Design of a Combined Unit for Reciprocal and Square Root Reciprocal

Yu Xiaofeng

Kongens Lyngby 2006
IMM - M.Sc. - 2006 - 45

Abstract

The reciprocal and square root reciprocal operations are important in several applications such as computer graphics and scientific computation. For the two operations, an algorithm that combines a digit-by-digit module and one iteration of the Newton-Raphson approximation is used. The latter is implemented by a digit-recurrence, which uses the digits produced by the digit-by-digit part. In this way, both parts execute in an overlapped manner, so that the total number of cycles is about half of the number that would be required by the digit-by-digit part alone. Since the approximation does not produce correct rounding in a few cases, for applications where exact rounding is required, the result is only computed by the digit-by-digit module. Radix-4 implementations for combined unit are described and have been synthesized. The result of the evaluation shows that the cycle time is the same as that of the digit-by-digit unit and that, as a consequence, the execution time is almost halved. Because of the approximation part, the area almost doubles of the digit-by-digit area. Finally, the layout of the combined unit has been created.

Preface

This thesis was prepared at Informatics and Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the M.Sc. degree in Computer System Engineering.

The thesis deals with the computation of double precision floating-point reciprocal and square root reciprocal. These two operations play an important role in the applications such as computer graphics and scientific computation. The objective of the work is to design a combined unit for the computation of reciprocal and square root reciprocal and implement it by using state-of-art tools and libraries of cells.

The main feature of the thesis is in providing a complete solution for the computation of reciprocal and square root reciprocal from algorithm to layout. The thesis consistently uses an algorithmic approach in presenting the design and implementation. To make the algorithm and design easy to read and understand, relevant theory and examples are given.

The thesis begins with a general introduction (Chapter 1). It then illustrates the algorithm (Chapter 2) to compute the reciprocal and square root reciprocal. The algorithms are introduced in accompany with the background theory and examples. Chapter 3 presented the complete architecture design based on the algorithm and explained in details using examples. The detailed diagram of the architecture has been provided with all the logic blocks and bit width. Reasons for design choice are explained.

Chapter 4 described the procedure of implementation and showed the synthesis results. The results are also evaluated in this chapter. At the end, chapter 5 summarized the work.

Acknowledgements

I thank many people in the Department of informatics and Mathematical Modelling who provide a lot of support and convenient to my thesis. In particular, I would like to give special thanks to my supervisor Alberto Nannarelli, I benefited greatly from his instructions, especially the discussions at the weekly project status meeting.

Yu Xiaofeng

April 29, 2006

Contents

Chapter 1
Introduction

Digital arithmetic operations are very important in the design of digital processors and application-specific systems. Among the arithmetic operations, the reciprocal and square root reciprocal operations are widely used in applications such as graphics and scientific computation. Therefore, different kind of schemes has been developed. There are several algorithms for the computation of reciprocal and square root reciprocal. Conventional algorithms include: 1) Digit-by-digit algorithms. 2) Quadratic convergent (Newton-Raphson) algorithms. 3) Polynomial approximations [1].

Digit-by-digit algorithms are based on a digit recurrence. In this method, the quotient of reciprocal computation is represented in a radix-r form and one digit of it is obtained per iteration. The digit recurrence involves a digit selection, a digit multiplication and an addition. The Newton-Raphson algorithms and polynomial approximations compute the reciprocal by iteratively improving an initial approximation. The most complex operation involved in the iteration is multiplication [2].

Newton-Raphson method and polynomial approximations have a quadratic convergence rate, which means that the number of bits of accuracy of the approximation doubles after each iteration, therefore, reduce the number of iterations. However, both Newton-Raphson method and polynomial approximations method require a dedicated parallel multiplier and additional tables. In contrast, digit-by-digit method has the advantage of low hardware overhead, but liner convergence with more iterations [1].

