AN ABSTRACT OF THE THESIS OF

Ajay Chordia Shantilal for the degree of Master of Science in Electrical and Computer Engineering presented on May 27, 2004.

Title: Improving Throughput of Networks of Radix-2 On-line Arithmetic Modules for Small Precision Applications.

Redacted for privacy

Abstract approved: _____

Alexandre F. Tenca

On-line arithmetic modules were proposed as a way to explore parallelism in arithmetic operations at digit level. Using always serial operators that receive inputs and compute the outputs from most-significant to least-significant digits, on-line arithmetic makes it possible to overlap all arithmetic operations and have a pipelined structure.

On-line division is one of the slowest operations among the basic arithmetic operations and naturally becomes a bottleneck in networks of on-line modules that use it. A higher radix divider has a good potential to attain higher throughput than radix-2 dividers and therefore improve the overall throughput of networks where division is needed. The improvement in throughput when using radix 4 is not straightforward since several components of the divider become more complex than in the radix-2 case. Previously proposed radix-4 designs were based on operand pre-scaling to simplify the selection function and reduce the critical path delay, at the cost of more complexity in the algorithm conditions and operations, plus a variable on-line delay, which is a very unattractive feature when small precision values are used (usually the case for DSP). These designs include several phases for pre-scaling

and actual division. In this thesis a design approach based on overlapped replication that results in a radix-4 on-line division module with low algorithm complexity, single division phase, less restrictions to the input values, and a small and fixed on-line delay is presented.

On-line arithmetic modules have an intrinsic on-line delay that impacts the maximum throughput of networks using these modules. The problem is particularly serious for small precision calculation or deep-pipelined networks. This work presents a solution to the problem. Results of an actual implementation of the solution for some basic arithmetic operators are shown and demonstrate the benefits of the proposed approach. The developed modules are also tested within the framework of an image processing application.

Improving Throughput of Networks of Radix-2 On-line Arithmetic Modules for Small Precision Applications

by

Ajay Chordia Shantilal

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed May 27, 2004
Commencement June 2005

Master of Science thesis of Ajay Chordia Shantilal presented on May 27, 2004

APPROVED:

Redacted for privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for privacy

Director of the School of Electrical Engineering and Computer Science

Redacted for privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for privacy

Ajay Chordia Shantilal, Author

ACKNOWLEDGMENTS

First and foremost I would like to thank Dr. Alexandre Tenca, for his constant support and encouragement during my graduate studies. His motivation and guidance in field of on-line arithmetic and digital design helped me a lot, without which this work would never have been possible.

Thanks to Mohammed Sinky who worked along in development of some modules and network for digital image processing algorithm.

I would like to thank my family for their love, encouragement and support during all stages of my studies.

TABLE OF CONTENTS

Page

1 INTRODUCTION 1

1.0.1 Least Significant Digit First (LSDF) Approach 2

1.0.2 Most Significant Digit First (MSDF) Approach 2

1.0.3 Comparison of different Arithmetic operation 3

1.1 Previous Work 4

1.2 Motivation 5

1.3 Organization Of The Thesis 7

2 ON-LINE ARITHMETIC 8

2.1 Digit Representation 8

2.2 On-line Arithmetic modules 9

2.2.1 Inputs and Outputs 9

2.2.2 Recurrence Equation 10

2.2.3 Selection Function 11

2.3 Performance Measurement of a Design 11

2.3.1 Latency 11

2.3.2 Throughput 12

2.3.3 Area 12

2.3.4 Precision 13

3 A RADIX-4 ON-LINE DIVISION DESIGN 15

3.1 Derivation of the Radix-4 on-line design 15

3.1.1 Basic concepts of on-line division 16

3.1.2 Radix-4 design using overlapped replication 17

3.1.3 Unfolded radix-2 recurrence equations 18

3.1.4 General organization 20

TABLE OF CONTENTS (Continued)

Page

3.1.5 predictive quotient digit selection 21

3.2 Circuit implementation 23

3.2.1 Radix-4 append registers 25

3.2.2 BS to CS conversion 26

3.2.3 Combining a digit n with vector dQ to obtain $n + dQ$ 27

3.2.4 Digit-by-vector multipliers 27

3.2.5 Selection function SEL2 28

3.2.6 Pipeline 29

3.3 Comparison with other designs 29

3.4 Experimental results 31

3.4.1 Throughput 35

4 HIGH-THROUGHPUT NETWORKS OF ON-LINE ARITHMETIC MODULES FOR DSP APPLICATIONS 37

4.1 Impact of on-line delay on network performance 37

4.2 Decoupling initialization and output generation 39

4.2.1 Addition, multiplication and MAC 39

4.2.2 Radix-2 on-line division 42

4.3 Design of new on-line modules 43

4.3.1 General design 43

4.3.2 Improved on-line adder 46

4.3.3 Overlapped on-line divider 48

4.4 Implementation 51

4.5 Case Study: Digital Signal Processing Applications 54

TABLE OF CONTENTS (Continued)

Page

5 CONCLUSION 58

5.1 Summary of the results 58

5.2 Consideration for usage of on-line arithmetic module 59

5.3 Further Work 60

5.4 Concluding Remarks 61

BIBLIOGRAPHY 62

LIST OF FIGURES

Figure		Page
1.1	Digit-Serial arithmetic timing characteristics	1
2.1	On-line arithmetic input-output characteristics	8
3.1	Organization of the Radix-4 division unit	21
3.2	Selection function based on predictive calculation of q_j.	23
3.3	Circuit for radix-4 on-line division	24
3.4	Radix-4 append register for $D[j+1]$	25
4.1	Network of on-line modules.	38
4.2	Improved design for on-line adder, multiplier, and mac modules	44
4.3	Generation of Next Residual	45
4.4	Simulation trace - improved on-line multiplier	46
4.5	On-line adder for improved throughput	47
4.6	Simulation of improved on-line adder	48
4.7	Circuit for radix-2 on-line division with auxiliary unit	49
4.8	Waveform for radix-2 on-line division with auxillary	50
4.9	Image processing algorithm for processing the chromatic signals of color images	54
4.10	Hardware configurations for back end of image processing algorithm	55
4.11	Timing of filter using (a) conventional on-line division (Architecture I) (b) improved-throughput on-line adder and overlapped on-line divider (Architecture II) and (c) SRT division (Architecture III)	56

LIST OF TABLES

Table		Page
2.1	BS, TC, and SM representation of signed-digit	9
2.2	Recurrence equations and main parameters for some on-line operators	14
3.1	Radix-4 and Radix-2 non-pipelined on-line divider for CMOS	34
3.2	Radix-4 and Radix-2 pipelined on-line divider for CMOS	34
3.3	Radix-4 and Radix-2 non-pipelined on-line divider for FPGA	35
3.4	Throughput comparison of Radix-4 and Radix-2-non-pipelined on-line divider for ASIC	36
3.5	Throughput comparison of Radix-4 and Radix-2-pipelined on-line divider for ASIC	36
3.6	Throughput comparison of Radix-4 and Radix-2-non-pipelined on-line divider for FPGA	36
4.1	Simplified selection function	41
4.2	Experimental results for on-line adders	52
4.3	Experimental results for on-line multipliers	53
4.4	Experimental results for on-line dividers	53
4.5	Experimental results obtained for Digital Image Processing network	57

1. INTRODUCTION

Computer arithmetic involves algorithm that are used to compute a particular mathematical operation. An arithmetic unit is a system or module which operates upon numbers and compute results of a particular operation. An arithmetic unit is broadly classified into three categories based upon the way the inputs are applied to them. If all the operands are applied in parallel to a unit, it is a parallel arithmetic unit. Whereas a unit is called as serial unit where the operands are applied digit-by-digit and the output is also obtained serially. A combination of both, where one of the inputs is applied serially and the other available in parallel, is also possible and it is called a serial-parallel unit.

In a serial arithmetic operation inputs are applied in digit-serial manner and the output is also obtained in a digit-serial fashion, one digit every clock cycle. One input operand is applied each cycle and output is also obtained every clock [18].

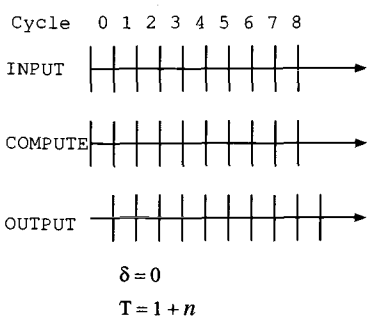


FIGURE 1.1. Digit-Serial arithmetic timing characteristics

In the Figure 1.1 input operands are applied starting at cycle 0 and output is obtained beginning cycle 1. The total number of cycles it takes to compute the result in the above case is $1 + n$, where n is the precision of operation. It is not

always the case that the output is obtained immediately at cycle 1. In some cases the input has to be accumulated for δ cycles until the output digits are to be generated. In this case the total number of cycles required to compute the output is δ + 1 + n cycles. There are two different modes for serial arithmetic operations Least Significant Digit First (LSDF) and Most Significant Digit First (MSDF) approach.

1.0.1. Least Significant Digit First (LSDF) Approach

In LSDF approach the digit of operands are applied serially with the least significant digit first [18], this method is also known as right to left approach and since this was the first proposed arithmetic approach, its also called as *conventional serial arithmetic.* Since digits are applied right to left, the order of indexing is simplified as in the case of integer representation, given as

$$X = \sum_{i=0}^{n-1} x_i r^i \tag{1.1}$$

1.0.2. Most Significant Digit First (MSDF) Approach

In MSDF approach the digit of operands are applied serially with the most significant digit first [18], with the output digits also obtained serially with MSD first after an intrinsic delay δ called the *on-line delay.* This method is also known as left to right approach or *on-line arithmetic.* Since digits are applied left to right the order of indexing is simplified as in case of fraction representation, given as

$$X = \sum_{i=1}^{n} x_i r^{-i} \tag{1.2}$$

1.0.3. Comparison of different Arithmetic operation

In a parallel arithmetic operation all the inputs are applied in parallel and also the output is obtained in parallel within one or few clock cycle. The latency of such a system would normally be better compared to any other system, and overall computational time lesser compared to serial arithmetic. The main reason for opting for serial arithmetic operation especially for Digital Signal Processing (DSP) application is that serial operation reduces the complexity of the system by reducing the number of signal lines and has a reduced area and power dissipation. The delay obtained due to more number of cycles used for serial operation could be compensated by using a pipelined operation i.e. overlapping of successive operation.

The advantage of conventional arithmetic operation over on-line arithmetic is its simplicity, and the potential to overlap several LSDF operations with one digit time delay. Whereas in on-line arithmetic there is an associated on-line delay δ, which in turn decreases the overall throughput of the system. Besides this disadvantage on-line arithmetic has several attractive features when compared to conventional serial arithmetic. In on-line arithmetic all arithmetic operations can be processed in the same direction therefore it is possible to overlap consecutive operations that could not be pipelined using conventional digit serial algorithms. For example if we have to compute $(a+b)/c$ we could pipeline the addition and division process because all the operand and results are obtained in the same direction right to left, this overlap is not feasible with conventional serial arithmetic because some algorithms, such as division and square root are inherently MSDF type.

On-line arithmetic operations make floating point implementation faster compared to its counterpart because in conventional approach normalization depends on MSD which is generated at last adding to latency whereas in online arithmetic

MSD is generated at beginning which could be normalized earlier and hence faster. Another advantage of on-line arithmetic is that computation could be stopped after desired precision is obtained. For example when we are operating with operands of 16-bit precision and would compute $a.b$; in the case of conventional arithmetic we would be wasting many cycles just to compute the LSB which would never be used thereby adding to latency whereas in on-line arithmetic the LSB are computed only if needed.

1.1. Previous Work

On-line arithmetic was proposed as an alternative to the implementation of serial arithmetic operators [4–6]. In [4, 5] principles and techniques of on-line arithmetic are discussed. The totally parallel addition proposed in [13] yields an on-line algorithm. On-line algorithm for multiplication and division were developed in [14]. On-line solutions are usually more economical in terms of area, and they are quite good in terms of throughput for the cases analyzed in [7, 8]. In [12] it is also claimed that On-line operators are more suitable to be used in A/D and D/A converters that usually operate MSD first. Studies have been done in the past to improve the efficiency of on-line modules [9], and operating frequency, but did not consider the basic problem of on-line delay. A complete library of ol-line operators for basic arithmetic operation was developed in [7]. Further development was made to this library in this work. The conclusion of [11] is that on-line arithmetic is very suitable to handle large numbers with higher precision but again fails to deal with small precision numbers.

Algorithms for on-line division were presented in [14–16]. In [17] a high radix on-line division unit is discussed but it fails to address the issue for small precision.

Design of a radix-4 division unit was proposed in [10]. It uses a technique based on pre-scaling of input operands and selection by rounding. The design is very complex, requires three different operation phases, and exhibits a large and variable on-line delay. For this reason, it is not appropriate for on-line networks used in digital signal processing, where the operand precision is usually small and the network throughput would be greatly affected by this behavior. The control system would need to be sophisticated enough to stall network modules depending on the on-line delay for division.

1.2. Motivation

There are two fundamental limitations to the performance of networks of on-line modules, especially for small precision of the operands: on-line delay and maximum operating clock frequency.

Usually, division units are the bottlenecks in networks of on-line modules. Division modules are slower than addition and multiplication. When a radix-2 divider is added to a network composed of only radix-2 on-line adders and multipliers, it reduces the maximum operating frequency by 35-40% [7]. Let's call this reduced frequency f_1. One approach to increase the performance of this type of on-line network is to use radix-4 dividers. The network would operate at a clock frequency f_2 for the adder and multiplier modules, and at a frequency $f_2/2$ for the radix-4 divider modules, improving the network throughput when $f_2 > f_1$. The solution has an advantage in performance when compared to networks using only radix-2 on-line operators. Previous radix-4 on-line division are too complex and difficult to implement. In this work a simple radix-4 ol-divider is shown.

