tag{B.3}$$

From (3.16) we have the bounds of $A[j]$,

$$\begin{cases} A[j] \le \beta_z + \dfrac{\beta_z}{r-1} - \dfrac{\max(\mathcal{H}_1[j])}{r-1} \\ A[j] \ge \alpha_z + \dfrac{\alpha_z}{r-1} - \dfrac{\min(\mathcal{H}_1[j])}{r-1} \end{cases} \tag{B.4}$$

Relations (B.1), (B.2) and (B.4) can be used to determine values of the design and interface parameters.

Let $p = 1$ and we get the expressions for on-line multiplication

$$\mathcal{H}_1[j] \;=\; x_{j+\delta_x-1}Y[j-1]r^{-\delta_x+1} + y_{j+\delta_y-1}X[j]r^{-\delta_y+1}$$

and from (B.3) we have

$$\frac{\beta_z}{r-1}\left(1 - \frac{\beta_x}{\beta_z}\cdot\max(Y)\cdot r^{-\delta_x+1} - \frac{\beta_y}{\beta_z}\cdot\max(X)\cdot r^{-\delta_y+1}\right) \ge 2^{-1} + 2^{-\psi_A-1}$$

Hence for a given radix, larger values of the ratio $\beta_z/(r-1)$ can result in smaller on-line delays or smaller value of ψ_A. For example, suppose that $X < 1$, $Y < 1$, $r = 4$, $\alpha_x = \alpha_y = -2$, $\beta_x = \beta_y = 2$, and let $\delta_x = \delta_y = \delta$. If $-\alpha_z = \beta_z = 2$, we have

$$\frac{2}{3}(1 - 2 \cdot 4^{-\delta+1}) \;\geq\; 2^{-1} + 2^{-\psi_A - 1}$$

and we may choose $\delta = 3$, $\psi_A = 3$ or $\delta = 4$, $\psi_A = 2$. Now let $-\alpha_z = \beta_z = 3$, then

$$1 - \frac{4}{3} \cdot 4^{-\delta+1} \;\geq\; 2^{-1} + 2^{-\psi_A - 1}$$

and we may have $\delta = 2$, $\psi_A = 2$ or $\delta = 3$, $\psi_A = 1$.

Let $p = 2$ and $Y^{(1)} = X^{(2)} = 1$, then we have the expressions for on-line addition

$$\mathcal{H}_1[j] \;=\; x_{j+\delta_x-1} r^{-\delta_x+1} + y_{j+\delta_y-1} r^{-\delta_y+1}$$

$$\frac{\beta_z}{r-1}\left(1 - \frac{\beta_x}{\beta_z} r^{-\delta_x+1} - \frac{\beta_y}{\beta_z} r^{-\delta_y+1}\right) \geq 2^{-1} + 2^{-\psi_A - 1}$$

Similar to multiplication, larger values of $\beta_z/(r-1)$ result in smaller values of the number of input bits for the digit selection function and the on-line delay.

B.2 Type 2 computations

For Type 2 computations we have

$$Z \;=\; \frac{\mathcal{A}}{\mathcal{B}}$$

where

$$\mathcal{A} \;\equiv\; \sum_{i=1}^{p} X^{(i)} \cdot Y^{(i)}$$
$$\mathcal{B} \;\equiv\; \sum_{i=1}^{q} V^{(i)}$$

and $X^{(i)}$, $Y^{(i)}$ and $V^{(i)}$ are single numbers. Assume that $\mathcal{B} > 0$. The partial residual function and on-line recurrence expression are

$$P[j] \;=\; P[j-1] - z_j \mathcal{B}[j] r^{-j}$$

$$
\begin{aligned}
&+\sum_{i=1}^{p}\left(x^{(i)}_{j+\delta_x-1}Y^{(i)}[j]r^{-j-\delta_x+1}+y^{(i)}_{j+\delta_y-1}X^{(i)}[j-1]r^{-j-\delta_y+1}\right)\\
&-\sum_{i=1}^{q}v^{(i)}_{j+\delta_v-1}Z[j-1]r^{-j-\delta_v+1}\\
A[j] \;=\;& r(A[j-1]-z_{j-1}\mathcal{B}[j-1])\\
&+\sum_{i=1}^{p}\left(x^{(i)}_{j+\delta_x-1}Y^{(i)}[j]r^{-\delta_x+1}+y^{(i)}_{j+\delta_y-1}X^{(i)}[j-1]r^{-\delta_y+1}\right)\\
&-\sum_{i=1}^{q}v^{(i)}_{j+\delta_v-1}Z[j-1]r^{-\delta_v+1}
\end{aligned}
$$

with

$$
\begin{aligned}
\mathcal{H}_1[j] \;=\;& \sum_{i=1}^{p}\left(x^{(i)}_{j+\delta_x-1}Y^{(i)}[j]r^{-\delta_x+1}+y^{(i)}_{j+\delta_y-1}X^{(i)}[j-1]r^{-\delta_y+1}\right)\\
&-\sum_{i=1}^{q}v^{(i)}_{j+\delta_v-1}Z[j-1]r^{-\delta_v+1}\\
\mathcal{H}_2[j] \;=\;& -\mathcal{B}[j]
\end{aligned}
$$

Then from (3.17) we get

$$
\begin{cases}
r\varepsilon_{j-1}-\varepsilon_j(\alpha_z)\le -\alpha_z\mathcal{B}[j]+\min(\mathcal{H}_1[j])\\
r\varepsilon_{j-1}-\varepsilon_j(\beta_z)\le \beta_z\mathcal{B}[j]-\max(\mathcal{H}_1[j])
\end{cases}
\tag{B.5}
$$

From the solution described in Appendix A we get

$$
\varepsilon_j \;=\; \begin{cases}
\dfrac{-\alpha_z}{r-1}\mathcal{B}[j]+\dfrac{-\alpha_z\sum\beta_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}+\dfrac{\min(\mathcal{H}_1[j])}{r-1}\\
\dfrac{\beta_z}{r-1}\mathcal{B}[j]+\dfrac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}-\dfrac{\max(\mathcal{H}_1[j])}{r-1}
\end{cases}
\tag{B.6}
$$

Hence ε_j is a function of $\mathcal{B}[j]$. Assuming the two expressions in (B.6) give the same solution which is taken as the value for ε_j, from expressions (2.25) and (2.26) we get the digit selection intervals for Type 2 computations as

$$
\widehat{L_k} \;\ge\; \begin{cases}
(k-\dfrac{\beta_z}{r-1})\mathcal{B}[j]+(k-\dfrac{\beta_z}{r-1})\nu+\xi\\
\qquad -\dfrac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}+\dfrac{\max(\mathcal{H}_1[j])}{r-1} & k\ge\dfrac{\beta_z}{r-1}\\
(k-\dfrac{\beta_z}{r-1})\mathcal{B}[j]-(k-\dfrac{\beta_z}{r-1})\mu+\xi\\
\qquad -\dfrac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}+\dfrac{\max(\mathcal{H}_1[j])}{r-1} & k<\dfrac{\beta_z}{r-1}
\end{cases}
$$

$$\widehat{U_k} \leq \begin{cases} (k+\frac{\beta_z}{r-1})\mathcal{B}[j]-(k+\frac{\beta_z}{r-1})\mu-\eta & \\ \qquad +\frac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}-\frac{\max(\mathcal{H}_1[j])}{r-1} & k>-\frac{\beta_z}{r-1} \\ (k+\frac{\beta_z}{r-1})\mathcal{B}[j]+(k+\frac{\beta_z}{r-1})\nu-\eta & \\ \qquad +\frac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}-\frac{\max(\mathcal{H}_1[j])}{r-1} & k\leq-\frac{\beta_z}{r-1} \end{cases}$$

