Alternative Implementations of SRT Division and Square Root Algorithms

Thomas Agermose Jensen

July 3, 1998

Adviser: Prof. Peter Kornerup

Preface

The work presented here is the product of my studies at the department of Mathematics and Computer Science (IMADA) at Odense University.

Thanks are in order to a number of people. First of all to my adviser Peter Kornerup for his endless patience, and for always having time for me. Secondly to Vitit Kantabutra who wrote the article which got me started in the first place. And to the people at Tegnestuen who made the time spent writing this thesis an enjoyable one.

Thomas Agermose Jensen,
Odense, July 1998.

Contents

Chapter 1

Introduction

1.1 Motivation

My interest for computer arithmetic was born when I took a course on computer arithmetic given by Peter Kornerup at Odense University. One of the topics of the course was the by then well-known Pentium FDIV bug. Later I came across an article [10] by Vitit Kantabutra, the subject of which was SRT division using the direct comparison method.

The aforementioned Pentium bug, has since spawned a considerable interest in verification of the table lookup method [3, 8], while little effort has been devoted to the direct comparison method.

The *main objective* of this thesis is to analyze the direct comparison method, and to compare it with the table lookup method.

1.2 Overview

- Chapter 2 gives an introduction to the concepts of number representations and fundamental operations on these.
- In Chapter 3 we present the general division problem, and digit serial division.
- In Chapter 4 we develop the necessary theory to study the SRT class of division algorithms. This theory is exemplified by the direct comparison method and the table lookup method. For both methods several possible implementations are analysed.
- Chapters 5 and 6 deal with square root.
- In Chapter 7 we examine a combined division and square root implementation, for the two possible SRT methods.
- A comparison of the direct comparison method and the table lookup method is presented in Chapter 8.
- Chapter 9 concludes our investigation.

Chapter 2

Number Representations

In this chapter we present some of the concepts on which the division and square root algorithms rely. Most of it is based on concepts presented in [12].

2.1 Radix Representations

Definition 2.1 *We define the radix β polynomials, $\mathcal{P}[\beta]$, as polynomials of the form:*

$$P([\beta]) = d_m[\beta]^m + \ldots + d_l[\beta]^l \tag{2.1}$$

where $d_i \in \mathbb{Z}$, and $-\infty < l \leq i \leq m < \infty$.

The value of the polynomial is:

$$\|P([\beta])\| = P(\beta) = d_m\beta^m + \ldots + d_l\beta^l. \tag{2.2}$$

We will use the following notation:

β: Radix or base.

d_i: i'th digit

d_l: least significant digit

l: least significant position

$[\beta]^l$: unit in last place, ulp $(P[\beta])$

m: most significant position

Although β may be positive, negative, and even real or complex, for our purpose we shall restrict β to the positive integers, that are powers of two, i.e. $\beta = 2^k$, for some $k \geq 1$. We shall usually employ the sum notation:

$$P([\beta]) = \sum_{i=l}^{m} d_i[\beta]^i, \tag{2.3}$$

or the shorthand notation,

$$P = P([\beta]). \tag{2.4}$$

In the following chapters we shall usually not distinguish between the representation of a radix polynomial and its value, the correct interpretation will be obvious from the context in which it appears.

2.2 Digit Sets

A subset $\mathcal{D}$ of the integers $\mathbb{Z}$, is called a *digit set*, and a member $d \in \mathcal{D}$ is called a digit. There are many types of digit sets, the most common being the *standard* digit set:

$$\mathcal{D}_\beta = \{0, \ldots, \beta - 1\}. \tag{2.5}$$

In the following we are particularly interested in the *symmetric* digit set:

$$\mathcal{D}_a = \{-a, \ldots, 0, \ldots, a\}, \tag{2.6}$$

since this is the digit set used in SRT algorithms.

2.3 Radix-Digit Set Combinations

To represent numbers we must choose a radix, β, together with a digit set, $\mathcal{D}$. We may then define:

$$\mathcal{P}[\beta, \mathcal{D}] = \{P \in \mathcal{P}[\beta] \mid d_i \in \mathcal{D}\} \tag{2.7}$$

In the case $\beta = 2$ we have several well-known base-digit set encodings. We will use the follow-

digit	encoding
0	00
1	01 or 10
2	11

Table 2.1: Carry-save encoding.

digit	encoding
$\overline{1}$	10
0	00 or 11
1	01

Table 2.2: Borrow-save encoding.

digit	encoding
$\overline{1}$	11
0	00 or 10
1	01

Table 2.3: Sign-digit encoding.

ing notation for carry-save and borrow-save representations, when a polynomial is expressed using the particular encodings:

carry-save

$$P = (P)^s + (P)^c \tag{2.8}$$

borrow-save

$$P = (P)^s - (P)^b \tag{2.9}$$

2.3.1 Redundancy

It is well-known that a digit set may be redundant in conjuction with a given radix, if some values have more than one possible representation. For such digit-set radix combinations we can define a reduncancy factor which measures how redundant the digit set is. For a symmetric digit set we have:

$$\rho = \frac{a}{\beta - 1}. \tag{2.10}$$

2.3.2 2's Complement Representation

Let the complement polynomials be the set:

$$\{P \mid P = \sum_{i=l}^{m+1} d_i[2]^i, d_i \in \{0,1\}, \text{for } l \leq i \leq m \text{ and } d_{m+1} = -d_m\}. \tag{2.11}$$

We have the additive inverse:

$$\begin{aligned}
-P &= \sum_{i=l}^{m+1}(-d_i)[2]^i \\
&= -d_{m+1}[2]^{m+1} + \sum_{i=l}^{m}(1 - d_i - 1)[2]^i \\
&= -d_{m+1}[2]^{m+1} + \sum_{i=l}^{m}(1 - d_i)[2]^i - \sum_{i=l}^{m}[2]^i \\
&= -d_{m+1}[2]^{m+1} + \sum_{i=l}^{m}(1 - d_i)[2]^i - \left([2]^{m+1} - [2]^l\right) \\
&= -(d_{m+1} + 1)[2]^{m+1} + \sum_{i=l}^{m}(1 - d_i)[2]^i + [2]^l.
\end{aligned}$$

We will use the notation (1's complement):

$$\overline{P} = \sum_{i=l}^{m}(1 - d_i)[2]^i. \tag{2.12}$$

we then have the following relation:

$$-P = \overline{P} + [2]^l. \tag{2.13}$$

That is, the additive inverse is obtained by bitcomplementing followed by an addition of 1 in the least significant bit position.

2.3.3 Sign-Magnitude Representation

Although we can easily determine the sign of a 2's complement radix polynomial from the most significant bit, the magnitude of the number depends on the sign. Sometimes it is more convenient to factor out the sign, so that both sign and magnitude may be determined easily. This is the basis for the sign-magnitude polynomials:

$$P^{\pm} \in \mathcal{P}[\beta, \mathcal{D}] = \{s \cdot P \mid s \in \{-1, 1\}, P \in \mathcal{P}[\beta, \mathcal{D}]\}. \tag{2.14}$$

2.4 Radix Conversions

Given a radix $\beta = 2^k$ polynomial $P([\beta]) = \sum_{i=l}^{m} d_i[\beta]^i$, we can convert it to a radix 2 polynomial P', by splitting each radix 2^k digit into k bits,

$$P'([2]) = \sum_{i=l\cdot k}^{m\cdot k} d'_i[2]^i, \tag{2.15}$$

where d'_i are the corresponding radix 2 digits. We may then define the truncated radix 2 polynomial $\{P'\}_t$:

$$\{P'([2])\}_t = \sum_{i=m\cdot k-t}^{m\cdot k} d'_i[2]^i \tag{2.16}$$

I.e. the t most significant bits. We also have,

$$\text{ulp}\left(\{P'\}_t\right) = [2]^{m\cdot k-t}. \tag{2.17}$$

2.5 Digit Set Conversions

In the following chapters we shall encounter situations, where we need to convert numbers from one digit set into another, or from one encoding into another.

2.5.1 Borrow-Save to Sign-Digit

This conversion requires a recoding of the digits from borrow-save to sign-digit. Inspection of Tables 2.2 and 2.3 reveals that the conversion is trivial. Let $a = (a^b, a^s)$ be a digit encoded in borrow-save and let $b = (b^s, b^d)$ be a digit encoded in sign-digit.

$$b^s = a^b, \quad b^d = a^b \oplus a^s \tag{2.18}$$

The conversion can thus be realized by a single XOR operation.

2.5.2 On-The-Fly Conversion

In particular we are interested in converting redundant borrow-save into standard binary representation. This conversion can trivially be accomplished by a subtraction as suggested in 2.9. In the algorithms described in this thesis, the result digits are generated in a sequential fashion. The result must usually be converted to non-redundant form. If we could perform the conversion into non-redundant form, in parallel with the digit generation we could then

avoid the slow addition operation on the redundant result. This parallel conversion can be accomplished by an *on-the-fly conversion*. This conversion is described in Section 1.10 of [12].

We wish to convert a polynomial Q' with digits, q_i from the symmetric digit set, $\mathcal{D}_a$, to another polynomial, Q, with digits from the standard digit set $\mathcal{D}_\beta$, [1] in radix β. First we derive the *conversion mapping* α, shown in Table 2.4. The carry set is seen to be $\{-1, 0\}$.

α	$\overline{a}$	...	0	...	a
0	$0+\overline{a} = \overline{1}\cdot\beta+\beta+\overline{a}$		$0+0 = 0\cdot\beta+0$		$0+a = 0\cdot\beta+a$
$\overline{1}$	$\overline{1}+\overline{a} = \overline{1}\cdot\beta+(\beta-1)+\overline{a}$		$\overline{1}+0 = \overline{1}\cdot\beta+(\beta-1)$		$\overline{1}+a = 0\cdot\beta+a-1$

Table 2.4: Conversion mapping.

We then have two conversion polynomials with the following update rules:

$$Q^0_{i+1} = \begin{cases} Q^{-1}_i + (\beta + q_{i+1})\beta^{-(i+1)} & \text{if} \quad q_{i+1} < 0 \\ Q^0_i + q_{i+1}\beta^{-(i+1)} & \text{if} \quad q_{i+1} \geq 0 \end{cases}$$

and

$$Q^{-1}_{i+1} = \begin{cases} Q^{-1}_i + ((\beta-1) + q_{i+1})\beta^{-(i+1)} & \text{if} \quad q_{i+1} \leq 0 \\ Q^0_i + (q_{i+1}-1)\beta^{-(i+1)} & \text{if} \quad q_{i+1} > 0 \end{cases}$$

with $Q^{-1}_0 = -1$ and $Q^0_0 = 0$. The final converted polynomial, Q is then equal to Q^0_n, if Q' has n digits.

2.6 Comparisons

We wish to perform comparisons of the form:

$$||P|| \geq a\beta^{m-t}, \quad t \geq 0, a \in \mathbb{Z}. \tag{2.19}$$

This comparison can be performed in time $O(\log t)$. This is a direct consequence of Theorem 2.10.13 of [12] which states that the sign of any polynomial can be found in time $O(\log(m-l))$.

If P is in redundant form with t digits then such a comparison can be performed in one of two ways:

Method 1 Assimilate P to non-redundant form. The comparison is then a matter of determining the sign of the borrow-save polynomial Q given by:

$$Q = (P)^s - \left(a[\beta]^{m-t}\right)^b. \tag{2.20}$$

Both of these operations run in time $O(\log t)$, if we use a carry-lookahead adder to convert P to non-redundant form.

[1] The definition of the on-the-fly algorithm states that the target digit set should be complete for radix β. The conversion given here is therefore only correct for non-negative numbers.

Method 2 First perform a redundant subtraction of the righthand side of (2.19) from the lefthand side, then determine the sign of the resulting redundant polynomial. The redundant subtraction runs in constant time while the sign-detection runs in time $O(\log t)$. When examining actual implementations in Chapter 8 we shall use Method 2, since it is faster. This method also has the advantage that it allows both operands to be in redundant form. Observation: Method 1 and Method 2 are equivalent.

Observation 2.10.14 in [12] states that we can compare two 2's complement numbers in the same manner, and in the same time. This implies that we can determine the sign of a carry-save polynomial in time $O(\log t)$.

2.7 IEEE Standard For Floating-Point Arithmetic

Nowadays most floating-point processors comply with the IEEE standard [9]. IEEE numbers are on the form:

$$(-1)^s 2^E (b_0.b_1 \cdots b_{p-1}), \tag{2.21}$$

where

$s \in \{0, 1\}$ is the sign,

$E \in [E_{min}, E_{max}]$ is the exponent,

$b_i \in \{0, 1\}$, $b_0 = 1$.

The well-known single and double-precision formats are shown in Figure 2.1 and Figure 2.2.

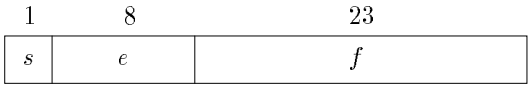

Figure 2.1: Single format.

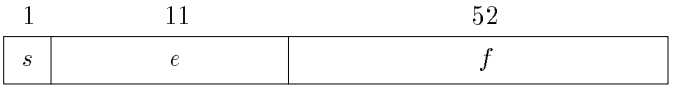

Figure 2.2: Double format.

Chapter 3

Division

Of the four fundamental arithmetic operations, division is the most complicated. In the paper and pencil method we have the problem that the least significant digit of the result depends on all the digits computed before it, so we have to determine the digits in a strictly serial fashion. We also recall that there are many possible results, depending on when we decide to terminate the division. Since computers can only represent a finite subset of the rational number system, any computerized version of the division operation is not closed. We now state the general division problem:

Definition 3.1 (The division problem.) *Given X and D, the dividend and divisor, and a required quotient precision,* $\operatorname{ulp} Q$*, find Q and R, the quotient and remainder, so that:*

$$X = Q \cdot D + R,$$

with

$$\left| \frac{X}{D} - Q \right| < \operatorname{ulp} Q, \qquad \operatorname{sign} R = \operatorname{sign} X.$$

Division algorithms can be divided into two groups. The first is the digit serial method, to which the well known paper and pencil method as well as the SRT algorithms belong. The other is the functional iteration method, which we will not be discussing here.

3.1 Digit Serial Division

Let β be the radix, and let the quotient digit set be $\mathcal{D} = \{a, \ldots, 0, \ldots, b\}$. Digit serial division produces the quotient digits one at a time, most significant first, by iterating a recurrence n times, where n is the number of iterations needed to produce the quotient with the required precision, $\operatorname{ulp} Q$. After i iterations, we have a *partial quotient*, Q_i. We then define the *error at step i*, ϵ_i, as follows:

$$\epsilon_i = \left| \frac{X}{D} - Q_i \right|. \tag{3.1}$$

One way of making sure that definition 3.1 is satisfied is to keep ϵ_i bounded in each iteration,

$$\epsilon_i < \operatorname{ulp} Q_i. \tag{3.2}$$

This will ensure that $\epsilon_n < \text{ulp}\, Q_n$. Using the notation from Chapter 2 we can write the quotient on the form:

$$Q = \sum_{i=l}^{m} q_i \beta^i, \tag{3.3}$$

with $q_i \in \mathcal{D}$. However, since division is a most significant digit first operation we shall write it on the equivalent form:

$$Q = \sum_{i=-m}^{-l} q_{-i} \beta^{-i}. \tag{3.4}$$

In both cases we have $\text{ulp}\, Q = \beta^l$. Likewise we can write the quotient after i iterations as:

$$Q_i = \sum_{j=-m}^{-m+i} q_{-j} \beta^{-j}, \tag{3.5}$$

with $\text{ulp}\, Q_i = \beta^{m-i}$. The number of iterations, n, is equal to $n = m - l$. For simplicity we will enumerate the quotient digits from 0 to n. After n iterations the quotient is then given by:

$$Q_n = \sum_{j=-m}^{-m+n} q_{j+m} \beta^{-j}. \tag{3.6}$$

Since we are doing most significant digit-first we have:

$$\text{ulp}\, Q_{i+1} = \beta^{-1} \text{ulp}\, Q_i. \tag{3.7}$$

The initial quotient is $Q_0 = 0$ with $\text{ulp}\, Q_0 = \beta^m$. We now develop a *division recurrence* based on the above. From (3.2) and (3.7) we get:

$$\begin{aligned} \left| \frac{X}{D} - Q_i \right| &< \beta^{-i} \text{ulp}\, Q_0 \\ |X - Q_i D| &< \beta^{-i+m} |D| \\ \beta^i |X - Q_i D| &< \beta^m |D|. \end{aligned} \tag{3.8}$$

Unscaled Partial Remainder

Define a *partial remainder*:

$$P_i = X - Q_i D. \tag{3.9}$$

We then obtain the recurrence:

$$\begin{aligned} P_{i+1} &= X - Q_{i+1} D \\ &= X - \left(Q_i + q_{i+1} \beta^{m-(i+1)} \right) D \\ &= \beta^{i+1} (X - Q_i D) - q_{i+1} D \beta^m \\ &= P_i - q_{i+1} D \beta^{m-(i+1)}, \end{aligned} \tag{3.10}$$

with initial remainder $P_0 = X$, since $Q_0 = 0$. The bound in (3.8) leads us to the following bound on the partial remainder:

$$\forall i: \quad |P_i| < \beta^{m-i} |D| = \text{ulp}\, Q_i |D|. \tag{3.11}$$

The initial remainder X also has to satisfy this bound, so:

$$|X| < \beta^m |D|. \tag{3.12}$$

We can easily see that the pair Q_n, P_n satisfies Definition 3.1, so we let $Q = Q_n$ and $R = P_n$.

Scaled Partial Remainder

We could also define a scaled *partial remainder*: [1]

$$R_i = \beta^i (X - Q_i D). \tag{3.13}$$

This partial remainder is the one we shall be using in the following chapters. We then obtain the following recurrence:

$$\begin{aligned} R_{i+1} &= \beta^{i+1}(X - Q_{i+1}D) \\ &= \beta^{i+1}\left(X - \left(Q_i + q_{i+1}\beta^{m-(i+1)}\right) D\right) \\ &= \beta^{i+1}(X - Q_i D) - q_{i+1} D \beta^m \\ &= \beta R_i - q_{i+1} D \beta^m, \end{aligned} \tag{3.14}$$

with the *initial remainder* $R_0 = X$, since $Q_0 = 0$ in (3.13). The bound in (3.8) leads us to the following bound on the scaled partial remainder:

$$\forall i: \quad |R_i| < \beta^m |D|. \tag{3.15}$$

Since X is the initial partial remainder, we require $|X| < \beta^m |D|$.

Final Remainder Formation

As before, Q_n satisfies Definition 3.1, however it is easily seen that the remainder R_n does not. We therefore have to form the *final remainder* R from R_n. We have:

$$R_n = \beta^n (X - Q_n \cdot D), \tag{3.16}$$

and we want

$$R = X - Q \cdot D. \tag{3.17}$$

so we form R by:

$$R = \beta^{-n} \cdot R_n. \tag{3.18}$$

Correction Step

We also have to make sure that the signs of the remainder R and the dividend are the same. If they disagree then we have to perform a *correction step* as follows:

$$\text{sign } R = \text{sign } D \quad \Rightarrow \quad \hat{Q} = Q + \text{ulp}\, Q,\ \hat{R} = R - \text{ulp}\, Q \cdot D, \tag{3.19}$$

$$\text{sign } R \neq \text{sign } D \quad \Rightarrow \quad \hat{Q} = Q - \text{ulp}\, Q,\ \hat{R} = R + \text{ulp}\, Q \cdot D. \tag{3.20}$$

The proof that the pair $\hat{Q}, \hat{R}$ satisfies Definition 3.1 is trivial.

[1] This is the same partial remainder we use in the paper and pencil method.

Selection Function

The condition (3.2) is only an upper bound; depending on the digit set used we may require a tighter bound, leading us to a tighter bound on the partial remainders as well. In the next chapter we shall see examples of this. We thus have to select the quotient digit q_{i+1} in the recurrence (3.14), in such a way that R_{i+1} is bounded. From (3.14) we see that the selection depends on βR_i and D. This selection procedure is visualized in the Robertson diagram 3.1. Consult Appendix A for the definition of Robertson diagrams. How this selection is

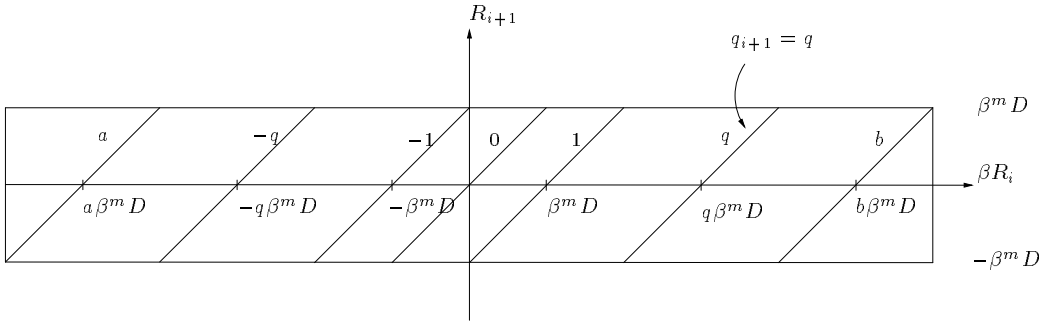

Figure 3.1: Robertson diagram for quotient digit selection.

done, depends on the actual implementation. For now we let the selection be handled by a quotient-digit selection function:

$$q_{i+1} = q\_select(\beta R_i, D). \tag{3.21}$$

3.1.1 Digit Serial Division Algorithm

The developments made in the preceeding section leads us to the following digit serial division algorithm, which uses the scaled partial remainder.

Algorithm 3.2 *(Digit serial division algorithm)*

Input: A radix β, a divisor D, a dividend X, and a required quotient precision β^{-n+m}.

Output: A quotient and remainder pair Q, R with $X = Q \cdot D + R$, $|X/D - Q| < \beta^{-n+m}$, and sign $R =$ sign X.

Method:

$R_0 := X$;
for $i := 0$ **to** $n - 1$ **do**
    $q_{i+1} := q\_select(\beta R_i, D)$;
    $R_{i+1} := \beta R_i - q_{i+1} D \beta^m$;
end;
$Q := \sum_{i=-m}^{-m+n} q_i \beta^{-i}$;
$R := \beta^{m-n} R_n$;
if sign $R \neq$ sign X **then**
    if sign $R =$ sign D **then**

$Q := Q + \beta^{-n};$
$R := R - \beta^{-n} D;$
else
$Q := Q - \beta^{-n};$
$R := R + \beta^{-n} D;$
end;
end;

The correctness of the algorithm follows from (3.6) and (3.26). One of the goals of an implementation of Algorithm 3.2 is to make the two steps in the **for** loop as fast as possible, and to generate the quotient Q as fast as possible.

3.1.2 Block Diagram of the Division Algorithm

Figure 3.2 shows a block diagram of one division iteration.

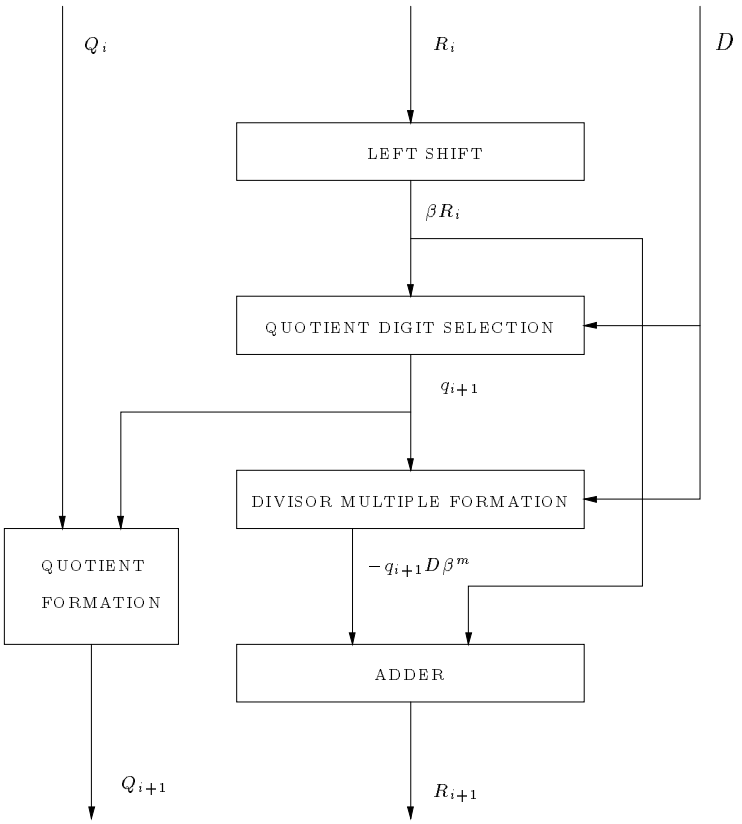

Figure 3.2: Block diagram of division iteration.

3.1.3 Integer Division

Integer division is just a special case of the above with $m = n$ and $l = 0$.

3.1.4 Floating-Point Division

Let X and D be IEEE floating-point numbers. We then have:

$$X = (-1)^{s_X} 2^{E_X} X', \tag{3.22}$$
$$D = (-1)^{s_D} 2^{E_D} D'. \tag{3.23}$$

The signs and exponents are handled trivially. In the following we shall focus on handling the significands, X' and D'. For simplicity we assume that the range of the significands are $[1/2, 1[$, which is just a matter of placing the binary point to the left of the significand. In the following the divisor and dividend will denote the significands of the true divisor and dividend. Let f be the number of fraction bits in the representation of X and D. Then n is the number of iterations needed to produce the quotient and remainder with f fraction bits,

$$n = \left\lceil \frac{f}{\log_2 \beta} \right\rceil \cdot \log_2 \beta. \tag{3.24}$$

We thus have $m = 0$ and $l = -n$, and the quotient can be written on the form:

$$Q_n = \sum_{i=0}^{n} q_i \beta^{-i}. \tag{3.25}$$

In this case we will use the scaled partial remainder 3.13; the recurrence is then:

$$R_{i+1} = \beta R_i - q_{i+1} D, \tag{3.26}$$

and the partial remainder bounds are:

$$\forall i : \quad |R_i| < D. \tag{3.27}$$

with initial remainder $X < D$. In the following chapters we shall assume that the operands are floating-point operands.

Rounding

In the IEEE standard rounding is required. However we refer to [5], for a description of how this rounding is performed.

Chapter 4

SRT Division

SRT division is named after D. Sweeney, J.E. Robertson [15] and T.D. Tocher [17], who discovered a new way of doing non-restoring radix-2 division, at about the same time. The method was then extended to higher radices by Robertson, and later D. Atkins [2] also included the use of redundant quotient representation. SRT division is not really a special kind of division, but rather a special class of digit serial division algorithms, and is characterized by the following:

- The partial remainders are normalized.
- The divisor is normalized.
- A redundant symmetric quotient digit set is used.
- Partial remainders may be in redundant representation.

In the past SRT division has been used in several floating-point processors from Cyrix, Sun and most famously Intel. The notorious Pentium FDIV bug used a radix-4 SRT division algorithm. Most of the previous litterature has focused on analysing a specific implementation, while there has been few attempts to develop a general theory on SRT division. In this chapter we will develop a general theory for SRT division. We will then use the theory to analyze two possible implementations of the quotient-digit selection function, namely the lookup-table and the direct comparison method.