In principle, on-line arithmetic should be efficient and have a high throughput. However, on-line modules have an intrinsic delay, related to the time when the modules are acquiring enough information from the input data stream before starting to generate the output stream. This delay is called on-line delay (δ) and affects the throughput significantly when the precision of the operands are small (e.g. Digital Signal Processing – DSP – applications use operands with tens of bits) or in deep networks (pipelining). All the previous implementations of on-line modules did not consider this problem. Theoretical results discuss the minimum unavoidable on-line delays for several arithmetic operation [11]. However, computer architecture techniques can be used to remove or reduce the impact of these delays on the throughput.

A more detailed study of the behavior of on-line modules forming a network for the implementation of DSP applications was done on [7, 8]. Such a study brought some insight to the problem of the on-line delay in the implementation of more complex networks and how the delay impacts the throughput, leading to design decisions that are not so obvious. Based on this study, it was clear that the reduction or elimination of on-line delays at the architecture level would be needed in order to improve the performance of systems implemented using on-line modules, and make them more competitive with other conventional implementations. A solution to improve the throughput of basic on-line modules commonly used in DSP applications is presented in this work. The basic on-line modules (adders, multipliers and dividers) show clear performance enhancement for networks that use these modules.

In [7] it was also concluded that SRT-division algorithm [18] is better in a network which uses on-line arithmetic modules both in terms of area, and operating frequency. But SRT-divider takes the input in parallel and produces output in non-

redundant form. Hence if we decide to use an on-line module in a network after SRT-divider it would cost extra area. Hence we need to compete on-line divider with this SRT-divider.

1.3. Organization Of The Thesis

Chapter 2 provides theory behind on-line arithmetic, performance measurement used and some results for basic on-line arithmetic modules such as adders, multipliers and dividers which are commonly used in networks of on-line modules. In Chapter 3 a new Radix-4 on-line division design, along with required derivations and optimization involved are shown. Most of this chapter is from work submitted in [1]. In chapter 4 the required architecture to hide the on-line delay in order to improve its throughput is discussed. The work in this chapter is in [2, 3]. The conclusion of this work and future work are finally presented in Chapter 5.

2. ON-LINE ARITHMETIC

In on-line arithmetic the operands as well as the results flow in a digit-by-digit manner with MSD first.

There is always an associated delay δ called the on-line delay when performing arithmetic operation using on-line modules. Figure 2.1 represents the input/ output characteristics of on-line arithmetic module [4]. Hence in-order to produce the result at the jth digit z_j we require the corresponding operand digits at j+δ cycle. (i.e. $x_{j+\delta}$ and $y_{j+\delta}$). On-line delay is also defined by the number of cycles between the input of the first fractional digit and the output of the first fractional digit.

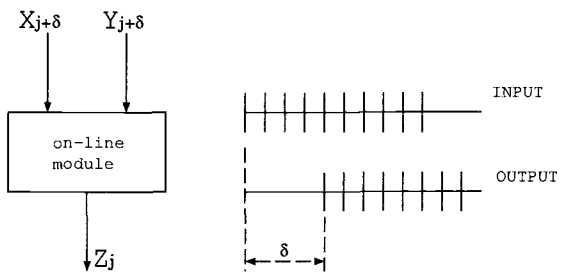


FIGURE 2.1. On-line arithmetic input-output characteristics

2.1. Digit Representation

In order to avoid carry propagation which could affect the MSD we use Redundant Number System (RNS), to represent digits used in on-line arithmetic. A redundant radix-r weighted system has more than r different possible values for one digit. A number X in such a system is defined [7] as

$$X = \sum_{i=-M}^{N} x_i r^i \tag{2.1}$$

where r is the radix of the system and x_i are the digits. One of the most used RNS is *signed digit* (SD). A digit in a SD form is in a digit set D = $-\rho$,...,0,...,ρ where $\frac{r-1}{2} < \rho \leq$ r-1 determines the amount of redundancy [4].

The most widely used representations for radix-2 SD number systems are *borrow-save* (BS), *two's complement* (TC), and *sign-magnitude* (SM). Table 2.1 shows these three representations.

Value	Notation	BS Representation a^+a^-	TC Representation a_1a_0	SM Representation sm
0	0	00 or 11	00	00 or 10
1	1	10	01	01
-1	$\bar{1}$	01	11	11

TABLE 2.1. BS, TC, and SM representation of signed-digit

There are few disadvantages of using RNS. There is an increase in the number of bits required to represent a redundant number. It is difficult to perform magnitude comparison of operands and detecting the sign of an operand represented in RNS. The effect of these disadvantages could be reduced by using *On-the-Fly Converters* OFC and short precision adders. The OFC described in [19] converts serial signed digit number with MSD first into TC numbers in parallel on the fly.

2.2. On-line Arithmetic modules

Most of on-line arithmetic are based upon certain principles, structures and properties. Some of the important properties are:

2.2.1. Inputs and Outputs

The operand (e.g. A) and results (e.g. Z) in on-line form at time j is

$$A[j] = \sum_{i=1}^{j+\delta} a_i r^{-i} = A[j-1] + a_{j+\delta} r^{-(j+\delta)} \tag{2.2}$$

$$Z[j] = \sum_{i=1}^{j+\delta} z_i r^{-i} = Z[j-1] + z_{j+\delta} r^{-j} \tag{2.3}$$

Where δ is on-line delay, r is the radix of the number system, and at any time in algorithm the output satisfies the condition

$$Z[j] = F(A[j], B[j]) + \epsilon_r r^j \tag{2.4}$$

where ϵ_r is the error and is bounded always between ± 1.

2.2.2. Recurrence Equation

Generally, on-line arithmetic is an iterative/ recursive process which uses intermediate residuals to compute the next output digit and the next residual. The residual at step j is represented as $W[j]$ and it is a function of input digits, previous residual and output digits generated. It is given by

W[j] = G (W[j-1], A[j], B[j], Z[j-1])

The function G is normally composed of addition, subtraction and multiplication with simple shift operation. The design of this function is normally kept simple and efficient.

Table 2.2 shows the basic on-line operators and its main parameters, such as on-line delay (δ) and truncation point (fractional digit position) used for estimate of residual in the selection function, as presented in [7]. These parameters are given as a function of the residual representation, that may be: carry-save (CS), signed-digit (SD), or non-redundant (NR).

2.2.3. Selection Function

The selection function S, is a function used to determine the output digit and it is a function of its partial residuals, input digits and output digits. It is given by

s_j = S (W[j-1], A[j], B[j], Z[j-1])

The selection function is defined in such a way that the result is bounded within given limits and the error converges to zero. The selection function when $r = 2$ for all the modules shown in the table 2.2 is given by (2.5). For on-line adders, multipliers and MACs, it uses the selection constants $m_0 = -0.5$ and $m_1 = 0.5$. For on-line dividers, $m_0 = -\frac{3}{16}$ and $m_1 = \frac{1}{16}$.

$$z_j = sel(\hat{W}[j]) = \begin{cases} 1 & \text{if } \hat{W}[j] \geq m_1 \\ 0 & \text{if } m_0 \leq \hat{W}[j] < m_1 \\ -1 & \text{otherwise} \end{cases} \tag{2.5}$$

2.3. Performance Measurement of a Design

Any digital design is measured in terms of speed and area. There are criteria for comparison of two systems. The efficiency of an arithmetic implementations can be measured by latency, throughput, area and accuracy besides simplicty of design. The design criteria are in turn highly dependant. We will define these measurement criteria in this section All results obtained here are based on these definitions:

2.3.1. Latency

Latency of a system is defined as the difference between the time when an input is applied to a system and time at which the output is obtained. In on-line

arithmetic, latency is defined based upon on-line delay δ. But this definition for latency may not be applicable to other arithmetic modules and hence there might be discripancies in these comparisons. Hence we generalize latency as the time between a parallel input and parallel output if applicable, else for a serial arithmetic latency is the time between the first input digit and last output digit.

2.3.2. Throughput

The throughput of a system design is defined as the number of input sets it can process per unit time. It gives the measure of a pipelining depth, and is a very important measurement criteria for such a system. Throughput of a system also depends upon maximum operating frequency and the data dependancy factor.

2.3.3. Area

The number of unit areas occupied by a design gives a measure of its cost and power consumption which might be an important factor for portable devices and need to be considered significantly. In this work we are mainly targetting FPGAs for area measurement, and it is mostly given by the number of CLBs used by a design. It gives a measurement of how well a given design fits into the given structure of the FPGA. When using traditional ASIC technology, area is measured based upon the number of equivalent gates in the design or the final design area obtained from layout.

2.3.4. Precision

Precision of a system is generally based upon its accuracy. In on-line arithmetic precision usally signifies the total number of fractional bits which are being operated upon. A lot of cycles is saved in on-line arithmetic by not evaluating the non-significant digits of output and maintaining the required precision. Some on-line modules produces additional integer digits, that are not needed to represent the possible range of the result. These extra digit can reduce the efficiency of a network of on-line modules. A way to handle them is to use a normalizer [20] component that eliminates all integer digits.

Addition $Z = A + B$ $W[j] = r \cdot (W[j-1] - z_{j-1}) + r^{-\delta} \cdot (a_{j+\delta} + b_{j+\delta})$ $W[0] = A[0] + B[0]$ for $r = 2$ and CS/SD/NR: $t = 2, \delta = 3$
Multiplication $Z = A \cdot B$ $W[j] = r \cdot (W[j-1] - z_{j-1}) +$ $r^{-\delta} \cdot (a_{j+\delta} \cdot B[j-1] + b_{j+\delta} \cdot A[j])$ $W[0] = A[0] \cdot B[0]$ for $r = 2$ $t = \{3(CS), 2(SD), 2(NR)\}, \delta = 4$ (pipelined)
MAC $Z = A \cdot B + C$ with B in parallel $W[j] = r \cdot (W[j-1] - z_{j-1}) + r^{-\delta} \cdot (a_{j+\delta} \cdot B + c_{j+\delta})$ $W[0] = A[0] \cdot B + C[0]$ for $r = 2$ $t = \{3(CS), 2(SD), 1(NR)\}, \delta = \{3(CS, SD), 2(NR)\}$
Division $Q = N/D$ $W[j] = r \cdot (W[j-1] - q_{j-1}D[j]) + r^{-\delta} \cdot (n_{j+\delta} - d_{j+\delta}Q[j-2])$ $W[0] = N[0]$ for $r = 2$ $t = \{4(CS)\}, \delta = 5$

TABLE 2.2. Recurrence equations and main parameters for some on-line operators

3. A RADIX-4 ON-LINE DIVISION DESIGN

On-line division is one of the most complex of the basic arithmetic operations. The design strategy proposed in this chapter consists in applying a technique already suggested for digit-recurrent division and called *overlapped replication* [21], which consists in obtaining a radix-4 design based on the concurrent execution of two iterations of a simple radix-2 division algorithm.

In the following sections we first show the derivation of the proposed design, present the architecture for the radix-4 on-line division, and define all the parameters required for proper operation. Section 3.2 shows details of the circuit implementation. Estimates of the design performance and comparison with other designs are provided in Section 3.3. Experimental results are shown in Section 3.4 for FPGA and CMOS implementations. The same section includes a discussion about the throughput of the proposed radix-4 divider.

3.1. Derivation of the Radix-4 on-line design

Since the design presented in this work is developed based on unfolded versions of a radix-2 on-line division operation, this section will first focus on the concepts, definitions, and results for on-line division, and in particular radix-2 on-line division. After that, we show the derivation of expressions for radix-4 on-line division.

3.1.1. Basic concepts of on-line division

From the work already presented by other authors [16, 26, 7], the recurrence equation that is required to perform on-line division of two m-bit fractional numbers N and D is given as:

$$W[j] = r(W[j-1] - q_{j-1}D[j]) + r^{-\delta}(n_{j+\delta} - d_{j+\delta}Q[j-2]) \tag{3.1}$$

where j represents the iteration step, δ represents the on-line delay (in steps), and W is called a scaled residual. Lower case letters represent digits, and the upper case letters represent digit vectors. Based on this recurrence equation, a division unit consumes one digit of N and one digit of D in each clock cycle, starting from the most-significant digit, and generates the output digits (q_j) after a delay of δ clock cycles. The digit vectors are defined as:

$$N = \sum_{i=0}^{m} n_i r^{-i}$$

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

$$Q[j] = \sum_{i=0}^{j} q_i r^{-i} = q_j r^{-j} + Q[j-1]$$

where n_i, q_i, and d_i are in a redundant digit set $\{-\rho, ..., \rho\}$, with $\frac{r-1}{2} \leq \rho \leq r-1$. The value of q_{j-1} is obtained from $W[j-1]$ by a selection function. It is always imposed that $N < D$ and that D be a normalized or quasi-normalized fraction.

For radix 2, with $1/4 \leq D < 1$ and residual in Carry-Save (CS) format, it was determined in [16] that $\delta = 5$. The selection function for this case is:

$$q_{j-1} = \begin{cases} 1 & \text{if } \hat{W}[j-1] \geq 1/16 \\ 0 & \text{if } -3/16 \leq \hat{W}[j-1] < 1/16 \\ -1 & \text{otherwise} \end{cases} \tag{3.2}$$

where $\hat{W}$ is an estimate of the residual, obtained by truncating the full-precision residual at the $t = 4$ fractional bit position. As mentioned before, the radix-2 on-line division is not as fast as the radix-2 on-line addition and multiplication [26]. Therefore, division is always a bottleneck for networks of on-line modules.