From expression (2.31) we have

$$\begin{aligned} &(\frac{\beta_z}{r-1}-\frac{1}{2})\mathcal{B}[j]+\frac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}-\frac{\max(\mathcal{H}_1[j])}{r-1} \\ &\quad\geq\begin{cases} (2k-1)\frac{\mu+\nu}{2}+\frac{\xi+\eta}{2}-2^{-\psi_A-1} & \frac{\beta_z}{r-1}\leq k\leq\rho \\ (\frac{2\beta_z}{r-1}-1)\frac{\mu+\nu}{2}+\frac{\xi+\eta}{2}-2^{-\psi_A-1} & -\frac{\beta_z}{r-1}+1<k<\frac{\beta_z}{r-1} \\ -(2k-1)\frac{\mu+\nu}{2}+\frac{\xi+\eta}{2}-2^{-\psi_A-1} & -\rho+1\leq k\leq-\frac{\beta_z}{r-1}+1 \end{cases} \end{aligned} \quad \text{(B.7)}$$

If $A[j]$ is in carry-save form, $\mathcal{B}[j]$ is in non-redundant form, and the estimates $\widehat{A[j]}$, $\widehat{\mathcal{B}[j]}$ are obtained by truncation, then we have $\eta=2^{-\psi_A+1}$, $\xi=0$, $\mu=2^{-\psi_B}$ and $\nu=0$, and

$$\begin{aligned} &(\frac{\beta_z}{r-1}-\frac{1}{2})\mathcal{B}[j]+\frac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}-\frac{\max(\mathcal{H}_1[j])}{r-1} \\ &\quad\geq\begin{cases} (2k-1)2^{-\psi_B}+2^{-\psi_A-1} & \frac{\beta_z}{r-1}\leq k\leq\rho \\ (\frac{2\beta_z}{r-1}-1)2^{-\psi_B}+2^{-\psi_A-1} & -\frac{\beta_z}{r-1}+1<k<\frac{\beta_z}{r-1} \\ -(2k-1)2^{-\psi_B}+2^{-\psi_A-1} & -\rho+1\leq k\leq-\frac{\beta_z}{r-1}+1 \end{cases} \end{aligned} \quad \text{(B.8)}$$

From (3.16) we get the bounds of $A[j]$,

$$\begin{cases} A[j]\leq(\beta_z+\frac{\beta_z}{r-1})\mathcal{B}[j]+\frac{\beta_z\sum\alpha_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}-\frac{\max(\mathcal{H}_1[j])}{r-1} \\ A[j]\geq(\alpha_z+\frac{\beta_z}{r-1})\mathcal{B}[j]+\frac{-\alpha_z\sum\beta_v r^{-\delta_v+1}}{(r-1)^2(r+1)}r^{-j+1}-\frac{\min(\mathcal{H}_1[j])}{r-1} \end{cases} \quad \text{(B.9)}$$

Relations (B.6), (B.7) and (B.9) can be used to determine the design and interface parameters.

Let $p=q=1$ and we have the expressions for on-line division

$$\begin{aligned} \mathcal{H}_1[j] &= n_{j+\delta_n-1}r^{-\delta_n+1}-d_{j+\delta_d-1}Q[j-1]r^{-\delta_d+1} \\ \mathcal{H}_2[j] &= -D[j] \end{aligned}$$

and the parameter relation (B.7) becomes

$$\frac{\beta_q}{r-1}\left(D[j]+\frac{\alpha_d}{r^2-1}r^{-j-\delta_d+2}-\frac{\beta_n}{\beta_q}r^{-\delta_n+1}+\frac{\alpha_d\cdot\max\left(Q[j-1]\right)}{\beta_q}r^{-\delta_d+1}\right)$$
$$\geq\begin{cases}\dfrac{D[j]}{2}+(2k-1)2^{-\psi_D}+2^{-\psi_A-1} & \dfrac{\beta_q}{r-1}\leq k\leq\rho\\ \dfrac{D[j]}{2}+(\dfrac{2\beta_q}{r-1}-1)2^{-\psi_D}+2^{-\psi_A-1} & -\dfrac{\beta_q}{r-1}+1<k<\dfrac{\beta_q}{r-1}\\ \dfrac{D[j]}{2}-(2k-1)2^{-\psi_D}+2^{-\psi_A-1} & -\rho+1\leq k\leq-\dfrac{\beta_q}{r-1}+1\end{cases}$$

For a given radix, a larger value for the ratio $\beta_q/(r-1)$ may result in smaller values of the on-line delay or ψ_A. As an example, suppose $r=4$ and $\alpha_n=\alpha_d=-2$, $\beta_n=\beta_d=2$, and let $\delta_n=\delta_d=\delta$. Assume that $D\in[4^{-1},1)$ and $Q[j]<1$. If $-\alpha_q=\beta_q=2$, we have

$$\frac{2}{3}(D[j]-\frac{2}{15}\cdot4^{-\delta+2-j}-2\cdot4^{-\delta+1})$$
$$\geq\begin{cases}\dfrac{D[j]}{2}+(2k-1)2^{-\psi_D}+2^{-\psi_A-1}, & k=1,2\\ \dfrac{D[j]}{2}-(2k-1)2^{-\psi_D}+2^{-\psi_A-1}, & k=-2,-1,0\end{cases}$$

and we may choose $\delta=4$, $\phi_A=3$, $\psi_A=4$, $\psi_D=11$. Now let $-\alpha_q=\beta_q=3$, then

$$D[j]-\frac{2}{15}\cdot4^{-\delta+2-j}-\frac{4}{3}\cdot4^{-\delta+1}$$
$$\geq\begin{cases}\dfrac{D[j]}{2}+(2k-1)2^{-\psi_D}+2^{-\psi_A-1}, & k=1,2,3\\ \dfrac{D[j]}{2}-(2k-1)2^{-\psi_D}+2^{-\psi_A-1}, & k=-3,-2,-1,0\end{cases}$$

and we may have $\delta=4$, $\phi_A=3$, $\psi_A=3$, $\psi_D=8$.

B.3 Type 3 computations

For Type 3 computations we have

$$Z = \sqrt{\mathcal{A}} \tag{B.10}$$

where

$$\mathcal{A} \equiv \sum_{i=1}^{p} X^{(i)}\cdot Y^{(i)}$$

and $X^{(i)}$ and $Y^{(i)}$ denote single numbers. The partial residual function and on-line recurrence expression are

$$\begin{aligned} P[j] &= P[j-1] - z_j(Z[j] + Z[j-1])r^{-j} + \\ &\quad + \sum_{i=1}^{p} \left(x^{(i)}_{j+\delta_x-1} Y^{(i)}[j] r^{-j-\delta_x+1} + y^{(i)}_{j+\delta_y-1} X^{(i)}[j-1] r^{-j-\delta_y+1} \right) \\ A[j] &= r(A[j-1] - z_{j-1}(Z[j-1] + Z[j-2])) + \\ &\quad + \sum_{i=1}^{p} \left(x^{(i)}_{j+\delta_x-1} Y^{(i)}[j] r^{-\delta_x+1} + y^{(i)}_{j+\delta_y-1} X^{(i)}[j-1] r^{-\delta_y+1} \right) \end{aligned}$$

with

$$\begin{aligned} \mathcal{H}_1[j] &= \sum_{i=1}^{p} \left(x^{(i)}_{j+\delta_x-1} Y^{(i)}[j] r^{-\delta_x+1} + y^{(i)}_{j+\delta_y-1} X^{(i)}[j-1] r^{-\delta_y+1} \right) \\ \mathcal{H}_2[j] &= -(Z[j] + Z[j-1]) \end{aligned}$$