4.1 Theory of SRT Division

As before, let β denote our radix, let X be the positive dividend, and let D be the positive divisor. In the following we shall use a redundant symmetric quotient digit set $\mathcal{D}_a = \{-a, .., 0, .., a\}$, with $a \geq \beta/2$. We then have a redundancy factor of:

$$\rho = \frac{a}{\beta - 1}. \tag{4.1}$$

The purpose of the quotient digit selection function is to select a quotient digit q_{i+1}, while keeping the partial remainder R_{i+1} bounded. These bounds are now determined. Let the *partial remainder bounds* in iteration i be $\underline{B}_i$ and $\overline{B}_i$

$$\forall i : \underline{B}_i \leq R_i \leq \overline{B}_i. \tag{4.2}$$

Let the *selection interval* $[L_q(i), U_q(i)]$ be the interval of βR_i for which we can select $q_{i+1} = q$, and keep R_{i+1} bounded by (4.2),

$$\beta R_i \in [L_q(i), U_q(i)] \Rightarrow \underline{B}_{i+1} \leq R_{i+1} = \beta R_i - qD \leq \overline{B}_{i+1}. \tag{4.3}$$

Now look at $q_{i+1} = a$, the maximal digit value, here we have:

$$\beta R_i = U_a(i) \Rightarrow \beta R_i - aD = \overline{B}_{i+1}, \tag{4.4}$$

and

$$\beta \overline{B}_i = U_a(i). \tag{4.5}$$

Combining these we get:

$$\beta \overline{B}_i \quad = \quad \overline{B}_{i+1} + aD. \tag{4.6}$$

The solution to this equation is $\overline{B}_i = \rho D$, which can be verified by inserting it in (4.6). Since the solution does not depend on the iteration index i we shall write $\overline{B}$ instead of $\overline{B}_i$. Analogously the value of $\underline{B}_i$, can be found by looking at $q_{i+1} = -a$, the minimal digit value,

$$\underline{B} = -\rho D, \quad \overline{B} = \rho D. \tag{4.7}$$

We then get the following bounds on the partial remainders,

$$\forall i : -\rho D \leq R_i \leq \rho D. \tag{4.8}$$

For $\rho = 1$ these bounds are not compatible with the bounds in (3.27). We therefore have to perform a correction step as described in Chapter 3. Note that since $X < D$ is the first partial remainder, we have to use $R_0 = X/\beta$, and postscale the results accordingly, in order to comply with these bounds. Let us now develop expressions for $L_q(i)$ and $U_q(i)$. Since the resulting expressions are again independent of i we simply write L_q and U_q instead of $L_q(i)$ and $U_q(i)$. Expressions (4.3) and (4.8) give us the following:

$$\begin{aligned} L_q &= \underline{B} + qD, & U_q &= \overline{B} + qD \\ L_q &= (q - \rho)D, & U_q &= (q + \rho)D. \end{aligned} \tag{4.9}$$

Expressions (4.3), (4.8), and (4.9) are visualized in the Robertson diagram in Figure 4.1. To make sure that we have at least one possible digit choice everywhere, every possible value of βR_i should correspond to a quotient digit. I.e every value of βR_i should belong to at least one digit selection interval:

$$U_{q-1} \geq L_q - \text{ulp}\,(\beta R_i). \tag{4.10}$$

The reason that $\text{ulp}\,(\beta R_i)$ appears on the right side is that it is exactly the difference between two consecutive representable values of βR_i. This condition is also known as the *continuity condition* [5]. In the following we will use the stricter bound:

$$U_{q-1} \geq L_q, \tag{4.11}$$

since it simplifies the resulting expressions. Equation (4.11) gives us the following requirement on ρ:

$$(q - 1 + \rho)D \quad \geq \quad (q - \rho)D \tag{4.12}$$

$$\rho \quad \geq \quad \frac{1}{2}, \tag{4.13}$$

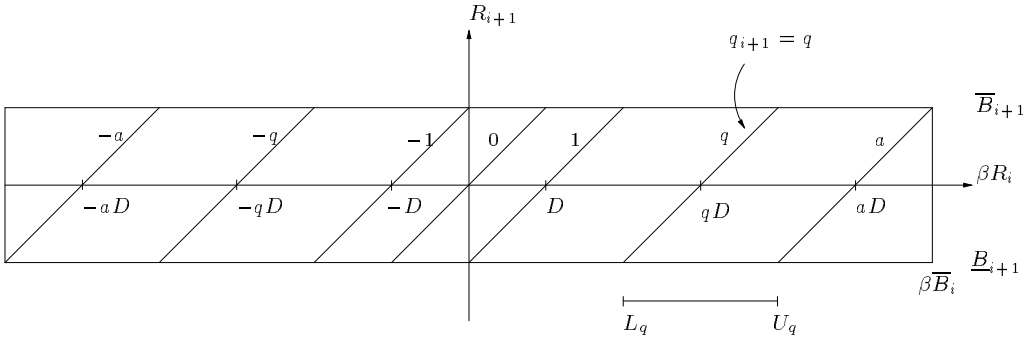

Figure 4.1: Partial remainder bounds and selection intervals.

which is always satisfied because of $a \geq \beta/2$. Since every value of βR_i belongs to at least one selection interval, consecutive selection intervals may overlap each other. The value of the selection interval bounds depend on ρ, the degree of redundancy in $\mathcal{D}_a$ therefore determines the amount of overlap we have between digit selection intervals. The overlap is given by:

$$U_{q-1} - L_q = (2\rho - 1)D > 0, \tag{4.14}$$

and is positive since $\rho > 1/2$ and $D > 0$. The amount of overlap thus depends on the digit set and on the divisor D. This overlap gives rise to the possibility of redundancy in the quotient digit selection, since in the interval $[L_q, U_{q-1}]$ we can select either $q_{i+1} = q$ or $q_{i+1} = q - 1$. Thus in order to select a correct quotient digit, we do not need to know the exact value of βR_i. The number of bits we need to know, to determine the next quotient digit, depends on the amount of overlap. The greater the value of D, the fewer bits we need examine. The closer we get to zero the more bits we need to examine. This is illustrated in Figure 4.2. If we

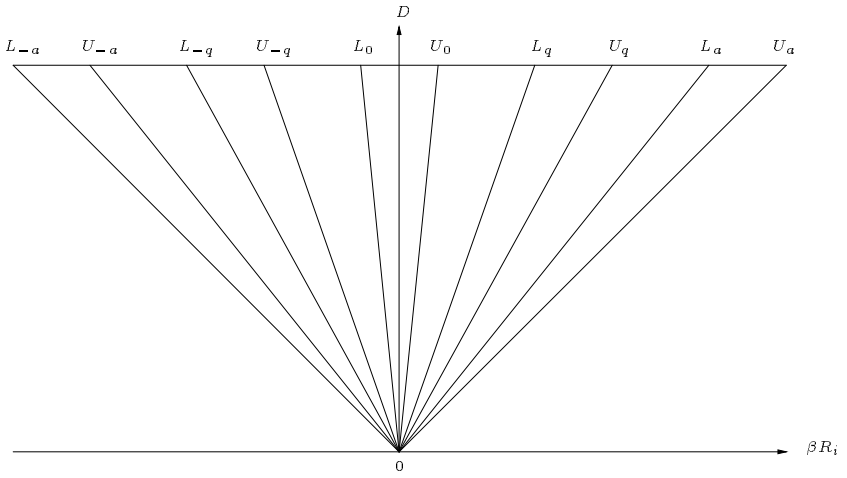

Figure 4.2: Digit selection interval width.

normalize the divisor to an interval $D \in [x, 1[$, where x is not too close to zero, then we only need to examine a few bits of βR_i, to determine the next quotient digit. The most common choice is $x = 1/2$, since this is easy to achieve, and since floating-point numbers are already

in this interval, as described in Chapter 2. This situation is illustrated in Figure 4.3. Once

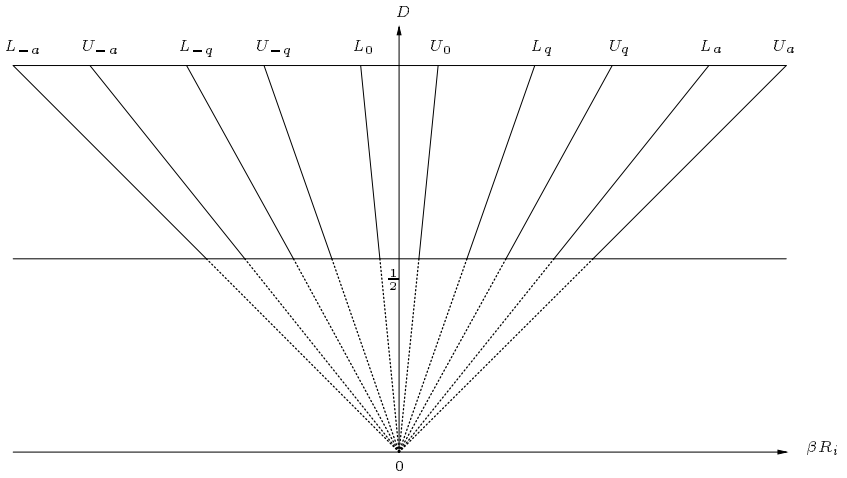

Figure 4.3: Digit selection interval width with normalized divisor.

we have normalized the divisor we can restrict the range of the divisor further by *scaling* it [1, 5, 13]. For our analysis we shall restrict ourselves to the unscaled normalized case.

4.2 The Quotient Digit Selection Function

Now we address the problem of choosing a quotient digit, so that the bounds on the partial remainders are satisfied. Since we have a redundancy in the choice of quotient digit, the piecewise linear function displayed in Figure 4.1 is not 1-1. That is, the same values of βR_i and D may result in two different values of q_{i+1}. A selection function that is 1-1 is now defined. For the moment we will keep D fixed. Partition the entire digit selection interval $[L_{-a}, U_a]$ into $2a$ parts, so that:

$$s_q \leq \beta R_i < s_{q+1} \Rightarrow q_{i+1} = q, \tag{4.15}$$

where s_q are the partition points. An example of this procedure is shown in Figure 4.4, for a maximally redundant digit set. If we do this for every possible value of D, the partition points correspond to a set of functions of D. We thus represent the selection function $q\_select(\beta R_i, D)$, by a set of *selection bounds* $\{s_q(D)\}, q \in \mathcal{D}_a$, with

$$s_q(D) \leq \beta R_i < s_{q+1}(D) \Rightarrow q_{i+1} = q. \tag{4.16}$$

For notational ease, we will write s_q instead of $s_q(D)$. Since the digit selection intervals are symmetric around $\beta R_i = 0$ ($L_{-q} = -U_q$ and $U_{-q} = -L_q$), we only need the positive selection bounds to determine the next quotient digit. If βR_i is non-negative then we proceed as usual. However if βR_i is negative then we compare $-\beta R_i$ with the positive selection bounds. The resulting quotient digit is then $-q$. This corresponds to the following quotient digit selection function:

$$\beta R_i \geq 0 : s_q \leq \beta R_i < s_{q+1} \quad \Rightarrow \quad q_{i+1} = q, \tag{4.17}$$

$$\beta R_i < 0 : s_q \leq -\beta R_i < s_{q+1} \quad \Rightarrow \quad q_{i+1} = -q. \tag{4.18}$$

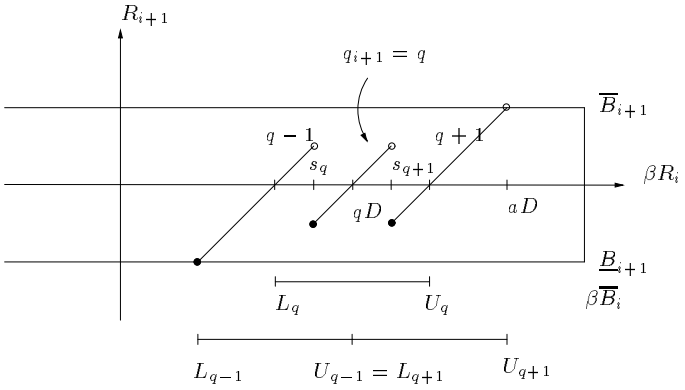

Figure 4.4: 1-1 digit selection function.

We now wish to find a requirement for the choice of s_q. From (4.3) and (4.16) we can deduce,

$$s_q \in [L_q, U_q]. \tag{4.19}$$

For $\beta R_i = s_q - \text{ulp}\,(\beta R_i)$ we must choose $q_{i+1} = q - 1$ because of (4.16). This is equivalent to:

$$s_q - \text{ulp}\,(\beta R_i) \leq U_{q-1}. \tag{4.20}$$

Combine this with (4.10), and we have:

$$s_q \in [L_q, U_{q-1} + \text{ulp}\,(\beta R_i)]. \tag{4.21}$$

We use the stricter upper bound:

$$s_q \in [L_q, U_{q-1}]. \tag{4.22}$$

This means that the selection bounds must be chosen to lie within the overlap between adjacent quotient digit selection intervals, as shown in Figure 4.5. Ihe next two sections we

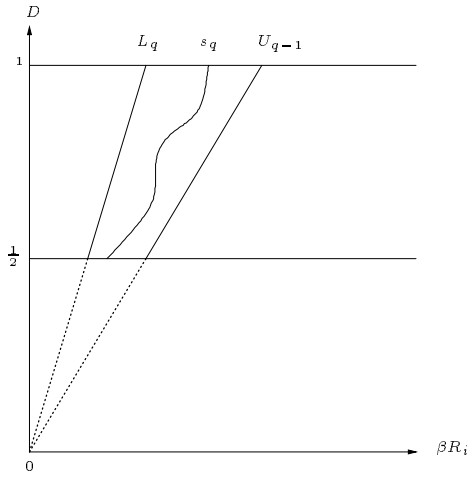

Figure 4.5: Selection function interval.

shall examine two different methods of implementing the function $q\_select(\beta R_i, D)$, namely the method of direct comparison and the table lookup method.

4.3 Direct Comparison

The simplest way to implement equation (4.16) is to perform a direct comparison between the shifted partial remainder βR_i, and the selection bounds s_q, to determine which interval βR_i lies in, and thus which quotient digit q_{i+1} we should select. These comparisons should take place in parallel, as shown in Figure 4.6. In this context the selection bounds are usually called comparison constants [5, 10], however since they are functions of D, we shall call them comparison functions. The method of direct comparison was first described by Robertson

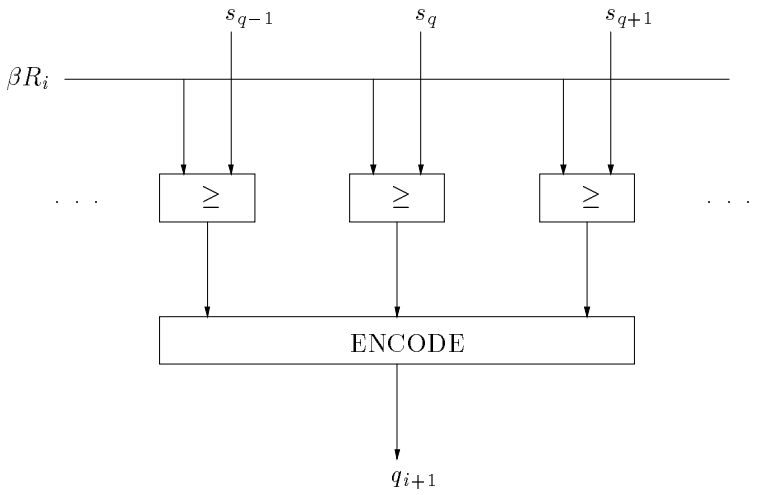

Figure 4.6: Direct comparison.

[15]. More recently Zurawski and Gosling [19], used it in a combined radix-4 square root, multiply, and divide unit. In [10] Kantabutra described a hybrid radix-2/radix-4 divider using direct comparison. To our knowledge all previous efforts have concentrated on particular implementations, failing to derive a general theory on the use of direct comparison. Such a generalization is the object of this section.

4.3.1 Truncated Comparison

Since the operand length may be very large, 53 bits for double-precision floating point [9], it is not feasible to perform a full-precision comparison, in terms of speed and space requirements. Luckily the overlap between digit selection intervals, permits us to perform a truncated comparison. Let $\{\beta R_i\}_\Delta$ denote truncation of βR_i to Δ bits. If βR_i is in redundant representation then we will allow different truncations of the redundant parts. Let the number of integer bits of $\{\beta R_i\}_\Delta$ be τ, and the number of fraction bits be δ. We then have $\Delta = \tau + \delta$. Let l be the number of fractional bits in the representation of βR_i, and s_q. Let ϵ_R be the error resulting from the truncation of βR_i:

$$\beta R_i = \{\beta R_i\}_\Delta + \epsilon_R. \tag{4.23}$$

The range of ϵ_R will depend on the truncations used and the chosen representation of R_i. The above is illustrated in Figure 4.7. Let $D \in [1/2, 1[$ be the normalized divisor, and let

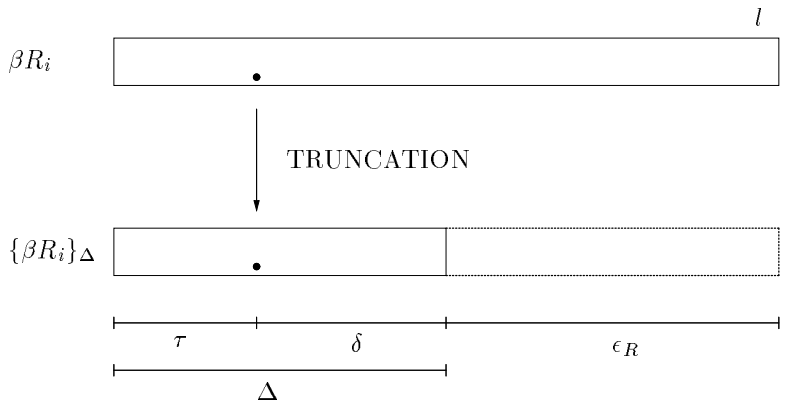

Figure 4.7: Truncating the partial remainder.

$\{s_q\}_\Delta$ denote truncation of s_q to Δ bits, and let ϵ_s be the error we get from truncating s_q:

$$s_q = \{s_q\}_\Delta + \epsilon_s, \tag{4.24}$$

where ϵ_s depends on the representation used for s_q. We implement the function $q\_select(\beta R_i, D)$ as follows:

$$\{s_q\}_\Delta \leq \{\beta R_i\}_\Delta < \{s_{q+1}\}_\Delta \Rightarrow q_{i+1} = q. \tag{4.25}$$

The comparison functions are now 'staircases' as shown in Figure 4.8. The case $\delta = l$ corresponds to a full-precision comparison. A δ which results in a valid quotient digit selection function therefore always exists.

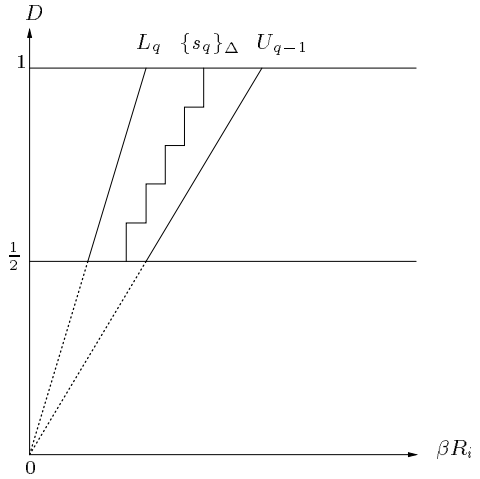

Figure 4.8: Truncated comparison.

4.3.2 Truncation Errors

Given that the partial remainder and the comparison functions may be in a number of different encodings and in either redundant or non-redundant representation we give the corresponding

truncation errors here, for later reference.

Comparison Functions

We shall usually require that s_q is in 2's complement encoding. The range of the truncation error is then, keeping in mind that s_q has l fraction bits:

$$\epsilon_s \in \left[0, 2^{-\delta} - 2^{-l}\right]. \tag{4.26}$$

Partial Remainder

The partial remainder is in one of these three encodings:

2's complement We truncate to δ fration bits, hence the range of ϵ_R:

$$\epsilon_{R(2c)} \in \left[0, 2^{-\delta} - 2^{-l}\right]. \tag{4.27}$$

Carry-Save We allow for different truncations of the carry and save registers, p and r bits respectively. The range of this truncation is:

$$\epsilon_{R(cs)} \in \left[0, 2^{-p} + 2^{-r} - 2 \cdot 2^{-l}\right]. \tag{4.28}$$

Borrow-Save The borrow and save parts are truncated to n bits of negative weight and p bits of positive weight:

$$\epsilon_{R(bs)} \in \left[-2^{-n} + 2^{-l}, 2^{-p} - 2^{-l}\right]. \tag{4.29}$$

4.3.3 Fraction Bits

Let us find an expression for δ, the number of fraction bits, required in the comparison; δ should be chosen so that (4.8) is satisfied. First look at the lefthand side in (4.25):

$$\begin{aligned} \{\beta R_i\}_\Delta &\geq \{s_q\}_\Delta \\ \beta R_i &\geq s_q - \epsilon_s + \epsilon_R. \end{aligned}$$

The selected digit is q so we have $\beta R_i \geq L_q$, leading to the requirement:

$$\begin{aligned} s_q - \epsilon_s + \epsilon_R &\geq L_q \\ s_q &\geq L_q + \epsilon_s - \epsilon_R \\ s_q &\geq L_q + \max(\epsilon_s - \epsilon_R) \\ s_q &\geq L_q + \max \epsilon_s - \min \epsilon_R. \end{aligned}$$

Now for the righthand side of (4.25):

$$\begin{aligned} \{\beta R_i\}_\Delta &< \{s_{q+1}\}_\Delta \\ \{\beta R_i\}_\Delta &\leq \{s_{q+1}\}_\Delta - \mathrm{ulp}\left(\{\beta R_i\}_\Delta\right) \\ \{\beta R_i\}_\Delta &\leq \{s_{q+1}\}_\Delta - 2^{-\delta} \\ \beta R_i &\leq s_{q+1} - 2^{-\delta} - \epsilon_s + \epsilon_R. \end{aligned}$$

We have $\beta R_i \leq U_q$ so we demand:

$$\begin{aligned} s_{q+1} - 2^{-\delta} - \epsilon_s + \epsilon_R &\leq U_q \\ s_{q+1} &\leq U_q + 2^{-\delta} + \epsilon_s - \epsilon_R \\ s_{q+1} &\leq U_q + 2^{-\delta} + \epsilon_s - \epsilon_R \\ s_{q+1} &\leq U_q + 2^{-\delta} + \min(\epsilon_s - \epsilon_R) \\ s_q &\leq U_{q-1} + 2^{-\delta} + \min \epsilon_s - \max \epsilon_R, \end{aligned}$$

where the final inequality follows from the fact that equation (4.25) is also valid for $q_{i+1} = q-1$. The chosen comparison functions and δ should thus satisfy the following two relations:

$$s_q \geq L_q + \max \epsilon_s - \min \epsilon_R, \tag{4.30}$$

$$s_q \leq U_{q-1} + 2^{-\delta} + \min \epsilon_s - \max \epsilon_R. \tag{4.31}$$

However, since it simplifies the following expressions, we shall usually use the following tighter bounds.

$$s_q \geq L_q + \sup \epsilon_s - \inf \epsilon_R, \tag{4.32}$$

$$s_q \leq U_{q-1} + 2^{-\delta} + \inf \epsilon_s - \sup \epsilon_R. \tag{4.33}$$

In order for such a s_q to exist we require:

$$\begin{aligned} U_{q-1} + 2^{-\delta} + \inf \epsilon_s - \sup \epsilon_R &\geq L_q + \sup \epsilon_s - \inf \epsilon_R \\ U_{q-1} - L_q &\geq + \sup \epsilon_s - \inf \epsilon_R - \inf \epsilon_s + \sup \epsilon_R - 2^{-\delta} \end{aligned}$$

Since the lefthand side is always positive and the righthand side only depends on δ, the last inequality can always be satisfied by making δ large enough. The number of fraction bits δ, required in the comparison, thus depends on the chosen comparison functions, the representation, and the encoding used for the comparison functions and the partial remainder.

4.3.4 Integer Bits

We now calculate the number of integer bits τ, required to represent $\{\beta R_i\}_\Delta$, and $\{s_q\}_\Delta$. The upper bound on the value of $\{\beta R_i\}_\Delta$ is:

$$\{\beta R_i\}_\Delta \leq \max(\beta R_i - \epsilon_R)$$

$$\{\beta R_i\}_\Delta \leq \left\lfloor \max(\beta R_i - \epsilon_R) \cdot 2^\delta \right\rfloor \cdot 2^{-\delta} \tag{4.34}$$

$$\{\beta R_i\}_\Delta \leq \left\lfloor (\beta(1 - \operatorname{ulp} D)\rho - \inf \epsilon_R) \cdot 2^\delta \right\rfloor \cdot 2^{-\delta}. \tag{4.35}$$

And the lower bound:

$$\{\beta R_i\}_\Delta \geq \left\lceil (\beta(-1 + \operatorname{ulp} D)\rho - \sup \epsilon_R) \cdot 2^\delta \right\rceil \cdot 2^{-\delta}. \tag{4.36}$$

To determine the number of integer bits needed, we calculate the number of bits needed for the upper bound and the lower bound respectively, and use the one that requires the largest number of integer bits.

The upper bound gives us the following lower bound on τ, which includes the sign bit:

$$2^{\tau-1} \geq \left\lfloor (\beta(1-\operatorname{ulp} D)\rho - \inf \epsilon_R) \cdot 2^{\delta} \right\rfloor \cdot 2^{-\delta} \tag{4.37}$$

$$\tau \geq \left\lceil \log_2 \left(\left\lfloor (\beta(1-\operatorname{ulp} D)\rho - \inf \epsilon_R) \cdot 2^{\delta} \right\rfloor \cdot 2^{-\delta} \right) \right\rceil + 1. \tag{4.38}$$

The lower bound leads to:

$$2^{\tau-1} \geq -\left\lceil (\beta(-1+\operatorname{ulp} D)\rho - \sup \epsilon_R) \cdot 2^{\delta} \right\rceil \cdot 2^{-\delta} \tag{4.39}$$

$$\tau \geq \left\lceil \log_2 \left(-\left\lceil (\beta(-1+\operatorname{ulp} D)\rho - \sup \epsilon_R) \cdot 2^{\delta} \right\rceil \cdot 2^{-\delta} \right) \right\rceil + 1. \tag{4.40}$$

Thus, given the number of fractional bits δ, we can determine the number of integer bits required.

4.3.5 Choice of Comparison Functions

When choosing our comparison functions s_q, we should choose ones which are simple to generate. The comparison functions must be chosen in the interval:

$$s_q \in \left[L_q + \sup \epsilon_s - \inf \epsilon_R, U_{q-1} + 2^{-\delta} + \inf \epsilon_s - \sup \epsilon_R \right]. \tag{4.41}$$

Possible choices are:

$s_q = U_{q-1} = (q-1+\rho)D$: This is only useable for $\rho = 1$, where we then have: $s_q = q \cdot D$. This choice is not possible for carry-save partial remainder, since the truncation error is too large. For $\rho < 1$ we get comparison functions which requires a division by a non-trivial multiplum of D, for example $(2/3)D$.

$s_q = \frac{U_{q-1}+L_q}{2} = \frac{(2q-1)D}{2}$. Since this is independent of ρ we can use this for all digit sets.

$s_q = L_q$: This choice is only valid for a full-precision comparison and using the bound (4.30).

$s_q = x \cdot 2^{-\delta'}$, where x is an integer representable in τ bits, and $\delta' \leq \delta$. Later we shall see when this choice is applicable.

In the following we shall restrict the comparison functions to the first two types, or to δ-bit constants.

4.3.6 Performing the Comparison

2's complement Partial Remainder

For a 2's complement partial remainder the comparison is performed by a sign detection on the implied borrow-save encoding:

$$(\{\beta R_i\}_{\Delta})^s - (\{s_q\}_{\Delta})^b. \tag{4.42}$$

Redundant Partial Remainder

Assume that we are using Method 1 of Section 2.6 to perform the comparison. After assimilating the redundant partial remainder we may choose to perform the sign detection on the resulting borrow-save number using δ bits, where δ is smaller than the initial truncations applied to the redundant parts. The advantage of this approach will become apparent later on. In the case of the equivalent Method 2 of Section 2.6, this second truncation corresponds to a redundant subtraction of size equal to the larger of the two truncations, followed by a δ-bit sign detection, on the redundant result. This addition and truncation introduces an additional error, subject of investigation.

Carry-save encoding

The truncated non-redundant partial remainder is formed as follows:

$$\{\beta R_i\}_\Delta = \left\{ \left(\{(\beta R_i)^s\}_{\tau+p} + \{(\beta R_i)^c\}_{\tau+r} \right) \right\}_{\tau+\delta}, \tag{4.43}$$

where $\delta \leq g = \max\{p, r\}$. The range of the error originating from the second truncation is:

$$\epsilon_{R(add)} \in \left[0, 2^{-\delta} - 2^{-g}\right]. \tag{4.44}$$

The total truncation error is then:

$$\begin{aligned} \epsilon_R &= \epsilon_{R(cs)} + \epsilon_{R(add)} \\ &\in \left[0, 2^{-p} + 2^{-r} - 2 \cdot 2^{-l} + 2^{-\delta} - 2^{-g}\right] \\ &= \left[0, 2^{-\delta} + 2^{-\min\{p,r\}} - 2 \cdot 2^{-l}\right]. \end{aligned} \tag{4.45}$$

Observe that the case $p = r = l$ is equivalent to the case with a 2's complement partial remainder, as expected.

Borrow-save encoding

The truncated partial remainder is again formed by an addition, and a second truncation:

$$\{\beta R_i\}_\Delta = \left\{ \left(\{(\beta R_i)^s\}_{\tau+p} - \left\{ (\beta R_i)^b \right\}_{\tau+n} \right) \right\}_{\tau+\delta}, \tag{4.46}$$

where $\delta \leq g = \max\{p, n\}$. The range of this error is:

$$\epsilon_{R(add)} \in \left[0, 2^{-\delta} - 2^{-g}\right]. \tag{4.47}$$

And the total error is:

$$\epsilon_R = \epsilon_{R(bs)} + \epsilon_{R(add)}$$

$$\in \left[-2^{-n} + 2^{-l}, 2^{-p} + 2^{-\delta} - 2^{-g} - 2^{-l}\right] \tag{4.48}$$

$$= \left[-2^{-n} + 2^{-l}, 2^{-p} + 2^{-\delta} - 2^{-\max\{p,n\}} - 2^{-l}\right]. \tag{4.49}$$

Observe that having $n > p$ always leads to a larger error range than $p \geq n$. The case $p = n = l$ is again equivalent to the 2's complement case.

4.3.7 Comparing only with Positive Comparison Functions

To reduce the size of the division circuit, we may perform only the comparisons involving the positive comparison functions. In Section 4.2 we saw that we can use only the positive selection bounds. We describe how to apply this principle to the truncated comparison functions.

The comparison can be divided into two cases. If $\{\beta R_i\}_\Delta \geq 0$ then we proceed with the comparison as before. However when $\{\beta R_i\}_\Delta < 0$, we wish to compare a transformed value $\{\beta R_i\}'_\Delta$, with the positive comparison functions, in such a way that $-q$ is the correct quotient digit, when q is the quotient digit determined by the comparison. This gives us the following implementation:

$$\{\beta R_i\}_\Delta \geq 0 : \{s_q\}_\Delta \leq \{\beta R_i\}_\Delta < \{s_{q+1}\}_\Delta \quad \Rightarrow \quad q_{i+1} = q, \tag{4.50}$$

$$\{\beta R_i\}_\Delta < 0 : \{s_q\}_\Delta \leq \{\beta R_i\}'_{\Delta'} < \{s_{q+1}\}_\Delta \quad \Rightarrow \quad q_{i+1} = -q. \tag{4.51}$$

Let $\epsilon_{R'}$ be the error corresponding to truncating $\beta R'_i$

$$\beta R'_i = \{\beta R_i\}'_{\Delta'} + \epsilon_{R'}. \tag{4.52}$$

The case $\{\beta R_i\}_\Delta < 0$ implies

$$\{\beta R_i\}_\Delta + \epsilon_R \in [L_{-q}, U_{-q}] \tag{4.53}$$

$$\{\beta R_i\}'_{\Delta'} + \epsilon_{R'} \in [L_q, U_q]. \tag{4.54}$$

Since the quotient digit selection intervals are symmetric around $\beta R_i = 0$, we have

$$L_{-q} = -U_q, \quad U_{-q} = -L_q \tag{4.55}$$

From 4.53 we see

$$\begin{aligned} \{\beta R_i\}_\Delta + \epsilon_R &\geq L_{-q} \\ \{\beta R_i\}_\Delta + \inf \epsilon_R &\geq -U_q \\ -\left(\{\beta R_i\}_\Delta + \inf \epsilon_R\right) &\leq U_q, \end{aligned}$$

and from 4.54 we have

$$\begin{aligned} \{\beta R_i\}'_{\Delta'} + \epsilon_{R'} &\leq U_q \\ \{\beta R_i\}'_{\Delta'} + \sup \epsilon_{R'} &\leq U_q, \end{aligned}$$

so we deduce

$$\{\beta R_i\}'_{\Delta'} + \inf \epsilon_{R'} = -\left(\{\beta R_i\}_\Delta + \sup \epsilon_R\right). \tag{4.56}$$

In an analogous fashion we get the equality,

$$\{\beta R_i\}'_{\Delta'} + \sup \epsilon_{R'} = -\left(\{\beta R_i\}_\Delta + \inf \epsilon_R\right). \tag{4.57}$$

Geometrically this means that we can determine the correct quotient digit by reflecting the area of uncertainty in the D-axis, as illustrated in Figure 4.9. The reflection of the area of uncertainty depends on the encoding of the partial remainder, so we analyze the three possibilities.