3.1.2. Radix-4 design using overlapped replication

One way to implement high-radix dividers consists in deriving the recurrence equation and selection function for the high radix case and implement the circuit based on this derivation. The resulting design is usually cumbersome and too slow because the selection function and other modules used in the circuit implementation are more complex than the radix-2 design. Some researchers proposed to simplify the selection function by scaling the input operands in an effort to speed up the final implementation. However, this approach, as shown in [10] for a radix-4 design, introduces different phases in the division process making both the algorithm derivation, condition for system operation, and actual circuit (specially control) reasonably complex. In addition, the resulting divider using pre-scaling exhibits a large and variable on-line delay. A variable on-line delay is undesirable when the division module is used in a network of on-line modules.

Another alternative to implement a radix-4 divider is to unfold several small-radix division iterations and operate them in parallel [21]. This design approach was called *overlapped replication* in [21]. It was not considered for on-line division before. In the following sections we detail the design process to obtain the proposed radix-4 on-line division design.

3.1.3. Unfolded radix-2 recurrence equations

Let's consider the case where two iterations of the radix-2 recurrence equation (steps j and $j+1$) are executed. When we take Eq. 3.1 for $r = 2$ and project the value of the residual for the next iteration we obtain the equation:

$$W[j+1] = 2(W[j] - q_j D[j+1]) + 2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1} Q[j-1]) \tag{3.3}$$

and when we use $W[j]$ as a function of $W[j-1]$ we obtain:

$$\begin{aligned} W[j+1] = 4(W[j-1] - q_{j-1} D[j]) + 2^{-\delta+1}(n_{j+\delta} - d_{j+\delta} Q[j-2]) - \\ - 2q_j D[j+1] + 2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1} Q[j-1]) \end{aligned} \tag{3.4}$$

At step j, the residual value is $W[j-1]$, and the quotient digit q_{j-1} can be obtained from an estimate ($\hat{W}[j-1]$) of this residual. The value q_j however requires the value $W[j]$ that is not available, and therefore must be computed during the same clock cycle. In order to have this value as soon as possible, a prediction unit evaluates an estimate of $W[j]$ for each possible value of q_{j-1}, the residual $\hat{W}[j-1]$, and other values available at step j. Therefore, three possible values of q_j are computed and the correct value is selected once the q_{j-1} is computed. Equation 3.4 shows one radix-4 iteration.

In this radix-4 step the index j increases by 2. Observe that the need for intermediate values, such as $D[j]$ or $Q[j-1]$ is a problem for circuit implementation because we are interested in using on-the-fly converters (OFC) [19] to generate non-redundant values of these vectors. OFC circuits that keep and provide these radix-2 values will require more memory elements than a circuit that works only on radix-4 digits. Furthermore, the need to have q_{j-1} to generate $Q[j-1]$ would imply in a very long combinational path in the design (notice that q_{j-1} is computed during the

radix-4 iteration). Therefore some extra transformations on the recurrence equation are necessary. Starting from Eq. 3.4, we look at the sub-expression:

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

Based on the fact that $Q[j] = \sum_{i=0}^{j} q_i 2^{-i}$ and $D[j] = \sum_{i=0}^{j+\delta} d_i 2^{-i}$ we have:

$$\begin{aligned} A &= 2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1}\overbrace{(Q[j-2] + q_{j-1}2^{-(j-1)})}^{Q[j-1]}) \\ &= 2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1}Q[j-2] - d_{j+\delta+1}q_{j-1}2^{-j+1}) \\ &= 2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1}Q[j-2]) - d_{j+\delta+1}q_{j-1}2^{-j-\delta+1} \\ &= 2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1}Q[j-2]) - 4d_{j+\delta+1}q_{j-1}2^{-j-\delta-1} \end{aligned}$$

Using this equation back into Eq. 3.4 we have:

$$\begin{aligned} W[j+1] = 4(W[j-1] - q_{j-1}D[j] - d_{j+\delta+1}q_{j-1}2^{-(j+\delta+1)}) + 2^{-\delta+1}(n_{j+\delta} - d_{j+\delta}Q[j-2]) \\ +2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1}Q[j-2]) - 2q_jD[j+1] \end{aligned}$$

and therefore:

$$\begin{aligned} W[j+1] = 4(W[j-1] - q_{j-1}D[j+1]) + 2^{-\delta+1}(n_{j+\delta} - d_{j+\delta}Q[j-2]) + \\ + 2^{-\delta}(n_{j+\delta+1} - d_{j+\delta+1}Q[j-2]) - 2q_jD[j+1] \end{aligned} \tag{3.5}$$

which makes the computation of $W[j+1]$ independent of the quotient digits computed based on $W[j-1]$ and input values.

Based on this last recurrence equation we can define another one that puts the binary digits together:

$$\begin{aligned} W[j+1] = 4(W[j-1] - (q_{j-1} + 2^{-1}q_j)D[j+1]) + \\ + 2^{-\delta}(n_{j+\delta} + 2n_{j+\delta+1} - (d_{j+\delta} + d_{j+\delta+1})Q[j-2]) \end{aligned} \tag{3.6}$$

and then translates it to radix-4 digits and radix-4 steps (k) as follows:

$$W[k] = 4(W[k-1] - q_k D[k]) + 4^{-\delta_2}(n_{k+\delta_2} - d_{k+\delta_2} Q[k-1]) \tag{3.7}$$

where $\delta_2 = \lceil \delta/2 \rceil$, and $D[k] = \sum_{i=0}^{k+\lceil \delta_2 \rceil} d_i 4^{-i}$, and $Q[k] = \sum_{i=0}^{k} q_i 4^{-i}$, and $d_i, q_i \in \{-3, -2, ..., 2, 3\}$.

Some important observations come out of this last recurrence equation for radix-4 on-line division. The on-line delay of the radix-4 on-line division using this technique is only 3!!! Much less than the minimum values presented in several other publications, where an on-line delay of 4 is only obtained for $r = 16$ and maximum redundancy in the digit set (with $r^{-2} \leq |D| < 1$) [22]. This is a very good result, based on the fact that this design is significantly simpler than designs using pre-scaling.

3.1.4. General organization

The proposed organization of a radix-4 on-line division unit is shown in Figure 3.1. It shows two quotient digit selection blocks (regular radix-2 selection and predictive quotient digit selection) and two unfolded iterations of the radix-2 recurrence equation for radix-2 on-line division, as derived in Eq. 3.5. The *predictive digit selection* block is explained in the next section.

It is important to emphasize that all the digits shown in the figure are radix-2 digits in the set $\{-1, 0, 1\}$ and the references to algorithm steps (j) are related to radix-2 computation. The system performs two radix-2 iterations in one clock cycle. The input and output digits, when taken as radix-4 digits, are in the range $\{-3, ..., 3\}$.

The values of $D[j]$, $D[j+1]$ and $Q[j-2]$ come from append registers. These registers are capable of concatenating incoming digits to the least significant position of a previously received digit vector. A redundant adder is used in the design, to

make the addition process independent of the precision. In our case we consider CS adders.

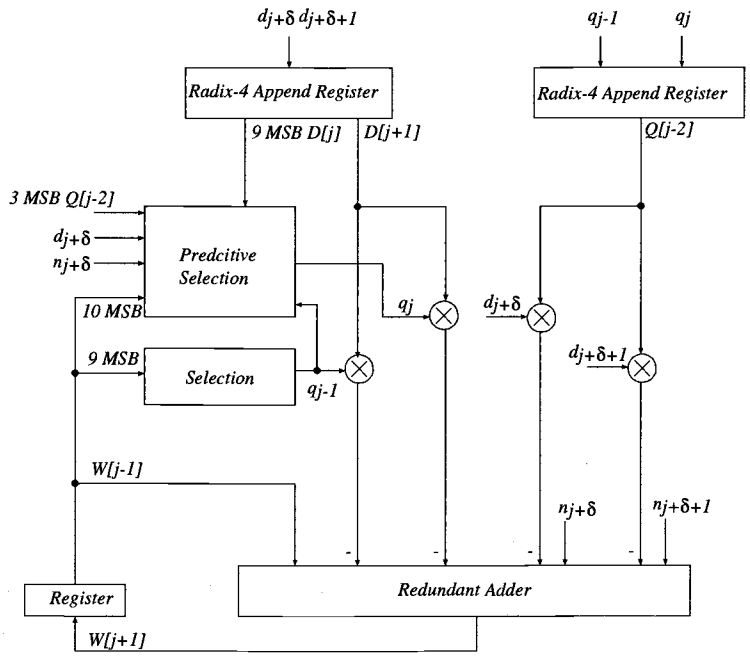


FIGURE 3.1. Organization of the Radix-4 division unit

Based on what is known about the ranges of W and the input operands, the internal representation of numbers must have at least 3 integer bits.

3.1.5. predictive quotient digit selection

The predictive quotient digit selection function is capable of generating the quotient digit q_j based on $W[j-1]$ and other information derived from the input digits. This task is accomplished by 3 simplified radix-2 modules (*prediction units*). The prediction units perform a predictive computation of a radix-2 division iteration (speculating on all possible values of a quotient digit q_{j-1}) and generate new quotient digits (q_j) for each case. The correct quotient digit q_j is then selected based on the

actual value of q_{j-1}, which is available some time after the predictive computation starts.

In order to have a simple predictive computation hardware, the prediction units must work with only the most-significant bits of the input vectors $D[j]$, $Q[j-2]$, and $W[j-1]$. Just enough to obtain a good approximation of $\hat{W}[j]$ and an accurate quotient digit (q_j). Each prediction unit needs to compute the equation:

$$\hat{W}[j]^* = 2(\hat{W}[j-1]|_v - q_{j-1}\hat{D}[j]|_v) + 2^{-5}(n_{j+5} - d_{j+5}\hat{Q}[j-2]|_v) \tag{3.8}$$

where $\hat{W}[j-1]|_v$, $\hat{D}[j]|_v$, and $\hat{Q}[j-2]|_v$ are estimates of the given vectors with v fractional bits, and $\hat{W}[j]^*$ is a valid approximation of $\hat{W}[j]$.

When the selection function presented at Section 3.1.1 was defined, it was considered that only the residual in CS form, and computed with full precision, was truncated at the fourth fractional bit position. When we introduced truncated values for $D[j]$, $Q[j-2]$, and $W[j-1]$ we also introduced approximation errors that make $\hat{W}[j]^* \neq \hat{W}[j]$, and therefore, the same selection function presented for radix-2 cannot be used in the predictive digit selection. We call the selection function for the predictive computation as $SEL2$. The actual error resulting from the truncated input operands will depend on the number representations used for them and for this reason this discussion is left for Section 3.2, where details of the circuit are provided.

The computation of the actual q_{j-1} happens in parallel with the predictive computation of q_j, and once this value is known, one of the predicted q_j digits is selected. Figure 3.2 shows the block diagram for this module with the recurrence equation implementation. Observe that the module that computes $-d_{j+\delta}\hat{Q}[j-2]|_v + n_{j+\delta}$ is not necessary since this value is computed in full precision in another part of the system. The adders are short precision CS adders.

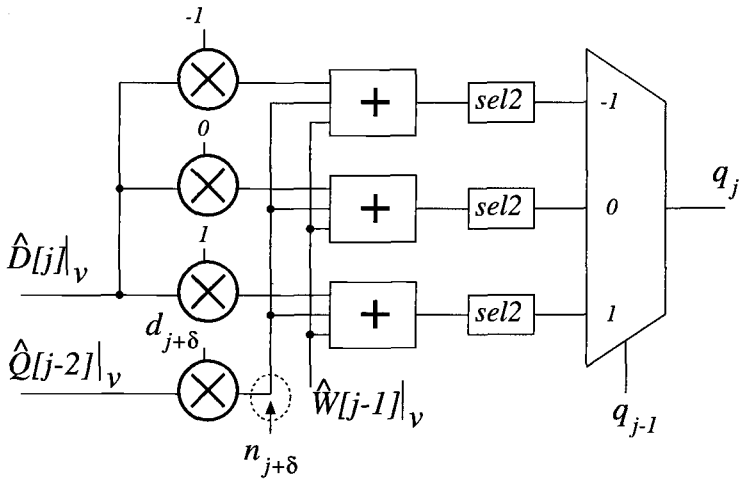


FIGURE 3.2. Selection function based on predictive calculation of q_j.

3.2. Circuit implementation

In this section we provided implementation details of the proposed radix-4 design. A direct implementation of the organization proposed in Section 3.1.4 generated poor results. The solution presented in this section is the result of several stages of refinement. The circuit that implement radix-4 on-line division is shown in Figure 3.3.