In this project, new algorithms are used, which combine a digit-by-digit part and one iteration of a quadratic convergence approximation [1]. The latter uses the digits produced by the former part, so that two parts can execute in parallel fashion, thus only half of the iterations are needed as compared to the digit-by-digit part alone. The execution time is reduced by implementing the approximation as digit-recurrence which is performed in an overlapped manner with the digit-by-digit part. This requires two data paths operating in parallel.

Radix-4 implementations have been done. This is based on the consideration of system complexity and execution time. High radix makes the implementation very complex. Since reciprocal and square root reciprocal operations have similar characteristics, it is considered advantageous to have a single scheme to implement both functions. The combined unit for both reciprocal and square root reciprocal are presented. This combined implementation needs a single selection function. The single selection function for radix-4 implementation has been developed by [3].

The comparison has been performed among different algorithms and different radix. The results show that the proposed new schemes have nearly the same cycle time as the corresponding digit-by-digit schemes. But the number of iterations is only half of the digit-by-digit schemes alone. The total delay is reduced roughly by half. From the evaluation, it can be concluded that the new algorithm implementation is a low latency solution with moderate hardware overhead.

The remainder of the paper is organized as following:

In Chapter 2, the floating point number system using the IEEE-754 standard is introduced at the beginning, in order to give a background theory. Then the algorithms for computation of reciprocal and square root reciprocal are described. The conventional digit-by-digit algorithm and Newton-Raphson approximation are introduced respectively, then the new algorithm which combined the

digit-recurrence and approximation together is described. The on-the-fly conversion and rounding algorithms are introduced as well, which is an important part for the combined unit implementation.

Chapter 3 presented the full architecture design. It started with the reciprocal architecture design, then the square root architecture design and finally, the combined unit architecture design. The quotient digit selection function is introduced in details in this chapter.

Chapter 4 described the procedure of implementation and showed the synthesis results. The critical path, area and power dissipation are estimated. The results are analyzed and evaluated. Chapter 5 gave a conclusion.

Chapter 2

Algorithm

This chapter gives a detailed description of digit-by-digit algorithm, Newton-Raphson approximation algorithm and new algorithm used in this work. The Floating Point Standard 754 is introduced firstly to provide a background theory.

2.1 IEEE Floating Point Standard 754 [4]

Since reciprocal and square root reciprocal computation require real numbers, the floating point representation is used in the design and implementation.

Parameters for representation

There are several parameters that define a floating point representation system.

- Sign S: One bit. S=1 if the number is negative.
- Magnitude (also called the significand): Represented in radix 2 with one integer bit.

$$1.F$$

Where F is fraction part with f bits and the most-significant 1 is the hidden bit.
The range of the (normalized) significand

$$1 \leq 1.F \leq 2 - 2^{-f}$$

- Exponent: base 2 and biased representation. The exponent field e, biased with bias

$$B = 2^{e-1} - 1$$

IEEE double precision standard

The double precision floating point number has 64 bits.

S	EEEEEEEEEEE	MMMMMM	. . .	MMMMMMMM
63	62 52	51		0

- First bit is the sign bit S.
- Next 11 bits are the exponent bits E.
- Final 52 bits are the significand, or magnitude M.

$$V = (-1)^S \cdot (1.M) \cdot 2^{E-1023} \quad \text{with } 0 < E < 2047$$

- Representation is normalized in $1.0 \leq 1.M < 2.0$. Normally, “one” (1.M) is implicit in representation.
- Exponent is biased (e.g. exp = 0 then E = 0 + 1023).

For single precision floating point representation (32 bits), S is 1 bit, E is 8 bits and F is 23 bits.

In this work, we use double precision floating point representation. During the implementation, the operands and result are in sign-and-magnitude representation. Only magnitudes are considered.

2.2 Digit-by-Digit Algorithm

The digit-by-digit algorithm is developed for division operation [2]. Naturally, it applies to the reciprocal and is the basis for square root reciprocal operation. The algorithm is based on digit recurrence method. In this method, the quotient is represented in a radix-r form and one digit of it is obtained per iteration. The digit-recurrence involves a digit selection, a digit multiplication, and an addition.