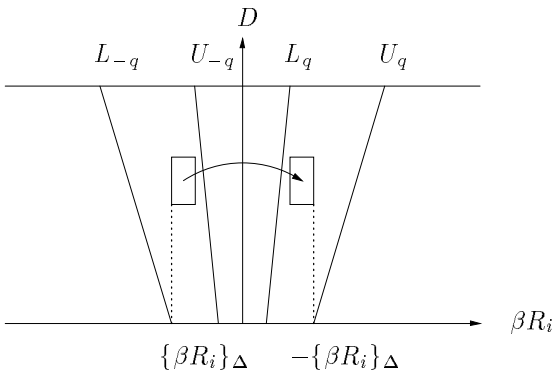

Figure 4.9: Reflecting the area of uncertainty.

2's complement encoding Using (4.56) and (2.13) we get:

$$\begin{aligned}\{\beta R_i\}'_{\Delta'} &= -\{\beta R_i\}_\Delta - \sup \epsilon_R - \inf \epsilon_{R'} \\ &= -\{\beta R_i\}_\Delta - 2^{-\delta} - 0 \\ &= \overline{\{\beta R_i\}_\Delta}\end{aligned}$$

and from (4.57) we get:

$$\begin{aligned}\{\beta R_i\}'_{\Delta'} &= -\{\beta R_i\}_\Delta - \inf \epsilon_R - \sup \epsilon_{R'} \\ &= -\{\beta R_i\}_\Delta - 0 - 2^{-\delta'} \\ &= \overline{\{\beta R_i\}_\Delta}\end{aligned}$$

where the last equality is valid if $\delta = \delta'$. So the encoding of $\{\beta R_i\}'_{\Delta'}$ is obtained by forming the ones complement of $\{\beta R_i\}_\Delta$.

Borrow-save encoding Using (4.56) we get:

$$\begin{aligned}\{\beta R_i\}'_{\Delta'} &= -\{\beta R_i\}_\Delta - \sup \epsilon_R - \inf \epsilon_{R'} \\ &= -\{\beta R_i\}_\Delta - 2^{-p} + 2^{-n'}\end{aligned}$$

(4.57) gives us:

$$\begin{aligned}\{\beta R_i\}'_{\Delta'} &= -\{\beta R_i\}_\Delta - \inf \epsilon_R - \sup \epsilon_{R'} \\ &= -\{\beta R_i\}_\Delta + 2^{-n} - 2^{-p'}\end{aligned}$$

If we let $p' = n$ and $n' = p$ we have:

$$\{\beta R_i\}'_\Delta = -\{\beta R_i\}_\Delta \tag{4.58}$$

So in this case we swap the borrow and save registers, to obtain encoding of $\{\beta R_i\}'_{\Delta'}$.

Carry-save encoding Using (4.56) we get:

$$\begin{aligned}\{\beta R_i\}'_{\Delta'} &= -\{\beta R_i\}_\Delta - \sup\epsilon_R - \inf\epsilon_{R'} \\ &= -\{\beta R_i\}_\Delta - 2^{-p} - 2^{-r} - 0 \\ &= -(\{(\beta R_i)^c\}_{\tau+p} + \{(\beta R_i)^s\}_{\tau+r}) - 2^{-p} - 2^{-r} \\ &= \overline{\{(\beta R_i)^c\}_{\tau+p}} + \overline{\{(\beta R_i)^s\}_{\tau+r}}\end{aligned}$$

and (4.57) implies:

$$\begin{aligned}\{\beta R_i\}'_{\Delta'} &= -\{\beta R_i\}_\Delta - \inf\epsilon_R - \sup\epsilon_{R'} \\ &= -\{\beta R_i\}_\delta - 0 - 2^{-p'} - 2^{-r'} \\ &= -(\{(\beta R_i)^c\}_{\tau+p} + \{(\beta R_i)^s\}_{\tau+r}) - 2^{-p'} - 2^{-r'} \\ &= \overline{\{(\beta R_i)^c\}_{\tau+p}} + \overline{\{(\beta R_i)^s\}_{\tau+r}}\end{aligned}$$

where the last equality is valid if $p' = p$ and $r' = r$. For carry-save we invert the truncated carry and save registers.

In all three cases the extra work, resulting from using only the positive comparison functions, is small. One problem remains, namely how to determine the sign of $\{\beta R_i\}_\Delta$. For R_i in 2's complement form we can just examine the sign bit. For R_i in redundant form we need to convert $\{\beta R_i\}_\Delta$ with δ fractional bits, to non-redundant form, from which an 'approximate' sign can be determined. If that number is negative then we use the transformed value $\{\beta R_i\}'_\Delta$. And if it is non-negative then for 2's complement and carry-save we know that βR_i is non-negative. For borrow-save we know that δ is selected so that (4.8) is satisfied, which in turn means that $\beta R_i \geq L_0$. So in all cases we know that βR_i is within the selection intervals corresponding to the non-negative digits.

4.3.8 Using only Positive Truncated Comparison Functions

Another way to reduce the size of the division circuit, is to perform all the comparisons, but using only the positive truncated comparison functions. The comparisons with the non-positive comparison functions are then performed using derivations of the positive comparison functions. This only makes sense whenever the non-positive comparison functions are the additive inverse of the positive ones, as is the case of $s_q = (2q-1)D/2$. This is equivalent to:

$$\{s_q\}'_{\Delta'} + \inf\epsilon_{s'} = -(\{s_q\}_\Delta + \sup\epsilon_s)\,, \tag{4.59}$$

$$\{s_q\}'_{\Delta'} + \sup\epsilon_{s'} = -(\{s_q\}_\Delta + \inf\epsilon_s)\,. \tag{4.60}$$

For a 2's complement encoding this implies that we must form the negative comparison functions as the ones complement of the positive comparison functions.

4.3.9 Using Constant Comparison Functions

Since we have to compute and possibly store the comparison functions, it would be preferable if we could choose a constant. I.e. one that is independent of D, and is representable in δ fraction bits. If we can use a constant that requires less than δ fractional bits, then the resulting comparison is simpler. Let us develop an expression for when we can use a constant, as a comparison function. Let s_q be a constant representable with δ fractional bits. We use

(4.32) and (4.33) as bounds on s_q. Since s_q is representable with δ fraction bits we have $\epsilon_s = 0$ so:

$$\begin{aligned} s_q &\geq \left\lceil \left(\max_D L_q - \inf \epsilon_R \right) 2^\delta \right\rceil 2^{-\delta}, \\ s_q &\leq \left\lfloor \left(\min_D U_{q-1} - \sup \epsilon_R + 2^{-\delta} \right) 2^\delta \right\rfloor 2^{-\delta}. \end{aligned}$$

Condition for the existence of such an s_q:

$$\left\lfloor \left(\min_D U_{q-1} - \sup \epsilon_R + 2^{-\delta} \right) 2^\delta \right\rfloor 2^{-\delta} \geq \left\lceil \left(\max_D L_q - \inf \epsilon_R \right) 2^\delta \right\rceil 2^{-\delta}. \tag{4.61}$$

This requirement is shown in Figure 4.10. For $q > 0$ we have:

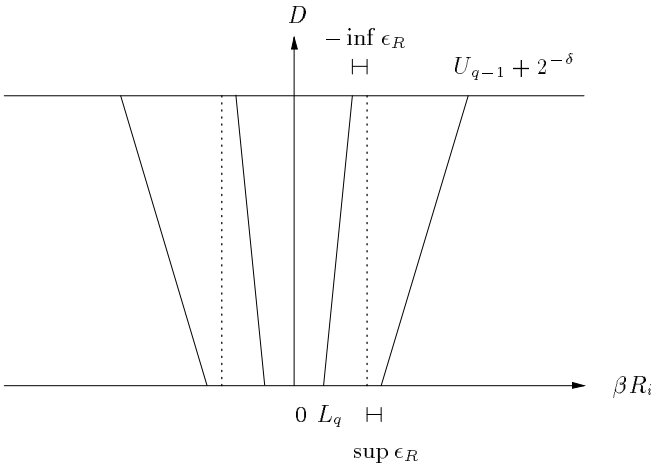

Figure 4.10: Condition for constant comparison function.

$$\left\lfloor \left((q - 1 + \rho)\frac{1}{2} - \sup \epsilon_R + 2^{-\delta} \right) 2^\delta \right\rfloor 2^{-\delta} \geq \left\lceil \left((q - \rho) - \inf \epsilon_R \right) 2^\delta \right\rceil 2^{-\delta}. \tag{4.62}$$

and for $q \leq 0$:

$$\left\lfloor \left((q - 1 + \rho) - \sup \epsilon_R + 2^{-\delta} \right) 2^\delta \right\rfloor 2^{-\delta} \geq \left\lceil \left((q - \rho)\frac{1}{2} - \inf \epsilon_R \right) 2^\delta \right\rceil 2^{-\delta}. \tag{4.63}$$

The positive quotient digits, for which we can use a constant comparison function are shown in Tables 4.1, 4.2 and 4.3.

Notice that in the carry-save case we sometimes need one bit less for δ, compared with the borrow-save case. In general we can only use a constant for $q = 1$ if the degree of redundancy is sufficiently high. Likewise for $\beta \geq 4$ we can use a constant in the case $q = 2$ if we use a non-redundant encoding of the partial remainder, and a maximally redundant digit set. When choosing the value of the constant comparison functions the following bounds should be satisfied for $q > 0$:

$$s_q \leq \left\lfloor \left((q - 1 + \rho)\frac{1}{2} - \sup \epsilon_R + 2^{-\delta} \right) 2^\delta \right\rfloor 2^{-\delta} \tag{4.64}$$

$$s_q \geq \left\lceil \left((q - \rho) - \inf \epsilon_R \right) 2^\delta \right\rceil 2^{-\delta}. \tag{4.65}$$

β	a	$\{q\}$	δ
2	1	1	1
4	2	$\emptyset$	
4	3	1,2	0
8	4	$\emptyset$	
8	5	1	4
8	6	1	2
8	7	1,2	0

Table 4.1: Constant comparison functions for 2's complement partial remainder.

β	a	$\{q\}$	p	r	δ
2	1	1	1	1	0
4	2	$\emptyset$			
4	3	1	1	1	0
8	4	$\emptyset$			
8	5	1	5	5	4
8	6	1	3	3	2
8	7	1	1	1	0
16	8	$\emptyset$			
16	9	$\emptyset$			
16	10	$\emptyset$			
16	11	1	5	5	4
16	12	1	3	3	2
16	13	1	3	3	2
16	14	1	3	3	2
16	15	1	1	1	0

Table 4.2: Constant comparison functions for carry-save partial remainder.

β	a	$\{q\}$	p	r	δ
2	1	1	1	1	1
4	2	$\emptyset$			
4	3	1	1	1	1
8	4	$\emptyset$			
8	5	1	5	5	5
8	6	1	3	3	3
8	7	1	1	1	1
16	8	$\emptyset$			
16	9	$\emptyset$			
16	10	$\emptyset$			
16	11	1	5	5	4
16	12	1	3	3	3
16	13	1	3	3	3
16	14	1	3	3	2
16	15	1	1	1	1

Table 4.3: Constant comparison functions for borrow-save partial remainder.

And for $q \leq 0$:

$$s_q \leq \left\lfloor \left((q - 1 + \rho) - \sup \epsilon_R + 2^{-\delta} \right) 2^{\delta} \right\rfloor 2^{-\delta} \tag{4.66}$$

$$s_q \geq \left\lceil ((q - \rho)\frac{1}{2} - \inf \epsilon_R) 2^{\delta} \right\rceil 2^{-\delta}. \tag{4.67}$$

4.3.10 Results

The number of bits required in the comparison depends on a number of factors, namely the representations used for the partial remainder and the comparison functions, as well as the choice of comparison function. A single expression for Δ, can therefore not be derived, but we can derive expressions for the possible encodings of the partial remainder.

Partial remainder in 2's complement encoding

From (4.32), we get:

$$\begin{aligned} \sup \epsilon_s - \inf \epsilon_R &\leq s_q - L_q \\ 2^{-\delta} - 0 &\leq s_q - L_q \\ 2^{-\delta} - 0 &\leq \min_D (s_q - L_q) \\ \delta &\geq \left\lceil -\log_2 \left(\min_D (s_q - L_q) \right) \right\rceil . \end{aligned} \tag{4.68}$$

From (4.33), we get:

$$\begin{aligned} \sup \epsilon_R - \inf \epsilon_s &\leq U_{q-1} - s_q + 2^{-\delta} \\ 2^{-\delta} - 0 &\leq U_{q-1} - s_q + 2^{-\delta} \\ 0 &\leq U_{q-1} - s_q. \end{aligned}$$

This is always satisfied, because of (4.22). To minimize δ, s_q should be chosen close to U_{q-1}. Note that the closer we get to $s_q = L_q$, the closer we get to requiring, $\delta = l$, i.e. a full-precision comparison.

Carry-Save

Assume that $p \geq r$. This assumption can be made since the carry and save registers are equivalent. We then get the following requirements on δ, p and r:
From (4.32) and (4.45), we get:

$$\begin{aligned} \sup \epsilon_s - \inf \epsilon_R &\leq s_q - L_q \\ 2^{-\delta} - 0 &\leq \min_D (s_q - L_q) \\ \delta &\geq \left\lceil -\log_2 \left(\min_D (s_q - L_q) \right) \right\rceil . \end{aligned} \tag{4.69}$$

From (4.33) and (4.45), we get:

$$\begin{aligned} \sup \epsilon_R - \inf \epsilon_s &\leq U_{q-1} - s_q + 2^{-\delta} \\ 2^{-\delta} + 2^{-\min\{p,r\}} - 0 &\leq \min_D (U_{q-1} - s_q) + 2^{-\delta} \\ r &\geq \left\lceil -\log_2 \left(\min_D (U_{q-1} - s_q) \right) \right\rceil . \end{aligned} \tag{4.70}$$

In this case we can determine δ and r once s_q is chosen. To minimize δ we should thus choose s_q close to the middle of the interval $[L_q, U_{q-1}]$. Choosing s_q close to L_q or U_{q-1} requires a full-precision comparison.

Borrow-Save

From (4.32) and (4.49), we get:

$$\begin{aligned} \sup \epsilon_s - \inf \epsilon_R &\leq s_q - L_q \\ 2^{-\delta} + 2^{-n} &\leq \min_D (s_q - L_q). \end{aligned}$$

From (4.33) and (4.49), we get:

$$\begin{aligned} \sup \epsilon_R - \inf \epsilon_s &\leq U_{q-1} - s_q + 2^{-\delta} \\ 2^{-p} + 2^{-\delta} - 2^{-\max\{p,n\}} - 0 &\leq \min_D (U_{q-1} - s_q) + 2^{-\delta} \\ 2^{-p} - 2^{-\max\{p,n\}} &\leq \min_D (U_{q-1} - s_q). \end{aligned}$$

For the case $\delta = p = n$, we get:

$$\begin{aligned} 2^{-\delta} + 2^{-\delta} &\leq s_q - L_q \\ \delta &\geq \left\lceil -\log_2 \left(\frac{s_q - L_q}{2} \right) \right\rceil , \end{aligned} \tag{4.71}$$

and

$$0 \leq U_{q-1} - s_q .$$

The latter is always satisfied, because of (4.22). Here we should also select s_q close to U_{q-1}, to minimize δ, and we cannot select $s_q = L_q$.

4.3.11 Examples

Table 4.4 shows, δ and Δ, the required number of bits in the comparison, for some of the common implementations. The number of bits required in the comparison is $\Delta = \tau + \delta$. However, if we perform the positive comparison then we need $\Delta - 1$ comparisons. Note that if we want to choose $s_q = U_{q-1}$, we can only use 2's complement or borrow-save encoding. But if we want to choose $s_q = (U_{q-1} + L_q)/2$ then carry-save encoding is preferred over borrow-save encoding. Vice versa if we want to use a specific representation for the partial remainder, then there are some s_q's which are more suited than others. All results were calculated using the equations derived in Section 4.3.10, and from Tables 4.1, 4.2, and 4.3. Notice that in the cases with redundant partial remainder and less than maximally redundant digit set there is an advantage in truncating the partial remainder to more than Δ bits.

Radix 2

Here we have $\beta = 2$, $a = 1$, and $\rho = 1$.

2's Complement Encoding of Partial Remainder This is the simplest possible implementation of the division algorithm. We choose $s_q = U_{q-1} = qD$, or equivalently $s_0 = 0$, and $s_1 = D$. However, it is evident from Table 4.1 that we can use a constant comparison function s_1. From (4.64) and (4.65) we see that we can choose $s_1 \in [0, 1/2]$. Since we require $s_1 > s_0$ we choose $s_1 = 1/2$.

Carry-Save Encoding of Partial Remainder We use $p = r = \delta = 1$. The bounds on the comparison functions are calculated using (4.64)-(4.67).

$$s_0 \in [-1/2, 0 - 2^{-\delta}]$$

and

$$s_1 \in [0, 1/2 - 2^{-\delta}]$$

With $\delta = 1$ we can choose $s_q = \{-1/2, 0\}$.

Radix 4, Minimally Redundant

Here we have $\beta = 4, a = 2$, and $\rho = 2/3$. For this case there are several possible implementations. We will examine some of the possibilities.

β	a	R_i encoding	s_q	τ	p	n, r	δ	Δ	comparators$\star$
2	1	2's complement	$q/2$	3	1	1	1$*$	3	2+0
2	1	borrow-save	$q/2$	3	1	1	1$*$	4	2+0
2	1	carry-save	$-q/2$	3	1	1	1$*$	4	2+0
4	2	borrow-save	$(U_{q-1}+L_q)/2$	3	5	5	5	8	0+4
4	2	borrow-save	$(U_{q-1}+L_q)/2$	3	6	4	4	7	0+4
4	2	carry-save	$(U_{q-1}+L_q)/2$	3	4	4	4	7 §	0+4
4	3	2's complement	U_{q-1}	4	1	1	1	5 †	4+2
4	3	borrow-save	U_{q-1}	4	2	2	2	6	2+4
4	3	carry-save	$(U_{q-1}+L_q)/2$	4	2	2	2	6	2+4
8	4	borrow-save	$(U_{q-1}+L_q)/2$	4	6	6	6	10 ‡	0+8
8	4	carry-save	$(U_{q-1}+L_q)/2$	4	5	5	5	9	0+8
8	7	2's complement	U_{q-1}	5	1	1	1	6 ‡	4+10
8	7	borrow-save	U_{q-1}	5	2	2	2	7	2+12

Table 4.4: Bits required for direct comparison.
$\star$: The first number is the number of comparators corresponding to the constant comparison functions.
$*$: Since the comparison functions are constants representable in 1 fraction bit, we only need 1 fractional bit in the comparison.
†: The hybrid radix-2/radix-4 implementation described in [10] can be modified into this by splitting case 1, into two radix-4 cases.
‡: Same as [11].
§: Same implementation as [19]. They required $\Delta = 8$.

Borrow-save encoding of partial remainder The choice $s_q = U_{q-1}$ leads to: $s_{-1} = -(4/3)D, s_0 = -(1/3)D, s_1 = (2/3)D, s_2 = (5/3)D$. But since we cannot form these comparison functions easily we turn our attention to another set of comparison functions. If we want simpler comparison functions, we can choose $s_q = (U_{q-1} + L_q)/2$, or equivalently: $s_{-1} = -(1/2)D, s_0 = 0, s_1 = (1/2)D, s_2 = (3/2)D$. Let us calculate δ for this choice. From (4.71) we get:

$$\begin{aligned}
\delta &\geq \left\lceil -\log_2 \left(\frac{\frac{(U_{q-1}+L_q)}{2} - L_q}{2} \right) \right\rceil \\
&= \left\lceil -\log_2 \left(\frac{U_{q-1} + L_q - 2L_q}{4} \right) \right\rceil \\
&= \left\lceil -\log_2 \left(\frac{U_{q-1} - L_q}{4} \right) \right\rceil \\
&= \left\lceil -\log_2 \left(\frac{(2\rho - 1)D}{4} \right) \right\rceil \\
&= \left\lceil -\log_2 \left(\frac{1}{12} D \right) \right\rceil .
\end{aligned}$$

Again the maximum value of the lower bound of δ is obtained at $D = 1/2$,

$$D = \frac{1}{2} \Rightarrow \delta \geq \left\lceil -\log_2 \left(\frac{1}{24} \right) \right\rceil = 5,$$
$$\delta \geq 5 \Rightarrow \Delta \geq 5 + \tau - 1 = 7.$$

This choice is shown graphically as a staircase in the PD-plot in Figure 4.11. From Table 4.4

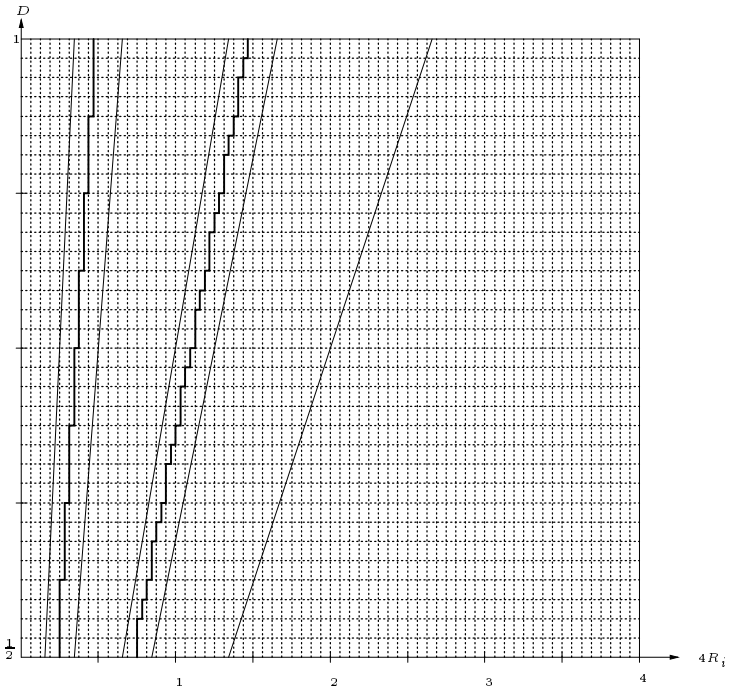

Figure 4.11: PD-plot for $\beta = 4, a = 2, s_q = \frac{(U_{q-1}+L_q)}{2}, \delta = 5$.

we see that by using 6 and 4 bits of the borrow and save parts we can use $\delta = 4$ and thus one bit less in the comparison.

Radix 4, Maximally Redundant

Here we have $\beta = 4, a = 3$, and $\rho = 1$.

Borrow-save encoding of partial remainder We can choose $s_q = U_{q-1} = qD$, giving us: $s_{-2} = -2D, s_{-1} = -D, s_0 = 0, s_1 = D, s_2 = 2D$, and $s_3 = 3D$. Let us calculate δ for this choice. From (4.71) we get:

$$\begin{aligned} \delta &\geq \left\lceil -\log_2 \left(\frac{U_{q-1} - L_q}{2} \right) \right\rceil \\ &= \left\lceil -\log_2 \left(\frac{(2\rho - 1)D}{2} \right) \right\rceil \\ &= \left\lceil -\log_2 \left(\frac{D}{2} \right) \right\rceil . \end{aligned}$$

The maximum value of the lower bound of δ is obtained at $D = 1/2$,

$$D = \frac{1}{2} \Rightarrow \delta \geq -\log_2\left(\frac{1}{4}\right) = 2,$$
$$\delta \geq 2 \Rightarrow \Delta \geq 2 + \tau - 1 = 5.$$

This choice is shown graphically as a staircase in the PD-plot in Figure 4.12.

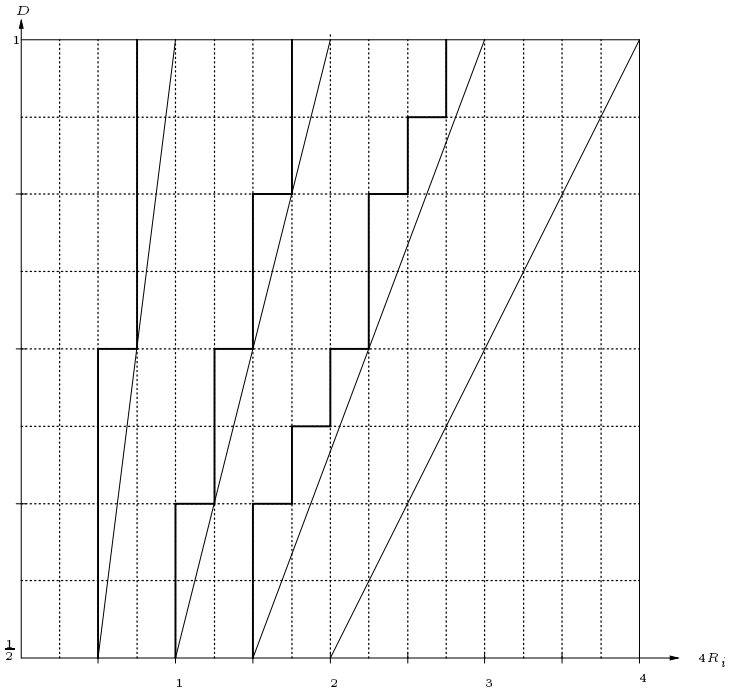

Figure 4.12: PD-plot for $\beta = 4, a = 3, s_q = U_{q-1}, \delta = 2$.

Carry-save encoding of partial remainder As mentioned above we should choose s_q close to the middle of the interval $[L_q, U_{q-1}]$, so we choose $s_q = (U_{q-1}+L_q)/2 = (2q-1)D/2$, or $s_{-2} = -(5/2)D, s_{-1} = -(3/2)D, s_0 = -(1/2)D, s_1 = (1/2)D, s_2 = (3/2)D$, and $s_3 = (5/2)D$. From (4.69) we get:

$$\begin{aligned}
\delta &\geq \left\lceil -\log_2\left(\frac{U_{q-1} + L_q}{2} - L_q\right)\right\rceil \\
&= \left\lceil -\log_2\left(\frac{U_{q-1} - L_q}{2}\right)\right\rceil \\
&= \left\lceil -\log_2\left(\frac{(2\rho - 1)D}{2}\right)\right\rceil \\
&= \left\lceil -\log_2\left(\frac{D}{2}\right)\right\rceil .
\end{aligned}$$

Note that this is exactly the same result we obtained for the borrow-save encoding with $s_q = U_{q-1}$.

4.4 Table Lookup

The table lookup method is by far the most popular implementation of the quotient digit selection function. It recently became famous due to the Intel Pentium bug [3].

In the direct comparison method we implemented equation (4.16) by performing the comparison directly. The flexibility in that implementation is limited by the constant relation between the comparison functions and the divisor D, over the interval $[\min D, \max D]$. A more flexible implementation can be obtained by allowing the selection bounds to variate over that interval. This method is known as the table lookup method [5] since the resulting quotient-digit selection function can be depicted as a table, where we use δ bits of the divisor D, and c bits of the shifted partial remainder, βR_i as inputs to a lookup table, to determine the next quotient digit, as shown in Figure 4.13. In [4] Neil Burgess and Ted Williams did an

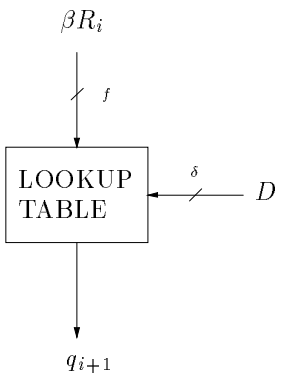

Figure 4.13: Table lookup.

analysis of the number of bits required to determine the next quotient digit. Oberman and Flynn [14] have investigated actual implementations of different tables, based on a similar analysis. In the following we shall develop an expression to determine valid truncations of the divisor and the shifted partial remainder respectively. For simplicity we will use the common term $\{\beta R_i\}_c$ to denote truncation of the partial remainder to c bits, just as we used $\{\beta R_i\}_\Delta$ in Section 4.3. If βR_i is in redundant representation we will allow different truncations of the two redundant parts. Let ϵ_R be the error corresponding to the truncation of βR_i:

$$\beta R_i = \{\beta R_i\}_c + \epsilon_R. \tag{4.72}$$

The range of ϵ_R will depend on the encoding of βR_i and the truncations of βR_i, subject to later investigation. Let τ be the number of integer bits needed to represent $\{\beta R_i\}_c$, let f be the number of fraction bits in the truncation, and let l be the number of fraction bits used in the representation of βR_i. We then have $c = \tau + f$. Let $D \in [1/2, 1[$ be the divisor, let $\{D\}_\delta$ denote truncation of D to δ bits, and let ϵ_D be the corresponding truncation error:

$$D = \{D\}_\delta + \epsilon_D. \tag{4.73}$$

Now partition the interval $[1/2, 1[$ into $2^{\delta-1}$ parts of equal size. Since D is truncated to δ fraction bits, and normalized to the interval $[1/2, 1[$ we can write the truncated divisor on the form:

$$\{D\}_\delta = \frac{1}{2} + m2^{-\delta}, \tag{4.74}$$

where m is an integer satisfying $m \in [0, 2^{\delta-1}[$. Let s_q be a function of D which is constant on each of the intervals $[\{D\}_\delta + \min \epsilon_D, \{D\}_\delta + \max \epsilon_D]$, and is representable with f fraction bits. We can then write s_q on the following form:

$$s_q = \begin{cases} x_0 2^{-f} & \text{if} \quad D \in [1/2, 1/2 + 2^{-\delta}[\\ \vdots & \\ x_m 2^{-f} & \text{if} \quad D \in [1/2 + m2^{-\delta}, 1/2 + (m+1)2^{-\delta}[\\ \vdots & \\ x_{2^{\delta-1}-1} 2^{-f} & \text{if} \quad D \in [1/2 + (2^{\delta-1} - 1)2^{-\delta}, 1[, \end{cases}$$

where $x_0, \ldots, x_{2^{\delta-1}-1}$ are integers representable with τ bits. We use the following implementation of $q\_select(\beta R_i, D)$:

$$\{s_q\}_c \leq \{\beta R_i\}_c < \{s_{q+1}\}_c \Rightarrow q_{i+1} = q. \tag{4.75}$$

We want to perform this comparison on each interval of D, in parallel. This can be achieved indirectly by constructing a two-dimensional table, using truncated values of D and βR_i.

4.4.1 Truncation Errors

Divisor

The divisor is in 2's complement encoding:

$$\epsilon_D \in [0, 2^{-\delta} - \text{ulp } D]. \tag{4.76}$$

Partial Remainder

The truncation errors are identical to the ones given in Section 4.3.2.