Then from (3.17) we get

$$\begin{cases} r\varepsilon_{j-1} - \varepsilon_j(\alpha_z) \le -\alpha_z(Z[j] + Z[j-1]) + \min(\mathcal{H}_1[j]) \\ r\varepsilon_{j-1} - \varepsilon_j(\beta_z) \le \beta_z(Z[j] + Z[j-1]) - \max(\mathcal{H}_1[j]) \end{cases}$$

From the solution described in Appendix A we get

$$\varepsilon_j = \begin{cases} \dfrac{-\alpha_z}{r-1}(Z[j] + Z[j-1]) - \left(\dfrac{\alpha_z}{r-1}\right)^2 r^{-j+1} + \dfrac{\min(\mathcal{H}_1[j])}{r-1} \\ \dfrac{\beta_z}{r-1}(Z[j] + Z[j-1]) + \left(\dfrac{\beta_z}{r-1}\right)^2 r^{-j+1} - \dfrac{\max(\mathcal{H}_1[j])}{r-1} \end{cases} \tag{B.11}$$

Hence ε_j is a function of $Z[j] + Z[j-1]$. Assuming that the first expression in (B.11) gives the solution for ε_j, from expressions (2.25) and (2.26) we get the digit selection intervals for Type 3 computations as

$$\widehat{L_k} \ge \begin{cases} (k - \dfrac{-\alpha_z}{r-1})\overline{(Z[j] + Z[j-1])} + (k - \dfrac{-\alpha_z}{r-1})\nu + \xi \\ \qquad + \left(\dfrac{\alpha_z}{r-1}\right)^2 r^{-j+1} - \dfrac{\min(\mathcal{H}_1[j])}{r-1}, & k \ge \dfrac{-\alpha_z}{r-1} \\ (k - \dfrac{-\alpha_z}{r-1})\overline{(Z[j] + Z[j-1])} - (k - \dfrac{-\alpha_z}{r-1})\mu + \xi \\ \qquad + \left(\dfrac{\alpha_z}{r-1}\right)^2 r^{-j+1} - \dfrac{\min(\mathcal{H}_1[j])}{r-1}, & k < \dfrac{-\alpha_z}{r-1} \end{cases}$$

$$\widehat{U_k} \le \begin{cases} (k + \dfrac{-\alpha_z}{r-1})\overline{(Z[j] + Z[j-1])} - (k + \dfrac{-\alpha_z}{r-1})\mu - \eta \\ \qquad - \left(\dfrac{\alpha_z}{r-1}\right)^2 r^{-j+1} + \dfrac{\min(\mathcal{H}_1[j])}{r-1}, & k > -\dfrac{-\alpha_z}{r-1} \\ (k + \dfrac{-\alpha_z}{r-1})\overline{(Z[j] + Z[j-1])} + (k + \dfrac{-\alpha_z}{r-1})\nu - \eta \\ \qquad - \left(\dfrac{\alpha_z}{r-1}\right)^2 r^{-j+1} + \dfrac{\min(\mathcal{H}_1[j])}{r-1}, & k \le -\dfrac{-\alpha_z}{r-1} \end{cases}$$

where $\overline{(Z[j]+Z[j-1])}$ denotes low precision estimate of $(Z[j]+Z[j-1])$. From expression (2.31) we have

$$\left(\frac{-\alpha_z}{r-1}-\frac{1}{2}\right)(Z[j]+Z[j-1])-\left(\frac{\alpha_z}{r-1}\right)^2 r^{-j+1}+\frac{\min(\mathcal{H}_1[j])}{r-1}$$
$$\geq\begin{cases}(2k-1)\dfrac{\mu+\nu}{2}+\dfrac{\xi+\eta}{2}-2^{-\psi_A-1}, & \dfrac{-\alpha_z}{r-1}\leq k\leq\rho\\ (\dfrac{-2\alpha_z}{r-1}-1)\dfrac{\mu+\nu}{2}+\dfrac{\xi+\eta}{2}-2^{-\psi_A-1}, & -\dfrac{-\alpha_z}{r-1}+1<k<\dfrac{-\alpha_z}{r-1}\\ -(2k-1)\dfrac{\mu+\nu}{2}+\dfrac{\xi+\eta}{2}-2^{-\psi_A-1}, & -\rho+1\leq k\leq-\dfrac{-\alpha_z}{r-1}+1\end{cases}\quad\text{(B.12)}$$

If $A[j]$ is in carry-save form, $(Z[j]+Z[j-1])$ is in non-redundant form, and the estimates $\widehat{A[j]}$, $\overline{Z[j]+Z[j-1]}$ are obtained by truncation, then we have $\eta=2^{-\psi_A+1}$, $\xi=0$, $\mu=2^{-\psi_{2Z}}$ and $\nu=0$, and

$$\left(\frac{-\alpha_z}{r-1}-\frac{1}{2}\right)(Z[j]+Z[j-1])-\left(\frac{\alpha_z}{r-1}\right)^2 r^{-j+1}+\frac{\min(\mathcal{H}_1[j])}{r-1}$$
$$\geq\begin{cases}(2k-1)2^{-\psi_{2Z}-1}+2^{-\psi_A-1}, & \dfrac{-\alpha_z}{r-1}\leq k\leq\rho\\ (\dfrac{-2\alpha_z}{r-1}-1)2^{-\psi_{2Z}-1}+2^{-\psi_A-1}, & -\dfrac{-\alpha_z}{r-1}+1<k<\dfrac{-\alpha_z}{r-1}\\ -(2k-1)2^{-\psi_{2Z}-1}+2^{-\psi_A-1}, & -\rho+1\leq k\leq-\dfrac{-\alpha_z}{r-1}+1\end{cases}$$

From (3.16) we have the boundary values of $A[j]$,

$$\begin{cases}A[j]\leq(\beta_z+\dfrac{\beta_z}{r-1})(Z[j]+Z[j-1])+\left(\dfrac{\beta_z}{r-1}\right)^2 r^{-j+1}-\dfrac{\max(\mathcal{H}_1[j])}{r-1}\\ A[j]\geq(\alpha_z-\dfrac{-\alpha_z}{r-1})(Z[j]+Z[j-1])+\left(\dfrac{\alpha_z}{r-1}\right)^2 r^{-j+1}-\dfrac{\min(\mathcal{H}_1[j])}{r-1}\end{cases}\quad\text{(B.13)}$$

Relations (B.11), (B.12) and (B.13) can be used to determine the design and interface parameters.