Operand and Result

The reciprocal operation is defined as

$$q = 1/d$$

Where dividend 1 and the divisor d are the operands and the result is the quotient q.
The operands and result are represented in double precision floating-point using the IEEE-754 standard, namely, with significand in the range [2,1) with a precision of 53 bits. The range of the quotient is

$$1 \leq q < 2$$

Corresponding to

$$½ \leq d < 1$$

To achieve the result in the specified range, the input significand d may be shifted within the interval [1/2,1). Correspondingly, the exponent has to be adjusted.

Define square root reciprocal as

$$P = 1/\sqrt{d}$$

Again, the range of the result

$$1 \leq p < 2$$

Correspondingly, the range of the operand

$$1/4 \leq d < 1$$

Digit-by-digit recurrence for reciprocal

The digit-by-digit algorithm consists of n iterations of a recurrence, in which each iteration produces one digit of the quotient. The value of the quotient after j iterations Q[j] is defined as

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

Where r is radix, Q[0] is determined by the initialization.
The residual (or partial remainder) w define as

$$w[j] = r^j(1 - dQ[j]) \qquad (2)$$

The expression for recurrence is

$$w[j+1] = rw[j] - q_{j+1}\, d \qquad (3)$$
$$q_{j+2} = SEL(rw[j+1], d) \qquad (4)$$

Expression (4) is the quotient-digit-selection function. It is used to select a suitable value of q_{j+1}. The digit q_{j+1} take values in the set

$$q_{j+1} \in \{-a, \ldots, -1, 0, 1, \ldots, a\}$$

$$\rho = a/r-1$$

where ρ is the redundancy factor.
The digit q_{j+1} is selected so that w[j+1]is bounded by

$$-\rho d \leq w[j] \leq \rho d$$

Quotient-digit-selection function is described in [2] in details. The scheme is shown in Figure 1.

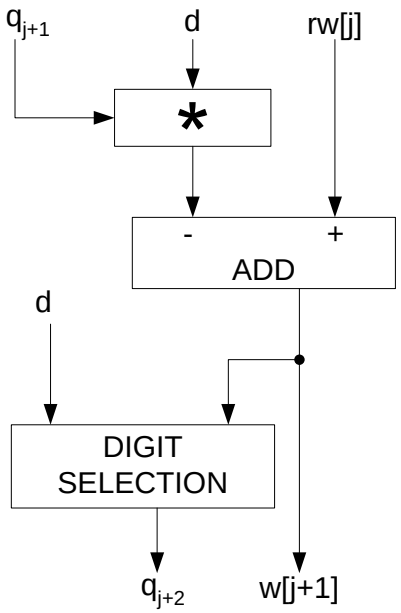

Figure 1 Scheme of digit-by-digit recurrence for reciprocal

Before recurrence, the initialization step is needed to initialize w[0] and Q[0] in order to assure convergence. Initialization will be introduced in chapter 3.

Digit-by-digit recurrence for Square Root Reciprocal [1][3][5][6]

For the square root reciprocal recurrence, the partial result up to iteration j, P[j] defines as

$$P[j] = P[0] + \sum_{i=1}^{j} p_i r^{-i} \qquad (5)$$

Where P[0] is initialization

The residual is defined as

$$w[j] = (1/2)\, r^{j} (1 - d\, p[j]^2) \qquad (6)$$

To simplify the description and the implementation, two variables are introduced:

$$D[j] = d\, p[j] \qquad (7)$$
$$C[j] = (1/2) r^{-(j+1)} d \qquad (8)$$

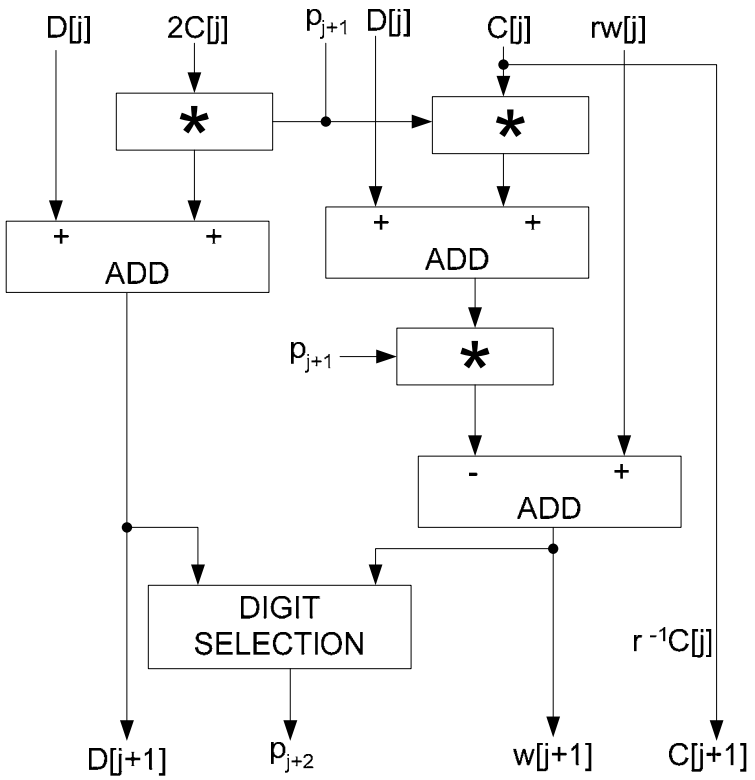

Figure 2 Scheme of digit-by-digit recurrence for square root reciprocal

Using these variables, the recurrence defined as

$$w[j+1] = rw[j] - p_{j+1} D[j] - p^2_{j+1} C[j] \quad (9)$$
$$D[j+1] = D[j] + 2p_{j+1} C[j] \quad (10)$$
$$C[j+1] = r^{-1} C[j] \quad (11)$$
$$p_{j+2} = SEL(rw[j+1], D[j+1]) \quad (12)$$

It is necessary to initialize W[0], D[0], C[0] and P[0] before recurrence, see chapter 3 for details. The digit selection function is developed in [3]. The scheme is shown in Figure 2.

2.3 Newton-Raphson Approximation

The Newton-Raphson method computes a function by iteratively improving an initial approximation. The most complex operation involved in the iteration is multiplication. This method has a quadratic convergence rate, which means that the number of bits of accuracy of the approximation doubles after each iteration. As a consequence, the number of iterations for a desired accuracy is smaller than for linear convergence (digit-by-digit). However, since full precision multiplications are involved, the delay of an iteration is larger.

Newton-Rahson method for reciprocal approximation [2]

The Newton-Rahson method illustrated here is the computation of the reciprocal function, since it is the basis for square root reciprocal.

The approximation is based on a general method to obtain the zero of a function. That is, the value of x for which f(x) = 0. If x[j] is an approximation of the zero, then a better approximation is

$$x[j+1] = x[j] - f(x[j]) / f'(x[j]) \quad (13)$$

where f '(x[j]) is the derivative of f(x) with respect to x, evaluated at x[j].

For the approximation of reciprocal,

$$f(R) = 1/R - d \text{ (whose zero is } 1/d)$$
$$f'(R) = -1/R^2$$

Apply to expression (13), the recurrence is obtained.

$$R[j+1] = R[j](2 - R[j]d) \quad (14)$$

The recurrence is initiated with an initial approximation R[0]. Each iteration requires two multiplications and one subtraction from the value 2. The scheme is shown in Figure 3.