4.4.2 Fraction bits

We will now determine what values of δ, and truncations of βR_i result in a valid selection function. In analogy with the direct comparison case we get the following equations, from (4.75):

$$\begin{aligned} s_q &\geq L_q + \max \epsilon_s - \min \epsilon_R & (4.77) \\ s_q &\leq U_{q-1} + \min \epsilon_s - \max \epsilon_R + 2^{-f}. & (4.78) \end{aligned}$$

Since s_q is given with c bits we have $\epsilon_s = 0$, which gives us the following bounds on s_q:

$$\begin{aligned} s_q &\geq L_q - \min \epsilon_R & (4.79) \\ s_q &\leq U_{q-1} - \max \epsilon_R + 2^{-f}. & (4.80) \end{aligned}$$

We will use the tighter bounds:

$$\begin{aligned} s_q &\geq L_q - \inf \epsilon_R & (4.81) \\ s_q &\leq U_{q-1} - \sup \epsilon_R + 2^{-f}. & (4.82) \end{aligned}$$

These expressions have to be satisfied for the interval $D' \in [\{D\}_\delta + \min \epsilon_D, \{D\}_\delta + \max \epsilon_D]$, so we get:

$$s_q \geq \max_{D'} L_q - \inf \epsilon_R \tag{4.83}$$

$$s_q \leq \min_{D'} U_{q-1} - \sup \epsilon_R + 2^{-f}. \tag{4.84}$$

And because s_q has f fraction bits:

$$s_q \geq \left\lceil \left(\max_{D'} L_q - \inf \epsilon_R \right) 2^f \right\rceil 2^{-f} \tag{4.85}$$

$$s_q \leq \left\lfloor \left(\min_{D'} U_{q-1} - \sup \epsilon_R + 2^{-f} \right) 2^c \right\rfloor 2^{-f} \tag{4.86}$$

Condition for the existence of such an s_q:

$$\left\lfloor \left(\min_{D'} U_{q-1} - \sup \epsilon_R + 2^{-f} \right) 2^f \right\rfloor 2^{-f} \geq \left\lceil \left(\max_{D'} L_q - \inf \epsilon_R \right) 2^f \right\rceil 2^{-f}, \tag{4.87}$$

where $\min_{D'} U_{q-1}$ and $\max_{D'} L_q$ are given by:
$q > 0$:

$$\max_{D'} L_q = (q - \rho)\left(\{D\}_\delta + \max \epsilon_D\right) \tag{4.88}$$

$$= (q - \rho)\left(\frac{1}{2} + (m+1)2^{-\delta} - \operatorname{ulp} D\right) \tag{4.89}$$

$$\min_{D'} U_{q-1} = (q - 1 + \rho)\left(\{D\}_\delta + \min \epsilon_D\right) \tag{4.90}$$

$$= (q - 1 + \rho)\left(\frac{1}{2} + m2^{-\delta}\right), \tag{4.91}$$

$q \leq 0$:

$$\max_{D'} L_q = (q - \rho)\left(\{D\}_\delta + \min \epsilon_D\right) \tag{4.92}$$

$$= (q - \rho)\left(\frac{1}{2} + m2^{-\delta}\right) \tag{4.93}$$

$$\min_{D'} U_{q-1} = (q - 1 + \rho)\left(\{D\}_\delta + \max \epsilon_D\right) \tag{4.94}$$

$$= (q - 1 + \rho)\left(\frac{1}{2} + (m+1)2^{-\delta} - \operatorname{ulp} D\right). \tag{4.95}$$

We then have the following equations:
$q > 0$:

$$\left\lfloor \left((q - 1 + \rho)\left(\frac{1}{2} + m2^{-\delta}\right) - \sup \epsilon_R + 2^{-f} \right) 2^f \right\rfloor 2^{-f} \geq$$
$$\left\lceil \left((q - \rho)\left(\frac{1}{2} + (m+1)2^{-\delta} - \operatorname{ulp} D\right) - \inf \epsilon_R \right) 2^f \right\rceil 2^{-f}, \tag{4.96}$$

$q \leq 0$:

$$\left\lfloor \left((q-1+\rho) \left(\frac{1}{2} + (m+1)2^{-\delta} - \operatorname{ulp} D \right) - \sup \epsilon_R + 2^{-f} \right) 2^f \right\rfloor 2^{-f} \geq \left\lceil \left((q-\rho) \left(\frac{1}{2} + m2^{-\delta} \right) - \inf \epsilon_R \right) 2^f \right\rceil 2^{-f}. \quad (4.97)$$

Equations (4.96) and (4.97) should be evaluated for every possible value of q and m to determine valid choices of δ and valid truncations of βR_i. If $2^f \sup \epsilon_R$ and $2^f \inf \epsilon_R$ are integers then equation (4.96) is identical to equation (14) in [4], where its derivation is based on Robertson diagrams. This is the case when c denotes the larger of the two truncations, and in the 2's complement case.

4.4.3 Integer bits

We now calculate the number of integer bits, τ, required to represent $\{\beta R_i\}_c$. The upper bound on the value of $\{\beta R_i\}_c$ is:

$$\{\beta R_i\}_c \leq \max(\beta R_i - \epsilon_R) \quad (4.98)$$

$$\{\beta R_i\}_c \leq \left\lfloor \max(\beta R_i - \epsilon_R) \cdot 2^f \right\rfloor \cdot 2^{-f} \quad (4.99)$$

$$\{\beta R_i\}_c \leq \left\lfloor (\beta(1 - \operatorname{ulp} D)\rho - \inf \epsilon_R) \cdot 2^f \right\rfloor \cdot 2^{-f}. \quad (4.100)$$

And the lower bound:

$$\{\beta R_i\}_c \geq \lceil (\beta(-1 + \operatorname{ulp} D)\rho - \sup \epsilon_R) \cdot 2^f \rceil \cdot 2^{-f}. \quad (4.101)$$

To determine the number of integer bits needed, we calculate the number of bits needed for the upper bound and the lower bound respectively, and use the one that requires the largest number of integer bits.
The upper bound gives us the following lower bound on the number of integer bits, including the sign bit:

$$\begin{aligned} 2^{\tau-1} &\geq \lfloor (\beta(1 - \operatorname{ulp} D)\rho - \inf \epsilon_R) \cdot 2^f \rfloor \cdot 2^{-f} \\ \tau &\geq \lceil \log_2(\lfloor (\beta(1 - \operatorname{ulp} D)\rho - \inf \epsilon_R) \cdot 2^f \rfloor \cdot 2^{-f}) \rceil + 1, \end{aligned} \quad (4.102)$$

Lower bound:

$$\begin{aligned} 2^{\tau-1} &\geq -\lceil (\beta(-1 + \operatorname{ulp} D)\rho - \sup \epsilon_R) \cdot 2^f \rceil \cdot 2^{-f} \\ \tau &\geq \lceil \log_2(-\lceil (\beta(-1 + \operatorname{ulp} D)\rho - \sup \epsilon_R) \cdot 2^f \rceil \cdot 2^{-f}) \rceil + 1. \end{aligned} \quad (4.103)$$

Thus, given the truncations of βR_i, we can determine the number of integer bits required.

4.4.4 Table Inputs

For a redundant partial remainder there are two possibilities when using the inputs of the partial remainder to the table.

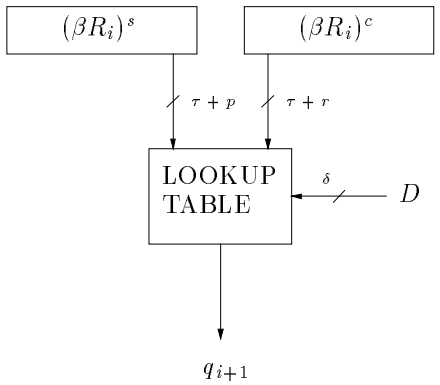

Figure 4.14: Direct input of bits.

Using Inputs Directly

Assume that we are using a carry-save encoding for the partial remainder. The first option is to use the $2 \cdot \tau + p + r$ inputs from the carry and save parts, together with the δ bits of D, as shown in Figure 4.14. In this case we have for the possible encodings of βR_i:

2's complement

$$\epsilon_R \in [0, 2^{-f} - 2^{-l}]. \tag{4.104}$$

carry-save

$$\epsilon_{R(cs)} \in \left[0, 2^{-p} + 2^{-r} - 2 \cdot 2^{-l}\right]. \tag{4.105}$$

borrow-save

$$\epsilon_{R(bs)} \in \left[-2^{-n} + 2^{-l}, 2^{-p} - 2^{-l}\right]. \tag{4.106}$$

Assimilating Redundant Inputs

Assume that the partial remainder is in carry-save encoding. To reduce the number of inputs to the table, we can assimilate the $p + r$ inputs from the partial remainder by using a small carry propagate adder to compute a 2's complement truncated partial remainder, and using $\tau + f \leq \tau + g$ bits from it as input to the table, where $g = \max\{p, r\}$. This procedure is illustrated in Figure 4.15. The total truncation error is then the sum of the error from truncating the partial remainder plus the error resulting from using the CPA, namely $\epsilon_{R(cpa)}$.

$$\epsilon_{R(cpa)} \in [0, 2^{-f} - 2^{-g}]. \tag{4.107}$$

If $f = g$ then we have the same error as in the case without CPA.

carry-save $f \leq g = \max\{p, r\}$

$$\begin{aligned} \epsilon_R &= \epsilon_{R(cs)} + \epsilon_{R(cpa)} \\ &\in \left[0, 2^{-p} + 2^{-r} - 2 \cdot 2^{-l} + 2^{-f} - 2^{-g}\right] \\ &= \left[0, 2^{-f} + 2^{-\min\{p,r\}} - 2 \cdot 2^{-l}\right]. \end{aligned} \tag{4.108}$$

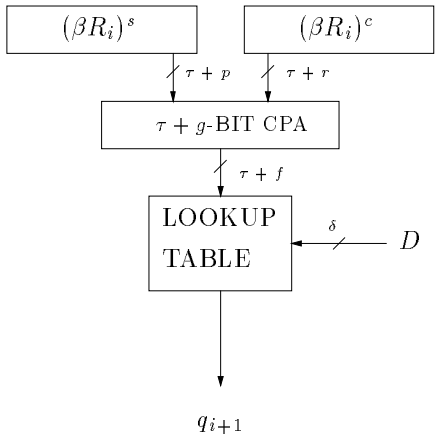

Figure 4.15: Assimilating table inputs.

borrow-save $f \leq g = \max\{p, n\}$

$$\begin{aligned} \epsilon_R &= \epsilon_{R(bs)} + \epsilon_{R(cpa)} \\ &\in \left[-2^{-n} + 2^{-l}, 2^{-p} + 2^{-f} - 2^{-g} - 2^{-l}\right]. \end{aligned} \tag{4.109}$$

Observe that $\sup \epsilon_R - \inf \epsilon_R$ is identical for carry-save and borrow-save in both cases. This means that equations (4.96) and (4.97) give the same result for either encoding, when $f = g$.

4.4.5 Using only Positive Selection Bounds

In Section 4.2 we saw that we can use only the positive selection bounds. The comparison can then be divided into two cases. If $\{\beta R_i\}_c \geq 0$ then we proceed with the comparison as before. However when $\{\beta R_i\}_c < 0$, we wish to compare a transformed value $\{\beta R_i\}'_{c'}$, with the positive selection bounds, in such a way that $-q_{i+1}$ is the correct quotient digit, when q_{i+1} is the quotient digit determined by the comparison. This gives us the following implementation:

$$\{\beta R_i\}_c \geq 0 : s_q \leq \{\beta R_i\}_c < s_{q+1} \quad \Rightarrow \quad q_{i+1} = q \tag{4.110}$$

$$\{\beta R_i\}_c < 0 : s_q \leq \{\beta R_i\}'_{c'} < s_{q+1} \quad \Rightarrow \quad q_{i+1} = -q. \tag{4.111}$$

Since the quotient digit selection intervals are symmetric around $\beta R_i = 0$, we can determine the correct quotient digit by reflecting the area of uncertainty in the D-axis, as illustrated in Figure 4.16. This gives us the following two expressions:

$$\{\beta R_i\}'_{c'} + \sup \epsilon_{R'} = -(\{\beta R_i\}_c + \inf \epsilon_R), \tag{4.112}$$

$$\{\beta R_i\}'_{c'} + \inf \epsilon_{R'} = -(\{\beta R_i\}_c + \sup \epsilon_R). \tag{4.113}$$

These equations are identical to the ones in Section 4.3.7. The encoding of $\{\beta R_i\}'_{c'}$ is therefore obtained in the same manner. For the results we refer to Section 4.3.7.

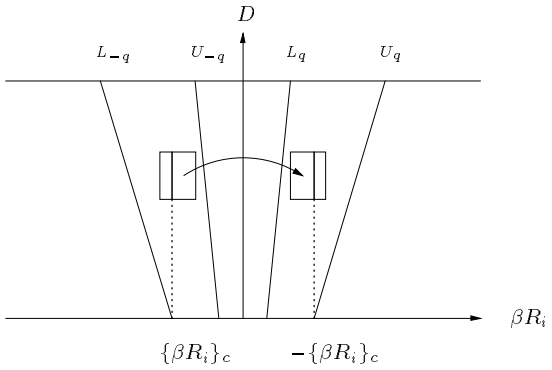

Figure 4.16: Reflecting the area of uncertainty.

4.4.6 Results

Table 4.5 shows the required number of inputs w to the lookup table, when using a 2's complement partial remainder. It was constructed by writing a small program which records valid truncations.

Tables 4.6 and 4.7 show the total number of bits, w, we require as input to the lookup table for carry-save representations of βR_i. The number of fractional bits of βR_i is denoted by p and r. The table was constructed by writing a small program which records values of p, r, δ and τ that result in a valid selection function, according to equations (4.96), (4.103), and (4.102). To reduce the table size we can insert a small $\tau + f$-bit carry-propagate adder to compute the non-redundant representation of the redundant partial remainder to $f \leq \max\{p, n\}$ bits. The total number of bits needed as input to the table is then given by $w = f + (\delta - 1) + (\tau - 1)$. Note that the most significant bit of D is not used since D is normalized. Also the msb of $\{\beta R_i\}_c$ is not used, because we only use the positive part of the table. The results are almost identical to the ones in [4], when $f = \max\{p, r\}$. The main difference is the number of integer bits needed, which is wrong in [4].

Although w is a measure of complexity for the table, a more detailed analysis was made in [14], by calculating the actual amount of logic needed to implement the tables as a PLA. Using $f < g$ does not necessarily result in a lower w, however it could reduce the number of PLA terms, since the resulting bounds on s_q are looser.

4.4.7 Examples

Once the required table size has been established, there are several ways to implement it. In this section we will look at some examples. We define $\tilde{L}_q$ and $\tilde{U}_{q-1}$ as:

$$\tilde{L}_q = \left\lceil \left(\max_{D'} L_q - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c} \tag{4.114}$$

$$\tilde{U}_{q-1} = \left\lfloor \left(\min_{D'} U_{q-1} - \sup \epsilon_R \right) 2^c \right\rfloor 2^{-c} + 2^{-c} \tag{4.115}$$

The lookup table can now be constructed by computing for all q the bounds $\tilde{L}_q$ and $\tilde{U}_{q-1}$, and selecting the $s_q \in [\tilde{L}_q, \tilde{U}_{q-1}]$, which needs the fewest fractional bits.

β	a	τ	f	δ	w
2	1	3	0	1	2
4	2	3	3	4	8
4	3	4	2	2	5
8	4	4	4	8	14
8	4	4	5	7	14
8	4	4	6	6	14
8	7	5	1	6	10
8	7	5	2	4	9
16	8	5	5	11	19
16	8	5	6	9	19
16	8	5	8	8	19
16	15	6	1	9	14
16	15	6	2	6	12
16	15	6	3	5	12

Table 4.5: Table Lookup truncations for 2's complement partial remainder.

Radix 2

2's complement From Table 4.5 we see that we can use $\delta = 1$, meaning that the quotient digit can be determined independently of the divisor. We can find the next quotient digit by inspecting the partial remainder only. Using only the positive part of the table, corresponds to using the sign of the partial remainder to determine the next quotient digit. This implementation is the well-known non-restoring division.

carry-save Let R_i be in carry-save form. From Table 4.6 we see that $\delta = 1$ indicating that the table is independent of D. The resulting table is shown in Table 4.8.

Radix 4, Minimally Redundant

Let R_i be in carry-save form. From Table 4.6 we see that we can use $\delta = 4$, $p = r = c = 4$ and $\tau = 3$ bits for the table. A possible implementation is shown in Table 4.9. The corresponding PD-plot for the positive half is shown in Figure 4.17. The constructed table is exactly the same as Table 3.4 in [5].

β	a	τ	p	r	f	δ	w
2	1	3	1	1	1	1	3
4	2	3	4	4	4	4	9
4	3	4	2	2	2	3	7
8	4	4	5	5	5	8	15
8	4	4	6	5	6	7	15
8	4	4	6	6	5	7	14
8	4	4	8	6	8	6	16
8	4	4	7	7	7	6	15
8	5	4	4	3	4	7	13
8	5	4	5	3	5	6	13
8	5	4	4	4	4	6	12
8	5	4	4	4	3	7	12
8	5	4	5	4	5	5	12
8	5	4	5	5	4	5	11
8	6	4	4	2	4	7	13
8	6	4	5	2	5	6	13
8	6	4	3	3	3	6	11
8	6	4	4	3	4	5	11
8	6	4	4	3	4	5	11
8	6	4	4	4	3	5	10
8	6	4	4	4	2	6	10
8	6	4	9	5	8	4	14
8	7	5	2	2	2	7	12
8	7	5	3	2	3	5	11
8	7	5	3	3	3	4	10
8	7	5	3	3	2	5	10

Table 4.6: Table Lookup truncations for redundant partial remainder radix 2-8.

β	a	τ	p	r	f	δ	w
16	8	5	6	6	6	12	21
16	8	5	7	6	7	10	20
16	8	5	8	6	8	9	20
16	8	5	7	7	7	9	19
16	8	5	7	7	6	10	19
16	8	5	8	8	6	9	18
16	8	5	9	9	9	8	20
16	9	5	5	4	5	10	18
16	9	5	6	4	6	9	18
16	9	5	7	4	7	8	18
16	9	5	5	5	5	8	16
16	9	5	7	5	7	7	17
16	9	5	6	6	6	7	16
16	9	5	6	6	5	8	16
16	9	5	6	6	4	9	16
16	9	5	7	7	5	7	15
16	9	5	7	7	4	8	15
16	10	5	5	3	5	10	18
16	10	5	6	3	6	9	18
16	10	5	7	3	7	8	18
16	10	5	4	4	4	8	15
16	10	5	5	4	5	7	15
16	10	5	5	5	4	7	14
16	10	5	6	6	6	6	15
16	10	5	6	6	4	7	14
16	11	5	4	3	4	8	15
16	11	5	5	3	5	7	15
16	11	5	4	4	4	7	14
16	11	5	4	4	3	8	14
16	11	5	6	4	6	6	15
16	11	5	5	5	5	6	14
16	11	5	5	5	3	7	14

β	a	τ	p	r	c	δ	w
16	11	5	6	6	4	6	13
16	11	5	6	6	3	7	13
16	12	5	5	2	5	9	17
16	12	5	7	2	7	8	18
16	12	5	3	3	3	8	14
16	12	5	4	3	4	7	14
16	12	5	4	4	4	6	13
16	12	5	4	4	3	7	13
16	12	5	6	6	3	6	12
16	12	5	7	7	2	8	13
16	13	5	4	2	4	8	15
16	13	5	6	2	6	7	16
16	13	5	3	3	3	7	13
16	13	5	5	3	5	6	14
16	13	5	4	4	4	6	13
16	13	5	4	4	3	7	13
16	13	5	4	4	2	8	13
16	13	5	5	5	3	6	12
16	13	5	6	5	2	7	12
16	14	5	3	2	3	8	14
16	14	5	4	2	4	7	14
16	14	5	3	3	3	6	12
16	14	5	4	4	2	7	12
16	14	5	7	6	7	5	15
16	14	5	7	7	2	6	11
16	15	6	2	2	2	10	16
16	15	6	3	2	3	7	14
16	15	6	4	2	4	6	14
16	15	6	3	3	3	6	13
16	15	6	3	3	2	7	13
16	15	6	4	4	4	5	13
16	15	6	4	4	2	6	12

Table 4.7: Table Lookup truncations for redundant partial remainder, radix 16.

$\{D\}_\delta$	1/2
$\tilde{L}_0, \tilde{U}_{-1}$	-1/2,-1/2
s_0	-1/2
$\tilde{L}_1, \tilde{U}_0$	0,0
s_1	0

Table 4.8: Selection intervals and selection bounds for radix 2, with carry-save partial remainder.

$\{D\}_\delta \cdot 16$	8	9	10	11	12	13	14	15
$\left(\tilde{L}_{-1}, \tilde{U}_{-2}\right) \cdot 16$	-13,-13	-15,-15	-16,-16	-18,-17	-20,-19	-21,-20	-23,-21	-25,-23
s_{-1}	-13/16	-15/16	-1	-9/8	-5/4	-5/4	-11/8	-6/4
$\left(\tilde{L}_0, \tilde{U}_{-1}\right) \cdot 16$	-5,-4	-6,-5	-6,-5	-7,-5	-8,-6	-8,-6	-9,-6	-10,-7
s_0	-1/4	-3/8	-3/8	-3/8	-1/2	-1/2	-1/2	-1/2
$\left(\tilde{L}_1, \tilde{U}_0\right) \cdot 16$	3,4	4,5	4,5	4,6	5,7	5,7	5,8	6,9
s_1	1/4	1/4	1/4	1/4	3/8	3/8	1/2	1/2
$\left(\tilde{L}_2, \tilde{U}_1\right) \cdot 16$	12,12	14,14	15,15	16,17	18,19	19,20	20,22	22,24
s_2	3/4	7/8	15/16	1	9/8	19/16	5/4	6/4

Table 4.9: Selection intervals and selection bounds for radix 4, minimally redundant with carry-save remainder.

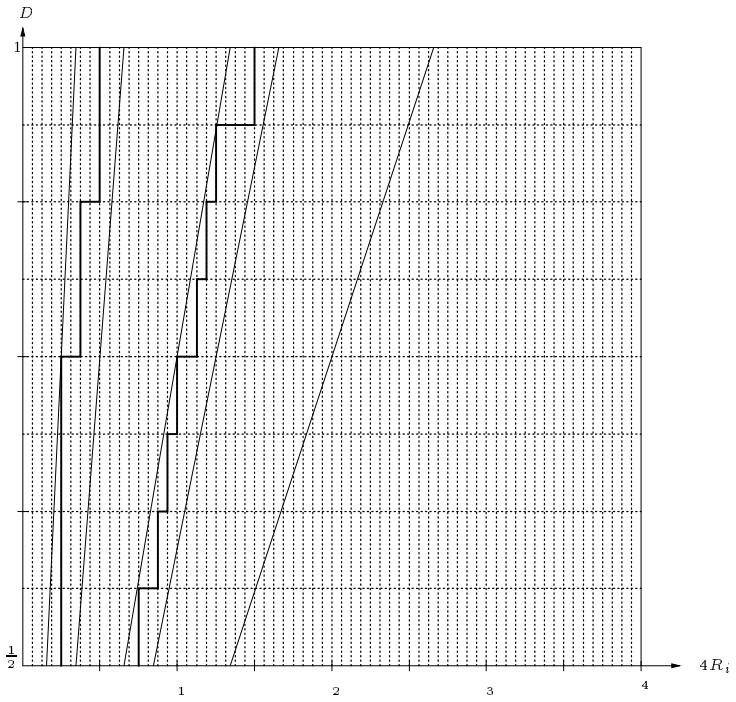

Figure 4.17: PD-plot for $\beta = 4$, $a = 2$.

Chapter 5

Square Root

Like division, the square root operation is in a way the inverse to multiplication. We therefore expect to find some similarities between the two. Unlike with division, we will only consider floating-point square root, since integer square root is of little use. The implementation of square root has usually received less resources than multiplication and division. Implementation of the IEEE floating point standard [9] requires a square root operation to be present.

Definition 5.1 (The square root problem.) *Given X, the radicand, and a required partial root precision* ulp S*, find S and R, the root and remainder, such that:*

$$X = S^2 + R,$$

with

$$\left|\sqrt{X} - S\right| < \text{ulp}\, S.$$

Just as for division, square root algorithms belong either to the digit serial class or the functional iteration class. As before we only discuss the former.

5.1 Digit Serial Square Root

Let β be the radix, and let the root digits s_i belong to some root digit set $\mathcal{D} = \{a, \ldots, 0, \ldots, b\}$. Digit serial square root produces the partial root digits one at a time, most significant first by iterating a recurrence n times, where n is the number of iterations needed to produce the root with the required precision. Let S_i be the *partial root* after i iterations. The *error at step i* is then defined as:

$$\epsilon_i = \left|\sqrt{X} - S_i\right|. \tag{5.1}$$

To ensure that Definition 5.1 is satisfied, we keep the error bounded in each iteration,

$$\epsilon_i = \left|\sqrt{X} - S_i\right| < \text{ulp}\, S_i. \tag{5.2}$$

And we get $\epsilon_n < \text{ulp}\, S_n$.

Using the notation from Chapter 2 we can write the partial root on the form:

$$S_i = \sum_{j=-m}^{-m+i} s_j \beta^{-j}, \tag{5.3}$$

with $s_i \in \mathcal{D}$. After n iterations we have the final partial root:

$$S_n = \sum_{j=-m}^{-m+n} s_j \beta^{-j}. \tag{5.4}$$

Since we are producing the root digits most significant first we have:

$$\operatorname{ulp} S_{i+1} = \beta^{-1} \operatorname{ulp} S_i, \tag{5.5}$$

with initial root S_0, and $\operatorname{ulp} S_0 = \beta^m$. We now develop a recurrence based on the above. From (5.2) and (5.5) we get:

$$\left| \sqrt{X} - S_i \right| < \beta^{-i} \operatorname{ulp} S_0 = \beta^{-i+m}. \tag{5.6}$$

Assume $S_i > 0$, which gives us:

$$\begin{aligned}
-\beta^{-i+m} &< \sqrt{X} - S_i &&< \beta^{-i+m} \\
S_i - \beta^{-i+m} &< \sqrt{X} &&< S_i + \beta^{-i+m} \\
S_i^2 + \beta^{-2i+2m} - 2S_i\beta^{-i+m} &< X &&< S_i^2 + \beta^{-2i+2m} + 2S_i\beta^{-i+m} \\
\beta^{-2i+2m} - 2S_i\beta^{-i+m} &< X - S_i^2 &&< \beta^{-2i+2m} + 2S_i\beta^{-i+m} \qquad (5.7) \\
\beta^{-i+2m} - 2S_i\beta^{m} &< \beta^i(X - S_i^2) &&< \beta^{-i+2m} + 2S_i\beta^{m}. \qquad (5.8)
\end{aligned}$$

If $S_i = 0$ we have:

$$\begin{aligned}
0 \le \sqrt{X} &< \beta^{-i+m} \\
0 \le X &< \beta^{-2i+2m}. \qquad (5.9)
\end{aligned}$$

Unscaled Partial Remainder

Define a *partial remainder*:

$$P_i = X - S_i^2. \tag{5.10}$$

We then obtain the following recurrence:

$$\begin{aligned}
P_{i+1} &= X - S_{i+1}^2 \\
&= X - \left(S_i + s_{i+1}\beta^{-(i+1)+m} \right)^2 \\
&= X - S_i^2 - s_{i+1}^2\beta^{-2(i+1)+2m} - 2S_i s_{i+1}\beta^{-(i+1)+m} \\
&= P_i - s_{i+1}^2\beta^{-2(i+1)+2m} - 2S_i s_{i+1}\beta^{-(i+1)+m} \qquad (5.11)
\end{aligned}$$

with *initial remainder*,

$$R_0 = X - S_0^2. \tag{5.12}$$

The bound in (5.7) gives us the following bound on P_i if $S_i > 0$:

$$\forall i, S_i > 0: \quad \beta^{-2i+2m} - 2S_i\beta^{-i+m} < P_i < \beta^{-2i+2m} + 2S_i\beta^{-i+m}. \tag{5.13}$$

If $S_i = 0$ then:

$$\forall i, S_i = 0: \quad 0 \le P_i < \beta^{-2i+2m}. \tag{5.14}$$

The pair P_n, S_n satisfies Definition 5.1 because of (5.2) and (5.10). So we let $R = P_n$ and $S = S_n$.

Scaled Partial Remainder

An alternate choice is to define a scaled partial remainder:

$$R_i = \beta^i(X - S_i^2). \tag{5.15}$$

We then obtain the following recurrence:

$$\begin{aligned} R_{i+1} &= \beta^{i+1}\left(X - S_{i+1}^2\right) \\ &= \beta^{i+1}\left(X - \left(S_i + s_{i+1}\beta^{-(i+1)+m}\right)^2\right) \\ &= \beta^{i+1}\left(X - S_i^2 - s_{i+1}^2\beta^{-2(i+1)+2m} - 2S_is_{i+1}\beta^{-(i+1)+m}\right) \\ &= \beta R_i - 2S_is_{i+1}\beta^m - s_{i+1}^2\beta^{-(i+1)+2m}. \end{aligned} \tag{5.16}$$

From (5.15) we get the *initial remainder*:

$$R_0 = X - S_0^2. \tag{5.17}$$

We then require, for $S_i > 0$

$$\forall i, S_i > 0: \quad -2S_i + \beta^{-i+m} < R_i < 2S_i + \beta^{-i+m}. \tag{5.18}$$

If $S_i = 0$ then:

$$\forall i, S_i = 0: \quad 0 \le R_i < 2S_i + \beta^{-i+m}. \tag{5.19}$$

Final Remainder Formation

The partial root S_n satisfies Definition 5.1 so we let $S = S_n$. However, the partial remainder R_n does not satisfy Definition 5.1, so we have to form the final remainder R from R_n. We have:

$$R_n = \beta^n(X - S_n^2), \tag{5.20}$$

and we want

$$R = X - S^2, \tag{5.21}$$

so we let

$$R = \beta^{-n}R_n. \tag{5.22}$$

We have to select the root-digit s_{i+1} in the recurrence (5.30), so that R_{i+1} is also bounded. This is handled by the root-digit selection function,

$$s_{i+1} = s\_select(\beta R_i, S_i). \tag{5.23}$$

5.1.1 Digit Serial Square Root Algorithm

The developments made in the previous section leads us to the following algorithm, using the scaled partial remainder:

Algorithm 5.2 *(Digit serial square root algorithm)*

Input: A radix β, a radicand X, a required root precision β^{-n+m}, and an initial partial root S_0.

Output: A partial root and remainder pair S, R with $X = S^2 + R$, and $|\sqrt{X} - S| < \beta^{-n+m}$.