Let $p=1$, $Y^{(1)}=1$ and we get the expressions for the on-line square root algorithm

$$\begin{aligned}\mathcal{H}_1[j] &= x_{j+\delta_x-1}r^{-\delta_x+1}\\ \min(\mathcal{H}_1[j]) &= \alpha_x r^{-\delta_x+1}\\ \max(\mathcal{H}_1[j]) &= \beta_x r^{-\delta_x+1}\end{aligned}$$

and the parameter relation is

$$\frac{-\alpha_z}{r-1}\left(Z[j]+Z[j-1]-\frac{\alpha_z}{r-1}r^{-j+1}+\frac{\alpha_x}{\alpha_z}r^{-\delta_x+1}\right)$$

$$\geq \begin{cases} \dfrac{Z[j]+Z[j-1]}{2} + (2k-1)2^{-\psi_{2Z}} + 2^{-\psi_A-1}, & \dfrac{-\alpha_z}{r-1} \leq k \leq \rho \\ \dfrac{Z[j]+Z[j-1]}{2} + (\dfrac{-2\alpha_z}{r-1} - 1)2^{-\psi_{2Z}} + 2^{-\psi_A-1}, & -\dfrac{-\alpha_z}{r-1} + 1 < k < \dfrac{-\alpha_z}{r-1} \\ \dfrac{Z[j]+Z[j-1]}{2} + (2k-1)2^{-\psi_{2Z}} + 2^{-\psi_A-1}, & -\rho + 1 \leq k \leq -\dfrac{-\alpha_z}{r-1} + 1 \end{cases}$$

Appendix C

Derivation of the selection function for the radix-2 on-line square root algorithm

In this appendix we derive the selection function for square root for $j = 1, 2, 3$. We have

$$\begin{aligned} A[j] &= 2(A[j-1] - y_{j-1}(Y[j-1] + Y[j-2])) + x_{next}2^{-4} \\ &= 2R[j-1] + x_{next}2^{-4} \\ R[j] &= A[j] - y_j(Y[j] + Y[j-1]) \\ U_k &= k \cdot 2Y[j-1] + k^2 2^{-j} + \varepsilon_j \\ L_k &= k \cdot 2Y[j-1] + k^2 2^{-j} - \varepsilon_j \end{aligned}$$

For $j \leq 4$ and $\psi_A = 4$, the truncation error for $\widehat{A[j]}$ is zero and we have

$$\widehat{A[j]} = A[j]$$

For $j = 1$ we have

$$\begin{aligned} A[1] &= 2 \cdot X[1] \cdot 2^{-1} \in (2^{-3}, 1) \\ R[1] &= A[1] - y_1 2^{-1} \\ U_1 &= 2^{-1} + \varepsilon_1 \\ L_1 &= 2^{-1} - \varepsilon_1 \\ U_0 &= \varepsilon_1 \\ L_0 &= -\varepsilon_1 \end{aligned}$$

To satisfy the containment condition, we have

$$U_1 \geq 1 \quad \longrightarrow \quad \varepsilon_1 \geq 2^{-1}$$

To satisfy the continuity condition, we have

$$U_0 - L_1 \geq 0 \quad \longrightarrow \quad \varepsilon_1 \geq 2^{-2}$$

We choose

$$I_1 \quad = \quad 2^{-2}$$

Then we have

$$\begin{array}{ll} y_1 = 1, & A[1] \in [2^{-2}, 1) \\ & R[1] \in [-2^{-2}, 2^{-1}) \\ y_1 = 0, & A[1] \in (2^{-3}, 2^{-2}) \\ & R[1] \in (2^{-3}, 2^{-2}) \end{array}$$

For $j = 2$, we have

$$\begin{aligned} A[2] &= 2 \cdot R[1] + x_{next} 2^{-4} \\ R[2] &= A[2] - y_2(Y[2] + Y[1]) \\ U_1 &= 2Y[1] + 2^{-2} + \varepsilon_2 \\ L_1 &= 2Y[1] + 2^{-2} - \varepsilon_2 \\ U_0 &= \varepsilon_2 \\ L_0 &= -\varepsilon_2 \\ U_{-1} &= -2Y[1] + 2^{-2} + \varepsilon_2 \\ L_{-1} &= -2Y[1] + 2^{-2} - \varepsilon_2 \end{aligned}$$

1. For $y_1 = 0$, we have

$$\begin{aligned} R[1] &\in (2^{-3}, 2^{-2}) \\ A[2] &\in (2^{-3} + 2^{-4}, 2^{-1} + 2^{-4}) \\ Y[1] &= 0 \\ U_1 &= 2^{-2} + \varepsilon_2 \\ L_1 &= 2^{-2} - \varepsilon_2 \end{aligned}$$

Then we have

$$\begin{array}{ll} y_2 = 1, & A[2] \in [2^{-3} + 2^{-4}, 2^{-1} + 2^{-4}) \\ & Y[2] + Y[1] = 2^{-2} \\ & R[2] \in [-2^{-4}, 2^{-2} + 2^{-4}) \end{array}$$

2. For $y_1 = 1$,

$$\begin{aligned} R[1] &\in [-2^{-2}, 2^{-1}) \\ A[2] &\in [-2^{-1} - 2^{-4}, 1 + 2^{-4}) \\ Y[1] &= 2^{-1} \\ U_1 &= 1 + 2^{-2} + \varepsilon_2 \\ L_1 &= 1 + 2^{-2} - \varepsilon_2 \\ U_0 &= \varepsilon_2 \\ L_0 &= -\varepsilon_2 \\ U_{-1} &= -1 + 2^{-2} + \varepsilon_2 \\ L_{-1} &= -1 + 2^{-2} - \varepsilon_2 \end{aligned}$$

To satisfy the containment condition, we have

$$\begin{aligned} U_1 \geq 1 + 2^{-4} &\longrightarrow \varepsilon_2 \geq -2^{-3} - 2^{-4} \\ L_{-1} \leq -2^{-1} - 2^{-4} &\longrightarrow \varepsilon_2 \geq -2^{-3} - 2^{-4} \end{aligned}$$

To satisfy the continuity condition, we have

$$U_0 - L_1 \geq 0 \longrightarrow \varepsilon_2 \geq 2^{-1} + 2^{-3}$$

We choose

$$I_1 = 2^{-3} + 2^{-4}$$

which allows $y_2 = 1$ for the first case. Then we have

$$\begin{aligned} y_2 = 1, \quad & A[2] \in [2^{-3} + 2^{-4}, 1 + 2^{-4}) \\ & Y[2] + Y[1] = 1 + 2^{-2} \\ & R[2] \in [-1 - 2^{-4}, -2^{-3} - 2^{-4}) \\ y_2 = 0, \quad & A[2] \in [-2^{-1} - 2^{-4}, 2^{-3} + 2^{-4}) \\ & Y[2] + Y[1] = 1 \\ & R[2] \in (-2^{-1} - 2^{-4}, 2^{-3} + 2^{-4}) \end{aligned}$$

For $j = 3$, we have

$$\begin{aligned} A[3] &= 2 \cdot R[2] + x_{next} 2^{-4} \\ R[3] &= A[3] - y_3(Y[3] + Y[2]) \\ U_1 &= 2Y[2] + 2^{-3} + \varepsilon_3 \end{aligned}$$

$$
\begin{aligned}
L_1 &= 2Y[2] + 2^{-3} - \varepsilon_3 \\
U_0 &= \varepsilon_3 \\
L_0 &= -\varepsilon_3 \\
U_{-1} &= -2Y[2] + 2^{-3} + \varepsilon_3 \\
L_{-1} &= -2Y[2] + 2^{-3} - \varepsilon_3
\end{aligned}
$$

1. For $y_1 = 1$, $y_2 = 1$, we have

$$
\begin{aligned}
R[2] &\in [-1 - 2^{-4}, -2^{-3} - 2^{-4}) \\
A[3] &\in [-2 - 2^{-3} - 2^{-4}, -2^{-2} - 2^{-4}) \\
Y[2] &= 2^{-1} + 2^{-2} \\
U_0 &= \varepsilon_3 \\
L_0 &= -\varepsilon_3 \\
U_{-1} &= -1 - 2^{-1} + 2^{-3} + \varepsilon_3 \\
L_{-1} &= -1 - 2^{-1} + 2^{-3} - \varepsilon_3
\end{aligned}
$$