The convergence of this method is quadratic, that is, if the error at step j is $\varepsilon[j]$, then the error at step j+1 is $\varepsilon[j]^2$. This can be shown as follows. Since R[j] is an approximation of 1/d, the relative

error is

$$\mathcal{E}[j] = 1 - dR[j]$$

Then from expression (14)

$$R[j+1] = (1 - \mathcal{E}[j]^2)/d$$
$$\mathcal{E}[j+1] = 1 - dR[j+1] = \mathcal{E}[j]^2 \qquad (15)$$

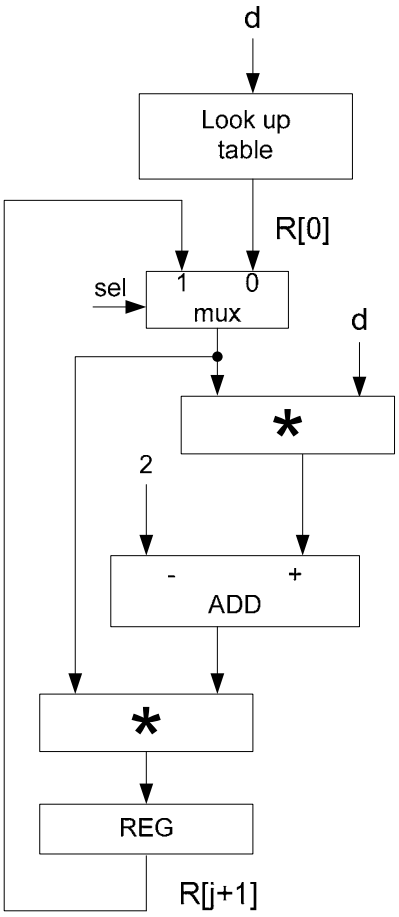

Figure 3 Scheme of Newton-Rahson method for reciprocal approximation

2.4 Proposed Algorithm [1]

The new algorithm used in this work consists of a digit-by-digit part followed by a Newton-Raphson approximation. Two parts operate in an overlapped manner to reduce the total delay. The scheme of new algorithm is illustrated in Figure 4. The algorithm consists two steps:

1. Module A (Digit-by-digit) obtains the approximation k using a digit recurrence and performing g iterations, roughly half of final required precision.

2. Module B (Newton-Raphson method) performs a better approximation by means of a digit recurrence.

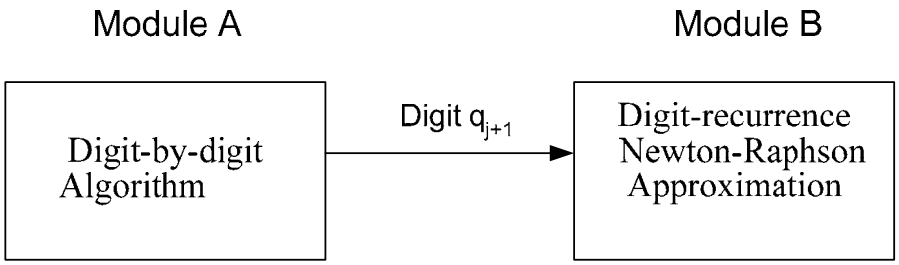

Figure 4 New algorithm scheme

The digit-by-digit algorithms are already given in section 2.2. This section introduces the approximation part.

Reciprocal approximation

The digit-by-digit part produces an approximation k(k=Q[g]), following by one Newton-Raphson iteration. As described by the Newton-Raphson method, a better approximation is given by

$$A = k(2 - kd) \qquad (16)$$

From expression (15), we know that the convergence of this iteration is quadratic, that is, if the error of k is δ, then the error of A is $\mathcal{E} = \delta^2$. This can be shown as follows. Since k is an approximation of 1/d, the relative error is

$$\delta = 1 - dk, \text{ and } k = (1 - \delta) / d$$

Applies to expression (16), then

$$A = (1 - \delta^2) / d$$

The relative error of A is

$$\mathcal{E} = 1 - dA = \delta^2$$

The approximation A is computed by means of a digit-recurrence to avoid multipliers and to speed up the computation. This approximation recurrence is overlapped with the computation of k.
From expression (16), the approximation recurrence can be defined as

$$A[j] = Q[j](2 - d\,Q[j]) \qquad (17)$$

To eliminate variable shifts, define

$$E[j] = r^{2j}A[j]$$

Then

$$E[j+1] - r^2E[j] = r^{2(j+1)}(Q[j+1](2 - dQ[j+1]) - Q[j](2 - dQ[j]))$$
$$E[j+1] - r^2E[j] = r^{2(j+1)}((Q[j] + q_{j+1}r^{-(j+1)})(2 - dQ[j+1]) - Q[j](2 - dQ[j]))$$