Some important decisions are the following:

- only the append register for vector D includes OFC. The append register for Q uses a Borrow-Save (BS) append register without conversion to non-redundant form;
- addition of a digit n_j and d_jQ is done with minimal extra hardware;
- a pipelined structure was created to reduce the critical path and improve throughput;

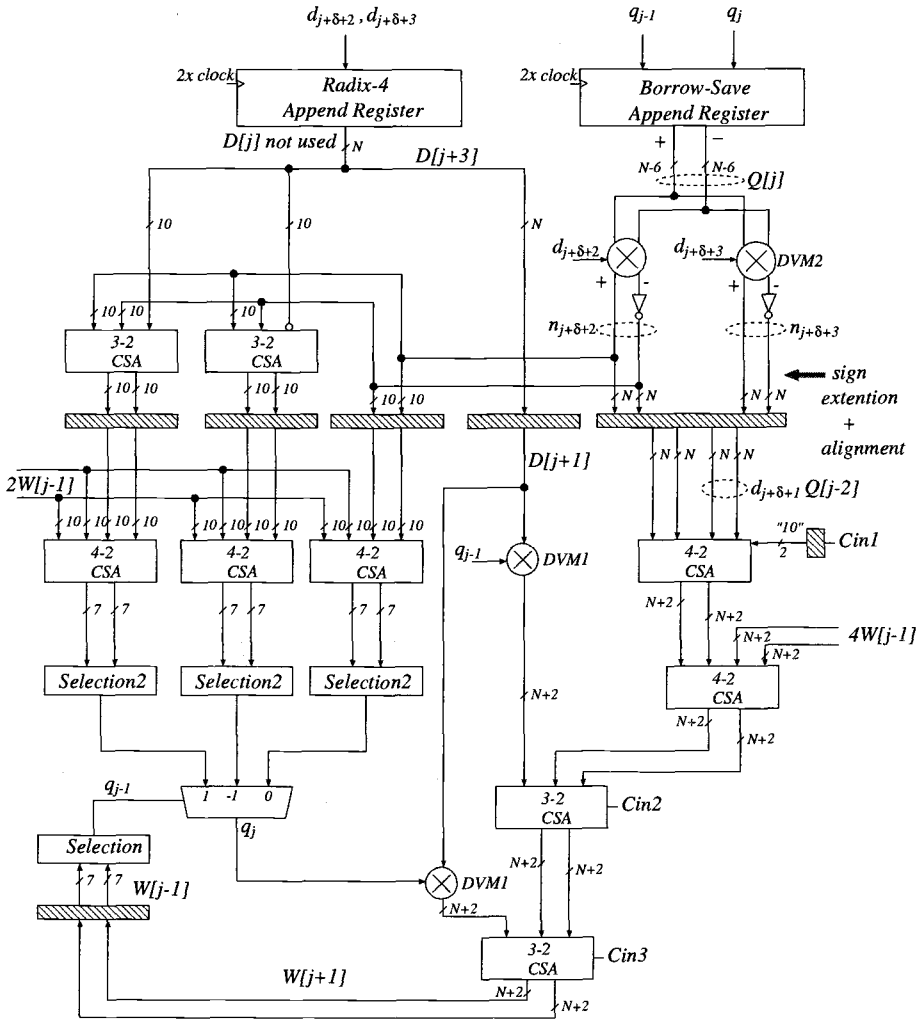


FIGURE 3.3. Circuit for radix-4 on-line division

- only CS adders should be used to compute the recurrence equation, therefore, the conversion from BS code to CS code had to be included in the design (Section 3.2.2).

3.2.1. Radix-4 append registers

It was a design decision to have non-redundant representation for the input operands, this way reducing the complexity of the redundant adders.

The append register for D (shown in Figure 3.4) uses a clock frequency that is twice the frequency used in other parts of the on-line divider. These two frequencies are easy to obtain in a network of on-line modules that include fast radix-2 modules (working in a $2\times$ clock). The conventional "radix-4" append registers were built on top of the radix-2 append register architecture proposed in other publications [26, 23]. An extra register loads its input value at half the frequency of the radix-2 append register this way storing the value of D each time a new pair of radix-2 digits are received.

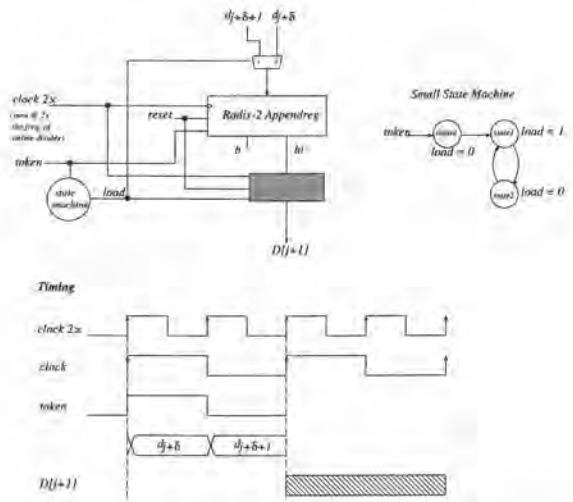


FIGURE 3.4. Radix-4 append register for $D[j+1]$

Another design decision is related to the use of $D[j]$. The value $D[j+1]$ is a better approximation of the actual D than $D[j]$. For this reason, $D[j]$ can be

replaced by $D[j+1]$ in the equations for the predictive calculation of q_j, and the predictive selection function can use only $D[j+1]$.

The straightforward implementation of the proposed design lead us to initially use the same append register to store Q. However, the result was not as efficient as we would like it to be. The critical path in the divider would start in the append register for Q, propagate through the predictive selection function (2 levels of 3-2 CSA + SEL + 4-1 MUX), and then return to the same append register which also requires logic to compute the two's complement representation of Q before the value is registered.

After several design configurations we decided to use $Q[j-2]$ vector in borrow-save format and a radix-4 append register for digits in BS code. By using this solution the implementation of the append register became very simple and the logic required to obtain Q was significantly reduced.

3.2.2. BS to CS conversion

Since the value of Q is available in BS code, and we would like to use simple CS adders in the design, we decided to convert the value of Q in BS code to CS code. The value of Q in BS code is given by two vectors Q^+ and Q^- that represent positive integers, such that the value of Q is obtained from the value of each vector, let's say q^+ and q^-, calculating $q^+ - q^-$. The conversion to CS form is easily performed based on the fact that two's complement is used in the design, and therefore, one of the vectors for the CS representation is the same as Q^+ and the other is $\overline{Q^-} + 1$, where $\overline{Q^-}$ represents bit complementation and '+' represents addition. The 2-bit signal $cin1$ in Figure 3.3 carries the 1 values needed for the two conversions required in the design.

This is a clear example of the advantage in using several number representations in a design to attain better performance.

3.2.3. Combining a digit n with vector dQ to obtain $n + dQ$

Directly inserting the digits $n_{j+\delta}$ and $n_{j+\delta+1}$ into the adder structure would increase the number of adder levels and significantly impact performance. For this reason, the inclusion of one of these digits in the computation (lets use n as a general reference) is done in the same way proposed in [16], where the digit value is combined with the vector dQ using a simple logic network. Let's call $dQ = P = (p_s p_0.p_1 p_2 ...)$, and let's consider the digit n represented in two's complement by the pair (n_s, n_0). The addition $P + n$ is given by the bit vector $(xy.p_1 p_2 ...)$ such that:

$$x = n_s + n_0' \cdot p_s$$

and

$$y = n_0 \oplus p_s$$

3.2.4. Digit-by-vector multipliers

There are two types of digit-by-vector multipliers (DVMs) used in the proposed design. DVM1 takes a vector in two's complement form and multiplies by a signed bit (a digit in the set $\{-1, 0, 1\}$) generating an output in two's complement. DVM2 takes a vector in BS form and multiplies it by a signed bit generating a vector in BS form. Call the input vector X and multiplier digit y.

Since DVM1 receives the input vector in two's complement form, we call the bit at position i as x_i. The input digit y comes to the unit using a 2-bit code (n, g),

where $n = 1$ indicates $y < 0$ and $g = 1$ indicates $|y| = 1$ (same as two's complement). Carry-input signals into the adders ($cin2$ and $cin3$) are used to obtain the change of sign for the input vector when $y < 0$. A single output bit z_i of DVM1 is implemented by the following switching expression:

$$z_i = (x_i \oplus n) \cdot g$$

For DVM2, a digit position i of the input vector is given as $x_i = (x_i^+, x_i^-)$, and the multiplier digit is given as $y = (y^+, y^-)$. The output bit of this DVM at position i is generated as:

$$z_i^+ = x_i^+ \cdot y^+ + x_i^- \cdot (y^+)'$$

$$z_i^- = x_i^+ \cdot y^- + x_i^- \cdot (y^-)'$$

3.2.5. Selection function SEL2

The selection function used to generate q_j, namely SEL2, is a little more complex than a regular radix-2 selection function. The extra complexity comes from the errors introduced by the truncation of the input operands used to compute the estimate $W[j-1]^*$ (as discussed in Section 3.1.5). To compensate for the truncation errors we use $v = 7$ bits in the operands used to compute $W[j-1]^*$, and then truncate $W[j-1]^*$ at position $t = 5$. Therefore, SEL2 has one more bit in the carry-ripple adder to obtain the non-redundant representation of $W[j-1]^*$, as compared to the original selection function. The selection is:

$$q_{j-1} = \begin{cases} 1 & \text{if } \hat{W}[j-1]^* \geq \frac{1}{8} \\ 0 & \text{if } -\frac{1}{8} \leq \hat{W}[j-1]^* < \frac{1}{8} \\ -1 & \text{if } \hat{W}[j]^* < -\frac{1}{8} \end{cases}$$

and it is implemented by the following switching expressions, considering that the non-redundant representation of $W[j-1]^*$ is given as $(w_s w_{-1} w_0 . w_1 w_2 w_3)$:

$$z^+ = en\_sel(SignD \oplus (w'_s(w_{-1} + w_0 + w_1 + w_2 + w_3)))$$
$$z^- = en\_sel(SignD \oplus (w_s(w'_{-1} + w'_0 + w'_1 + w'_2 + w'_3)))$$

where *en_sel* is used to enable the selection function (this signal is 0 during the first δ clock cycles), *SignD* is the sign of the divisor, and the output digit is generated using BS code.

3.2.6. Pipeline

The introduction of a time barrier in the middle of the datapath was the last phase in the performance improvement of this radix-4 design. The two-stage pipeline reduced and balanced the delay paths in the circuit. As shown in Figure 3.3, the output of the append registers and the index of input digits were adjusted to represent the correct system operation. In particular, the output of the append register for Q has an output that depends on the input in this case, while the output didn't depend on the input in the previous case ($Q[j]$ instead of $Q[j-2]$). Other cases are indicated in the Figure.

3.3. Comparison with other designs

The radix-4 on-line divider by overlapped replication has the following critical path:

4-2 CSA $\rightarrow$ SEL2 $\rightarrow$ 4-1 MUX $\rightarrow$ BS-APPREG $\rightarrow$ DVM_2 $\rightarrow$ 3-2 CSA

where DVM_r is the DVM component for radix-r digit.

The critical path of the non-pipelined radix-2 on-line divider is:

$$\text{SEL} \rightarrow DVM_2 \rightarrow \text{3-2 CSA} \rightarrow \text{3-2 CSA}$$

Cancelling out common delay terms between the two and assuming negligible delay for the BS-APPREG (append register for BS format), we obtain:

$$\text{4-2 CSA} + \text{SEL2} + \text{4-1 MUX}$$

versus

$$\text{SEL} + \text{3-2 CSA}$$

The difference between SEL and SEL2 is a full adder delay (3-2 CSA) given that SEL2 uses 7 bits of the residual estimate as opposed to 6 bits used in SEL. Therefore SEL + 3-2 CSA = SEL2 and the difference in the critical path delay between the radix-4 and radix-2 designs is estimated as 4-2 CSA + 4-1 MUX. Considering the delay of a 4-2 adder as 1.5 × 3-2 CSA, and the delay of a 4-1 MUX as 0.5 × 3-2 CSA, and considering that the delay of the radix-2 design is around 9 × 3-2 CSA (6 for the SEL, and 1 for the DVM), it is estimated that the radix-4 design will have 22% more delay than the radix-2 design. However, since the radix-4 design process twice as much bits then the radix-2 design, it is expected that it shows 64% more throughput for large precision of the operands. For small precision values, the on-line delays will impact the performance, and make the throughput smaller than this maximum estimate.

The critical path for another radix-4 design that uses pre-scaling [10] is given as:

$$q_j \rightarrow APPREG \rightarrow DVM_4 \rightarrow \text{3-2 CSA} \rightarrow \text{3-2 CSA} \rightarrow \text{SEL} \rightarrow q_{j+1}$$

The normal APPREG doesn't have a negligible delay. Since the SEL function for this design is slightly more complex than the radix-2 design, it is expected that the proposed radix-4 design have at most 20% more delay than the radix-4 divider under comparison. The worst-case on-line delay for the other on-line divider is 8, excluding the reset and start cycles required to complete and begin a division operation. Thus $N+8+2 = N+10$ clock cycles are used for each division operation (worst-case), where N is the number of digits in the operands. Given this large delay, the throughput won't be very good for the design using pre-scaling when the operand precision is small. Based on the previous considerations, we can estimate that the throughput of the proposed design is 14.5% bigger for 16-bit operands than the design using pre-scaling. For long precision, the proposed design looses its gain. Besides, the control system for the design using pre-scaling is significantly more complicated.

3.4. Experimental results

The proposed design was described using VHDL. Functional simulation was used to validate the design operation. The synthesis tools considered in this work are:

- FPGA Express Xilinx Edition 3.6.1;
- Leonardo Spectrum Level 3 v20001a2.72 (ASIC), from Mentor Graphics.

For the FPGA synthesis, optimization was set for speed and effort was high for the Xilinx chip XC4062XLA-09bg560. For the ASIC synthesis, the AMI05 fast CMOS library ($0.5\mu m$ CMOS) was used along with area and speed optimizations.

Unfortunately, when performing synthesis using the FPGA Express tool, we encountered inefficiencies with respect to how the designs were synthesized. It was

unpredictable as to what types of delays would be generated from certain components. For example, in one case a 4-2 carry-save adder within the critical path generated an unacceptable delay of just over 4ns whereas a 3-2 carry-save adder within the same critical path only had a 0.77ns delay. This behavior did not exhibit how a 4-2 carry-save adder can be better in delay than two 3-2 carry-save adders. Most of the delay was not related to the logic of the carry-save adder itself but rather routing.