   To satisfy the containment condition,

$$
L_{-1} \le -1 - 2^{-2} - 2^{-4} \quad \longrightarrow \quad \varepsilon_3 \ge -2^{-4}
$$

   To satisfy the continuity condition,

$$
U_{-1} - L_0 \ge 0 \quad \longrightarrow \quad \varepsilon_3 \ge 2^{-1} + 2^{-3} + 2^{-4}
$$

   We choose

$$
I_0 = -2^{-2} - 2^{-3} - 2^{-4}
$$

   Then we have

$$
\begin{aligned}
y_3 = 0, \qquad & A[3] \in [-2^{-2} - 2^{-3} - 2^{-4}, -2^{-2} - 2^{-4}) \\
& R[3] \in [-2^{-2} - 2^{-3} - 2^{-4}, -2^{-2} - 2^{-4}) \\
y_3 = -1, \qquad & A[3] \in [-2 - 2^{-3} - 2^{-4}, -2^{-2} - 2^{-3} - 2^{-4}) \\
& Y[3] + Y[2] = 1 + 2^{-2} + 2^{-3} \\
& R[3] \in (-2^{-1} - 2^{-2} - 2^{-4}, 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4})
\end{aligned}
$$

2. For $y_1 = 1$, $y_2 = 0$, we have

$$
R[2] \in (-2^{-1} - 2^{-4}, 2^{-1} + 2^{-3})
$$

$$
\begin{aligned}
A[3] &\in (-1-2^{-3}-2^{-4}, 1+2^{-2}+2^{-4}) \\
Y[2] &= 2^{-1} \\
U_1 &= 1+2^{-3}+\varepsilon_3 \\
L_1 &= 1+2^{-3}-\varepsilon_3 \\
U_0 &= \varepsilon_3 \\
L_0 &= -\varepsilon_3 \\
U_{-1} &= -1+2^{-3}+\varepsilon_3 \\
L_{-1} &= -1+2^{-3}-\varepsilon_3
\end{aligned}
$$

To satisfy the containment condition,

$$
\begin{aligned}
U_1 \geq 1+2^{-2}+2^{-4} &\longrightarrow \varepsilon_3 \geq 2^{-3}+2^{-4} \\
L_{-1} \leq -1-2^{-3}-2^{-4} &\longrightarrow \varepsilon_3 \geq 2^{-2}+2^{-4}
\end{aligned}
$$

To satisfy the continuity condition,

$$
\begin{aligned}
U_0 - L_1 \geq 0 &\longrightarrow \varepsilon_3 \geq 2^{-1}+2^{-4} \\
U_{-1} - L_0 \geq 0 &\longrightarrow \varepsilon_3 \geq 2^{-1}-2^{-4}
\end{aligned}
$$

We choose

$$
\begin{aligned}
I_1 &= 2^{-2}+2^{-4} \\
I_0 &= -2^{-2}-2^{-3}-2^{-4}
\end{aligned}
$$

Then we have

$$
\begin{aligned}
y_3 = 1, \quad & A[3] \in [2^{-2}+2^{-4}, 1+2^{-2}+2^{-4}) \\
& Y[3]+Y[2] = 1+2^{-3} \\
& R[3] \in [-2^{-1}-2^{-2}-2^{-4}, 2^{-3}+2^{-4}) \\
y_3 = 0, \quad & A[3] \in [-2^{-2}-2^{-3}-2^{-4}, 2^{-2}+2^{-4}) \\
& R[3] \in [-2^{-2}-2^{-3}-2^{-4}, 2^{-2}+2^{-4}) \\
y_3 = -1, \quad & A[3] \in [-1-2^{-3}-2^{-4}, -2^{-1}+2^{-4}) \\
& Y[3]+Y[2] = 1-2^{-3} \\
& R[3] \in (-2^{-2}-2^{-4}, 2^{-2}+2^{-3}+2^{-4})
\end{aligned}
$$

3. For $y_1 = 0$, $y_2 = 1$,

$$
R[2] \in (-2^{-4}, 2^{-2}+2^{-4})
$$

$$
\begin{aligned}
A[3] &\in (-2^{-3}-2^{-4}, 2^{-1}+2^{-3}+2^{-4}) \\
Y[2] &= 2^{-2} \\
U_1 &= 2^{-1}+2^{-3}+\varepsilon_3 \\
L_1 &= 2^{-1}+2^{-3}-\varepsilon_3 \\
U_0 &= \varepsilon_3 \\
L_0 &= -\varepsilon_3
\end{aligned}
$$

To satisfy the containment condition,

$$
U_1 \geq 2^{-1}+2^{-3}+2^{-4} \quad \longrightarrow \quad \varepsilon_3 \geq 2^{-4}
$$

To satisfy the continuity condition,

$$
U_0 - L_1 \geq 0 \quad \longrightarrow \quad \varepsilon_3 \geq 2^{-2}+2^{-4}
$$

We choose

$$
I_1 = 2^{-2}+2^{-4}
$$

Then we have

$$
\begin{aligned}
y_3 = 1, \qquad & A[3] \in [2^{-2}+2^{-4}, 2^{-1}+2^{-3}+2^{-4}) \\
& Y[3]+Y[2] = 2^{-1}+2^{-3} \\
& R[3] \in [-2^{-2}-2^{-4}, 2^{-4}) \\
y_3 = 0, \qquad & A[3] \in [-2^{-3}-2^{-4}, 2^{-2}+2^{-4}) \\
& R[3] \in [-2^{-3}-2^{-4}, 2^{-2}+2^{-4})
\end{aligned}
$$

It is easy to show that the values of $R[3]$ satisfy the containment condition for $j = 4$. The selection function for square root operation for $j = 1, 2, 3$ is

$$
\begin{aligned}
I_1 2^{-4} &= \begin{cases} 2^{-2}, & j = 1 \\ 2^{-3}+2^{-4}, & j = 2 \\ 2^{-2}+2^{-4}, & j = 3 \end{cases} \\
I_0 2^{-4} &= -2^{-2}-2^{-3}-2^{-4}, \qquad j = 3
\end{aligned}
$$

Appendix D

Complexity and performance of conventional designs

In this appendix we derive cost and performance data for conventional parallel implementations of floating-point arithmetic operations so that a comparison with on-line approaches can be made. We make the following assumptions.

1. Each input and output data is in double precision floating-point format, with an 8-bit exponent and a 56-bit mantissa. For conventional implementations, input and output data of all arithmetic units are in non-redundant form.

2. For conventional design, detailed control mechanisms and drivers to accommodate large fanouts are ignored.

3. For the ease of comparison, all designs use components of the LSI gate array libraries as described in [41]. The complexity of a design is measured in terms of its *gate count* and *pin count*. The gate count is the number of *equivalent gates*[41], which is defined to be equal to two P-channel and two N-channel transistors. The pin count is the number of wires needed for input and output data of the arithmetic operation.

4. The timing of the designs are measured in terms of *gate delays*, denoted as t_g. A *nand* gate is assumed to have a delay of 1 t_g if the number of inputs is 2 to 4, and 2 t_g for 5 to 16 inputs. The effect of fanout is ignored since it is dependent on low level circuit design and implementation technology, and we assume that similar effects would apply to both conventional and on-line schemes.