Which simplifies to

$$E[j+1] - r^2E[j] = q_{j+1}r^{j+1}\,(2 - dQ[j+1] - dQ[j])$$

As shown of expression (2),

$$w[j] = r^j(1 - dQ[j])$$
$$Q[j+1] = Q[j] + q_{j+1}r^{-(j+1)}$$

The expression for approximation recurrence is obtained as

$$E[j+1] = r^2E[j] + q_{j+1}(2rw[j] - q_{j+1}\,d) \qquad (18)$$

Where w[j] and $-\,q_{j+1}\,d$ are computed by the digit-by-digit recurrences.

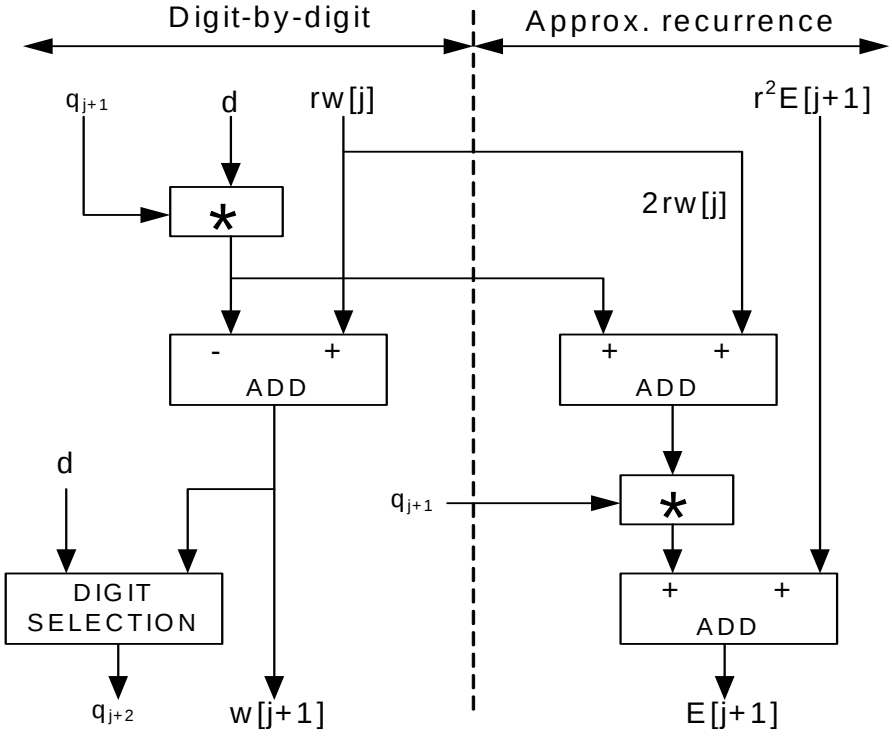

Figure 5 Scheme for reciprocal algorithm

Figure 5 shows the scheme of reciprocal algorithm. The digit-by-digit recurrence and approximation recurrence are in overlapped manner. After g iterations, g is roughly half of the iterations as compared to use the digit-by-digit algorithm alone, the final result E[g] obtained.

Square Root Reciprocal approximation

Similarly to the case for reciprocal, the square root reciprocal approximation is defined as

$$B = k(3/2 - k^2 d/2) \qquad (19)$$

Again, in this case the convergence is quadratic. If δ and $\mathcal{E}$ are the relative error of k and B respectively, then

$$\delta = 1 - k\sqrt{d} \quad \text{and} \quad \mathcal{E} = 1 - B\sqrt{d}$$

Therefore,

$$k\sqrt{d} = 1 - \delta \text{ and } \mathcal{E} = 1 - (k\sqrt{d}/2)(3 - dk^2)$$

Consequently,

$$\mathcal{E} = 1 - \tfrac{1}{2}(1 - \delta)(3 - (1 - \delta)^2) = (\delta^2/2)(3 - \delta)$$

Then the relative error of B has a complexity of $Q(\delta^2)$.