Method:

$R_0 := X - S_0^2$;
for $i := 0$ **to** $n - 1$ **do**
    $s_{i+1} := s\_select(\beta R_i, S_i)$;
    $R_{i+1} := \beta R_i - 2S_i s_{i+1}\beta^m - s_{i+1}^2\beta^{-(i+1)+m}$;
end;
$S := \sum_{i=0}^{n} s_i\beta^{-i}$;
$R := \beta^{-n}R_n$;

The correctness of the algorithm follows immediately from the developments made in the previous section. The object of an implementation is to make every step in the **for** loop as fast as possible.

5.1.2 Block Diagram of the Digit Serial Square Root Algorithm

Figure 5.1 shows a block diagram of one square root iteration.

5.1.3 Floating-Point Square Root

Let X be an IEEE floating-point number:

$$X = (-1)^s 2^E X_1. \tag{5.24}$$

If $s = 1$ then the result is NaN, so assume that $s = 0$. We can write X on the form:

$$X = 2^{E'} X_2, \tag{5.25}$$

where E' is even, and X_2 is normalized to the range:

$$X_2 \in [1/4, 1[. \tag{5.26}$$

With result

$$\sqrt{X} = 2^{E'/2}\sqrt{X_2}, \tag{5.27}$$

and the range:

$$\sqrt{X_2} \in [1/2, 1[. \tag{5.28}$$

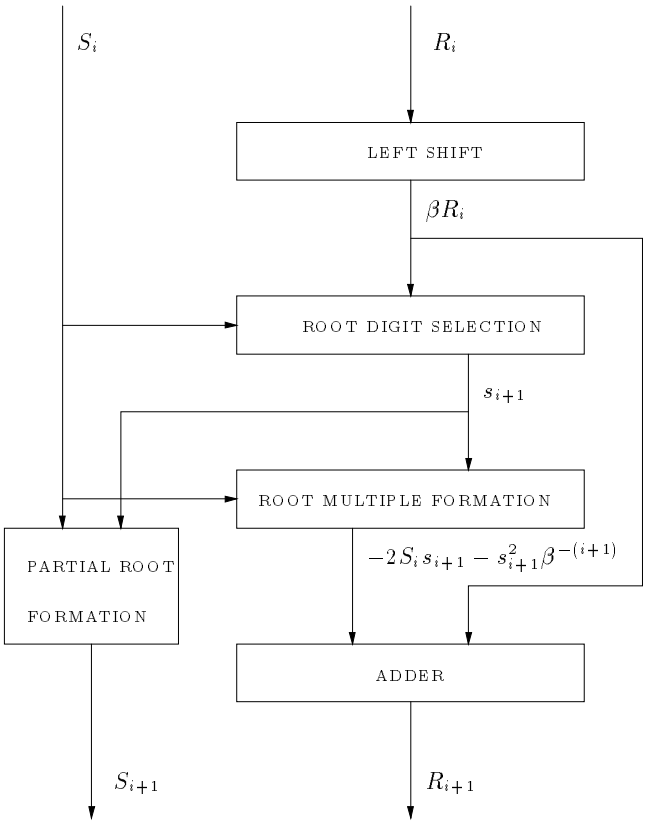

Figure 5.1: Block diagram of square root iteration.

Because the exponent is handled trivially we shall focus on calculating the square root of the significand X_2. We have $m = 0$ and $l = -n$ so the root is on the form:

$$S_n = \sum_{i=0}^{n} s_i \beta^{-i}. \tag{5.29}$$

We will use the scaled partial remainder, with recurrence:

$$R_{i+1} = \beta R_i - 2S_i s_{i+1} - s_{i+1}^2 \beta^{-(i+1)} \tag{5.30}$$

and bounds:

$$\forall i, S_i > 0: \qquad \beta^{-i} - 2S_i < R_i < \beta^{-i} + 2S_i \tag{5.31}$$

$$\forall i, S_i = 0: \qquad 0 \leq R_i < \beta^{-i}. \tag{5.32}$$

Since the result is scaled, we cannot have $S_i = 0$ in every iteration. We now find a requirement on i,

$$R_i = \beta^i X. \tag{5.33}$$

X is normalized, so (5.32) implies:

$$\frac{1}{4} \leq X \leq \beta^{-2i}. \tag{5.34}$$

Since $\beta \geq 2$ we have $i = 0$. The only iteration in which we can use $S_i = 0$ is therefore $S_0 = 0$.

Chapter 6

SRT Square Root

SRT square root has received little attention over the years, compared with division. An analysis of the table lookup method is made in [5].

SRT square has the same characteristics as SRT division namely:

- The partial remainders are normalized.
- The radicand is normalized.
- A redundant digit set is used.
- Partial remainders may be in redundant representation.

In this chapter we shall develop a general theory for SRT square root in close analogy with the developments made in Chapter 4.

6.1 Theory of SRT Square Root

Let β be the radix, and let $X \in [1/4, 1[$ be the normalized radicand. We will again use a redundant symmetric digit set $\mathcal{D}_a$, with $a \geq \beta/2$. As in division the purpose of the root-digit selection function, is to select a root-digit s_{i+1}, while keeping the partial remainder R_{i+1} bounded. Let the bounds on the partial remainders be $\underline{B}_i$ and $\overline{B}_i$

$$\forall i: \quad \underline{B}_i \leq R_i \leq \overline{B}_i. \tag{6.1}$$

Let $[L_s(i), U_s(i)]$ be the selection interval of βR_i for which we can select $s_{i+1} = s$ and keep R_{i+1} bounded.

$$\beta R_i \in [L_s(i), U_s(i)] \Rightarrow \underline{B}_{i+1} \leq R_{i+1} = \beta R_i - 2S_i s - s^2 \beta^{-(i+1)} \leq \overline{B}_{i+1}. \tag{6.2}$$

Now look at $s_{i+1} = a$, the maximal root digit value. Here we have:

$$\beta R_i = U_a(i) \Rightarrow \beta R_i - 2S_i a - a^2 \beta^{-(i+1)} = \overline{B}_{i+1}, \tag{6.3}$$

and

$$\beta \overline{B}_i = U_a(i). \tag{6.4}$$

Combining these we get:

$$\beta\overline{B}_i = \overline{B}_{i+1} + 2S_i a + a^2\beta^{-(i+1)}. \tag{6.5}$$

The solution to this equation is $\overline{B}_i = 2\rho S_i + \rho^2\beta^{-i}$ which can be verified by inserting it in (6.5), and keeping in mind that we are looking at $s_{i+1} = a$. Unlike in division the bound depends on the iteration index i, which complicates matters. The expression for $\underline{B}_i$ can be found in an analogous way, giving us:

$$\underline{B}_i = -2\rho S_i + \rho^2\beta^{-i}, \quad \overline{B}_i = 2\rho S_i + \rho^2\beta^{-i}. \tag{6.6}$$

The partial remainders thus have to satisfy the following bounds:

$$\forall i : -2\rho S_i + \rho^2\beta^{-i} \le R_i \le 2\rho S_i + \rho^2\beta^{-i}. \tag{6.7}$$

The initial remainder R_0 must also satisfy this bound. If $S_0 = 1$ we demand:

$$\begin{aligned} -2\rho S_0 + \rho^2 \le \quad & X - S_0^2 \quad & \le 2\rho S_0 + \rho^2 \\ |S_0 - \rho| \le \quad & \sqrt{X} \quad & \le S_0 + \rho \\ 1 - \rho \le \quad & \sqrt{X} \quad & \le 1 + \rho. \end{aligned}$$

implying:

$$\rho \ge 1 - \sqrt{X}, \qquad \rho \ge \sqrt{X} - 1 \tag{6.8}$$

$$\rho \ge 1 - \min\sqrt{X}, \qquad \rho \ge \max\sqrt{X} - 1 \tag{6.9}$$

$$\rho \ge \frac{1}{2}, \qquad \rho \ge 0 - \mathrm{ulp}\,\sqrt{X}, \tag{6.10}$$

which is satisfied for any redundant digit set. In the case $S_0 = 0$ we require:

$$\begin{aligned} 0 \le \quad & X \quad & \le \rho^2 \\ 0 \le \quad & \sqrt{X} \quad & \le \rho. \end{aligned}$$

So we have the following requirement on ρ:

$$\rho \ge \max\sqrt{X} \tag{6.11}$$

$$\rho \ge 1 - \mathrm{ulp}\,\sqrt{X}. \tag{6.12}$$

In this case we need a maximally redundant digit set. Expressions (6.2) and (6.6) gives us the following expressions for $L_s(i)$ and $U_s(i)$,

$$L_s(i) = \underline{B}_{i+1} + s^2\beta^{-(i+1)} + 2S_i s, \quad U_s(i) = \overline{B}_{i+1} + s^2\beta^{-(i+1)} + 2S_i s \tag{6.13}$$

$$L_s(i) = 2S_i(s-\rho) + (s-\rho)^2\beta^{-(i+1)}, \quad U_s(i) = 2S_i(s+\rho) + (s+\rho)^2\beta^{-(i+1)}. \tag{6.14}$$

The degree of redundancy gives an *overlap* between adjacent digit selection intervals

$$U_{s-1}(i) - L_s(i) = (2\rho - 1)\left(2S_i + (2s-1)\beta^{-(i+1)}\right). \tag{6.15}$$

Generally we have to make sure that every value of βR_i belongs to a selection interval:

$$U_{s-1}(i) - L_s(i) \ge -\mathrm{ulp}\,(\beta R_i). \tag{6.16}$$

So we require:

$$(2\rho-1)\left(2S_i+(2s-1)\beta^{-(i+1)}\right) \geq -\text{ulp}\,(\beta R_i) \tag{6.17}$$

$$S_i \geq \frac{-\text{ulp}\,(\beta R_i)}{4\rho-2}+\max\left(\frac{(1-2s)\beta^{-(i+1)}}{2}\right) \tag{6.18}$$

$$S_i \geq \frac{-\text{ulp}\,(\beta R_i)}{4\rho-2}+\left(\frac{(2a-1)\beta^{-(i+1)}}{2}\right). \tag{6.19}$$

In particular this gives us the following requirement on S_0,

$$S_0 \geq \frac{-\text{ulp}\,(\beta R_i)}{4\rho-2}+\frac{2a-1}{2\beta}. \tag{6.20}$$

Depending on the granularity of the numbers we have to represent we may use $S_0=0$. For $S_0=1$, $\text{ulp}\,(\beta R_i)=\beta^{-n}$ with n small, equation (6.20) is satisfied. Likewise for $S_i=1/2$ and small n equation (6.19) is satisfied. This means that it is possible to have a positive overlap in every iteration. As we shall see later, it is convenient to have the same overlap in every iteration

$$\min_i U_{s-1}(i)-\max_i L_s(i) \geq -\text{ulp}\,(\beta R_i), \tag{6.21}$$

where

$$\min_i U_{s-1}(i) = 2S_i(s-1+\rho)+(s-1+\rho)^2\beta^{-(n+1)} \tag{6.22}$$

$$\max_i L_s(i) = 2S_i(s-\rho)+(s-\rho)^2\beta^{-1}. \tag{6.23}$$

The second term in $\min_i U_{s-1}(i)$ is very small for large n, and always positive, we shall therefore discard that term, and use the resulting lower minimum value. This gives us the following requirement:

$$\begin{aligned} 2S_i(s-1+\rho)+(s-1+\rho)^2-2S_i(s-\rho)-(s-\rho)^2\beta^{-1} &\geq -\text{ulp}\,(\beta R_i) \\ 2S_i(2\rho-1)-(s-\rho)^2\beta^{-1} &\geq -\text{ulp}\,(\beta R_i) \\ \beta 2S_i(2\rho-1) &\geq (s-\rho)^2-\text{ulp}\,(\beta R_i) \end{aligned} \tag{6.24}$$

The right side of (6.24) has its maximum at $s=-a+1$, and the left side has its minimum at either $S_0=0$ or $S_i=1/2$, so for $S_i=1/2$:

$$\beta(2\rho-1) \geq -\text{ulp}\,(\beta R_i)+(-a+1-\rho)^2, \tag{6.25}$$

and for $S_0=0$ we have:

$$0 \geq -\text{ulp}\,(\beta R_i)+(-a+1-\rho)^2. \tag{6.26}$$

As we shall see later there are situations where there are no solutions to these equations. In these cases we have to be content with using the same overlap from iteration I and onwards, for some $I \geq 1$. Similar calculations to the above give us the following lower bound on I:

$$\begin{aligned} \beta^{I+1}(2\rho-1) &\geq (-a+1-\rho)^2-\text{ulp}\,(\beta R_i) \\ I &\geq \left\lceil \log_\beta\left(\frac{(-a+1-\rho)^2-\text{ulp}\,(\beta R_i)}{2\rho-1}\right)\right\rceil-1. \end{aligned} \tag{6.27}$$

Later we shall see how this influences particular implementations of the square root algorithm.

6.2 The Root Digit Selection Function

We represent the function $s\_select(R_i, S_i)$ by a set of selection bounds $\{p_s(i)\}, s \in \mathcal{D}_a$:

$$p_s(i) \leq \beta R_i < p_{s+1}(i) \Rightarrow s_{i+1} = s. \tag{6.28}$$

From (5.30) we can see that the $p_s(i)$ are functions of S_i, and i, since the selection of the next root digit depends on R_i, S_i and i. For notational convenience we shall write $p_s(i)$. We now find a requirement for the choice of $p_s(i)$. From (6.2) and (6.28) we see that:

$$p_s(i) \in [L_s(i), U_s(i)]. \tag{6.29}$$

For $\beta R_i = p_s(i) - \text{ulp}\,(\beta R_i)$ we have to choose $s_{i+1} = s - 1$, because of (6.28). Combined with (6.16) we get:

$$p_s(i) \in [L_s(i), U_{s-1}(i) + \text{ulp}\,(\beta R_i)]. \tag{6.30}$$

We shall use the stricter bound:

$$p_s(i) \in [L_s(i), U_{s-1}(i)]. \tag{6.31}$$

Note that since we always have a positive overlap, as demonstrated in the previous section, such a set of functions will always exist. If we want the same set of selection bounds, $p_s(I)$ from iteration I and onwards we must require:

$$p_s(I) \in \left[\max_{i \geq I} L_s(i), \min_{i \geq I} U_{s-1}(i)\right]. \tag{6.32}$$

Having established the theory, we analyse the two main examples of it, namely the table lookup method, and direct comparison. Given the number of similarities between square root and division we shall not analyse the square root case to the extent that we did in the division case.

6.3 Direct Comparison

6.3.1 Truncated Comparison

Let $X \in [1/4, 1[$ be the normalized radicand, and let $\{\beta R_i\}_\Delta$ denote truncation of βR_i to Δ bits, and let $\{\beta R_i\}_\Delta$ be in 2's complement form. Let l be the number of fraction bits used in the representation of βR_i. Let ϵ_R be the error resulting from the truncation:

$$\beta R_i = \{\beta R_i\}_\Delta + \epsilon_R. \tag{6.33}$$

The range of ϵ_R depends on the representation used, and the truncation. Let τ and δ be the number of integer and fraction bits respectively in $\{\beta R_i\}_\Delta$. Let $\{p_s(i)\}_\Delta$ denote truncation of $p_s(i)$ to Δ bits, τ integer bits and δ fraction bits, and let ϵ_p be the corresponding truncation error:

$$p_s(i) = \{p_s(i)\}_\Delta + \epsilon_p. \tag{6.34}$$

We now implement the function $s\_select(\beta R_i, S_i)$ as:

$$\{p_s(i)\}_\Delta \le \{\beta R_i\}_\Delta < \{p_{s+1}(i)\}_\Delta \Rightarrow s_{i+1} = s. \tag{6.35}$$

6.3.2 Truncation Errors

Comparison Functions

We shall usually require that $p_s(i)$ is in 2's complement encoding. Assume that $p_s(i)$ is a multiplum of S_i, and that S_i is in 2's complement encoding, obtainable by an on-the-fly conversion. The range of the truncation error is then:

$$\epsilon_p \in \left[0, \max\{2^{-\delta} - \text{ulp}\, S_i\}\right]. \tag{6.36}$$

Partial Remainder

The truncation errors are identical to the ones given in Section 4.3.2.

6.3.3 Fraction Bits

Let us find an expression for δ, the number of fraction bits used in the comparison.
The lefthand side of (6.35) gives:

$$\begin{aligned} \{\beta R_i\}_\Delta &\ge \{p_s(i)\}_\Delta \\ \beta R_i &\ge p_s(i) - \epsilon_p + \epsilon_R. \end{aligned}$$

For selection of the digit q we have $\beta R_i \ge L_s(i)$ so we demand:

$$\begin{aligned} p_s(i) - \epsilon_p + \epsilon_R &\ge L_s(i) \\ p_s(i) &\ge L_s(i) + \epsilon_p - \epsilon_R \\ p_s(i) &\ge L_s(i) + \max(\epsilon_p - \epsilon_R) \\ p_s(i) &\ge L_s(i) + \max \epsilon_p - \min \epsilon_R. \end{aligned}$$

Now for the righthand side of (6.35):

$$\begin{aligned}
\{\beta R_i\}_\Delta &< \{p_{s+1}(i)\}_\Delta \\
\{\beta R_i\}_\Delta &\leq \{p_{s+1}(i)\}_\Delta - \mathrm{ulp}\left(\{\beta R_i\}_\Delta\right) \\
\{\beta R_i\}_\Delta &\leq \{p_{s+1}(i)\}_\Delta - 2^{-\delta} \\
\beta R_i &\leq p_{s+1}(i) - 2^{-\delta} - \epsilon_p + \epsilon_R.
\end{aligned}$$

We have $\beta R_i \leq U_s(i)$ so:

$$\begin{aligned}
p_{s+1}(i) - 2^{-\delta} - \epsilon_p + \epsilon_R &\leq U_s(i) \\
p_{s+1}(i) &\leq U_s(i) + 2^{-\delta} + \epsilon_p - \epsilon_R \\
p_{s+1}(i) &\leq U_s(i) + 2^{-\delta} + \min(\epsilon_p - \epsilon_R) \\
p_{s+1}(i) &\leq U_s(i) + 2^{-\delta} + \min \epsilon_p - \max \epsilon_R \\
p_s(i) &\leq U_{s-1}(i) + 2^{-\delta} + \min \epsilon_p - \max \epsilon_R,
\end{aligned}$$

where the final inequality follows from the fact that the right hand side in equation (6.35) is also valid for $s_{i+1} = s - 1$. The chosen comparison functions should thus satisfy these two inequalities:

$$\begin{aligned}
p_s(i) &\geq L_s(i) + \max \epsilon_p - \min \epsilon_R && (6.37) \\
p_s(i) &\leq U_{s-1}(i) + 2^{-\delta} + \min \epsilon_p - \max \epsilon_R. && (6.38)
\end{aligned}$$

For simplicity we shall use the tighter bounds:

$$\begin{aligned}
p_s(i) &\geq L_s(i) + \sup \epsilon_p - \inf \epsilon_R && (6.39) \\
p_s(i) &\leq U_{s-1}(i) + 2^{-\delta} + \inf \epsilon_p - \sup \epsilon_R. && (6.40)
\end{aligned}$$

The number of fraction bits, δ, required in the comparison depends on the chosen comparison functions $p_s(i)$, and the chosen representation for the partial remainders and the partial root S_i. For existence of such an $p_s(i)$ we require:

$$\begin{aligned}
U_{s-1}(i) + 2^{-\delta} + \inf \epsilon_p - \sup \epsilon_R &\geq L_s(i) + \sup \epsilon_p - \inf \epsilon_R \\
U_{s-1}(i) - L_s(i) &\geq \sup \epsilon_p - \inf \epsilon_R - \inf \epsilon_p + \sup \epsilon_R - 2^{-\delta}
\end{aligned}$$

We saw earlier that the lefthand side can be made positive for all i. The righthand side depends only on δ, so we can always satisfy the last inequality by making δ large enough.

6.3.4 Integer bits

Let τ be the number of integer bits required to represent the shifted partial remainder and the comparison functions. We now calculate the number of integer bits τ, needed to represent the truncated shifted partial remainder. The upper bound on the value of $\{\beta R_i\}_\Delta$ is:

$$\begin{aligned}
\{\beta R_i\}_\Delta &\leq \max\left(\beta R_i - \epsilon_R\right) \\
\{\beta R_i\}_\Delta &\leq \left\lfloor \max\left(\beta R_i - \epsilon_R\right) 2^\delta \right\rfloor 2^{-\delta} \\
\{\beta R_i\}_\Delta &\leq \left\lfloor \left(\beta\left(2\rho + \rho^2\right) - \inf \epsilon_R\right) 2^\delta \right\rfloor 2^{-\delta}. && (6.41)
\end{aligned}$$

Lower bound:

$$\{\beta R_i\}_\Delta \geq \left\lceil \left(-2\beta\rho - \sup \epsilon_R\right) 2^\delta \right\rceil 2^{-\delta}. \tag{6.42}$$

Upper bound:

$$2^{\tau-1} \geq \left\lfloor \left(\beta\left(2\rho+\rho^2\right) - \inf \epsilon_R\right) \cdot 2^\delta \right\rfloor \cdot 2^{-\delta} \tag{6.43}$$

$$\tau \geq \left\lceil \log_2 \left(\left\lfloor (\beta(2\rho+\rho^2) - \inf \epsilon_R) \cdot 2^\delta \right\rfloor \cdot 2^{-\delta} \right) \right\rceil + 1. \tag{6.44}$$

Lower bound:

$$2^{\tau-1} \geq -\left\lceil (-2\beta\rho - \sup \epsilon_R) \cdot 2^\delta \right\rceil \cdot 2^{-\delta} \tag{6.45}$$

$$\tau \geq \left\lceil \log_2 \left(-\left\lceil (-2\beta\rho - \sup \epsilon_R) \cdot 2^\delta \right\rceil \cdot 2^{-\delta} \right) \right\rceil + 1. \tag{6.46}$$

Thus, given the number of fractional bits δ, we can determine the number of integer bits required. The total number of bits, Δ, required for the comparison, is then $\Delta = \tau + \delta$, including the sign bit.

6.3.5 Same Comparison Functions in Every Iteration

We would like to use the same set of comparison functions, in every iteration. I.e a set of comparison functions p_s that are independent of i. As noted in Section 6.1 such a set of comparison functions may not exist. In this case we have to find the first I digits of the partial root, using another method, and then use a set of comparison functions $p_s(I)$, whose formation is independent of i, from iteration I and onwards. The function $s\_select(\beta R_i, S_i)$ from iteration I is then given by:

$$\forall i \geq I : \{p_s(I)\}_\Delta \leq \{\beta R_i\}_\Delta < \{p_{s+1}(I)\}_\Delta \Rightarrow s_{i+1} = s. \tag{6.47}$$

Similar calculations to the ones in Section 6.3.3 gives us the following two expressions:

$$\forall i \geq I : p_s(I) \geq L_s(i) + \sup \epsilon_p - \inf \epsilon_R \tag{6.48}$$

$$\forall i \geq I : p_s(I) \leq U_{s-1}(i) + 2^{-\delta} + \inf \epsilon_p - \sup \epsilon_R. \tag{6.49}$$

These are equivalent to:

$$p_s(I) \geq \max_{i \geq I} \left(L_s(i) + \sup \epsilon_p - \inf \epsilon_R\right)$$

$$p_s(I) \leq \min_{i \geq I} \left(U_{s-1}(i) + 2^{-\delta} + \inf \epsilon_p - \sup \epsilon_R\right).$$

Since ϵ_R is independent of i we have:

$$p_s(I) \geq \max_{i \geq I} \left(L_s(i) + \sup \epsilon_p\right) - \inf \epsilon_R \tag{6.50}$$

$$p_s(I) \leq \min_{i \geq I} \left(U_{s-1}(i) + \inf \epsilon_p\right) - \sup \epsilon_R + 2^{-\delta}. \tag{6.51}$$

A set of $p_s(I)$ satisfying (6.50) and (6.51) can always be found by making I large enough, since as I approaches n, $p_s(I)$ will approach $p_s(n)$, $\max_{i \geq I} L_s(i)$ will approach $L_s(n)$, and

$\min_{i \geq I} U_{s-1}(i)$ will approach $U_{s-1}(n)$. Since the values of δ and I depend on each other, there are several possible solution pairs. Also note that $I = 0$ corresponds to using the same comparison functions in every iteration.

Due to the fact that $p_s(I)$ depends on S_i and the value of S_i could possibly change in every iteration, one could be mislead into believing that we would have to generate the comparison functions $p_s(I)$ in every iteration. This is however not necessarily the case. Using the same comparison functions after I iterations, introduces another error besides the truncation error.

6.3.6 Choice of Comparison Functions

When choosing our comparison functions $p_s(I)$, we should choose ones which are as simple as possible to generate. We therefore define the following digit selection intervals.

$$L_s^* = \max_{i \geq I} L_s(i) - (s-\rho)^2 \beta^{-(I+1)} = 2S_i(s-\rho) \tag{6.52}$$

$$U_{s-1}^* = \min_{i \geq I} U_{s-1}(i) - (s-1+\rho)^2 \beta^{-(n+1)} = 2S_i(s-1+\rho) \tag{6.53}$$

Note that we can not choose $p_s(I) = L_s^*$. Discarding the second term to obtain U_{s-1}^* poses no problem since the resulting bound is lower. Observe that these selection intervals are identical to the ones in the division case, with $2S_i$ taking the place of D. In relation to these selection intervals we can choose the comparison functions in the interval:

$$p_s(I) \in \left[L_s^* + 2^{-\delta} + \sup \epsilon_p - \inf \epsilon_R, U_{s-1}^* + 2^{-\delta} + \inf \epsilon_p - \sup \epsilon_R \right], \tag{6.54}$$

where we require that $(s-\rho)^2 \beta^{-(I+1)} \leq 2^{-\delta}$. That is the term we subtract to obtain L_s^* must be smaller than the granularity of the truncation, which is a matter of making I large enough.
Possible choices are:

$p_s(I) = U_{s-1}^*$: This is only usable for $\rho = 1$, where we then have: $p_s(I) = 2S_i s$. As in division this choice is not viable for carry-save partial remainder.

$p_s(I) = \frac{U_{s-1}^* + L_s^*}{2}$. Since this is independent of ρ we can use this for all digit sets. However we must make sure that (6.54) is satisfied.

$p_s(I) = L_s^* + 2^{-\delta}$: Only full-precision comparison.

For all these choices we have the truncation error given earlier:

$$\epsilon_p \in \left[0, \max\{0, 2^{-\delta} - \text{ulp}\, S_i\} \right]. \tag{6.55}$$

The question of how we compute non-redundant non-trivial multiples of S_i arises. I.e. how do we compute $3S_i$ for example. In this case we store $3D$ in redundant form using an additional bit of precision. The corresponding truncation error is then the same as before.

6.3.7 Performing the Comparison

The truncated partial remainder is formed in the same way as in division. However, since the truncation error on $p_s(I)$ depends on S_i, the result depends on the particular implementation and can not be derived analytically.

6.3.8 Using only Positive Comparison Functions

Since the digit selection intervals $[L_s^*, U_{s-1}^*]$ are symmetric around $\beta R_i = 0$, we can reflect the area of uncertainty in the S_i axis, in the same manner as in the division case.

6.3.9 Using Constant Comparison Functions

In analogy with the division case we get the following requirement for the existence of a constant comparison function, for $s > 0$

$$\left\lfloor \left(\min_{S_i} U_{s-1} - \sup \epsilon_R + 2^{-\delta} \right) 2^{\delta} \right\rfloor \quad \geq \quad \left\lceil \left(\max_{S_i} L_s(i) - \inf \epsilon_R \right) 2^{\delta} \right\rceil \tag{6.56}$$

$$\begin{aligned} \left\lfloor \left(s - 1 + \rho + (s - 1 + \rho)^2 \beta^{-(i+1)} - \sup \epsilon_R + 2^{-\delta} \right) 2^{\delta} \right\rfloor \quad \geq \\ \left\lceil \left(2(s - \rho) + (s - \rho)^2 \beta^{-(i+1)} - \inf \epsilon_R \right) 2^{\delta} \right\rceil . \end{aligned} \tag{6.57}$$

6.3.10 Results and Examples

Since the digit selection intervals L^* and U^* are identical to the ones in division we get identical results with the exception that the first I iterations have to be handled differently, and we need an integer bit more due to $2S_i$ replcing D. The valid choices of I and δ for a few implementations are shown in Table 6.1.

β	a	R_i encoding	s_q	δ	I
2	1	carry-save	$(2s-1)S_i$	1	0
4	2	carry-save	$(2s-1)S_i$	3	2

Table 6.1: Valid truncations and number of initial iterations needed for direct comparison.

Radix 4, Minimally Redundant

From Table 6.1 we see that we can use $s_q = (2s-1)S_i, I = 2$ and $\delta = 3$. The comparison functions $3S_i$ and $-3S_i$ are non-trivial to generate. Therefore we perform this comparison with $3D$ in carry-save form: $3S_i = (2S_i)^c + (S_i)^s$, where the carry and save parts are computed to 4 bits.
Case $i = 0$: With $S_0 = 1$ we and using (6.39) and 6.40 we have the bounds:

$$\begin{aligned} p_s(0) &\geq 2(s - \frac{2}{3}) + (s - \frac{2}{3})^2 4^{-1} \\ p_s(0) &\leq 2(s - \frac{1}{3}) + (s - \frac{1}{3})^2 4^{-1} - \frac{1}{8} \end{aligned}$$

Simple calculations give us the following comparison functions: $p_{-1}(0) = (2s-1) + 1 = -2$, $p_0(0) = (2s-1) = -1$, $p_1(0) = (2s-1) = 1$, and $p_2(0) = (2s-1) + 1/2 = 7/2$.

Case $i = 1$: Analogously we can use $p_{-1}(1) = S_i(2s-1) + 1/8 = -3S_i + 1/8$, and $p_s(1) = (2s-1)$ for $s \in \{0, 1, 2\}$.

In all there are three special cases which have to be handled in the first 2 iterations. From then on we proceed with the same comparison functions.

6.4 Table Lookup

We will use the same notation as we did for division, i.e. $\{\beta R_i\}_c$ will denote the truncated partial remainder. Let ϵ_R be the error corresponding to the truncation of βR_i:

$$\beta R_i = \{\beta R_i\}_c + \epsilon_R. \tag{6.58}$$

Let τ be the number of integer bits needed to represent $\{\beta R_i\}_c$, and let l be the number of fraction bits used in the representation of βR_i. Let $S_i \in [1/2, 1]$ be the partial root. Let $\{S_i\}_\delta$ denote truncation of S_i to δ bits, and let ϵ_S be the error resulting from the truncation. We then have:

$$S_i = \{S_i\}_\delta + \epsilon_S. \tag{6.59}$$

Since S_i is truncated to δ fraction bits, and normalized to the interval $[1/2, 1]$ we can write it as:

$$\{S_i\}_\delta = \frac{1}{2} + m \cdot 2^{-\delta}, \tag{6.60}$$

where $m \in \left[0, 2^{\delta-1}\right]$.