5. For conventional designs, an arithmetic unit is reused for different instances of the same arithmetic operation. Consequently, each input of an arithmetic operation is associated with a multiplexer.

6. The clock cycle time of the conventional designs is the time delay of the 56-bit carry-lookahead adder unit.

7. Post-normalization of result and rounding are not considered in the designs.

In the following, we describe briefly the cost and performance of arithmetic units for addition, multiplication, division and square root operations. The performance and cost parameters of the components used in the designs are listed at the end of this appendix.

D.1 Conventional addition unit design

The conventional floating-point add operation is performed in two clock cycles. In the first cycle, the exponent calculation and mantissa alignment take place. The mantissa calculation is performed in the second cycle. Figure D.1 is a diagram of the conventional adder design. The 8-bit adder for the exponent calculation consists of two FA4's and a CLA1 from the LSI component library. The mantissa alignment is implemented by 2 56-bit multiplexers and a 56-bit barrel shifter. Each of the switches in the barrel shifter consists of a pair of tri-state drivers. The 56-bit adder for the mantissa calculation uses 2 levels of carry-lookahead. The first level consists of 14 4-bit carry-lookahead units, and the second level consists of 4 of the same units.

The delay of the exponent calculation and mantissa alignment is 17 t_g. The delay of the mantissa calculation, which is the delay of the 56-bit adder, is 21 t_g. The total cost of the addition unit is 4773 equivalent gates. Table D.1 summarizes the cost and performance of the design.

D.2 Conventional multiplication unit design

The conventional multiplication operation is performed in radix-4 with multiplier recoding. The recurrence is computed in carry-save form and the total number of terms to be added is reduced from 56 to 28 due to recoding. After the recurrence steps are completed, the product term is converted from carry-save form to non-redundant from by a carry-assimilation adder, which is the same as that used in

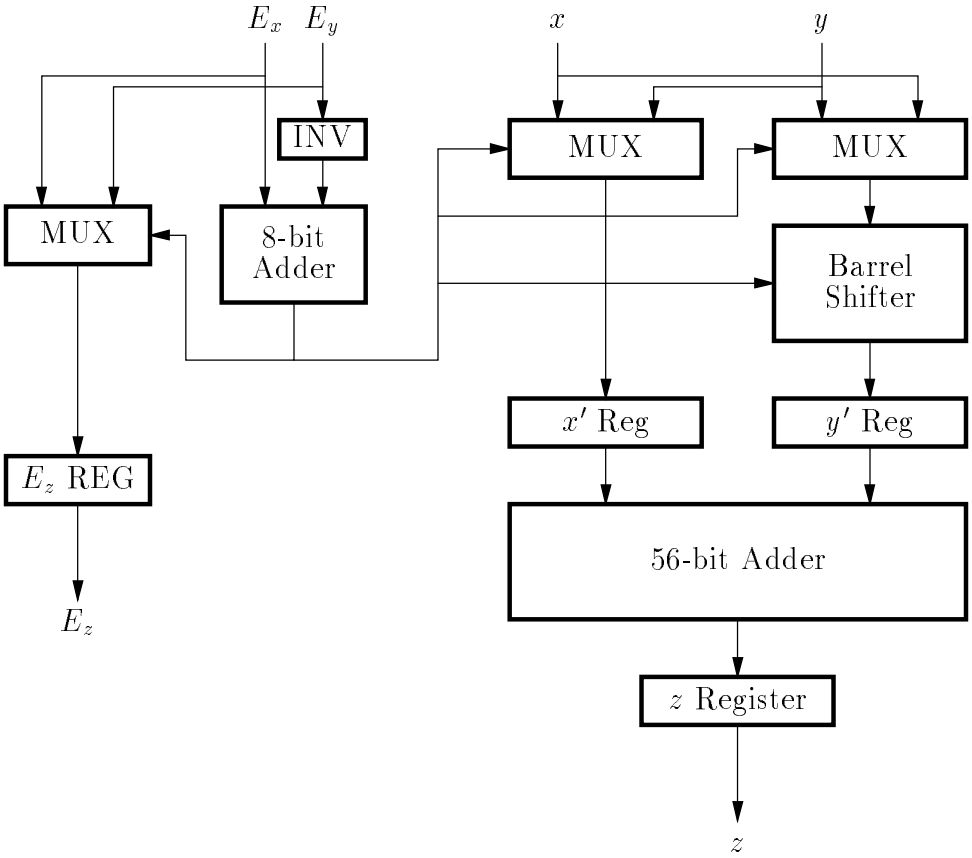


Figure D.1: Block diagram of conventional adder design

Functional unit	Functional components		Gate count	Delay	Pin count
Exponent calculation	8-bit MUX	(1)	26	2 t_g	
	8-bit INV	(1)	8	1 t_g	
	8-bit Adder	(1)	120	8 t_g	
	8-bit Register	(1)	65	2 t_g	
Mantissa calculation	56-bit MUX	(2)	364	2 t_g	
	56-bit Barrel shifter	(1)	2016	6 t_g	
	56-bit Register	(3)	1347	–	
	56-bit Adder	(1)	827	21 t_g	
Total			4773	21 t_g	197

Table D.1: Conventional addition unit design data

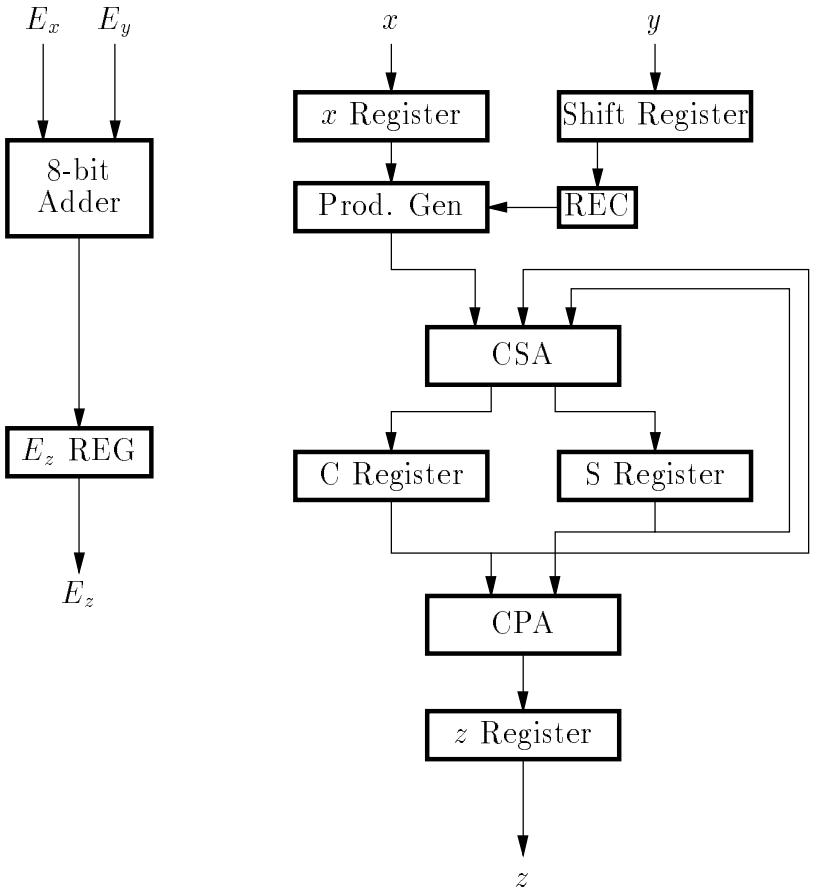


Figure D.2: Block diagram of conventional multiplier design

the mantissa calculation of addition. Figure D.2 is a diagram of the multiplication unit design.