Since k = P[g] (g is the total number of iterations), similar to the case for reciprocal, we define

$$H[j] = r^{2j}P[j](3/2 - dP^2[j]/2) = r^{2j}P[j](1 + r^{-j}w[j])$$

So that $B = r^{-2g}P[g]$. The recurrence for H[j] obtained by

$$H[j+1] = r^2H[j] + r^{2(j+1)}[(P[j] + p_{j+1}r^{-(j+1)}) \cdot (1 + r^{-(j+1)}w[j+1]) - P[j](1 + r^{-j}w[j])]$$

Using the recurrence given in (9) results in

$$H[j+1] = r^2H[j] + p_{j+1}(2rw[j] + w[j+1] - p_{j+1}D[j]/2) \qquad (20)$$

The initial condition is

$$H[0] = P[0](3/2 - dP^2[0]/2)$$

The scheme of square root reciprocal algorithm is shown in Figure 6.

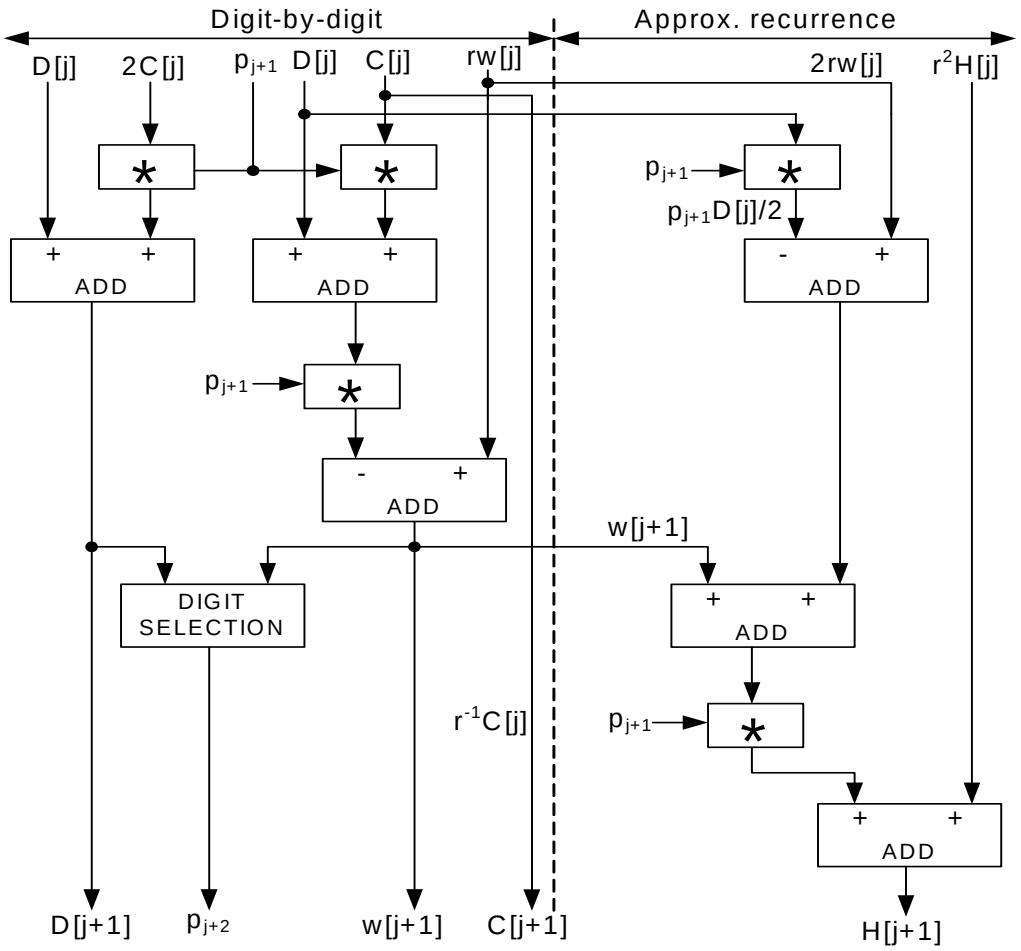

Figure 6 Scheme for square root reciprocal algorithm

2.5 On-the-fly Conversion and Rounding [2]

The on-the-fly conversion is to convert quotient from signed-digit representation to conventional representation. A simple alternative is to do an addition step after the quotient is completely computed. However, this addition would increase the delay. The on-the-fly conversion algorithm performs the conversion as the digits are produced. The on-the-fly conversion algorithm for digit-by-digit part can be expressed as following [2]:

Let Q[j] be the digit vector of the converted quotient consisting of the j most-significant digits, that is,

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

Then obtain

$$Q[j+1] = Q[j] + q_{i+1} r^{-(i+1)}$$

Since q_{j+1} can be negative, express algorithm as

$$Q[j+1] = Q[j] + q_{i+1}r^{-(i+1)} \quad \text{if } q_{i+1} \geq 0$$
$$Q[j+1] = Q[j] - r^{-i} + (r - |q_{i+1}|)r^{-(i+1)} \quad \text{if } q_{i+1} < 0$$