Partition the interval $[1/2, 1]$ into $2^{\delta-1}$ parts of equal size. Let $p_s(i)$ be a function of S_i which is constant on each of the intervals $[\{S_i\}_\delta + \min \epsilon_S, \{S_i\}_\delta + \max \epsilon_S]$, and is representable in c fractional bits. We then have the following implementation of $s\_select(\beta R_i, S_i)$.

$$\{p_s(i)\}_c \leq \{\beta R_i\}_c < \{p_{s+1}(i)\}_c \Rightarrow s_{i+1} = s. \tag{6.61}$$

6.4.1 Truncation Errors

Partial Root

Let f be the number of fraction bits of S_i. Let $2^{-\delta} \geq \operatorname{ulp} S_i$. If we use the truncated signed-digit representation of S_i then the range of the error is given by:

$$\epsilon_S \in \left[-\rho \cdot 2^{-\delta} + \operatorname{ulp} S_i, \rho \cdot 2^{-\delta} - \operatorname{ulp} S_i\right]. \tag{6.62}$$

Note that depending on the value of S_i the range of the error may be restricted further. For example $S_i = 1/2$, gives a non-negative truncation error and so forth. However, if we use an on-the-fly converted partial root then the error is given by:

$$\epsilon_S \in \left[0, 2^{-\delta} - \operatorname{ulp} S_i\right]. \tag{6.63}$$

Since the total error range is smaller in the latter case we shall use it in the following. If $\operatorname{ulp} S_i \geq 2^{-\delta}$ then we have zero truncation error since all bits beyond the δ'th are zero.

Partial Remainder

The truncation errors are identical to the ones given in Section 4.3.2.

6.4.2 Fraction Bits

We now determine what truncations of βR_i, and values of δ result in a valid root digit selection function. Using similar calculations to the ones in Section 4.4 we get the following two expressions:

$$p_s(i) \geq L_s(i) - \inf \epsilon_R \tag{6.64}$$
$$p_s(i) \leq U_{s-1}(i) - \sup \epsilon_R + 2^{-c}. \tag{6.65}$$

These expressions have to be satisfied for the interval $S'_i \in [\{S_i\}_\delta + \min \epsilon_S, \{S_i\}_\delta + \max \epsilon_S]$, so we get:

$$p_s(i) \geq \max_{S'_i} L_s(i) - \inf \epsilon_R \tag{6.66}$$
$$p_s(i) \leq \min_{S'_i} U_{s-1}(i) - \sup \epsilon_R + 2^{-c}. \tag{6.67}$$

And because $p_s(i)$ has c fraction bits:

$$p_s(i) \geq \left\lceil \left(\max_{S'_i} L_s(i) - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c} \tag{6.68}$$
$$p_s(i) \leq \left\lfloor \left(\min_{S'_i} U_{s-1}(i) - \sup \epsilon_R + 2^{-c} \right) 2^c \right\rfloor 2^{-c}. \tag{6.69}$$

The condition for the existence of such a $p_s(i)$:

$$\left\lfloor \left(\min_{S'_i} U_{s-1}(i) - \sup \epsilon_R + 2^{-c} \right) 2^c \right\rfloor 2^{-c} \geq \left\lceil \left(\max_{S'_i} L_s(i) - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c}. \tag{6.70}$$

Equation (6.70) should be evaluated to determine valid truncations of βR_i and S_i. However since the value of c and δ depends on i we could end up with different tables for different i. We therefore use the following alternative implementation.

6.4.3 Using Same Selection Bounds in Every Iteration

In Section 6.1 we saw that we cannot use the same selection bounds in every iteration. We therefore use one set of selection bounds from iteration I and onwards, and one or more different selection bounds for $i < I$. We then have the following implementation from iteration I and onwards:

$$\forall i \geq I : \{p_s(I)\}_c \leq \{\beta R_i\}_c < \{p_{s+1}(I)\}_c \Rightarrow s_{i+1} = s. \tag{6.71}$$

Condition for the existence of such a $p_s(I)$:

$$\left\lfloor \left(\min_{S'_i, i \geq I} U_{s-1}(i) - \sup \epsilon_R + 2^{-c} \right) 2^c \right\rfloor 2^{-c} \geq \left\lceil \left(\max_{S'_i, i \geq I} L_s(i) - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c}. \tag{6.72}$$

The expressions $\min_{S'_i, i \geq I} U_{s-1}(i)$ and $\max_{S'_i, i \geq I} L_s(i)$ are given by, using (6.14), (6.60) and (6.63):

$s > 0$:

$$\begin{aligned}
\min_{S_i', i \geq I} U_{s-1}(i) &= \min_{S_i', i \geq I} \left(2S_i'(s-1+\rho) + (s-1+\rho)^2 \beta^{-(i+1)} \right) \\
&= \min_{i \geq I} \left(2(\{S_i\}_\delta + \min \epsilon_S)(s-1+\rho) + (s-1+\rho)^2 \beta^{-(i+1)} \right) \\
&= \min_{i \geq I} \left(2\{S_i\}_\delta (s-1+\rho) + (s-1+\rho)^2 \beta^{-(i+1)} \right) \\
&= 2\{S_i\}_\delta (s-1+\rho) \\
&= 2\left(\frac{1}{2} + m 2^{-\delta} \right) (s-1+\rho), \qquad (6.73)
\end{aligned}$$

$$\begin{aligned}
\max_{S_i', i \geq I} L_s(i) &= \max_{S_i', i \geq I} \left(2S_i'(s-\rho) + (s-\rho)^2 \beta^{-(i+1)} \right) \\
&= \max_{i \geq I} \left(2(\{S_i\}_\delta + \max \epsilon_S)(s-\rho) + (s-\rho)^2 \beta^{-(i+1)} \right) \\
&= \max_{i \geq I} \left(2(\{S_i\}_\delta + 2^{-\delta} - \beta^{-i})(s-\rho) + (s-\rho)^2 \beta^{-(i+1)} \right) \\
&= 2\left(\{S_i\}_\delta + 2^{-\delta} \right) (s-\rho) + \max_{i \geq I} \left((s-\rho)^2 \beta^{-(i+1)} - 2\beta^{-i}(s-\rho) \right) \\
&= 2\left(\{S_i\}_\delta + 2^{-\delta} \right) (s-\rho) + \max_{i \geq I} \left((s-\rho)\left((s-\rho) - 2\beta\right) \beta^{-(i+1)} \right) \\
&= 2\left(\{S_i\}_\delta + 2^{-\delta} \right) (s-\rho) \\
&= 2\left(\frac{1}{2} + (m+1) 2^{-\delta} \right) (s-\rho), \qquad (6.74)
\end{aligned}$$

where we have used the fact that $(s-\rho) - 2\beta < 0$, implying that the last term is negative and converges towards zero for large i.
$s \leq 0$:

$$\begin{aligned}
\min_{S_i', i \geq I} U_{s-1}(i) &= \min_{S_i', i \geq I} \left(2S_i'(s-1+\rho) + (s-1+\rho)^2 \beta^{-(i+1)} \right) \\
&= \min_{i \geq I} \left(2(\{S_i\}_\delta + \max \epsilon_S)(s-1+\rho) + (s-1+\rho)^2 \beta^{-(i+1)} \right) \\
&= \min_{i \geq I} \left(2(\{S_i\}_\delta + 2^{-\delta} - \beta^{-i})(s-1+\rho) + (s-1+\rho)^2 \beta^{-(i+1)} \right) \\
&= 2\left(\{S_i\}_\delta + 2^{-\delta} \right) (s-1+\rho) + \min_{i \geq I} \left((s-1+\rho)^2 \beta^{-(i+1)} - 2(s-1+\rho)\beta^{-i} \right) \\
&= 2\left(\{S_i\}_\delta + 2^{-\delta} \right) (s-1+\rho) + \min_{i \geq I} \left((1-s-\rho)((1-s-\rho) + 2\beta)\beta^{-(i+1)} \right) \\
&= 2\left(\{S_i\}_\delta + 2^{-\delta} \right) (s-1+\rho) \\
&= 2\left(\frac{1}{2} + (m+1) 2^{-\delta} \right) (s-1+\rho), \qquad (6.75)
\end{aligned}$$

$$\max_{S_i', i \geq I} L_s(i) = \max_{S_i', i \geq I} \left(2S_i'(s-\rho) + (s-\rho)^2 \beta^{-(i+1)} \right)$$

$$
\begin{aligned}
&= \max_{i\geq I}\left(2(\{S_i\}_\delta + \min\epsilon_S)(s-\rho) + (s-\rho)^2\beta^{-(i+1)}\right)\\
&= \max_{i\geq I}\left(2\{S_i\}_\delta(s-\rho) + (s-\rho)^2\beta^{-(i+1)}\right)\\
&= 2\{S_i\}_\delta(s-\rho) + \max_{i\geq I}\left((s-\rho)^2\beta^{-(i+1)}\right)\\
&= 2\{S_i\}_\delta(s-\rho) + (s-\rho)^2\beta^{-(I+1)}\\
&= 2\left(\frac{1}{2} + m2^{-\delta}\right) + (s-\rho)^2\beta^{-(I+1)}. \qquad (6.76)
\end{aligned}
$$

Inserting these in (6.72) we have:
$s > 0$:

$$
\left\lfloor\left(2\left(\frac{1}{2}+m2^{-\delta}\right)(s-1+\rho) - \sup\epsilon_R + 2^{-c}\right)2^c\right\rfloor 2^{-c} \geq
\left\lceil\left(2\left(\frac{1}{2}+(m+1)2^{-\delta}\right)(s-\rho) - \inf\epsilon_R\right)2^c\right\rceil 2^{-c}. \qquad (6.77)
$$

$s \leq 0$:

$$
\left\lfloor\left(2\left(\frac{1}{2}+(m+1)2^{-\delta}\right)(s-1+\rho) - \sup\epsilon_R + 2^{-c}\right)2^c\right\rfloor 2^{-c} \geq
\left\lceil\left(2\left(\frac{1}{2}+m2^{-\delta}\right) + (s-\rho)^2\beta^{-(I+1)} - \inf\epsilon_R\right)2^c\right\rceil 2^{-c}. \qquad (6.78)
$$

Equations (6.77) and (6.78) should be evaluated for every possible value of s and m to determine valid choices of c, δ, I, and truncations of βR_i.

6.4.4 Integer Bits

We now calculate the number of integer bits τ, needed to represent the truncated shifted partial remainder. The upper bound on the value of $\{\beta R_i\}_c$ is:

$$
\begin{aligned}
\{\beta R_i\}_c &\leq \max(\beta R_i - \epsilon_R)\\
\{\beta R_i\}_c &\leq \lfloor\max(\beta R_i - \epsilon_R)\,2^c\rfloor\,2^{-c}\\
\{\beta R_i\}_c &\leq \left\lfloor\left(\beta\left(2\rho+\rho^2\beta^{-i}\right) - \inf\epsilon_R\right)2^c\right\rfloor 2^{-c}. \qquad (6.79)
\end{aligned}
$$

Lower bound:

$$
\{\beta R_i\}_c \geq \lceil(-2\beta\rho - \sup\epsilon_R)\,2^c\rceil\,2^{-c}. \qquad (6.80)
$$

Upper bound:

$$
\begin{aligned}
2^{\tau-1} &\geq \left\lfloor\left(\beta\left(2\rho+\rho^2\right) - \inf\epsilon_R\right)\cdot 2^c\right\rfloor\cdot 2^{-c} \qquad (6.81)\\
\tau &\geq \left\lceil\log_2\left(\left\lfloor(\beta(2\rho+\rho^2) - \inf\epsilon_R)\cdot 2^c\right\rfloor\cdot 2^{-c}\right)\right\rceil + 1. \qquad (6.82)
\end{aligned}
$$

Lower bound:

$$
\begin{aligned}
2^{\tau-1} &\geq -\lceil(-2\beta\rho - \sup\epsilon_R)\cdot 2^c\rceil\cdot 2^{-c} \qquad (6.83)\\
\tau &\geq \left\lceil\log_2\left(-\lceil(-2\beta\rho - \sup\epsilon_R)\cdot 2^c\rceil\cdot 2^{-c}\right)\right\rceil + 1. \qquad (6.84)
\end{aligned}
$$

Given the number of fraction bits in the truncation, we can determine the number of integer bits needed for the representation.

6.4.5 Table Implementations

Like in division there are two methods of feeding the value of the truncated redundant shifted partial remainder to the table. The analysis is identical to division so we only state the results here:

2's complement

$$\epsilon_R \in [0, 2^{-c} - 2^{-l}]. \tag{6.85}$$

Carry-save

$$\epsilon_{R(cs)} \in \left[0, 2^{-p} + 2^{-r} - 2 \cdot 2^{-l}\right] \tag{6.86}$$

$$\epsilon_R \in \left[0, 2^{-c} + 2^{-\min\{p,r\}} - 2 \cdot 2^{-l}\right]. \tag{6.87}$$

borrow-save

$$\epsilon_{R(bs)} \in \left[-2^{-n} + 2^{-l}, 2^{-p} - 2^{-l}\right] \tag{6.88}$$

$$\epsilon_R \in \left[-2^{-n} + 2^{-l}, 2^{-p} + 2^{-c} - 2^{-g} - 2^{-l}\right]. \tag{6.89}$$

6.4.6 Using only Positive Selection Bounds

In order to use the same half of the table for all operands we need symmetric digit selection intervals. In the direct comparison method we found an expression for these. Since this narrows the overlap it could result in larger tables. Alternatively we could use the positive half and fold the other half into it [6].

6.4.7 Results

Table 6.2 shows the required number of inputs w to the lookup table, when using a 2's complement partial remainder. It was constructed by writing a small program which recorded valid truncations.

Table 6.3 shows the total number of bits needed as input to the table for various radices and digit sets, when βR_i is in carry-save representation. Also shown is the number of initial iterations, I, we have to perform before we can use the same selection bounds in every iteration.

6.4.8 Examples

We define $\tilde{L}_s(I)$ and $\tilde{U}_{s-1}(I)$ as:

$$\tilde{L}_s(I) = \left\lceil \left(\max_{S_i', i \geq I} L_s(i) - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c} \tag{6.90}$$

$$\tilde{U}_{s-1}(I) = \left\lfloor \left(\min_{S_i', i \geq I} U_{s-1}(i) - \sup \epsilon_R \right) 2^c \right\rfloor 2^{-c} + 2^{-c}. \tag{6.91}$$

The lookup table, for $i \geq I$, can now be constructed by computing for all s the bounds $\tilde{L}_s(I)$ and $\tilde{U}_{s-1}(I)$, and selecting the $p_s(I) \in [\tilde{L}_s(I), \tilde{U}_{s-1}(I)]$, which needs the fewest number of fractional bits.

β	a	τ	c	δ	I	w
2	1	4	0	1	0	3
4	2	4	2	5	2	9
4	2	4	3	4	2	9
4	3	5	1	3	2	7
4	3	5	2	4	1	9
8	4	5	4	7	2	14
8	4	5	5	6	3	14
8	7	6	1	5	2	10
8	7	6	2	4	2	10
16	8	6	5	9	3	18
16	8	6	6	9	2	19
16	15	7	1	6	2	12
16	15	7	4	9	1	18

Table 6.2: Table Lookup truncations for 2's complement partial remainder.

Radix 2

Let R_i be in carry-save form. From Table 6.3 we see that we can use $\delta = 1$, $c = 1$, $\tau = 4$ and $I = 0$. Even though we can use the same selection bounds in every iteration, the case $i = 0$ is so special that we will consider it separately. In Section 6.1 we saw that we can use $S_0 = 0$. This implies that we have to select $s_1 = 1$ to have $S \geq 1/2$. Now for the other case ($i > 0$). The bound $\tilde{L}_0(1)$ is calculated as follows ($\{S_i\}_\delta = 1/2$):

$$\begin{aligned}
\tilde{L}_0(1) &= \left\lceil \left(\max_{S_i', i \geq 1} L_s(i) \right) 2 \right\rceil 2^{-1} - \inf \epsilon_R \\
&= \left\lceil \left(-2 \max_i \left(\{S_i\}_\delta + \min \epsilon_S + 2^{-(i+1)} \right) \right) 2 \right\rceil 2^{-1} \\
&= \left\lceil \left(-2 \max_i \left(\frac{1}{2} - 2^{-i} + 2^{-(i+1)} \right) \right) 2 \right\rceil 2^{-1} \\
&= \left\lceil \left(-2 \max_i \left(\frac{1}{2} - 2^{-i} + 2^{-(i+1)} \right) \right) 2 \right\rceil 2^{-1} \\
&= -1.
\end{aligned}$$

Table 6.4 shows the resulting lookup table.

Radix 4, Minimally Redundant

Let R_i be in carry-save form. From Table 6.3 we see that we can use $\delta = 4$, $p = n = 4$, $c = 3$, $\tau = 4$, and $I = 3$. This implementation is also described in [5], and is simpler than the one in [6], which uses 5 root bits. First we construct the table for $i \geq 3$, shown in Table 6.5. We then have to construct the selection function for $i < I$. One method is to use a small lookup

β	a	τ	p	r	c	δ	I	w
2	1	4	1	1	1	1	0	4
4	2	4	3	3	3	5	3	10
4	2	4	3	3	3	6	2	11
4	2	4	4	4	3	4	3	9
4	2	4	4	4	4	4	2	10
4	2	4	4	4	2	5	3	9
4	2	4	4	4	4	6	1	12
4	3	5	2	1	2	4	2	9
4	3	5	2	2	2	3	2	8
4	3	5	3	2	3	5	1	11
4	3	5	3	3	3	4	1	10
8	4	5	4	4	4	8	4	15
8	4	5	4	4	4	9	3	16
8	4	5	5	4	5	7	4	15
8	4	5	5	4	5	8	2	16
8	4	5	6	4	6	7	3	16
8	4	5	5	5	5	7	2	15
8	4	5	6	6	6	6	3	15
8	7	6	2	1	2	6	2	12
8	7	6	2	2	2	5	2	11
8	7	6	3	3	3	4	2	11
8	7	6	3	3	1	5	2	10
8	7	6	4	3	4	8	1	16
8	7	6	4	4	4	7	1	15
8	7	6	5	5	5	6	1	15

Table 6.3: Table lookup truncations for redundant partial remainder, $w = \tau + p + \delta - 1$.

$\{S_0\}_\delta$	1/2	1
$\tilde{L}_0(1), \tilde{U}_{-1}(1)$	-1,-1/2	-2,-1/2
$p_0(1)$	-1	-1
$\tilde{L}_1(1), \tilde{U}_0(1)$	0,1	0,2
$p_1(1)$	0	0

Table 6.4: Radix 2 table.

table to give us S_I, and use it for the first I iterations, to calculate R_I. This approach is used in [6, 7]. We shall however use the following approach presented in [5]. Since $\delta = 4$ and the granularity of S_2 is 2^{-4}, we have no truncation error on S_0, S_1 and S_2. That is $S_i = \{S_i\}_\delta$, for $i < 3$. Therefore we may be able to construct tables for $i < 3$, and match them with the table for $i \geq 3$. For all $i < 3$ we then calculate the bounds:

$$\tilde{L}_s(i) = \left\lceil \left(\max_{S'_i} L_s(i) - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c} \tag{6.92}$$

$$\tilde{U}_{s-1}(i) = \left\lfloor \left(\min_{S'_i} U_{s-1}(i) - \sup \epsilon_R \right) 2^c \right\rfloor 2^{-c} + 2^{-c} \tag{6.93}$$

$i = 0$: Since $S_0 = 1$ we can only select $s_1 \in \{-2, -1, 0\}$. We then get Table 6.6.
$i = 1$: Since S_1 can only assume the values $1/2, 3/4$ and 1, we have $S_1 = \{S_1\}_\delta$. We then get Table 6.7. If $\{S_i\}_\delta = 1/2$ then we cannot select $s_{i+1} = -1$, because that would make $S < 1/2$.
$i = 2$: Since the granularity of S_2 and $\{S_2\}_\delta$ is $1/16$, we have $S_2 = \{S_1\}_\delta$. We then get Table 6.8.

Table 6.9 show the combined table for all values of i.

It is evident that the only case in which we cannot use the same selection bounds in every iteration, is $i = 0$, $\{S_i\}_\delta = 16/16$ and $s = -1$. However we are not confined to using $\{S_0\}_\delta = 16/16$. In order to use the same table as in all the other cases we have to match the truncated value of S_0, with a table entry whose selection bound can be chosen in the range [-21,-19]. We see that we can choose either $\{S_0\}_\delta = 13/16$ or $14/16$. For $\{S_0\}_\delta = 13/16$ we can keep $p_{-1} \cdot 8 = -20$, while for $\{S_0\}_\delta = 14/16$ we have to change $p_{-1} \cdot 8$ to -21 which require a bit more to represent resulting in a more complex table. We therefore choose $\{S_0\}_\delta = 13/16$. It is also evident that the selection functions can be chosen identical for $\{S_i\}_\delta = 15/16$ and $\{S_i\}_\delta = 16/16$, except for the case $i = 0$ which we have taken care of. This means that whenever $\{S_i\}_\delta = 1/16$ we convert it to $15/16$, which eliminates the need to use the integer bit os S_i as input to the lookup table. The final lookup table is then given in Table 6.10. So although we require $I \geq 3$, we have shown that it is possible to construct a table which results in a valid selection function for all i. There is therefore no need to consider implementations which require $I < 3$.

$\{S_i\}_\delta \cdot 16$	8	9	10	11	
$\tilde{L}_{-1}(3) \cdot 8, \tilde{U}_{-2}(3) \cdot 8$	-13,-13	-14,-14	-16,-16	-18,-17	
$p_{-1}(3)$	-13/8	-7/4	-2	-9/4	
$\tilde{L}_0(3) \cdot 8, \tilde{U}_{-1}(3) \cdot 8$	-5,-4	-5,-4	-6,-5	-7,-5	
$p_0(3)$	-1/2	-1/2	-3/4	-3/4	
$\tilde{L}_1(3) \cdot 8, \tilde{U}_0(3) \cdot 8$	3,4	4,5	4,6	4,6	
$p_1(3)$	1/2	1/2	1/2	1/2	
$\tilde{L}_2(3) \cdot 8, \tilde{U}_1(3) \cdot 8$	12,12	14,14	15,16	16,17	
$p_2(3)$	3/2	7/4	2	2	
$\{S_i\}_\delta \cdot 16$	12	13	14	15	16
$\tilde{L}_{-1}(3) \cdot 8, \tilde{U}_{-2}(3) \cdot 8$	-19,-18	-21,-20	-23,-21	-24,-22	-26,-24
$p_{-1}(3)$	-9/4	-5/2	-11/4	-3	-3
$\tilde{L}_0(3) \cdot 8, \tilde{U}_{-1}(3) \cdot 8$	-7,-5	-8,-6	-9,-6	-9,-6	-10,-7
$p_0(3)$	-3/4	-1	-1	-1	-1
$\tilde{L}_1(3) \cdot 8, \tilde{U}_0(3) \cdot 8$	5,7	5,8	5,8	6,9	6,10
$p_1(3)$	3/4	1	1	1	1
$\tilde{L}_2(3) \cdot 8, \tilde{U}_1(3) \cdot 8$	18,19	19,21	20,22	22,24	23,26
$p_2(3)$	9/4	5/2	5/2	1	1

Table 6.5: $i \geq 3$.

$\{S_0\}_\delta \cdot 16$	16
$\tilde{L}_{-1}(0) \cdot 8, \tilde{U}_{-2}(0) \cdot 8$	-21,-19
$\tilde{L}_0(0) \cdot 8, \tilde{U}_{-1}(0) \cdot 8$	-9,-6
$\tilde{L}_1(0) \cdot 8, \tilde{U}_0(0) \cdot 8$	
$\tilde{L}_2(0) \cdot 8, \tilde{U}_1(0) \cdot 8$	

Table 6.6: $i = 0$.

$\{S_1\}_\delta \cdot 16$	8	12	16
$\tilde{L}_{-1}(1) \cdot 8, \tilde{U}_{-2}(1) \cdot 8$		-18,-16	-25,-21
$\tilde{L}_0(1) \cdot 8, \tilde{U}_{-1}(1) \cdot 8$	-5,-4	-7,-5	-10,-6
$\tilde{L}_1(1) \cdot 8, \tilde{U}_0(1) \cdot 8$	3,5	5,7	6,10
$\tilde{L}_2(1) \cdot 8, \tilde{U}_1(1) \cdot 8$	12,14	17,20	23,27

Table 6.7: $i = 1$.

$\{S_i\}_\delta \cdot 16$	8	9	10	11	
$\tilde{L}_{-1}(2) \cdot 8, \tilde{U}_{-2}(2) \cdot 8$		-14,-13	-16,-14	-17,-15	
$\tilde{L}_0(2) \cdot 8, \tilde{U}_{-1}(2) \cdot 8$	-5,-4	-5,-4	-6,-4	-7,-5	
$\tilde{L}_1(2) \cdot 8, \tilde{U}_0(2) \cdot 8$	3,4	4,5	4,6	4,6	
$\tilde{L}_2(2) \cdot 8, \tilde{U}_1(2) \cdot 8$	11,13	13,14	14,16	15,18	
$\{S_i\}_\delta \cdot 16$	12	13	14	15	16
$\tilde{L}_{-1}(2) \cdot 8, \tilde{U}_{-2}(2) \cdot 8$	-19,-17	-21,-18	-22,-19	-24,-21	-26,-22
$\tilde{L}_0(2) \cdot 8, \tilde{U}_{-1}(2) \cdot 8$	-7,-5	-8,-5	-9,-6	-9,-6	-10,-6
$\tilde{L}_1(2) \cdot 8, \tilde{U}_0(2) \cdot 8$	5,7	5,8	5,8	6,9	6,10
$\tilde{L}_2(2) \cdot 8, \tilde{U}_1(2) \cdot 8$	17,19	18,21	19,23	21,24	22,26

Table 6.8: $i = 2$.

$\{S_i\}_\delta \cdot 16$	8	9	10	11
$\tilde{L}_{-1}(i) \cdot 8, \tilde{U}_{-2}(i) \cdot 8$				
$i = 0$				
$i = 1$				
$i = 2$		-14,-13	-16,-14	-17,-15
$i \geq 3$	-13,-13	-14,-14	-16,-16	-18,-17
$p_{-1} \cdot 8$	-13	-14	-16	-17
$\tilde{L}_0(i) \cdot 8, \tilde{U}_{-1}(i) \cdot 8$				
$i = 0$				
$i = 1$	-5,-4			
$i = 2$	-5,-4	-5,-4	-6,-4	-7,-5
$i \geq 3$	-5,-4	-5,-4	-6,-5	-7,-5
$p_0 \cdot 8$	-4	-4	-6	-6
$\tilde{L}_1(i) \cdot 8, \tilde{U}_0(i) \cdot 8$				
$i = 0$				
$i = 1$	3,5			
$i = 2$	3,4	4,5	4,6	4,6
$i \geq 3$	3,4	4,5	4,6	4,6
$p_1 \cdot 8$	4	4	4	4
$\tilde{L}_2(i) \cdot 8, \tilde{U}_1(i) \cdot 8$				
$i = 0$				
$i = 1$	12,14			
$i = 2$	11,13	13,14	14,16	15,18
$i \geq 3$	12,12	14,14	15,16	16,17
$p_2 \cdot 8$	12	14	16	16

Table 6.9: Combined table.

$\{S_i\}_\delta \cdot 16$	12	13	14	15	16
$\tilde{L}_{-1}(i) \cdot 8, \tilde{U}_{-2}(i) \cdot 8$					
$i = 0$					-21,-19
$i = 1$	-18,-16				-25,-21
$i = 2$	-19,-17	-21,-18	-22,-19	-24,-21	-26,-22
$i \geq 3$	-19,-18	-21,-20	-23,-21	-24,-22	-26,-24
$p_{-1} \cdot 8$	-18	-20	-22	-24	-
$\tilde{L}_0(i) \cdot 8, \tilde{U}_{-1}(i) \cdot 8$					
$i = 0$					-9,-6
$i = 1$	-7,-5				-10,-6
$i = 2$	-7,-5	-8,-5	-9,-6	-9,-6	-10,-6
$i \geq 3$	-7,-5	-8,-6	-9,-6	-9,-6	-10,-7
$p_0 \cdot 8$	-6	-8	-8	-8	-8
$\tilde{L}_1(i) \cdot 8, \tilde{U}_0(i) \cdot 8$					
$i = 0$					
$i = 1$	5,7				6,10
$i = 2$	5,7	5,8	5,8	6,9	6,10
$i \geq 3$	5,7	5,8	5,8	6,9	6,10
$p_1 \cdot 8$	6	8	8	8	8
$\tilde{L}_2(i) \cdot 8, \tilde{U}_1(i) \cdot 8$					
$i = 0$					
$i = 1$	17,20				23,27
$i = 2$	17,19	18,21	19,23	21,24	22,26
$i \geq 3$	18,19	19,21	20,22	22,24	23,26
$p_2 \cdot 8$	18	20	20	24	24

Table 6.9: Combined table (continued).

$\{S_i\}_\delta \cdot 16$	8	9	10	11	12	13	14	15
p_{-1}	-13/8	-7/4	-2	-17/8	-9/4	-5/2	-11/4	-3/2
p_0	-1/2	-1/2	-3/4	-3/4	-3/4	-1	-1	-1
p_1	1/2	1/2	1/2	1/2	3/4	1	1	1
p_2	3/2	7/4	2	2	9/4	5/2	5/2	3

Table 6.10: Final lookup table.

Chapter 7

Combined Division and Square Root

Since an implementation of the IEEE standard requires both the division and square root operation to be present, and given the number of similarities between the division and square root algorithms presented in the previous chapters, one might wonder if a combined implementation is possible. It turns out that this is indeed the case. Shared radix-4 and 8 table lookup algorithms have been presented by Fandrianto [6, 7]. A radix-4 direct comparison implementation was given in [19].