The number of carry-save adders used in calculating the recurrence and the number of levels of the CSA tree affects the delay of the recurrence computation and the total number of cycles required for the multiplication operation. Table D.2 shows the delays and number of cycles needed for different number of CSA levels and CSA units. We assume that 7 carry-save adders are used, so the recurrence takes 4 cycles to generate 28 radix-4 digits. It takes 1 cycle to load the input operands and 1 cycle to convert the output into non-redundant form, hence the multiplication operations takes a total of 6 cycles to complete. The total cost of the design is 12063 gates. Table D.3 summarizes the cost and performance of the design.

CSA Levels	Num. of CSA's	Reduction	delay		number of cycles
			CSA only	with rec. & mult.	
1	1	3 to 2	3 t_g	7 t_g	26
2	2	4 to 2	6 t_g	10 t_g	13
3	4	6 to 2	9 t_g	13 t_g	7
4	5	7 to 2	12 t_g	16 t_g	6
4	6	8 to 2	12 t_g	16 t_g	5
4	7	9 to 2	12 t_g	16 t_g	4
5	10	12 to 2	15 t_g	19 t_g	3
5	11	13 to 2	15 t_g	19 t_g	3
6	17	19 to 2	18 t_g	22 t_g	2
7	26	28 to 2	21 t_g	25 t_g	1

Table D.2: Cycle times for various CSA levels

Functional unit	Functional components		Gate count	Delay	Pin count
Exponent calculation	8-bit Adder	(1)	120	8 t_g	
	8-bit Register	(1)	65	2 t_g	
Mantissa calculation	56-bit Register	(5)	2245	–	
	57-bit Prod. Gen.	(7)	4788	2 t_g	
	Multiplier recoder	(7)	98	2 t_g	
	56-bit CSA	(7)	3920	3 t_g	
	56-bit Adder	(1)	827	21 t_g	
Total			12063	16 t_g	197

Table D.3: Conventional multiplication unit design data

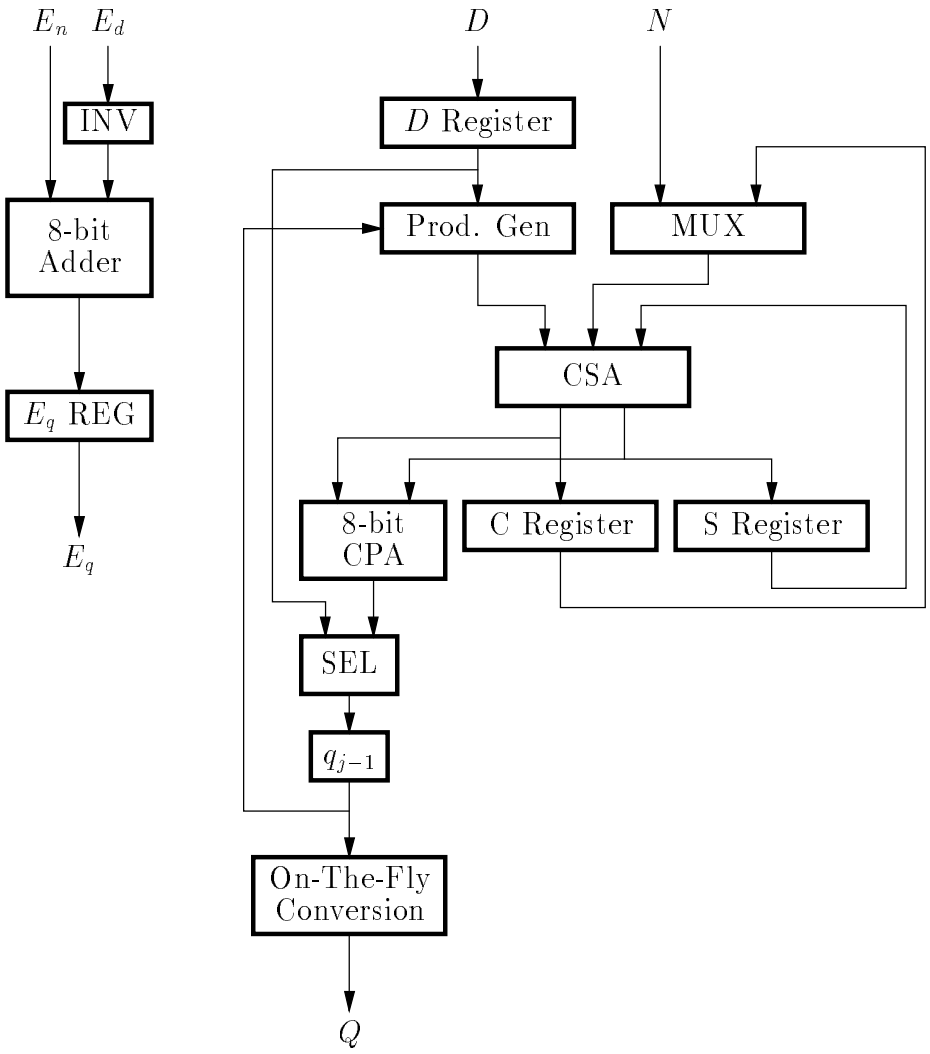


Figure D.3: Block diagram of conventional divider design

D.3 Conventional division unit design

The division scheme realizes a radix-4, non-restoring algorithm. The quotient selection function generates quotient digits in signed digit form, which are then converted to non-redundant form on-the-fly. The recurrence steps are performed by carry-save adders. Figure D.3 is a diagram of the division unit design.

The quotient digit selection design is based on data given by Talor in Figure 5(b) in [42]. Its design consists of an 8-bit adder and selection logic with 10 inputs, 2 outputs and 44 product terms. Each product term is assumed to have 6 inputs, and each output is assumed to have 22 product terms.

Algorithm D.1 is the radix-4 on-the-fly conversion algorithm.

Algorithm D.1 (Radix-4 on-the-fly conversion)

```
/* Initialization, reset */
if RS = 0 then
  for all i do
    C'[i] = 1;      a'[i] = 0
/* At digit position j, where incoming digit x[j] is loaded */
else if i ≠ j then
  C'[i] = 0
  if x[j] ≥ 0 then
    a'[i] = x[j]
  else
    a'[i] = 4 − |x[j]|
/* At all digit positions i ≠ j */
else
  if C[i] = 1 then
    C'[i] = 1;      a'[j] = a[j]
  else if x[j] > 0 then
    C'[i] = 1;      a'[j] = a[j]
  else if x[j] = 0 then
    C'[i] = 0;      a'[j] = a[j]
  else
    C'[i] = 1;      a'[j] = (a[i] − 1) mod 4

end.
```

It is assumed that the input digits are in the range

$$x[j] \quad \in \quad \{-2, -1, 0, 1, 2\}$$

Each digit position of the output is associated with a flag, which indicate whether that digit of the output is *confirmed* or *unconfirmed*.

The load signal ld for digit i is realized by a shift register $sh[i]$. Each digit slice of the conversion operation consists of 4 registers and the logic to generate $c'[i]$ and $a'[i]$. Table D.4 is the truth table for the $c'[i]$ and $a[i]$ logic. Figure D.4 is a block diagram of the digit-slice of the on-the-fly conversion unit. The cost of a digit-slice is given in Table D.5. The cost of a 56-bit conversion unit is 1652 equivalent gates.