Obviously for FPGA designs the delays are greatly dependent on the the actual place & route and technology being used. Post-map timing analysis already contained substantial delays whereas the Post-place & route timing analysis gave results that were extremely slow for radix-2 or radix-4 cases. We only take into consideration the post-map timing in our report.

ASIC synthesis did provide much better results. The theoretical gain which we could estimate by the difference in critical paths of both designs was tangibly observable in the ASIC synthesis results. This made much more sense when compared to what we got for the FPGA synthesis.

The compiled tables of results are listed below for the final design of the radix-4 on-line divider.

CMOS AMI05 - Radix-4 Divider					
operand length (bits)	12	16	24	32	36
Area (gates)	1954	2515	3644	4771	5337
max delay (ns)	14.8	15.35	16.01	16.49	16.59
max clock rate (MHz)	67.57	65.15	62.46	60.64	60.28

CMOS AMI05 - Radix-2 Divider (non-pipelined)					
operand length (bits)	12	16	24	32	36
Area (gates)	953	1275	1918	2565	2883
max delay (ns)	11.13	10.21	10.9	11.03	13.24
max clock rate (MHz)	89.85	97.94	91.74	90.66	75.53

CMOS AMI05 - Radix-2 Divider (pipelined)					
operand length (bits)	12	16	24	32	36
Area (gates)	974	1294	1935	2581	2902
max delay (ns)	10.57	11.25	11.28	11.12	11.25
max clock rate (MHz)	94.61	88.89	88.65	89.93	88.89

Xilinx FPGA - Radix-4 divider					
operand length (bits)	12	16	24	32	36
Area (CLBs)	268	327	448	568	628
max clock rate (MHz)	47.863	47.863	47.863	47.863	47.863

Xilinx FPGA - Radix-2 divider (non-pipelined)					
operand length (bits)	12	16	24	32	36
Area (CLBs)	83	111	167	223	251
max clock rate (MHz)	88.449	88.449	88.449	88.449	88.449

The *relative gain* in terms of bit-rate of the radix-4 on-line divider over the radix-2 on-line divider are defined to be the bit-rate of a radix-4 system compared to that of a radix-2 system. Basically, if the operating frequencies were the same for both cases, the radix-4 design would double the bit-rate. Thus the relative gain can be obtained by the formula:

Relative gains of Radix-4 divider over Radix-2-np divider - CMOS AMI05				
12	16	24	32	36
%50.4	%33	%36.17	%33.77	%59.59

TABLE 3.1. Radix-4 and Radix-2 non-pipelined on-line divider for CMOS

Relative gains of Radix-4 divider over Radix-2-p divider - CMOS AMI05				
12	16	24	32	36
%42.84	%46.59	%40.91	%34.86	%35.61

TABLE 3.2. Radix-4 and Radix-2 pipelined on-line divider for CMOS

$$\frac{2 \cdot (\text{R4-freq.}) - (\text{R2-freq.})}{(\text{R2-freq.})}$$

Table 3.1, 3.2, 3.3 are the relative gains of the radix-4 divider compared to the radix-2 dividers (np→non-pipelined, p→pipelined).

There seems to be no particular pattern to the relative bit-rate gain which makes it hard to gauge how efficient the radix-4 system is compared with the radix-2 system. It is important to note that the relative gain looks only at the speed. In order to compute the actual throughput, it is necessary to include the on-line delay. Besides that, the relative gain does not consider the divider area. The listed gains are fairly expensive, nearly double the area of the radix-2 design on average for the ASIC results and from 2.5 to 3.25 the area for the FPGA implementation.

Relative gains of Radix-4 divider over Radix-2-np divider - FPGA				
12	16	24	32	36
%8.23	%8.23	%8.23	%8.23	%8.23

TABLE 3.3. Radix-4 and Radix-2 non-pipelined on-line divider for FPGA

3.4.1. Throughput

The number of cycles required to complete an on-line division is given as:

$$C = 2 + \delta + N$$

where δ is the on-line delay, N is the number of output digits, and the 2 extra cycles are caused by a *reset* cycle and a *start* cycle. The throughput (10^6 divisions/sec = Mdiv/sec) is computed as the maximum clock rate divided by the total number of cycles to complete an operation (C).

The following results show the throughput of the Radix-4 and Radix-2 dividers from ASIC synthesis. Table 3.4 shows gain in throughput For radix-4, $\delta = 3$ and for radix-2, $\delta = 5$.

In the case of the radix-2 pipelined version the on-line delay is 6. The gains in throughput when using the radix-4 design are shown in table 3.5:

The throughput of the on-line dividers when synthesized for the FPGA gave the results as shown in table3.6:

Table3.6 which shows that there is no gain in using the proposed radix-4 design when the target technology is FPGAs, without further optimization in the VHDL code.

operand length (bits)	12	16	24	32	36
Radix-4 (Mdiv/sec)	6.14	5.01	3.67	2.89	2.62
Radix-2-np (Mdiv/sec)	4.73	4.26	2.96	2.32	1.76
Gain	%29.81	%17.61	%24	%24.57	%48.86

TABLE 3.4. Throughput comparison of Radix-4 and Radix-2-non-pipelined on-line divider for ASIC

operand length (bits)	12	16	24	32	36
Radix-4 (Mdiv/sec)	6.14	5.01	3.67	2.89	2.62
Radix-2-p (Mdiv/sec)	4.73	3.70	2.77	2.25	2.02
Gain	%29.81	%35.41	%32.49	%28.44	%29.70

TABLE 3.5. Throughput comparison of Radix-4 and Radix-2-pipelined on-line divider for ASIC

operand length	12	16	24	32	36
Radix-4 (Mdiv/sec)	4.35	3.68	2.82	2.28	2.08
Radix-2-np (Mdiv/sec)	4.66	3.85	2.85	2.27	2.06
Gain	-%6.65	-%4.42	-%1.05	%0.44	%0.97

TABLE 3.6. Throughput comparison of Radix-4 and Radix-2-non-pipelined on-line divider for FPGA

4. HIGH-THROUGHPUT NETWORKS OF ON-LINE ARITHMETIC MODULES FOR DSP APPLICATIONS

On-line arithmetic modules have an intrinsic on-line delay that impacts the maximum throughput of networks using these modules can reach. The problem is particularly serious for small precision calculation or deep-pipelined networks. In this chapter a solution to the problem is presented. Results of an actual implementation of the solution for some basic arithmetic operators are shown along with the benefits of the proposed approach. The developed modules are also tested within the framework of an image processing algorithm.

In the next section a detailed explanation of the impact of on-line delays on network performance is shown. In Section 4.2, a general solution to improve the throughput of on-line adders, multipliers, and dividers is discussed. After that, the new designs based on the given solution are presented in detail in Section 4.3. Experimental results and set-up, are given in Section 4.4. Section 4.5 provides a case study of the on-line modules discussed in this paper in the implemention of an image processing algorithm [27].

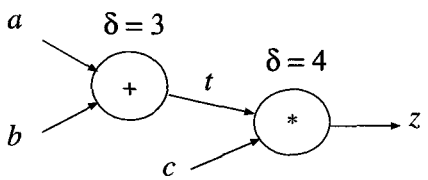
4.1. Impact of on-line delay on network performance

To compare the design options of on-line networks we use the following parameters:

- *Latency*: number of clock cycles between the first digit of the input and the last digit of the output.
- *Throughput*: the number of input operands the network can process per unit time. In on-line networks, the throughput is limited by the component that is busy for the most clock cycles.

- *Area*: number of equivalent gates used to implement the network.

Consider a simple network as shown in Figure 4.1. The network is manipulating 8-bit fractional numbers. It is easy to derive an abstract timing diagram, as shown in Figure 4.1, that shows the clock cycle count and the position (weight) of every digit for inputs, intermediate results, and outputs. The diagram also shows the cycles for which each module is busy (shaded lines), during the present computation. Since the on-line multiplier is the module that is busy for the longest time in this network, it becomes the limiting module for throughput.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
OL-adder																		
Inputs (a,b)	1	2	3	4	5	6	7	8										
Output (t)				0	1	2	3	4	5	6	7	8						
busy																		
OL-mult																		
Input (t,c)				0	1	2	3	4	5	6	7	8						
Output (z)									-1	0	1	2	3	4	5	6	7	8
busy																		

FIGURE 4.1. Network of on-line modules.

The latency of this network is 18 clock cycles. The network can start to work with another input every 16 clock cycles because the multiplier needs a reset cycle, resulting in a throughput of 0.06 inputs/cycle.

It is important to point out that the on-line multiplier is not receiving any input after cycle 12, and it is only generating output digits based on previous data. Thus, if we are able to decouple the generation of the initial state $W[0]$ for the next multiplication from the generation of output digits of the present multiplication, we could overlap these two phases, and improve the throughput. Another advantage

would be the ability to remove the reset cycle from the multiplier operation, that comes as a result of the overlapped operation.

In our example, the on-line multiplication with decoupled initialization phase could start as soon as cycle 14, with the on-line adder starting at cycle 11. The latency would be the same, but the throughput would be improved to one input every 10 cycles (0.1 inputs/cycle), which corresponds to a network that is 60% faster than the original network. Of course this conclusion comes from the assumption that the clock cycle is not affected much by the proposed modifications, which is not exactly true. The experimental results show the actual figure. It is also the case that such gain is reduced when numbers with more fractional bits are used, but for DSP applications where normally a small number of bits per operand is used this overlapping is very beneficial. Besides that, The improvement dependents on the network topology, and the type of on-line arithmetic modules involved.

4.2. Decoupling initialization and output generation

As already suggested, this proposal to improve the network throughput is based on overlapping the phase when the on-line module is receiving initial digits (over the initial δ clock cycles) and the phase when the module is generating the last output digits of the previous computation. This section identifies the modifications that need to be made to the on-line modules in order to obtain an efficient design for the proposed solution.

4.2.1. Addition, multiplication and MAC

For on-line addition, multiplication, and multiply-add modules, the recurrence equations take the form:

$$W[j] = r \cdot (W[j-1] - z_{j-1}) + \mathcal{F}(inputs) \tag{4.1}$$

where z_{j-1} are output digits, and W is the residual. When the inputs are not applied anymore, the recurrence is reduced to only $W_k[j] = r \cdot (W_k[j-1] - z_{j-1})$, where W_k is a truncated version of the residual with $k > t$ fractional bit positions. The value of k depends on the on-line delay and the number of cycles during which the overlapped operation takes place.

Using this simplified recurrence equation, and the condition that $|W[j]| < 2$ after each iteration of the recurrence equations shown in Table 2.2 when $r = 2$, it is possible to develop a simpler selection function relaxing the conditions to obtain the output digit. The overlap between the two phases cannot be more than $\delta+1$ cycles, so the number of fractional digits in the residual estimate W_k is small (slightly more than $\delta+1$). In this case, a Carry Ripple Adder could be used to assimilate the carries of any redundant representation of the residual. Once the non-redundant residual is obtained, the selection function is as shown in Table 4.1. It is very similar to the selection function presented in Section 2.2.3, however, (i) less fractional bits are needed in the residual estimate and (ii) it is the same for all the operations considered in this section, since all the recurrence equations are reduced to the same simple form (Eq.(4.1)) after the input digits become zero. For example, consider Eq. (4.1) and a residual value of $W_k[j-1] = (W_k[j-1]_1, W_k[j-1]_0, W_k[j-1]_{-1}, ...) = 10.1xyz$, since $|W_k[j-1]| < 2$ (two integer bits in TC) we could generate $z_{j-1} = -1$, and $W_k[j] = 11.yz$. The other cases are shown in Table 4.1.

From Table 4.1 it is easy to conclude that an output digit represented in BS form as $z_j = (z_j^+, z_j^-)$ is defined by the expressions:

$$z_{j-1}^+ = W_k[j-1]_1' \cdot W_k[j-1]_0 \tag{4.2}$$

and

Residual represented in NR form

MS dig. $(W_k[j-1])$	MS dig. $(W_k[j])$	z_{j-1}
00.0x	00.x	0
00.1x	01.x	0
01.0x	00.x	1
01.1x	01.x	1
10.0x	10.x	-1
10.1x	11.x	-1
11.0x	00.x	-1
11.1x	01.x	-1

TABLE 4.1. Simplified selection function

$$z_{j-1}^{-} = W_k[j-1]_1 \tag{4.3}$$

For output digits in TC form the expressions are:

$$z_1 = W_k[j-1]_1 \tag{4.4}$$

and

$$z_0 = W_k[j-1]_1 + W_k[j-1]_0 \tag{4.5}$$

Observe that a selection by truncation of the residual value is not enough. To implement the selection we cannot use only a simple shift register and take the MS digit because the addition of $-z_{j-1}$ to $W_k[j-1]$ must be performed in each step in order to obtain $W_k[j]$. It is necessary to process the most-significant bits of the residual in order to perform the correct calculation of the new residual. This behavior may be observed from Table 4.1. Bit 0 of $W_k[j]$ is equal to bit 1 (first fractional bit) of $\hat{W}_k[j-1]$ and bit -1 is obtained as:

$$W_k[j]_{-1} = W_k[j-1]_1 \cdot W_k[j-1]'_0 \tag{4.6}$$

The other bits of the new residual are obtained by a left shift as follows:

$$W_k[j]_i = W_k[j-1]_{i+1} \quad ; \quad W_k[j]_k = 0 \tag{4.7}$$

with $0 \leq i \leq k-1$.