	Division	Square Root
problem	$X = Q \cdot D + R, \lvert X/D - Q \rvert < \operatorname{ulp} Q$	$X = S^2 + R, \lvert \sqrt{X} - S \rvert < \operatorname{ulp} S$
result	$Q = \sum q_i \beta^{-i}$	$S = \sum s_i \beta^{-i}$
operands	$D \in [1/2, 1[, \lvert X \rvert < \lvert D \rvert, Q \in]-1, 1[$	$X \in [1/4, 1[, S \in [1/2, 1[$
partial rem.	$R_i = \beta^i (X - Q_i D)$	$R_i = \beta^i (X - S_i^2)$
recurrence	$R_{i+1} = \beta R_i - q_{i+1} D$	$R_{i+1} = \beta R_i - s_{i+1} 2 S_i - s_{i+1}^2 \beta^{-(i+1)}$
bounds	$\lvert R_i \rvert \leq \rho D$	$\lvert R_i - \rho^2 \beta^{-i} \rvert \leq \rho 2 S_i$
selection	$L_q = (q - \rho) D$	$L_s(i) = (s - \rho) 2 S_i + (s - \rho)^2 \beta^{-(i+1)}$
interval	$U_q = (q + \rho) D$	$U_s(i) = (s + \rho) 2 S_i + (s - \rho)^2 \beta^{-(i+1)}$
overlap	$(2\rho - 1) D$	$(2\rho - 1) \left(2 S_i + (2s - 1) \beta^{-(i+1)} \right)$

Table 7.1: Division and square root comparison.

Table 7.1 compares division and square root. It is evident that they are very similar with $Q_i D$ corresponding to S_i^2, and $2S_i$ taking the place of D. In order to construct a combined implementation we have to make some requirements.

- For division we require that $Q \in [1/2, 1[$, this will bring $Q_i D$ into the same range as S_i^2.
- For square root we scale R_i by $1/2$; the range of R_i in both cases is then the same except for the i-dependent term in the square root case.

- D and S_i should be represented in the same encoding. To meet this requirement we do an on-the-fly conversion of S_i, which in turn requires an on-the-fly conversion of Q_i.

Recurrence Let Y correspond to the divisor in the division case, and the partial root in the square root case. We then have the recurrence:

$$R_{i+1} = \beta R_i - W_i \tag{7.1}$$

where $W_i = d_{i+1}Y$ for division and $W_i = d_{i+1}2Y + d_{i+1}^2\beta^{-(i+1)}$ for square root.

7.1 Implementation

Figure 7.1 shows a block diagram of one iteration of the combined algorithm.

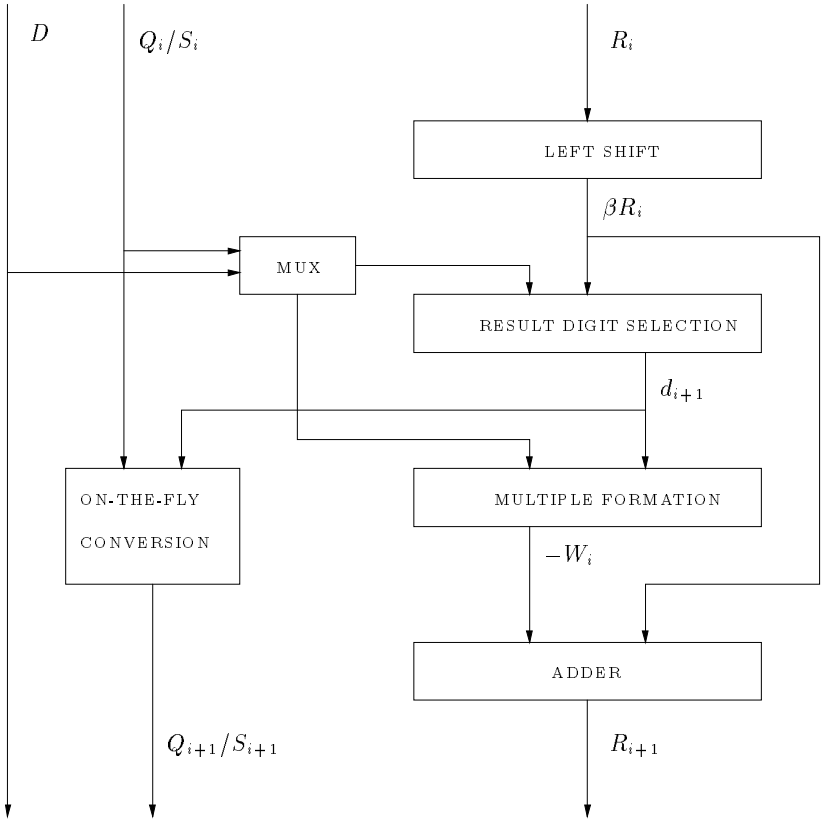

Figure 7.1: Block diagram of combined iteration.

7.2 The Digit Selection Function

The function $d\_select(R_i, Y)$ is represented by z_d, $d \in \mathcal{D}_a$.

$$z_d \leq \beta R_i < z_{d+1} \Rightarrow d_{i+1} = d \tag{7.2}$$

As before this selection function can be implemented using either direct comparison or table lookup.

7.3 Direct Comparison

With $D = S_i$ and $q = s$ the digit selection intervals $L_s^*/2$ and $U_s^*/2$ given in (6.52) and (6.53) are identical to the intervals L_q and U_q given in (4.9).

This means that we can use the same comparison functions in both cases, except for the first I iterations. These iterations have to be handled separately as we did in Section 6.3.1.

7.3.1 Formation of Comparison Functions

Because the bound on the comparison functions are identical, we can use the same comparison functions as in the previous chapters.

7.4 Table Lookup

In order to construct a combined table, we require that the selection bounds are identical. As mentioned above we need to scale the partial remainder by 1/2 in the square root case. The selection bounds are then in the range:

Division

$$\tilde{L}_q = \left\lceil \left(\max_{D'} L_q - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c} \tag{7.3}$$

$$\tilde{U}_{q-1} = \left\lfloor \left(\min_{D'} U_{q-1} - \sup \epsilon_R \right) 2^c \right\rfloor 2^{-c} + 2^{-c}. \tag{7.4}$$

Square root

$$\tilde{L}_s(I) = \left\lceil \left(\max_{S_i', i \geq I} \frac{1}{2} L_s(i) - \inf \epsilon_R \right) 2^c \right\rceil 2^{-c} \tag{7.5}$$

$$\tilde{U}_{s-1}(I) = \left\lfloor \left(\min_{S_i', i \geq I} \frac{1}{2} U_{s-1}(i) - \sup \epsilon_R \right) 2^c \right\rfloor 2^{-c} + 2^{-c}. \tag{7.6}$$

The condition for the existence of combined table is:

$$\forall q, s, m : \tilde{L}_s(I) \leq \tilde{U}_{q-1} \wedge \tilde{L}_q \leq \tilde{U}_{s-1}(I) \wedge \tilde{L}_q \leq \tilde{U}_q. \tag{7.7}$$

Hence:

$$z_d \in \left[\tilde{L}_q, \tilde{U}_{q-1} \right] \tag{7.8}$$

$$z_d \in \left[\tilde{L}_s(I), \tilde{U}_{s-1}(I) \right]. \tag{7.9}$$

The cases $i < I$ have to be handled by either an initial lookup table or by fitting the tables to the case $i \geq I$, as we did in the square root case. We now have to find values of c, δ, I, and truncations of βR_i, for which a combined table exists.

7.4.1 Integer bits

The number of integer bits required to represent the truncated partial remainder in the square root case is one less than what was required in the standalone square root algorithms, due to the scaling by 1/2.

From Tables 4.6 and 6.3 we see that the number of integer bits is then equal to the number used in the division case.

7.4.2 Results

Table 7.2 shows the number of inputs w to the lookup table for a redundant partial remainder. This table was generated by calculating the condition (7.7) for the existence of a combined table

β	a	τ	p	n	c	δ	I	w
2	1	3	2	2	2	1	0	4
4	2	3	4	4	4	5	3	10
4	2	3	4	4	4	6	2	11
4	2	3	5	4	5	5	2	11
4	2	3	5	5	4	4	3	9
4	2	3	5	5	3	5	3	9
4	2	3	5	5	5	6	1	12
4	3	4	3	2	3	4	2	9
4	3	4	3	3	3	3	2	8
4	3	4	4	3	4	5	1	11
4	3	4	4	4	2	4	2	8
4	3	4	4	4	4	4	1	10

Table 7.2: Table lookup truncations.

Chapter 8

Comparison of Implementations

In this chapter we shall compare the direct comparison method with the table lookup method, in terms of speed and area consumption. The technology used is described in Appendix B, and is identical to the technology used in [5]. We shall also make a theoretical comparison based on the table results in [14].

Only the division algorithms will be examined since no data on the square root tables is available. As we have seen the hardware needed to implement the square root and combined algorithms is very similar. The results we obtain for the division case, should therefore also give an indication of the performance of these algorithms. For the implementations, we shall abide to the following conventions:

- We assume that one carry-lookahead adder is available in the main arithmetic unit. Usage of this adder will not take up any area, but will of course incure some delay. This assumption is also made in [5].
- The carry-lookahead adder can be used two times per cycle.
- Critical paths are shown in gray in diagrams.
- Non-critical delays are shown in parentheses in tables.
- Cycle time is rounded to nearest integer.
- Total area is rounded to nearest 100.

8.1 Division

The main difference between the direct comparison method and the table lookup method is the implementation of the quotient digit selection. Now we describe how we implement the comparisons in hardware. Two alternatives are available.

- Method 1: Compute the comparison functions to Δ bits beforehand. In every iteration we perform a Δ-bit 3-2 subtraction of the comparison functions from $\{\beta R_i\}_\Delta$, followed by a sign detection on the redundant result. We then need the comparison functions in non-redundant form. The ones whose formation is nontrivial, i.e $3D$ etc., need to be computed beforehand.

- Method 2: We use 4-2 adders instead, for subtracting the comparison functions from the truncated partial remainder. With this approach there is no need to store the comparison functions, since they are only needed in redundant representation. For example in carry-save $3D$ can be encoded as $3D = (2D)^s + (D)^c$, and in borrow-save as $3D = (4D)^s - (D)^b$. Note that the 4-2 adders need not be full-precision. If we need Δ bits of for example $3D$ then we use $\Delta + 1$ bits of $2D$ and D. The corresponding 4-2 adder then needs to be $\Delta + 1$ bits wide, as does the sign detection circuit.

The two methods may be mixed so that some of the comparison functions are computed and stored at the beginning, while the rest are computed on-the-fly. Each comparison function that is computed at the beginning needs one half of a cycle of computation time, due to the use of the carry-lookahead adder.

Since 4-2 adders are slower and use more area than the 3-2 adders Method 1 seems better. However, this advantage is diminished by the initial cycles needed to compute the comparison functions. A number of factors influence our choice between the two methods including:

- The time needed to compute the comparison function to begin with should be less than the sum of the speed increase due to using 3-2 adders, taken over the rest of the cycles.

- For higher radix we need less cycles, so the weight of the initial cycles increase. But we should also expect higher a cycle time, implying that we could compute more than two comparison functions in one cycle. For simplicity we shall stay with our initial assumption that one use of the adder takes one half of a cycle, regardless of the cycle time.

The best approach is very technology dependent, as one could imagine.

Depending on the implementation we may choose to perform only the comparisons involving positive comparison functions. In this case we first factor out the sign of $\{\beta R_i\}_\Delta$ then subtract, determine quotient digit and finally multiply the result by the sign. The advantage here is that we only need half the number of comparators, each $\Delta - 1$ bits wide. This comes at an additional delay for computing the sign of $\{\beta R_i\}_\Delta$, and multiplying the result by it.

8.1.1 Radix 2, with Carry-Save Adder

In Section 4.3.11 we saw that we can use comparison functions $s_q = \{-1/2, 0\}$ and $\Delta = 4$. Figure 8.1 shows a diagram of this implementation. We use Method 1 since the comparison functions do not need any storage. For the comparisons we first perform a 1-bit carry-save addition of $1/2$ to $\{\beta R_i\}_\Delta$. The two borrow-save numbers are then converted to sign-digit through a single XOR-gate. Then we determine the sign of the two sign-digit numbers. The most significant bits are negated before feeding them to the sign-determination tree. For each comparison we need the sign-detection tree shown in Figure 8.2. The sign-digit output (r^s, r^d) of the two trees is fed into two NAND-gates. Let the outputs of the two NAND-gates be e_0 and e_1.

e_0	e_1	q_{i+1}
0	0	-1
1	0	0
1	1	1

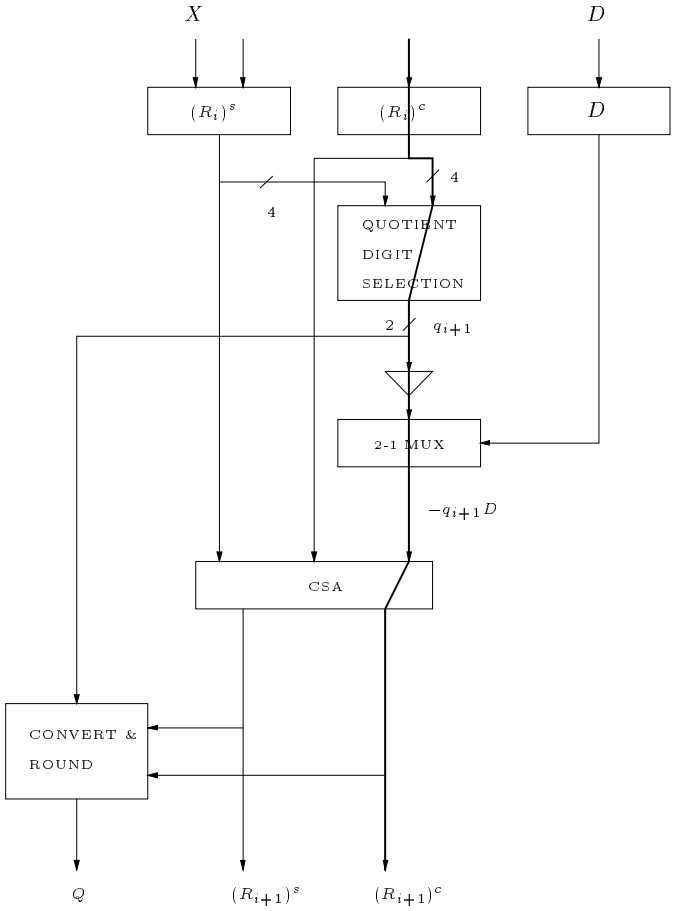

Figure 8.1: Implementation of radix-2 division with carry-save adder.

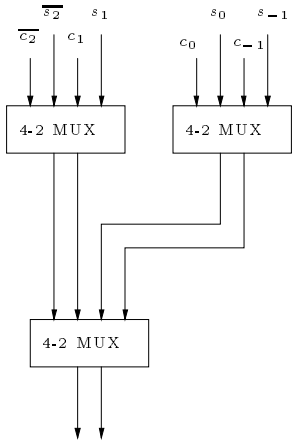

Figure 8.2: Sign detection tree, $\Delta = 4$, sign-digit operands.

These now have to be recoded into borrow-save with the encoding.

q^+	q^-	q_{i+1}
0	1	-1
0	0	0
1	0	1

The recoding is given by:

$$q^- = \overline{e_0}, \quad q^+ = e_1. \tag{8.1}$$

Thus the recoding can be accomplished by a single NOT-gate. The delays and area used for quotient digit selection are shown in Table 8.1. Table 8.2 shows the delay and area, for

event	delay	area
3-2 CSA 1-bit	4.2	6.7
conversion to SD	2.2	17.6
negation of msb	1.0	2.0
o-tree 4-bit ($\cdot 2$)	2.8	36.0
encode	1.4	3.0
total	11.6	65.3

Table 8.1: Radix-2 delay and area for quotient-digit selection.

the whole circuit. Note that the delay of the CSA is the delay of the input from the MUX, since this input is fed to fast output of the CSA The two inputs to the CSA from the partial remainder, which are fed to the slower outputs of the CSA, are available much earlier. As

event	delay	area
registers ($\cdot 3$)	8.0	650
quotient-digit selection	11.6	70
buffers ($\cdot 2$)	1.8	5
2-1 MUX	1.4	160
3-2 CSA	2.2	360
convert and round	(13)	1900
cycle time	25	
total area		3100

Table 8.2: Radix-2 total delay and area.

shown in Example 3.4.2 of [5] the circuitry needed to implement the comparisons is very simple. Implementing the two comparisons with comparators is a bad design choice.

8.1.2 Radix 4, Minimally Redundant with Carry-Save Adder

We use $s_q = \{-D/2, 0, D/2, 3D/2\}$ and $\Delta = 7$. Figure 8.3 shows the implementation of Method 1. We compute the comparison function $3D/2$ at the beginning. For each comparison

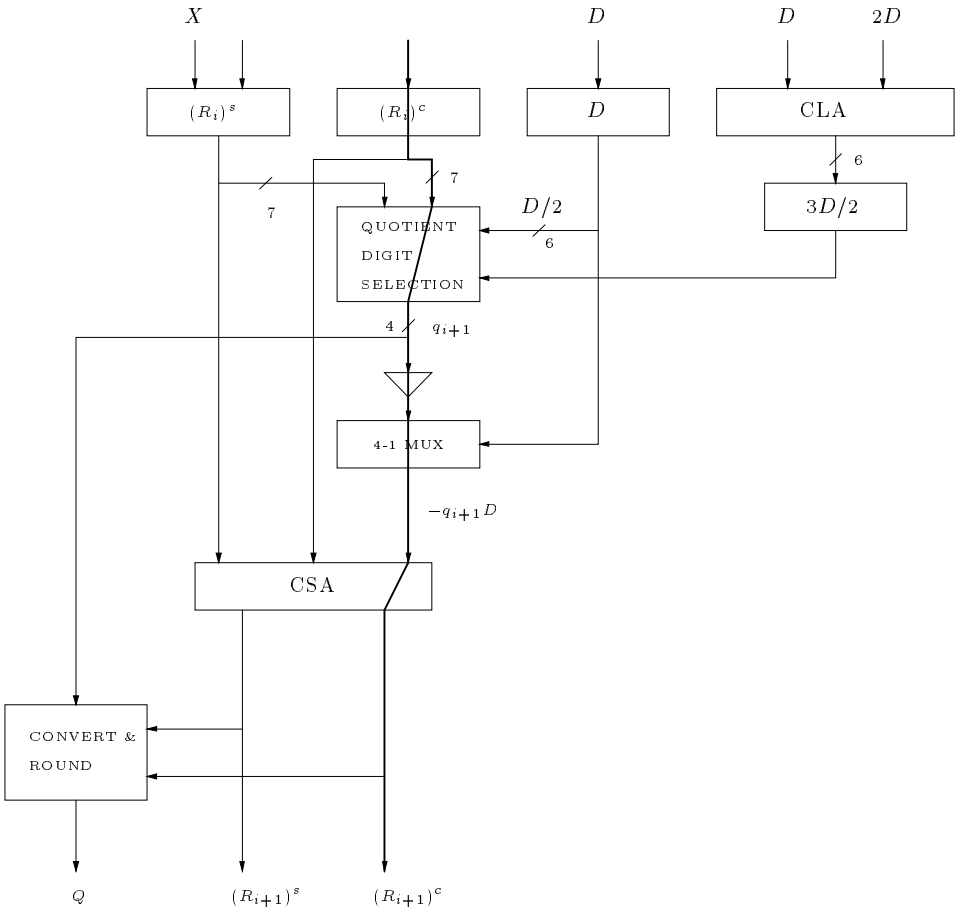

Figure 8.3: Implementation of radix-4 division.

we need the 7-bit sign-detection tree shown in Figure 8.4. The sign-digit outputs from the trees are fed into four NAND-gates, to produce four $\geq$-signals. Let the outputs from the NAND-gates be e_{-1}, e_0, e_1, e_2.

e_{-1}	e_0	e_1	e_2	q_{i+1}
0	0	0	0	-2
1	0	0	0	-1
1	1	0	0	0
1	1	1	0	1
1	1	1	1	2

We want the borrow-save encoding.

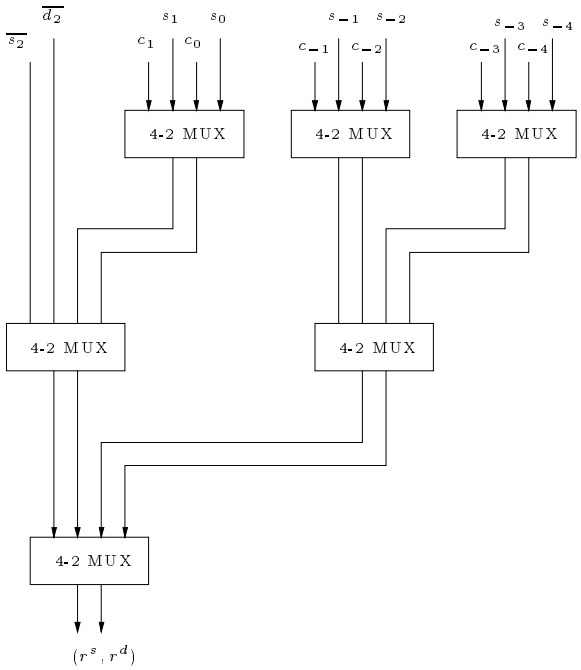

Figure 8.4: Sign detection tree, $\Delta = 7$, sign-digit operands.

q^{2+}	q^{+}	q^{-}	q^{2-}	q_{i+1}
0	0	0	1	-2
0	0	1	0	-1
0	0	0	0	0
0	1	0	0	1
1	0	0	0	2

So we recode as follows:

$$q^{2-} = \overline{e_{-1}}, \quad q^{-} = e_{-1} \oplus e_0, \quad q^{+} = e_1 \oplus e_2, \quad q^{2+} = e_2. \tag{8.2}$$

Table 8.3 shows the area and delay for quotient digit selection. The delays and area con-

event	delay	area
3-2 CSA 7-bit (·3)	4.2	140.7
convert to SD	2.2	61.6
o-tree 7-bit (·4)	4.2	144.0
encode	3.2	7.2
total	13.8	353.5

Table 8.3: Radix-4 delay and area for quotient-digit selection.

sumption for the entire implementation is shown in Table 8.4.

event	delay	area
CLA	(1 cycle)	0
registers (·3)	8.0	650
register 6-bit	(8.0)	24
quotient-digit selection	13.8	350
buffers (·4)	1.8	10
4-1 MUX	1.8	300
3-2 CSA	2.2	360
convert and round	(13)	1900
cycle time	28	
total area		3600

Table 8.4: Radix-4 delay and area.

8.1.3 Radix 4, Maximally Redundant with Carry-Save Adder

We can use $s_q = \{-2D, -D, 0, D, 2D, 3D\}$ and $\Delta = 6$. We store the comparison function, $3D$, in a full-length register.

Encoding of the sign-digit output from the tree into borrow-save for q_{i+1}. This encoding can be achieved with the same logic as in the previous example. Table 8.5 shows the area and delay for quotient digit selection. The delays and area consumption for the entire implementation is shown in Table 8.6. The results are almost identical to the minimally redundant case.

event	delay	area
3-2 CSA 6-bit (·5)	4.2	201.0
convert to SD	2.2	79.2
o-tree 6-bit (·6)	4.2	180.0
encode	3.2	7.2
total	13.8	467.4

Table 8.5: Radix-4 maximally redundant, delay and area for quotient-digit selection.

8.1.4 Radix 8, Minimally Redundant with Carry-Save Adder

We can use $s_q = \pm\{D/2, 3D/2, 5D/2, 7D/2\}$, and $\Delta = 9$. Calculate non-redundant truncated versions of $5D$ and $7D$ to begin with, and a full-length non-redundant $3D$. All comparison functions are then available in non-redundant form. And all divisor multipla are available.

The rest of the comparison functions are computed in every iteration. Figure 8.5 shows this implementation. Table 8.7 shows the area and delay for quotient digit selection. The delays and area consumption for the entire implementation is shown in Table 8.8.

event	delay	area
CLA	(1 cycle)	0
registers (·4)	8.0	860
quotient-digit selection	13.8	470
buffers (·6)	1.8	15
6-1 MUX	2.2	400
3-2 CSA	2.2	360
convert and round	(13)	1900
cycle time	28	
total area		4000

Table 8.6: Radix-4 maximally redundant, delay and area.

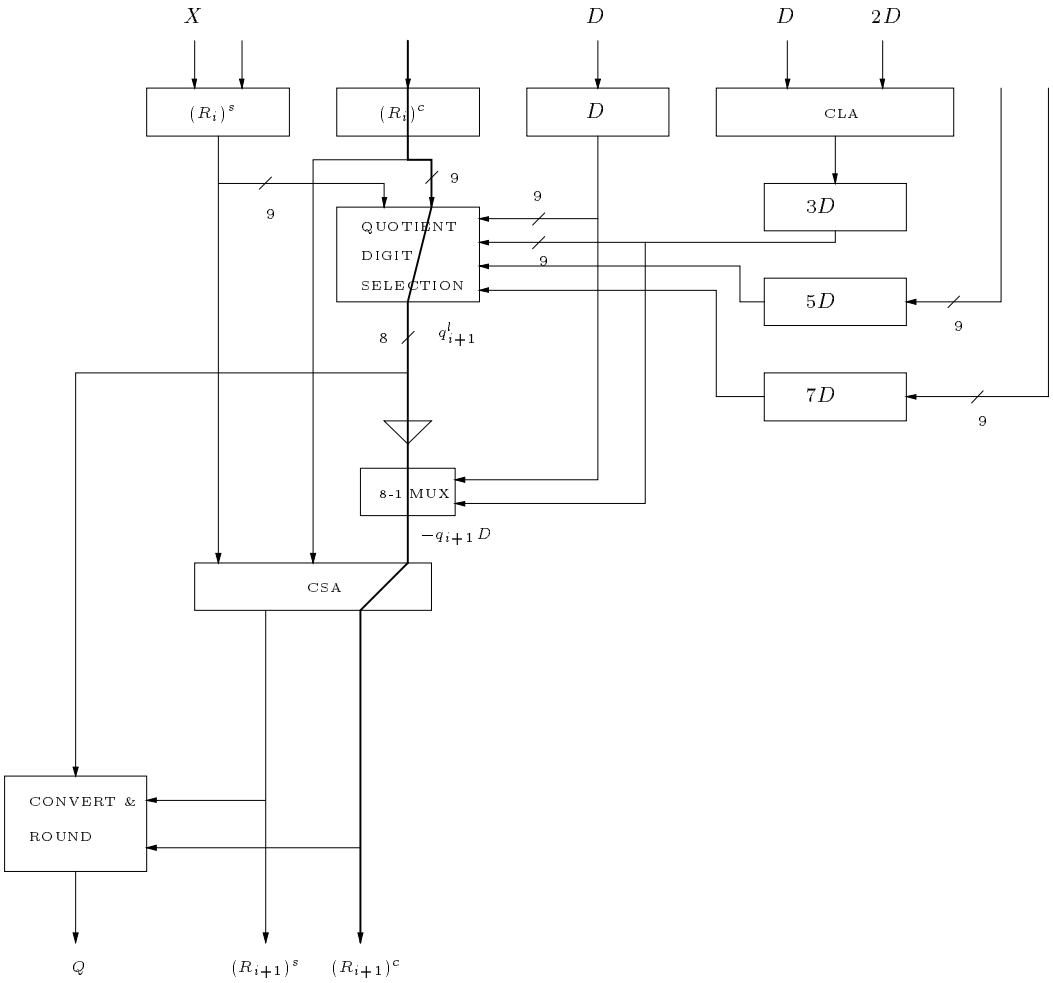

Figure 8.5: Implementation of Radix-8 Minimally Redundant Division.

event	delay	area
3-2 CSA 9-bit (·8)	4.2	482.4
convert to SD	2.2	158.4
o-tree 9-bit (·8)	5.6	384.0
encode	3.2	8.4
total	15.2	1033.2

Table 8.7: Radix-8 minimally redundant delay and area for quotient-digit selection.

event	delay	area
CLA	(2 cycles)	0
registers 53-bit (·4)	8.0	860
registers 9-bit (·2)	(8.0)	70
quotient-digit selection	15.2	1000
buffers (·8)	1.8	20
8-1 MUX	2.0	400
3-2 CSA	2.2	360
4-1 MUX	(1.8)	300
convert and round	(13)	1900
cycle time	29	
total area		4900

Table 8.8: Radix-8 minimally redundant, delay and area.

8.1.5 Radix 8, Maximally Redundant with Borrow-Save Adder

We can use $s_q = U_{q-1}$ and $\Delta = 7$. To avoid generating $3D, 5D$ and $7D$ to full precision for the divisor multiplum generation, we split the quotient digit into two components, $q^l \in \{-2, -1, 0, 1, 2\}$ and $q^h \in \{-8, -4, 0, 4, 8\}$, as suggested in [5].

We store truncated versions of $3D, 5D, 6D$, and $7D$. These require 2 cycles to compute, and 4 7-bit registers. We also need 12 3-2 6-bit BSA's, and 2 1-bit BSA's, since we can use a 1-bit constant in the case of the comparison with D, as shown in Table 4.3. The delays and area for quotient digit selection are shown in Table 8.9. The delays and area consumption for

event	delay	area
3-2 BSA 7-bit (·12)	4.2	562.8
3-2 BSA 1-bit (·2)	(4.2)	13.4
convert to SD	2.2	215.6
o-tree 7-bit (·14)	4.2	504
encode to geq	1.0	7.0
encode (q^h)	2.2	5.4
encode (q^l)	(4.0)	20.0
total	13.8	1338.2

Table 8.9: Radix-8 maximally redundant, delay and area for quotient-digit selection.

the entire implementation are shown in Table 8.10.

event	delay	area
CLA	(2 cycles)	0
registers 53-bit (·3)	8.0	650
registers 7-bit (·4)	(8.0)	110
quotient-digit selection	13.8	1300
buffers (·8)	1.8	20
4-1 MUX	1.8	300
3-2 CSA	2.2	360
4-1 MUX	(1.8)	300
3-2 CSA	4.2	360
convert and round	(13)	1900
cycle time	32	
total area		5300

Table 8.10: Radix-8 maximally redundant, delay and area.

8.1.6 Radix 16, with Carry-Save Adder

The implementation is radix-16 with $a = 10$, i.e $\rho = 2/3$. The reason is that the decomposition of the quotient digit into q^h and q^l described in the previous implementation, allows a maximum digit of 10, and minimum digit -10.
We can use $s_q = \pm\{D/2, 3D/2, 5D/2, 7D/2, 9D/2, 11D/2, 13D/2, 15D/2, 17D/2, 19D/2\}$, and $\Delta = 9$, as seen in the division chapter. Due to the large number of comparison functions, and the fact that we have to use 4-2 adders for some of them We compute $3D, 5D$ and $7D$ to 10 bits to begin with and store them for later use. The rest of the comparison functions can then be generated from these three and from shifted values of D. Due to the large number of comparison functions, and the fact that we have to use 4-2 adders for some of them, we choose to factor out the sign before quotient digit selection, to save area. Table 8.11 shows the area and delay for quotient digit selection. The delays and area consumption for the entire

event	delay	area
sign-detect 9-bit	4.2	48.0
3-2 CSA 8-bit ($\cdot 4$)	(4.2)	214.4
4-2 CSA 9-bit ($\cdot 6$)	6.0	837.0
convert to SD	2.2	189.2
o-tree 8-bit ($\cdot 4$)	(4.2)	168.0
o-tree 9-bit ($\cdot 6$)	4.2	288.0
encode to geq	1.0	10.0
encode (q^h)	2.2	3.2
encode (q^l)	(4.0)	10.0
sign-multiply	2.2	4.4
total	22.0	1772.2

Table 8.11: Radix-16 delay and area for quotient-digit selection.

implementation are shown in Table 8.12.