Input								Output		
RS	$sh[i]$	$x[j]$			$c[i]$	$a[i]$		$c'[i]$	$a'[i]$	
0	–	–	–	–	–	–	–	1	0	0
–	0	–	1	–	–	–	–	1	0	0
–	0	–	–	1	–	–	–	1	0	0
–	0	–	–	–	1	–	–	1	0	0
1	–	1	–	–	0	0	0	0	1	0
1	1	1	–	–	–	–	–	0	1	0
1	1	–	1	–	–	–	–	0	1	0
1	0	–	–	–	1	1	–	0	1	0
1	0	0	–	–	–	1	–	0	1	0
1	0	–	–	–	–	1	1	0	1	0
1	1	–	–	1	–	–	–	0	0	1
1	0	1	–	–	0	–	0	0	0	1
1	0	0	–	–	–	–	1	0	0	1
1	0	–	–	–	1	–	1	0	0	1

Table D.4: Truth table for the generation logic of $c'[i]$ and $a'[i]$

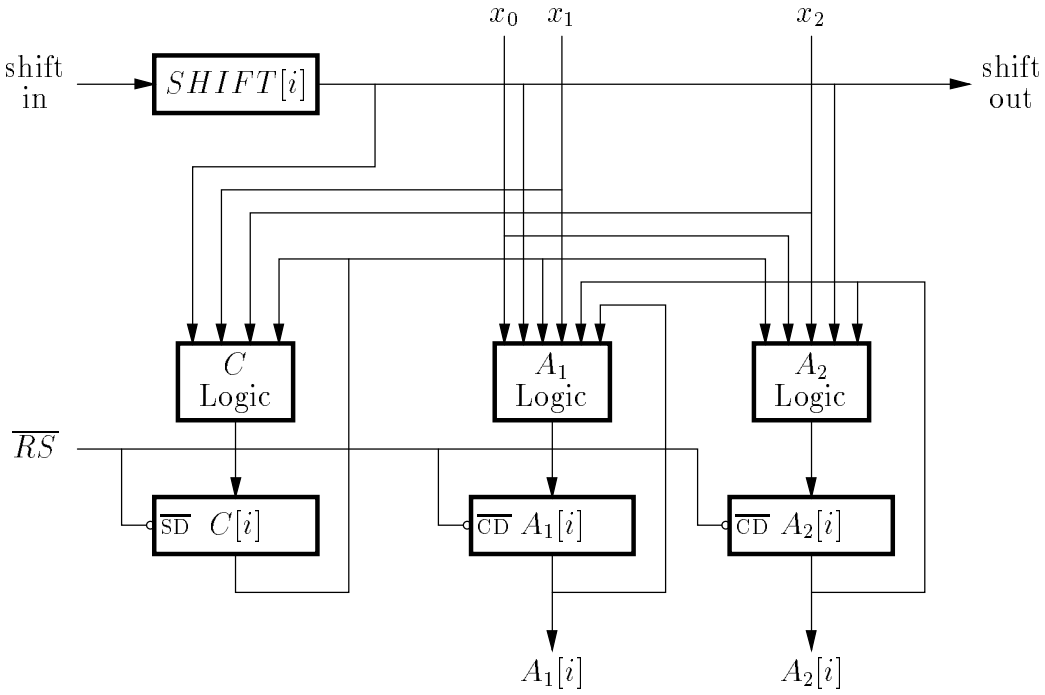


Figure D.4: Digit-slice of the on-the-fly conversion unit

Component	LSI comp. used	cost
Shift register	FD2(1)	6
Flag	FD2(1)	6
$a_1[i], a_2[i]$	FD2(2)	12
Inverters	IV(3)	3
$C[i]$ logic	ND2(3) ND4(1)	5
$a_1[i]$ logic	ND2(1) ND3(3) ND4(1) ND5(1)	14
$a_2[i]$ logic	ND3(1) ND4(3) ND6(1)	13
Digit-slice cost		59

Table D.5: Cost of radix-4 on-the-fly conversion digit slice

The delay of the conversion is 3 t_g.

The floating-point division operation takes 29 cycles and the cost of the design is 4993 equivalent gates. Table D.6 summarizes the cost and performance of the design.

D.4 Conventional square root unit design

It is assumed that the square root unit has the same complexity and performance as the division unit.

Functional unit	Functional components		Gate count	Delay	Pin count
Exponent calculation	8-bit INV	(1)	8	1 t_g	
	8-bit Adder	(1)	120	8 t_g	
	8-bit Register	(1)	65	2 t_g	
Mantissa calculation	56-bit Register	(3)	1347	–	
	57-bit Prod. Gen.	(1)	684	2 t_g	
	56-bit MUX	(1)	182	2 t_g	
	56-bit CSA	(1)	560	3 t_g	
	8-bit Adder	(1)	120	8 t_g	
	Digit selection	(1)	237	5 t_g	
	3-bit Register	(1)	18	–	
	56-bit conversion	(1)	1652	3 t_g	
Total			4993	18 t_g	197

Table D.6: Conventional division unit design data

Functional component	LSI comp.(count)	equiv. gate	gate delay
8-bit INV	IV(8)	8	1 t_g
8-bit Adder	CLA1(1) FA4(2)	120	8 t_g
8-bit MUX	MUX24H(2)	26	2 t_g
8-bit Register	FD1(8) ND2(24) IV(1)	65	2 t_g
56-bit MUX	MUX24H(14)	182	2 t_g
56-bit Barrel shifter	BTS4(672)	2016	6 t_g
56-bit Adder	ND2(164) ND3(72) ND4(90) ND6(18) EO(112) IV(90)	827	21 t_g
56-bit Register	FD1(56) ND2(168) IV(1)	449	2 t_g
Multiplier recoder	ND2(7) ND3(2) IV(3)	14	2 t_g
57-bit Prod. Gen.	ND3(228) ND4(57) IV(114)	684	2 t_g
56-bit CSA	FA1(56)	560	3 t_g
Digit selection	ND6(44) ND4(6) ND3(1) NR2(3)	237	5 t_g
3-bit Register	FD2(3)	18	–
64-bit 2 input MUX	MUX24H(16)	208	2 t_g
64-bit 3 input MUX	MUX34H(16)	352	3 t_g
64-bit 4 input MUX	MUX44H(16)	416	3 t_g
64-bit 5 input MUX	MUX54H(16)	560	4 t_g

Table D.7: Cost and delay of functional components

LSI component	Description	equiv. gate	gate delay
IV	invertor	1	1 t_g
ND2	2 input NAND gate	1	1 t_g
ND3	3 input NAND gate	2	1 t_g
ND4	4 input NAND gate	2	1 t_g
ND6	6 input NAND gate	5	2 t_g
NR2	2 input NOR gate	1	1 t_g
EO	2 input exclusive OR gate	3	2 t_g
BTS4	3 state driver	3	1 t_g
FD1	D flip flop	5	–
FD2	D flip flop with clear	6	–
FA1	full adder	10	3 t_g
FA4	4 bit binary full adder	44	7/4 t_g
CLA1	4-bit carry look ahead	22	4 t_g
MUX24H	quad 2 bit multiplexer	13	2 t_g
MUX34H	quad 3 bit multiplexer	22	3 t_g
MUX44H	quad 4 bit multiplexer	26	3 t_g
MUX54H	quad 5 bit multiplexer	35	4 t_g

Table D.8: Cost and delay of LSI library components