Another important fact is that the initial condition for these operations may be computed at the same time the output digits for the previous operation is still being generated. They are given as $W[0]$ in Table 2.2. The initial conditions can only be obtained when almost all the hardware usually used for the operation is available. Only the selection function is deactivated while the initial condition is computed. Based on this observation we propose the solution to improve throughput. Once another sequence of input digits is about to start a copy of the residual ($W_k[j]$) is made and a separate hardware takes care of the output digit selection and computes the next residual, while the usually used hardware builds the initial condition for the new operation. A more detailed explanation of this approach is discussed in the Section 4.3.

4.2.2. Radix-2 on-line division

On-line division unit has one of the highest on-line delays and it is the slowest among most of the basic on-line modules. On-line division forms a major bottleneck when used in a network of on-line modules especially for small precision applications. A non-pipelined on-line division module [28] has an on-line delay of 5 clock cycles.

The recurrence equation of an on-line division is given by

$$W[j] = 2(W[j-1] - q_{j-1}D[j]) + 2^{-\delta}(n_{j+\delta} - d_{j+\delta}Q[j-2]) \tag{4.8}$$

where $\delta = 5$. The computation of Eq. 4.8 can be split into 3 phases:

- *Phase (A)*: When $q_{j-1} = 0$, i.e before the beginning of generation of output digits (occurs during first 6 clock cycles starting from first input digit) the recurrence equation is:

$$W[j] = 2(W[j-1]) + 2^{-5}.(n_j) \tag{4.9}$$

The above equation is nothing but accumulation of numerator such that $W[5] = 2^{-5}.(2^5 n_0 + 2^4 n_1 + 2^3 n_2 + 2^2 n_3 + 2 n_4 + n_5)$

- *Phase (B)*: When q_{j-1} is being generated at the same time the input digits are being applied. The recurrence equation given by Eq. 4.8 is used in its complete form.

- *Phase (C)*: When all the input digits have been applied (occurs when $n_{j+\delta} = d_{j+\delta} = 0$). The recurrence equation is reduced to:

$$W[j] = 2(W[j-1] - q_{j-1}D[j]) \tag{4.10}$$

Phases (A) and (C) are totally independent and can be executed simultaneously for different sets of inputs. We take advantage of this fact to hide the on-line delay between two consecutive computations.

4.3. Design of new on-line modules

This Section presents the design of improved modules for on-line operations that use the overlapping technique presented in the previous sections.

4.3.1. General design

A general solution applicable to on-line adders, multipliers and MACs listed in Section 2.2.3 is shown in Figure 4.2.

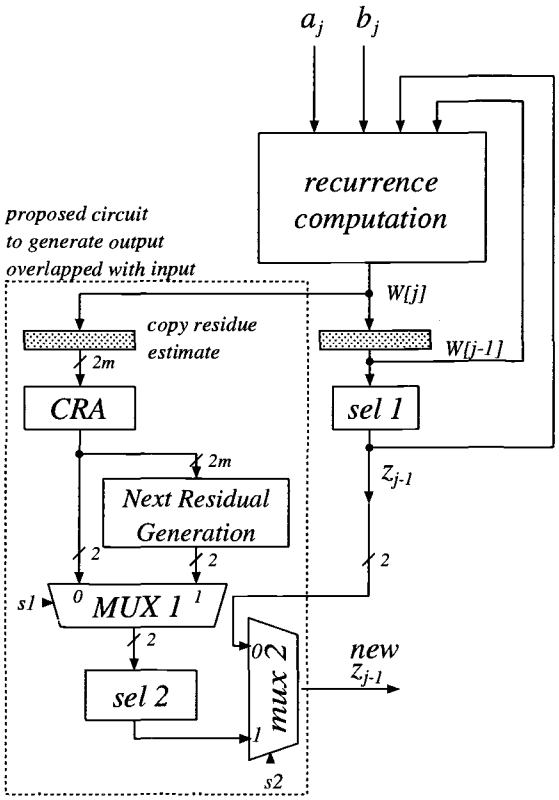


FIGURE 4.2. Improved design for on-line adder, multiplier, and mac modules

The dashed box in Figure 4.2 shows the extra circuit that is being proposed where $m = \delta + 4$. When it is used, the original on-line circuit is freed to work on the new input data, while the output digits continue to be generated by $Sel2$. The control signals for the modules must be applied in the following sequence (one for each clock cycle):

- Register a copy of the MS digits of the residual ($W_k[j-1]$) into the copy register and switch MUX2 to take the output of $Sel2$ instead of $Sel1$ ($s2 = 1$);
- Start the "Next Residual Generation" block. Keep $s2 = 1$ and $s1 = 0$;

- For the following δ cycles generate a new residual value (activating the "Next Residual Generation" block) and keep $s2 = s1 = 1$.

- Switch MUX2 back to $Sel1$. The output digits for the new set of inputs are already coming.

The block diagram for the "Next Residual Generation" is shown in Figure 4.3. It implements equations (4.6) and (4.7). The multiplexer was included because the value of the MS bit of $W_k[j]$ must be immediately updated from the non-redundant representation of the residual estimate coming from the CRA. Since the selection function is implemented by a simple two-input gate, only the two MS bits of the non-redundant residual are used (outputs of this module).

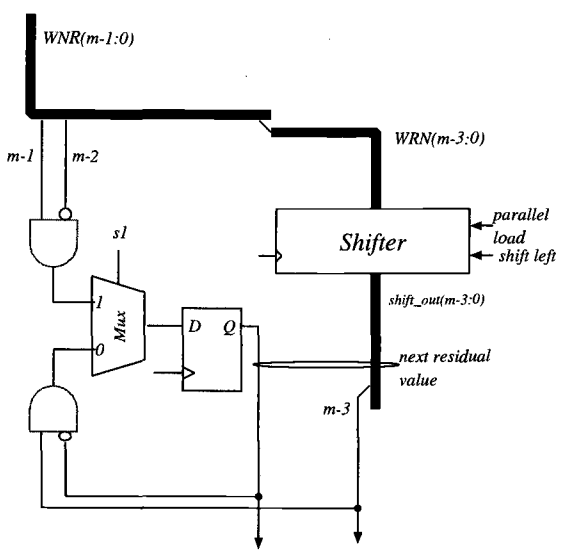


FIGURE 4.3. Generation of Next Residual

The block named $Sel2$ implements Eq. (4.2) and Eq. (4.3) when the output digit is coded in BS form, or Eq. (4.4) and Eq. (4.5) when the output digit is coded in TC form.

The simulation of the improved on-line multiplier is shown in Figure 4.4. The trace shows the same points (A to E) used for the on-line adder. The input operands are $x1 = 0.11\bar{1}11\bar{1}0 = 0.70312$ and $y1 = 0.\bar{1}\bar{1}011 = -0.461425$. The beginning of a data set is indicated by a *token*. *sync_reset* signal indicates that another set of inputs is about to start. Another pulse in the *token* input starts the next data set of $x2 = y2 = 0.\overline{1111111} = -0.972185$. The value $z1 = x1 * y1 = 0.\bar{1}000100 = -0.46175$ (between points B and D) matches the correct value. Note that the second output directly follows the output of the first multiplication without any delay (point D), hence we get a gain of 4 cycles for every set of inputs.

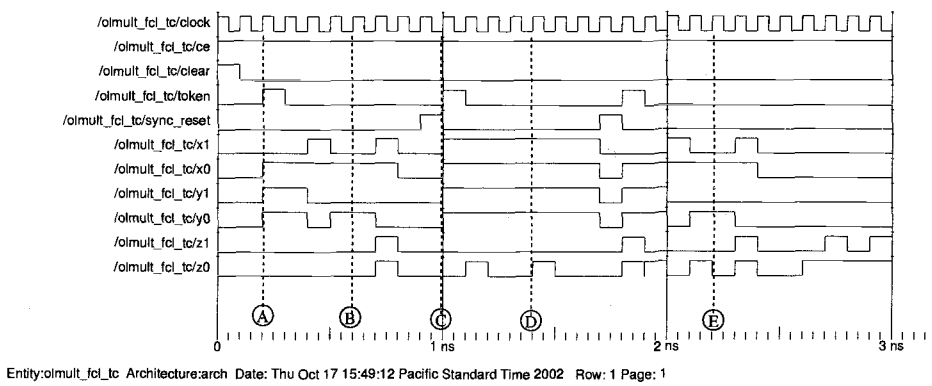


FIGURE 4.4. Simulation trace - improved on-line multiplier

4.3.2. Improved on-line adder

The on-line adders based on SD adders (called PPM - plus-plus-minus adders) do not use a recurrence equation and are very small and fast [24, 25]. For this type of adders, since they are so small, it is possible to use two adders and this way overlap the two phases (initialization and end of previous operation) as shown in Figure 4.5. When oladder1 is working and another addition is about to start, the input is

routed to oladder2. Oladder1 continues to generate the output. Some cycles later the output of oladder2 is connected to the output. A similar procedure happens when oladder2 is working and another addition starts.

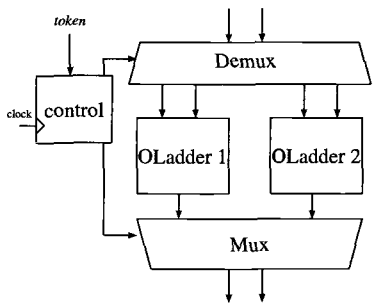


FIGURE 4.5. On-line adder for improved throughput

The simulation of the improved ol-adder is shown in Figure 4.6 for input operand $x1 = 0.\bar{1}111\bar{1}\bar{1}\bar{1} = -0.1171875$ and $y1 = 0.\bar{1}\bar{1}101\bar{1}1 = -0.6015625$, including all the internal states ($\bar{1} = -1$). Digits are presented in TC code. Observe the overlapped operation between points C and D. The *token* signal used to indicate that the first valid input digit is being applied occurs while the output digits for the first data set is being generated. The next data input is $x2 = 0.11\bar{1}\bar{1}011 = 0.5859375$ and $y2 = 0.1100111\bar{1} = 0.7890625$. The circuit output is $z1 = x1 + y1 = \bar{1}.0101\bar{1}00 = -0.71875$. Note that the second output, $z2 = x2 + y2 = 1.1\bar{1}10000 = 1.375$, directly follows the output of first addition without any delay (point D). Hence there is a gain of two clock cycles when compared to the original adder for every set of inputs. The improved adder has 25% more throughput than the original on-line adder. Dashed line on the Figure shows the following events: A - beginning of data set 1, B - beginning of output digits for data set 1, C - start of data set 2, D - beginning of output digits for data set 2 (end of outputs for data set 1), E - end of outputs for data set 2.

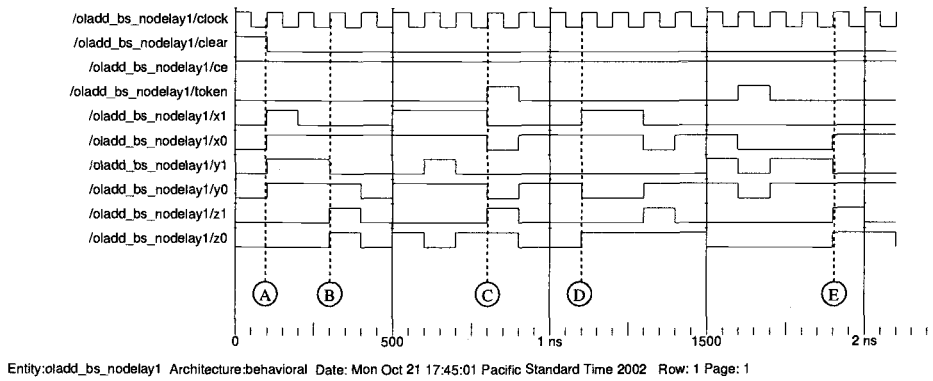


FIGURE 4.6. Simulation of improved on-line adder

4.3.3. Overlapped on-line divider

In this section we provide implementation details of the proposed radix-2 on-line division module with hidden on-line delay. Figure 4.7 shows the circuit that implements the overlapped radix-2 on-line division.

The *adder network* shown in Figure 4.7 is formed by two 3-2 Carry Save Adders (CSA). The input to these carry save adders needs to be applied in such a way that the delay is reduced. The longest delay occurs along the path of selection function generating q_{j-1}, whereas $2W[j-1]$ and the other vector $n_{j+\delta} - d_{j+\delta}Q[j-2]$ are available earlier. The input of first 3-2 CSA takes this last value and its output is combined with $q_{j-1}D[j]$ in the second adder. The append register in this circuit accepts inputs in BS form and has an on-the-fly converter which converts the input into two's complement notation [26, 23].