8.2 Comparison

A comparison between direct comparison and table lookup is made based on previous results in the litterature.

- A theoretical comparison is made based on the results in [14].
- Based on [5] we make a comparison of a few implementations.

8.2.1 Theoretical Comparison

According to a study performed by Obermann and Flynn [14] the delay grows linearly with the radix, while the area grows quadratically with the radix. The average delay and area for radix 2 through 32 is shown if Figures 8.6 and 8.7. The area is fitted to a polynomial of degree 2,

event	delay	area
CLA	(2 cycles)	0
registers 53-bit (·3)	8.0	650
registers 10-bit (·3)	(8.0)	120
quotient-digit selection	22	1800
buffers (·8)	1.8	10
4-1 MUX	1.8	300
3-2 CSA	2.2	360
4-1 MUX	(1.8)	300
3-2 CSA	4.2	360
convert and round	(13)	1900
cycle time	40	
total area		5800

Table 8.12: Radix-16 delay and area.

confirming the quadratical dependence. These delays are close to a linear dependency except for the radix-2 delay, which is too small compared with the other delays. We now analyse the direct comparison method for maximally redundant digits sets, borrow-save, and $s_q = U_{q-1}$ we have from (4.38),(4.40) and (4.71):

$$\Delta = \log_2 \beta + 4. \tag{8.3}$$

The delays and area for a single Δ-bit comparison are shown in Table 8.13. Assume that we

event	delay	area
3-2 BSA Δ-bit	4.2	$6.7 \cdot \Delta$
convert to SD	2.2	$2.2 \cdot \Delta$
o-tree Δ-bit	$1.4 \cdot \lceil \log_2 \Delta \rceil$	$6.0 \cdot (\Delta - 1)$
encode to $\geq$	1.0	1.0

Table 8.13: Delay and area for one comparison.

just output the quotient digit in borrow-save. Then we get an additional delay for encoding the $2a$ outputs to borrow-save, corresponding to a row of XOR's, and a single NOT-gate. The total delay is then

$$\begin{aligned} delay &= 9.6 + 1.4 \cdot \lceil \log_2 \Delta \rceil \\ &= 9.6 + 1.4 \cdot \lceil \log_2 (\log_2 \beta + 4) \rceil. \end{aligned}$$

And the total area is:

$$\begin{aligned} area &= 2a \cdot (15.9 \cdot \Delta + 3.2 - 6.0) \\ &= 2(\beta - 1)(9.9 \cdot (\log_2 \beta + 4) - 2.8). \end{aligned}$$

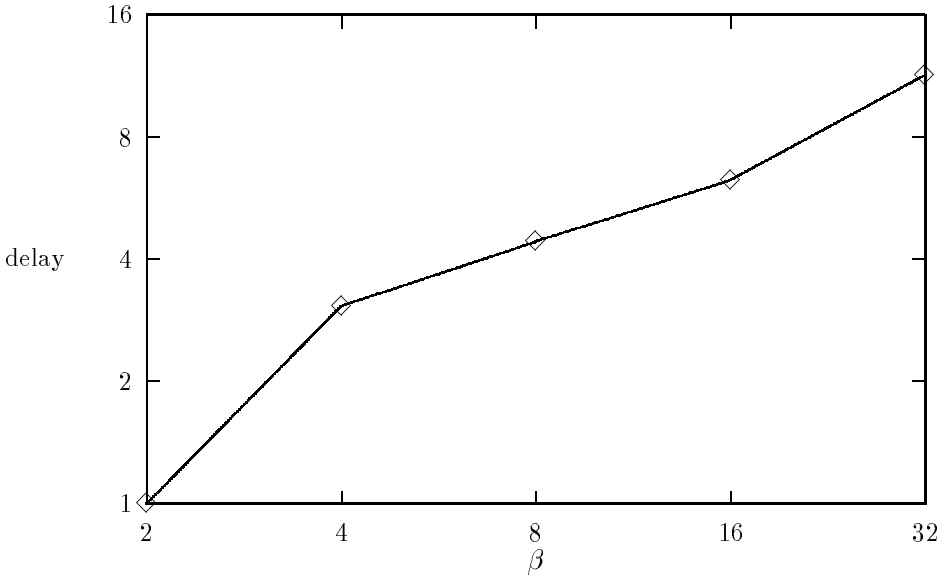

Figure 8.6: Relative table delay according to [14].

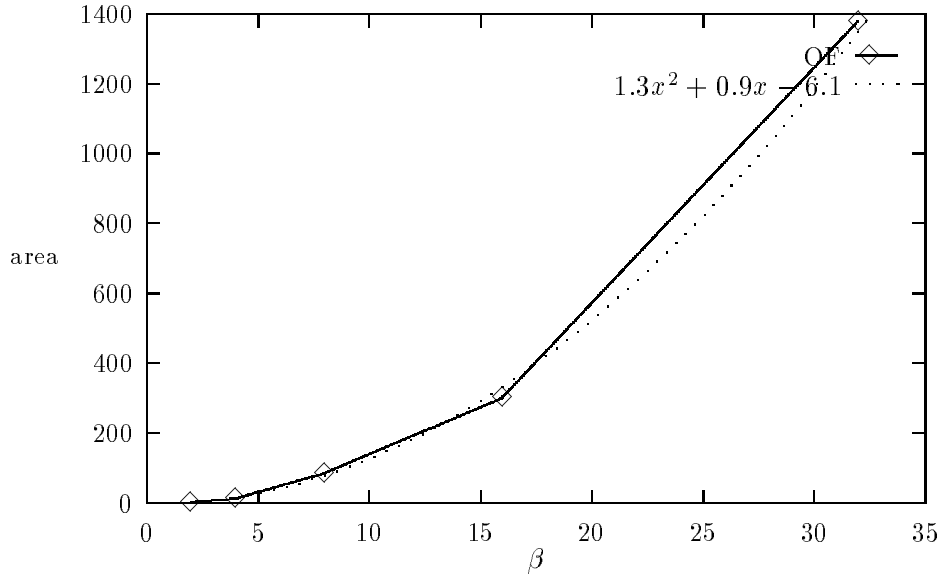

Figure 8.7: Relative table area according to [14].

So the delay grows as $\log^2 \beta$, while the area grows with $\beta \cdot \log \beta$. In both cases these growth rates are asymtotically better than the ones found by Obermann and Flynn for the tables. For radices 2 through 16 we have identical delay. The delay is a piecewise linear function, and we have the same delay for radix-32 and up to radix-8192.

The relative delays and area of the quotient digit selection for the direct comparison method is shown as a function of the radix, in Figures 8.8 and 8.9.

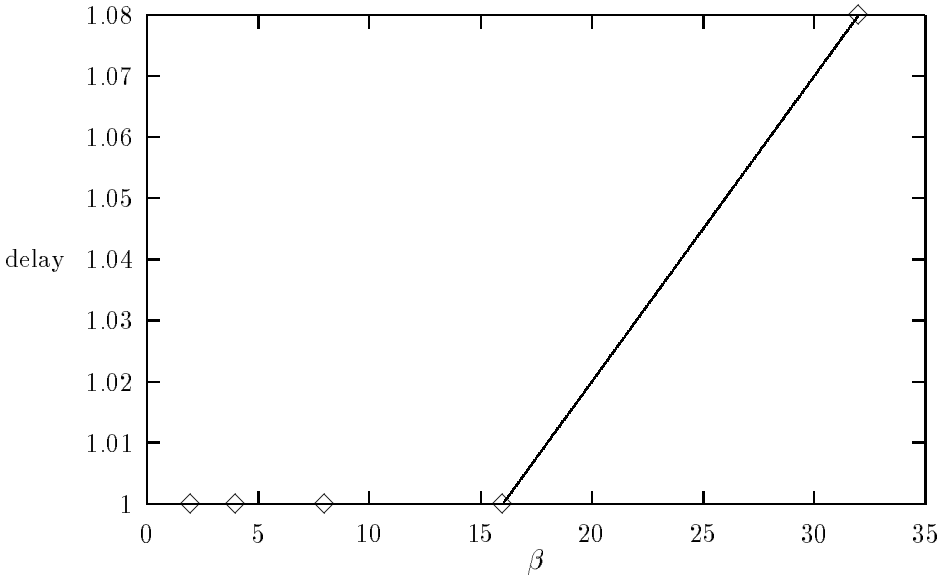

Figure 8.8: Direct comparison relative theoretical delay for quotient digit selection.

These results are of course only theoretical. Practical limitations such as generating the divisor multipla are not considered. To summarize the above:

method	delay	area
table	$O(\beta)$	$O(\beta^2)$
comparators	$O(\log^2 \beta)$	$O(\beta \cdot \log_2(\beta))$

Although the comparators have better asymptotical performance, constant factors can make this advantage disappear. We therefore need to investigate some actual implementations.

8.2.2 Comparison of Implementations

The implementations presented in Section 8.1 are now compared with the implementations in [5]. These comparisons should provide a clearer picture of whether the direct comparison method can compete with the tables or not.

Table 8.14 shows the delay and area for the implementations:

- Radix-2 with carry-save
- Radix-4 a=2 with carry-save

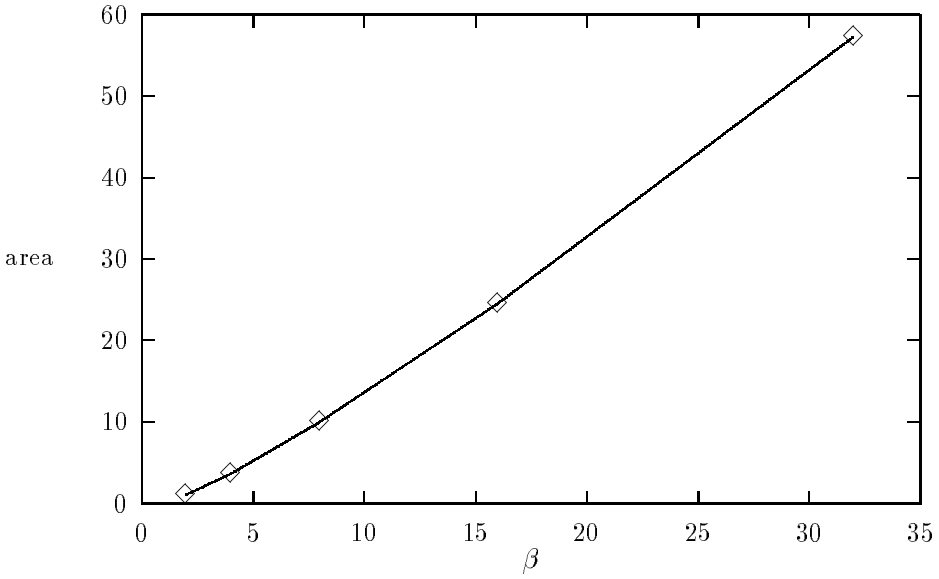

Figure 8.9: Direct comparison relative theoretical area for quotient digit selection.

- Radix-8 a=7, and carry-save for table lookup, borrow-save for direct comparison.
- Radix-16 a=10. Direct comparison only.

implementation	$\beta = 2$		$\beta = 4$		$\beta = 8$		$\beta = 16$
	tab	dc	tab	dc	tab	dc	dc
relative cycle-time	1.0	1.3	1.3	1.4	1.5	1.6	2.0
number of cycles	56	56	28	29	19	21	17
relative speedup	1.0	0.8	1.6	1.4	2.0	1.7	1.7
relative area	1.0	1.0	1.1	1.2	1.4	1.7	1.9
relative speedup/area	1.0	0.8	1.5	1.2	1.4	1.0	0.9

Table 8.14: Comparison of implementations

Figure 8.10 shows the results from Table 8.14 in graphical form. The results show that the table lookup method is slightly superior to direct comparison. However, a radix-16 table is impractical, while it is possible for direct comparison albeit at a large area expense. For radix-8, minimally redundant the cycle time is lower for direct comparison than it is for the maximally redundant radix-8 table implementation, but this advantage is lost due to the two overhead cycles needed. Since the tables are optimized and the comparators are not, these results do not give a true picture, of the actual relationship between the two methods. The tools used for optimizing the tables (ESPRESSO, SIS etc.) are not at our disposal, so there is no way that we can achieve the same level of optimization for the comparators. But we may take some comfort in the fact that even without optimization we can achieve a lower cycle time.

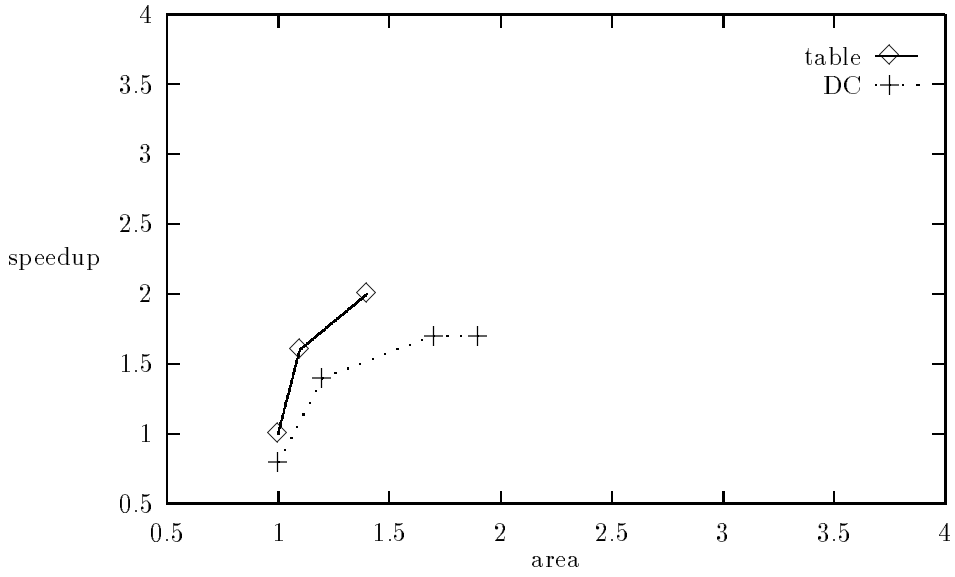

Figure 8.10: Comparison of implementations.

8.2.3 Comparison of Quotient Digit Selection

The cycle times obtained by Ercegovac and Lang seem small compared to Obermann and Flynn. Assume that the delay for the radix-2 table is identical in both [14] and [5]. Since the logic needed to implement the table is very simple, this is a fair assumption. The relative delay and area are shown in Figures 8.11 and 8.12.

It is evident from these figures that direct comparison lies somewhere between the two table lookup results. This just goes to show that results can be very implementation dependent.

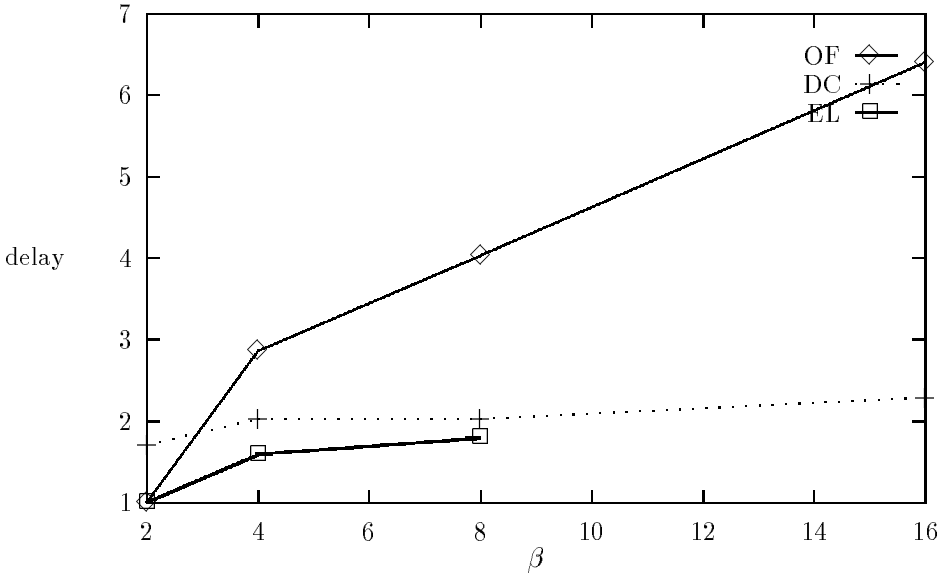

Figure 8.11: Comparison of relative quotient digit selection delay.

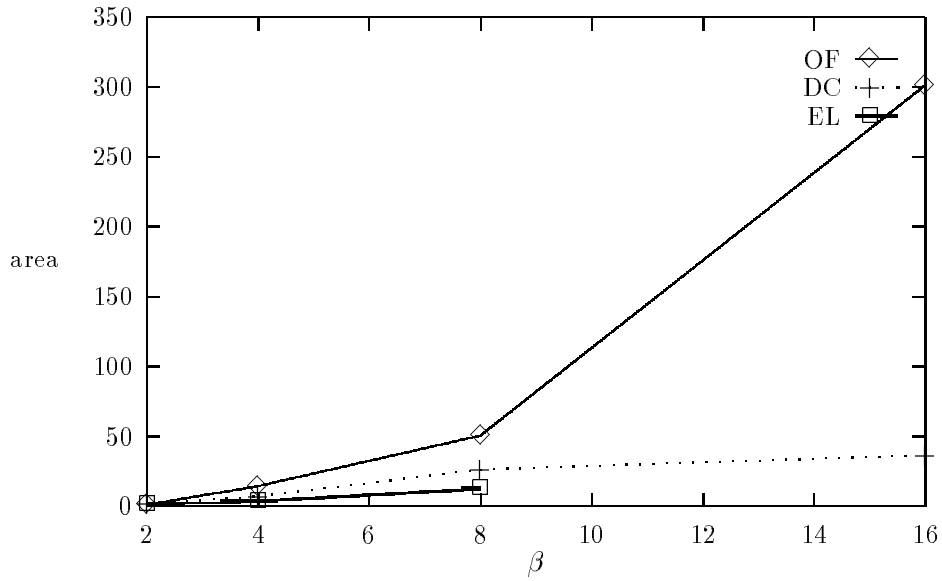

Figure 8.12: Comparison of relative quotient digit selection area.

Chapter 9

Conclusion

To conclude our investigation we summarize the results found in the previous chapters.

- The comparators have a better asymptotic performance than the tables, in terms of delay and area
- The implementations compared in Chapter 8 showed that the highly optimized tables were slightly faster than the unoptimized comparators. If we had the tools to optimize the comparators we this advantage would shrink, and probably turn the other way.
- The implementation of the comparators is far simpler than the lookup table This should not be underestimated, given the number of errors made in previous litterature, and the Pentium incident. This simplicity should also allow a chip designer to produce a design in less time.

Overall it is our opinion, that based on the results presented here demonstrates that direct comparison is a more than adequate alternative to the table lookup approach.

Future developments

- Direct comparison in conjuction with scaling, and very high radix.
- A more exact comparison could be achieved if we had used the same optimizations etc. as Ercegovac and Lang used for their implementations.
- Analysis of square root and shared implementations.

Appendix A

Robertson and Taylor Diagrams

The 'Robertson diagram' was introduced by J.Robertson in [15], as a means of illustrating 'arithmetic procedures during division'.

The 'Taylor diagram', a term coined by Williams and Horowitz [18], is named after G.Taylor based on the article [16]. These diagrams are used in this thesis several times, and their construction is therefore explained here.

A.1 Robertson Diagrams

The Robertson diagram is a useful tool for visualising digit selection functions, for both division and square root digit serial algorithms. Let β be the radix, and assume that we are using a digit set $\mathcal{D}$, and let the digit selection function be:

$$d_{i+1} = d\_select(R_i, Y), \tag{A.1}$$

where $Y = D$ in division and $Y = S_i$ for square root.

Assume that the recurrence is given by:

$$R_{i+1} = \beta R_i - d_{i+1} W, \tag{A.2}$$

and let the boundaries on R_{i+1} be $\underline{B}_{i+1}$ and $\overline{B}_{i+1}$, where $W = D$ for division and $W = 2S_i + d_{i+1}\beta^{i+1}$ for square root.

The Robertson diagram has as axes the shifted partial remainder βR_i, and the next remainder R_{i+1}. The selection functions is then plotted in as follows: For each digit in $\mathcal{D}$ we plot the line $R_{i+1} = \beta R_i - d_{i+1} W$.

For each possible value of βR_i, d_{i+1} is determined by moving up or down along the R_{i+1}-axis until we meet one of the straight lines, say $R_{i+1} = \beta R_i - dW$, the returned digit d_{i+1} is then d.

Note that depending on the digit set the function d_select may have more than one possible digit choice, however if we use a complete digit set, we always have at least one choice. The resolution of the possible digit choice conflicts, depends on the actual implementation. Also note that how one resolves the conflicts may depend on D. An example of a Robertson diagram is illustrated in figure A.1.

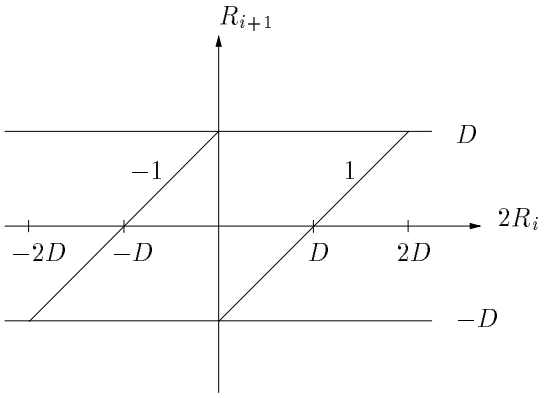

Figure A.1: Robertson diagram for binary non-restoring division.

A.2 Taylor Diagrams

Another useful instrument, used in the design of a digit selection function is the Taylor diagram, or PD-plot as it is known in the division case. The PD-plot has as axes the shifted partial remainder, βR_i and the divisor D. The digit selection intervals which are functions of D are then drawn in the diagram. These diagrams are sometimes drawn with the axes swapped. For historical reasons we shall draw them as is illustrated in figure A.2, showing a PD-plot for SRT division.

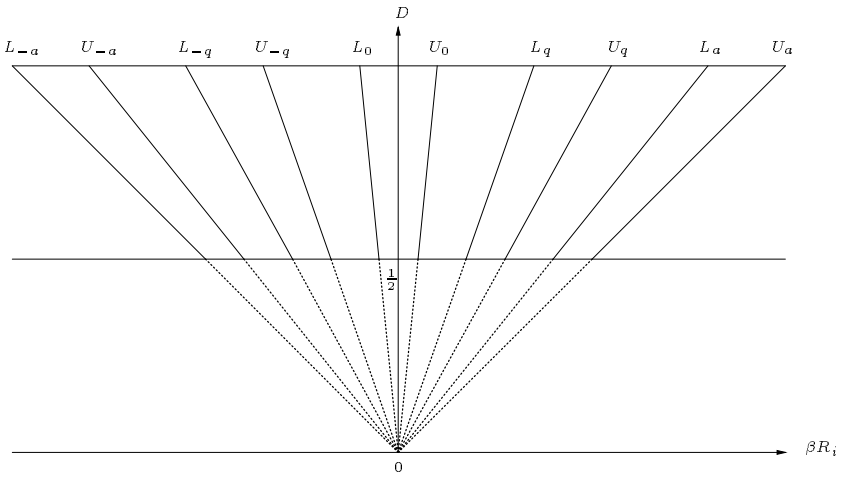

Figure A.2: Taylor diagram for SRT division.

Appendix B

Logic

We will use the same technology as in Appendix B of [5].

- 2-input NAND gates are used as units, for delay and area.
- Operands are 53-bit floating-point.
- Interconnections are not included.
- Delays within modules are the maximum of the delay for a high to low transition and a low to high transition. The delay of a number of gates is not just the sum of the delays of the individual gates.

B.1 Gates

gate	delay	area
NAND	1.0	1.0
NOT	1.0	1.0
XOR	2.2	2.2

B.2 Basic Modules

module	delay	area
2-1 MUX	1.4	3.0
2-1 MUX (FA)	2.0	2.3
3-1 MUX	1.6	4.4
4-1 MUX	1.8	5.6
6-1 MUX	2.0	6.6
8-1 MUX	2.2	7.4
Register (1-bit)	8.0	4.0
Buffer (1-bit)	1.8	2.6

The reason we have two different 2-1 MUX'es is that sometimes we optimize for speed and sometimes for area.

B.3 Complex Modules

3-2 CSA: A 3-2 CSA is just an array of full-adders. The delays and area for a full-adder are:

module	delay	area
full-adder		6.7
$a, b \rightarrow sum$	4.2	
$c_{in} \rightarrow sum$	2.2	
$a, b \rightarrow c_{out}$	3.8	
$c_{in} \rightarrow c_{out}$	2.0	

4-2 CSA: This takes two carry-save inputs and outputs one carry-save operand. Like the 3-2 CSA the outputs have different delays. For our purpose we only need the maximum delay of the adder.

module	delay	area
4-2 CSA	6.0	15.5

4-2 BSA: Although a 4-2 BSA is not included in [5] we shall assume that it is identical in delay and area to a 4-2 CSA.

module	delay	area
4-2 BSA	6.0	15.5

Signum-node: The signum node has two sign-digit input and one sign-digit output. The o-composition node given in Section 2.10.2 of [12] can be realized by a 4-2 MUX, which is equivalent to two 2-1 MUX'es.

module	delay	area
Signum-node	1.4	6.0

Comparator: As stated in Section 2.6 a Δ-bit comparison of an operand in redundant representation with a Δ-bit constant can be performed by first subtracting the constant from the redundant operand and then determining the sign of the resulting redundant operand. If the constant is in non-redundant form we can use a 3-2 adder, and if it is in redundant representation we can use a 4-2 adder.

Observation 2.10.14 in [12] states that the sign-determination can be performed using a tree of o-nodes. If the operand we wish to determine the sign of is in borrow-save, then we convert it to sign-digit, and feed the converted operand to the signum tree. This conversion can be realized through a single XOR gate. For carry-save we can do the same with the exception that we invert the most significant bits, after conversion to sign-digit. The output of the final signum-node is two bits in sign-digit encoding,

signifying one of the three relations $<, =, >$. A single signal giving the relation $\geq$ can be obtained by performing a NAND on the two output bits.

Observation: The number of signum nodes needed for a Δ-bit comparison is $\Delta - 1$.

Observation: The depth of a sign determination tree is: $\lceil \log_2(\Delta) \rceil$.

Observation: The delay of this sign-detection is less than the zero and sign-detection network in EL.

Encoder: The ouputs from the comparators is a string of $\geq$ signals, which have to be encoded in borrow-save, since the quotient digit q_{i+1} is needed in this representation. The logic needed to perform this encoding is implementation variant.

Convert and Round: The convert and round network perform the on-the-fly conversion and rounding of the quotient.

module	delay	area
Sign and Zero Detect	15	600
Convert and Round	13	1300

Sign and zero detection is only used for postcorrection and rounding, while convert and round is used in every iteration.

Lookup table: We shall not implement the tables in logic, but instead use the results obtained by other sources.

Bibliography

[1] E. Antelo, T. Lang, and J.D. Bruguera. Computation of $\sqrt{x/d}$ in a Very High Radix Combined Division/Square-Root Unit with Scaling and Selection by Rounding. *IEEE Transactions on Computers*, 47(2):152–161, Feb. 1998.

[2] D.E. Atkins. Higher-Radix Division Using Estimates of the Divisor and partial remainders. *IEEE Transactions on Computers*, C-17(10):925–934, Oct. 1968.

[3] R.E. Bryant. Bit-Level Analysis of an SRT Divider Circuit. Technical report, School of Computer Science, Carnegie Mellon University, 1995.

[4] N. Burgess and T. Williams. Choices of Operand Truncation in the SRT Algorithm. *IEEE Trans. Computers*, 44(7):933–938, July 1995.

[5] M.D. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, 1994.

[6] J. Fandrianto. Algorithms for High Speed Shared Radix 4 Division and Radix 4 Square Root. In *Proceedings of the 8th IEEE Symposium on Computer Arithmetic*, Como, Italy, 1987. IEEE Computer Society Press, Los Alamitos, CA.

[7] J. Fandrianto. Algorithms for High-Speed Shared Radix 8 Division and radix 8 Square-Root. In *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, Santa Monica, USA, September 1989. IEEE Computer Society Press, Los Alamitos, CA.

[8] H. Rueß, N. Shankar, and M.K. Srivas. Modular Verification of SRT Division. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 123–134, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.

[9] IEEE Standard for Binary Floating-Point Arithmetic, 1985.

[10] V. Kantabutra. A New algorithm for Division in Hardware. In *IEEE International Conference on Computer Design*, 1996.

[11] V. Kantabutra. A New Theory for High-Hadix Division in Hardware: Two Direct, Comparison-Based Radix-8 Cases, unpublished. 1997.

[12] Peter Kornerup. Manuscript of book in progress.

[13] S.F. Oberman and M.J. Flynn. An Analysis of Division Algorithms and Implementations. Technical Report CSL-TR-95-675, Computer Systems Laboratory, Stanford University, July 1995.

[14] S.F. Oberman and M.J. Flynn. Measuring the Complexity of SRT Tables. Technical Report CSL-TR-95-679, Computer Systems Laboratory, Stanford University, Nov. 1995.

[15] J.E. Robertson. A New Class of Digital Division Methods. *IRE Trans. Electronic Computers*, EC-7:218–222, Sept. 1958.

[16] G. S. Taylor. Compatible Hardware For Division and Square Root. In *Proceedings of the 5th Symposium on Computer Arithmetic*, pages 127–134. IEEE Computer Society Press, 1981.

[17] T.D. Tocher. Techniques of Multiplication and Division for Automatic Binary Computers. *Quarter. J. Mech. App. Math.*, 2:364–384, 1958.

[18] T.E. Williams and M. Horowitz. SRT Division Diagrams and Their Usage in Designing Custom Integrated Circuits for Division. Technical Report CSL-TR-87-326, Computer Systems Laboratory, Stanford University, Nov. 1986.

[19] J. Zurawski and J. Gosling. Design of a High-Speed Square Root Multiply and Divide Unit. *IEEE Transactions on Computers*, C-36(1):13–23, Jan. 1987.