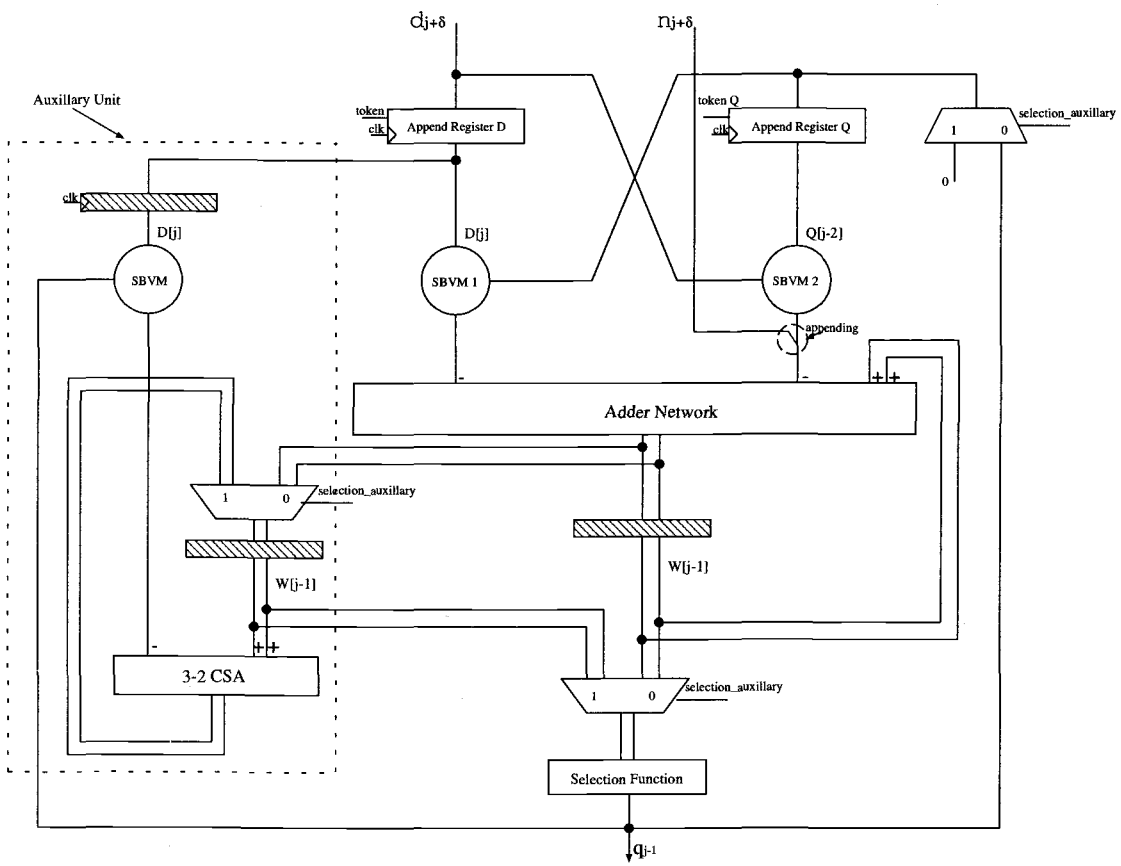


FIGURE 4.7. Circuit for radix-2 on-line division with auxiliary unit

The Auxiliary unit has a very simple circuit implementation, as shown in Figure 4.7. It implements phase C of the recurrence equation computation, which produces output digits once the input digit is no longer applied. This circuit basically implements Eq. 4.10 using a 3-2 CSA, a shifter and registers. The size of this unit is equal to either the precision of the input operand or 11 bits (whichever is smaller), and the error in the residual is kept within the limits because this circuit is active only for δ clock cycles.

Input digits are applied serially. The first input digit is applied when token is high. We force $q_{j-1} = 0$ for the first δ cycles (phase(A)) and thus SBVM1 generates 0 output. Also, since Q[j-2] is zero during this stage, the accumulation of n_j digits onto the residual register happen. Once the first output digit is obtained (after an online delay of 5 cycles) a token to Append register Q is activated and the circuit implemens the whole recurrence equation 4.8 (phase (B)). Once all the input digits are applied, a control signal $selection - auxillary$ is issued forcing to transfer the residual value to auxillary unit. During this cycle all the registers in main modules are reset. The $selection - auxillary$ signal once trigerred, is high for the next $5(\delta)$ clock cycles there by implementing phase (C) of recurrence equation. Observe that the computation on the auxillary unit overlaps with the next set of inputs which occurs simultaneously on main module. There is one cycle delay *reset* between two sets of consecutive inputs.

Figure 4.8 shows all these control signals along with input and outputs being applied for n=7 bits. In this case we have a gain of 5 cycles for every set of inputs, without considering the loss due to increase in delay there is a gain of approximately 65%.

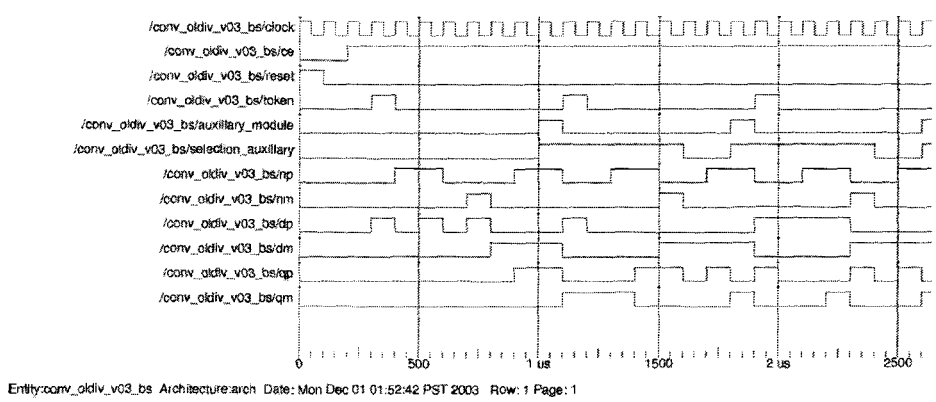


FIGURE 4.8. Waveform for radix-2 on-line division with auxillary

4.4. Implementation

The on-line modules were described using VHDL code and synthesized for Xilinx Spartan and XC4000E chips using Leonardo Spectrum Level 3 v2001a2.72 from Mentor Graphics. An ASIC synthesis using the same tool was done with CMOS AMI 0.5 micron technology optimized for speed. For FPGA technology area is referred to as Function Generators (FG) whereas the area for the ASIC design is presented as gates. Frequency is given in MHz and throughput in Msamples/second.

The number of cycles required to complete an on-line operation is given as: $C = 2 + \delta + N$ where N is the number of digits and δ is the on-line delay. The 2 extra cycles are caused by a *reset* cycle and *start* cycle. The throughput is computed as the maximum clock rate divided by the total number of cycles to complete the operation (C). In the case of our proposed architecture, due to overlapped operation the on-line delay is hidden, hence it takes a lesser number of cycles to complete a set of computations and is given by: $C = 1 + N$. Based on these observations and definitions all our results were generated.

The experimental results for the on-line adder using recurrence equation (Recurrence), the on-line adder using PPMs (PPM), and the proposed on-line adder (Improved) are shown in Table 4.2. The area of the improved on-line adder is still smaller than the adder using recurrence, with a significant advantage in speed. This is clearly a good solution. There is a gain of three clock cycles for every set of inputs when compared to the recurrence-based adder.

Clearly the proposed adder has a larger area when compared to the ppm-based on-line adder. There is a significant improvement in the throughput as well. An on-line adder never forms a bottleneck in a network of on-line modules, in terms

Technology	On-line adders								
	Recurrence			PPM			Improved		
	Area	Freq.	Thru.	Area	Freq.	Thru.	Area	Freq.	Thru.
Spartan	19	45.1	3.76	3	99.9	9.08	10	115.3	16.47
XC4000E	19	37.7	3.14	3	91.7	8.34	10	70.1	10.01
ASIC	184	181.4	15.12	54	369.5	33.59	136	226.1	32.33

TABLE 4.2. Experimental results for on-line adders

of operating frequency. There is a gain of two clock cycles for every set of inputs hence the proposed adder is the best solution.

For the on-line multiplier, the design obtained in [7] was used as a reference where both inputs are in on-line form. An important observation is related to the use of reset signals. We avoid the use of synchronous clear whenever possible. Synchronous clear forces the use of more CLBs and introduces more delay.

It was concluded in [7, 8] that RCAs and signed digits in TC form generate the best implementation of on-line multipliers for DSP on FPGAs. Our proposed multiplier was hence targeted to improve the throughput of the most efficient results available for this multiplier design. Table 4.3 shows the area (FGs), frequency (MHz) and throughput (Msamples/sec.) for the original and improved on-line multipliers, with precision of 7 bits.

The improved on-line multiplier has a better throughput than the conventional on-line multiplier at the expense of around 35% more area. The gain in throughtput for the results presented in Table 4.3 is around 60%.

Technology	On-line multipliers					
	ol-mult TC			improved ol-mult		
	Area	Freq.	Thru.	Area	Freq.	Thru.
Spartan	48	58.4	4.5	64	57.7	7.21
XC4000E	48	40.6	3.12	64	39.8	4.98
ASIC	806	116.7	8.98	1107	115.1	14.39

TABLE 4.3. Experimental results for on-line multipliers

Technology	On-line dividers					
	ol-div			overlapped ol-div		
	Area	Freq.	Thru.	Area	Freq.	Thru.
Spartan	66	33.5	2.8	104	33.7	4.2
XC4000E	66	27.8	2.32	104	28.3	3.54
ASIC	606	103.2	7.37	955	88.4	11.05

TABLE 4.4. Experimental results for on-line dividers

Table 4.4 shows the results obtained for on-line division with a precision of 7 bits. It can be observed that our proposed architecture has a significant gain in terms of throughput. The gain in throughput is approximately 50% at a cost of 57% increased area.

4.5. Case Study: Digital Signal Processing Applications

For this analysis we are interested in the color image processing algorithm presented in [27] and shown in Figure 4.9. This DSP application is an attractive candidate for on-line arithmetic because it is initiated with divsion followed by operations of mutiplication and addition. In this paper, we are concerned with the back end of the algorithm and will analyze two 9-tap spatial filter masks followed by division as shown by the dashed-line box in Figure 4.9. The filter mask denoted by $h_k[n]$ in the figure multiplies the inputs by fixed coefficients and adds the 9 results to produce the output.

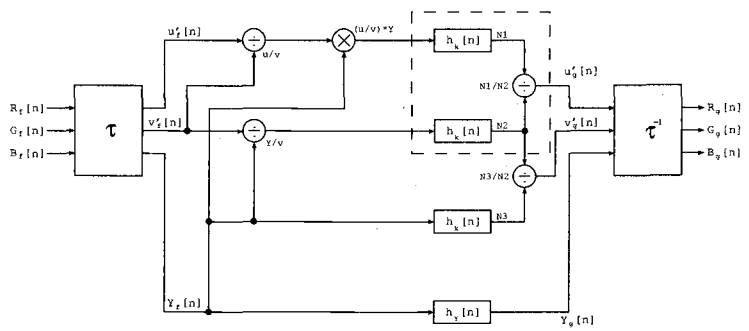


FIGURE 4.9. Image processing algorithm for processing the chromatic signals of color images

The experimental set up involves implementing the back end of the algorithm with conventional radix-2 division by digit recurrence (SRT division), conventional radix-2 on-line division, and using the improved on-line divider. Due to the input behavior of the algorithm, if the three channels (u, v, Y) are represented by 8 bits, we have nine 11-bit operands entering the filters $h_k[n]$ serially. Following the tree of adders, 15 bits emerge from the filter in on-line fashion. The numerical behavior of the algorithm ensures that the numerator (N) is always less than the denominator (D). This fact simplifies the design significantly when dealing with on-line division,

as shown in Figure 4.10(a). For SRT division shown in Figure 4.10(b) we do not know when the first significant bit coming out of the filter will be available. Therefore, we must accumulate the extended precision of 15 bits before dividing. This is done with a normalizing on-the-fly converter which contains an extra bit for the sign. The divider output is then sent to an on-the-fly converter to produce the final result.

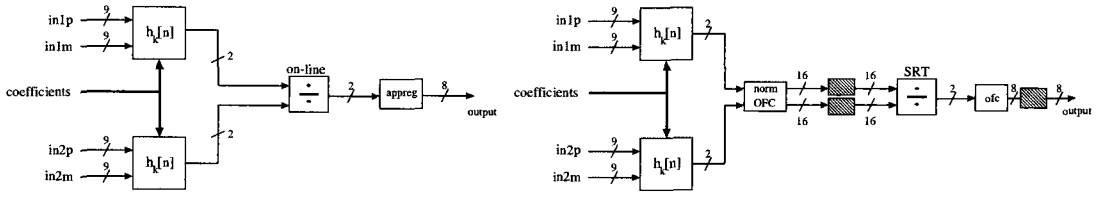


FIGURE 4.10. Hardware configurations for back end of image processing algorithm

The last module (output module) of the filter is an on-line adder. The timing diagrams of the three implementations of the image processing application are shown in Figure 4.11. We only show the timing of the on-line modules that cause the bottleneck in the network. Figure 4.11(a) gives the timing when a conventional on-line divider is used (Architecture I). Figure 4.11(b) shows the timing of the system when the final on-line adder in the filter is replaced with the improved on-line adder and on-line divider (Architecture II). Conventional SRT division is used in the diagram of Figure 4.11(c) (Architecture III).

By observing these diagrams, it is clear that when compared to using conventional on-line components, the improved on-line modules generate a gain in the perceived throughput, going from producing a pixel every 22 clock cycles to one every 16 clock cycles (a gain of nearly 40%). When on-line modules are followed by a conventional divider, the throughput is one pixel every 17 clock cycles as shown in Figure 4.11(c).

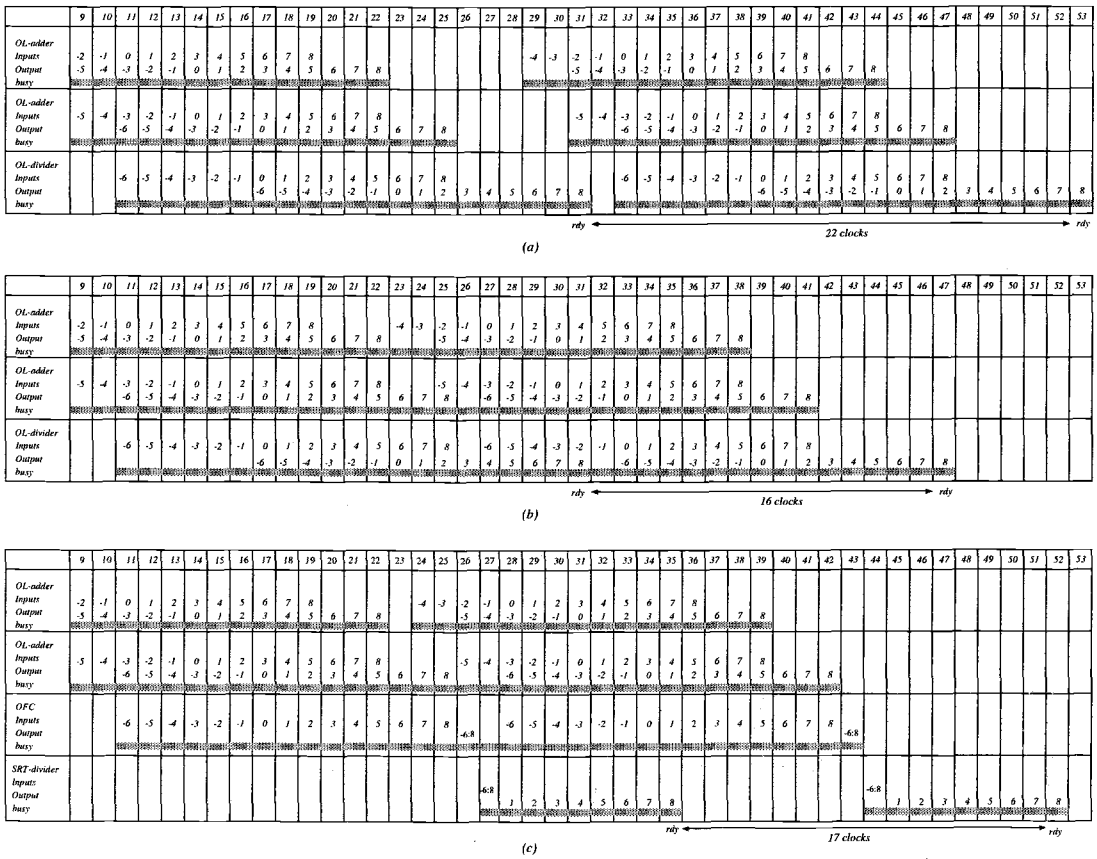


FIGURE 4.11. Timing of filter using (a) conventional on-line division (Architecture I) (b) improved-throughput on-line adder and overlapped on-line divider (Architecture II) and (c) SRT division (Architecture III)

Table4.5 shows the results obtained for the back end of the image processing application implemented with the three architectures explained above. It is clear that Architecture I has the least area and throughput. There is a gain of 10% in throughput at a cost of 10% in area when we use Architecture II over Architecture I. This gain in throughput, when we use Architecture III, is 30% more compared to Architecture I and 20% more compared to Architecture II with a significant increase in area of 40% and 25% respectively. Architecture II works at a lower clock frequency due to high loading. In principle, it is possible to reduce the delay on the

Technology	Network								
	Architecture I			Architecture II			Architecture III		
	Area	Freq.	Thru.	Area	Freq.	Thru.	Area	Freq.	Thru.
Spartan	441	35.3	1.6	504	28.1	1.76	533	35.3	2.46
XC4000E	441	27	1.23	504	22.3	1.39	533	26.1	1.54
ASIC	6439	94.1	4.28	6947	71.2	4.45	7737	110.6	6.51

TABLE 4.5. Experimental results obtained for Digital Image Processing network

critical path at the cost of extra area. It can also be observed from Figure 4.11 that Architecture II has a better latency compared to Architecture III.

5. CONCLUSION

The goal of this work was to improve the throughput of network of basic radix-2 on-line arithmetic modules. There has been a gain in throuhput of network using newly developed modules instead of the previous on-line modules. In this chapter the results of this work would be summarized.

5.1. Summary of the results

The throughput of a network of on-line modules is a very important parameter to measure the network performance.

This work presents a new design approach for the implementation of a radix-4 on-line division in chapter 3. The use of this technique resulted in some clear benefits: fixed small on-line delay, simple implementation of selection function and components, less restriction for the input operands. The proposed division unit has a better throughput than radix-2 on-line division when implemented on CMOS devices, and could be used to improve the performance of networks of radix-2 on-line modules that include radix-2 division. Because of the small on-line delay, this solution is especially useful on networks of on-line modules used in digital signal processing applications, where the operand precision is usually small, and the on-line delay will significantly impact the system throughput. The throughput improvements ranged from 17% to 49%, however nearly double the area was required for such gains. Synthesis for various FPGA chips didn't provided good results, indicating that there are specific transformations that must be made to the design in order to reach the estimated gain in throughput.

Overall, the radix-4 implementation by overlapped replication proves to be simpler than the design proposed in [10] and produces better throughput than a

radix-2 on-line divider. Estimation of delays indicate that the proposed design will also have a better throughput than the design proposed in [10] for small precision of the operands.

Conventional on-line modules fail to deal with the impact of on-line delay on throughput. The proposed solution in chapter 4 and design for improved on-line operators proved to be efficient for small precision of the operands, when the on-line delay will cause the most loss in performance. An improvement of approximately 35% in throughput was obtained for 7-bit operands. This gain is even higher for networks with deep pipelining. It is clear, that the impact of on-line delay is not significant for large precision. However, in DSP applications, the precision of the operands is usually small. The proposed overlapped architecture in chapter 4 is vital for an efficient implementation of networks using on-line modules.

In [7] it was concluded that SRT is the best choice when division is used in network of on-line modules. SRT divider has less area, and is efficient compared to on-line divider for same precision. But with our proposed ol-line divider we concluded that on-line divider have much lesser area, has less latency for a comparable throughput.

5.2. Consideration for usage of on-line arithmetic module

The usage of on-line arithmetic modules on a digital system is based upon lots of factors.

- *Range of operation*: This is one of the most important design issue to be considered. Since on-line operators have certain constraints on the input values, there usage may always not be efficient. Based upon the module an integer bit

may be added which needs to be normalized and then send to other modules increasing the latency, efficiency and area of the system.

- *Network topology*: Network topology is an important factor for implementation of a system.

- *Latency*: For a system that has to react extremely fast low latency has to be very important. The on-line designs show for all applications the lowest latency.

- *Throughput*: This is probably most important performance measure of a system. It is this factor that decides efficiency. In this work we show that by making modifications at architectural level of modules a throughput of a network could be enhanced. But this modifications are made at cost of extra area.

- *Complexity and precision of operation*: The area in on-line arithmetic grows linearly with increasing precision as in case of other bit-serial arithmetic. In a system which shows a lot of data dependencies it is advisable to use on-line arithmetic as it increases parallelism and thereby have further gain in terms of latency. The design of new modules presented in this work would not have significant gain where precision of inputs are large. They are significant in cases of small precision application as analyzed in case study of digital image processing or other digital signal processing application.

5.3. Further Work

Some issues were not completely solved in this thesis. This Section identifies some of those issues which needs further investigation.

The control structure of the hidden on-line delay networks needs to be developed with a greater care. Instead of using decentralized control structure a centralized control structure needs to be explored further. Hiding on-line delay of radix-4 on-line divider needs to be solved. There was less effort to optimize some modules inorder to have best performances. The CAD tools did most of optimizations. Some more work should be put into these modules to optimize their performaces and area.

A simple network using hidden on-line modules was tested. A Design of network using radix-4 on-line divider needs to be investigated along with some arithmetic modules which are not looked upon in this thesis such as improved on-line square root modules also needs to be developed. A library of all the devloped modules must also be created.

5.4. Concluding Remarks

In this thesis we identified the problem associated in a network using on-line modules with small precision. We tried to improve the throughput of a network of radix-2 on-line arithmetic modules by hiding the on-line delay. For small precision application such as in case of DSP applications the developed modules when replaced by on-line modules which causes bottleneck in throughput has a clear performance enhancement. We know, that on-line arithmetic are always not the best solutions compared to distributed arithmetic but we have tried to make it competitive and presented a strong case for its usage for practical application. The developed a radix-4 on-line divider which is much simpler to design could be used on a network for higher throughput. As long as there are no complete libraries of modules and tools that help design data flow in on-line arithmetic modules it is less likely that on-line modules would be used in any practical applications.

BIBLIOGRAPHY

[1] A. F. Tenca, Ajay Shantilal and Mohammed Sinky. "A Radix-4 On-line Division Design and Its Application To Networks of On-line Arithmetic Modules". *Proceedings SPIE Advanced Signal Processing Algorithms, Architectures and Implementations XIII*, vol. 5205-57, pp. 529–551, 2003.

[2] A. F. Tenca, Ajay Shantilal and Mohammed Sinky. "Improved Throughput Networks of Basic On-line Arithmetic Modules for DSP Application". *To appear towards the proceedings of 15th IEEE conference on Application-specific Systems, Architectures and Processors*, 2004.

[3] Mohammed Sinky, A. F. Tenca, L. Lucchese and Ajay Shantilal. "Design of a Color Image Processing Algorithm Using On-line Arithmetic Modules". *To appear in proceedings of SPIE Advanced Signal Processing Algorithms, Architectures and Implementations XIV*, 2004.

[4] M. D. Ercegovac. "On-line arithmetic: an Overview". *SPIE Real Time Signal Processing VII*, vol. 495, pp. 86–93, 1984.

[5] M. D. Ercegovac and T. Lang "On-line Arithmetic: A Design Methodology and Applications in Digital Signal Processing". *VLSI Signal Processing III*, pp. 252–263, 1988.

[6] D. Lau and A. Schneider and M. D. Ercegovac and J. Villasenor "A FPGA-based Library for On-line Signal Processing". *The Journal of VLSI Signal Processing-Systems for Signal Image and Video Technology*, vol. 28, pp. 129–143, May 2001.

[7] R. Galli "Design and Evaluation of On-line Arithmetic Modules and Networks for Singal Processing Applications on FPGAs". *Oregon State University Master Thesis*, June 2001.

[8] R. Galli and A. Tenca "A design methodology for networks of on-line modules to implement DSP algorithms". *Oregon State University, in prep.*, 2001.

[9] J. S. Fernando and M. D. Ercegovac "Conventional and on-line arithmetic designs for high-speed recursive digital filters". *VLSI Signal Processing III*, pp. 189–197, 1994.

[10] P. K.-G. Tu and M. D. Ercegovac "A Radix-4 On-line Division Algorithm". *IEEE 8th Symposium on Computer Arithmetic*, pp. 181–187, 1987.

[11] J. Duprat and Y. Herreros and J.-M. Muller "Some results about On-line computation of functions". *Proceedings 9th IEEE symposium on Computer Arithmetic*, September 1989.

[12] M. Dimmler, A. Tisserand, U. Holmberg and R. Longchamp "On-line arithmetic for real-time control of microsystems". *IEEE/ASME Transactions on Mechatronics*, vol. 4, no. 2, pp. 213–217, June 1999.

[13] A. Avizenis "Signed-digit number representation for fast parallel arithmetic". *IEEE Transactions on Electronic Computers*, vol. EC10, pp. 389–400, Sept. 1961.

[14] K. S. Trivedi and M. D. Ercegovac "On-line Algorithm for Division and Multiplication". *IEEE Transactions on Computers*, Vol. C-26 no. 7, pp. 681–687, July 1977.

[15] K. S. Trivedi and J. G. Rusnak "Higher Radix On-line Division". *Proceedings of 4th IEEE Symposium on Computer Arithmetic"*, pp. 164–174, Oct. 1978.

[16] P. K.-G. Tu and M. D. Ercegovac "Design of On-line Division Unit". *IEEE 9th Symposium on Computer Arithmetic*, pp. 42–49, 1989.

[17] A. F. Tenca and M. D. Ercegovac "On the Design of High-Radix On-Line Division for Long Precision". ,

[18] M. D. Ercegovac and Thomas Lang "Digital Arithmetic". *Morgan Kaufmann Publishers*, 2003.

[19] M. D. Ercegovac and Thomas Lang "On-the-fly Conversion of Redundant into conventional representations". *IEEE Transactions on Computers*, Vol. C-36, no. 7, pp. 895–897, July 1987.

[20] A. F. Tenca "Theory and Implementation of Radix-2 On-line Multipliers". *Oregon State University in preperation*, 2001.

[21] M. D. Ercegovac and Thomas Lang "Division and Square Root: Digit-Recurrence Algorithms and Implementation". *Kluwer Academic Publishers*, 1994.

[22] P. K.-G. Tu "On-line Arithmetic Algorithms for Efficient Implementation". *PhD thesis, University of California, Los Angeles*, Sept 1990.

[23] D. M. Tullsen and M. D. Ercegovac, "Design and VLSI Implementation of an On-line Algorithm". *Real Time Signal Processing IX - 698*, pp. 92–99, SPIE, 1986.

[24] A. Guyot, Y. Herreros, and J. M. Muller "JANUS, an on-line multiplier/ divider for manipulating large numbers". *Proceedings 9th IEEE symposium on Computer Arithmetic*, pp. 106–111, Sept. 1989.

[25] J. C. Bajard, J. Duprat, S. Kla and J. M. Muller, "Some operators for on-line radix-2 computations". *Journal of parallel and distributed computing*, vol. 22, pp. 336–345, 1994.

[26] R. Galli and A. F. Tenca, "Design and evaluation of on-line arithmetic for signal processing applications on FPGA". *Proceedings of the SPIE International Conference on High-speed computing, digital signal processing, and filtering using reconfigurable logic*, August 2001.

[27] L. Lucchese and S. k. Mitra "A new class of non-linear filters for color image processing. part I: Theory". *IEEE Transactions on Image Processing*, 2002

[28] A. F. Tenca and S. U. Hussaini "A design of radix-2 on-line division using lsa organization". *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pp. 266–273, June 2001.