This algorithm has the disadvantage that the subtraction $Q[j] - r^{-j}$ requires the propagation of a borrow and, therefore, is slow. To avoid this propagation, another form is defined as

$$QM[j] = Q[j] - r^{-i}$$

Using this form, the conversion algorithm becomes

$$Q[j+1] = (Q[j], q_{i+1}) \quad \text{if } q_{i+1} \geq 0$$
$$Q[j+1] = (QM[j], (r - |q_{i+1}|)) \quad \text{if } q_{i+1} < 0$$

$$QM[j+1] = (Q[j], q_{i+1} - 1) \quad \text{if } q_{i+1} > 0$$
$$QM[j+1] = (QM[j], ((r-1) - |q_{i+1}|)) \quad \text{if } q_{i+1} \leq 0$$

So that the subtraction is replaced by loading the form QM[j], then no carry/borrow is propagated. The form QM[j] also needs to be updated as shown in above expression. An example is given in Figure 7 to illustrate the on-the-fly conversion algorithm.

j	q_j	Q	QM
1	1	x x x x x x x x 1	x x x x x x x x 0
2	2	x x x x x x x 1 2	x x x x x x x 1 1
3	0	x x x x x x 1 2 0	x x x x x x 1 1 3
4	-1	x x x x x 1 1 3 3	x x x x x 1 1 3 2
5	0	x x x x 1 1 3 3 0	x x x x 1 1 3 2 3
6	0	x x x 1 1 3 3 0 0	x x x 1 1 3 2 3 3
7	2	x x 1 1 3 3 0 0 2	x x 1 1 3 3 0 0 1
8	-2	x 1 1 3 3 0 0 1 2	x 1 1 3 3 0 0 1 1
9	-1	1 1 3 3 0 0 1 1 3	1 1 3 3 0 0 1 1 2

Figure 7 Example of radix-4 on-the-fly conversion

To perform the on-the-fly conversion and round, three registers are needed to store Q, QM, QP as shown in Figure 8, However, when the rounding is done in the least-significant position, and a< r -1, two registers Q and QM are sufficient. These registers are shifted one digit left each cycle, while new least significant digits are inserted into registers, depending on the value of q_j . Registers Q

and QM are updated each iteration by the following rules:

$$
\begin{aligned}
&Q[j] <= (shl(Q[j-1]), q_j) && \text{if } q_j > 0\\
&QM[j] <= (shl(Q[j-1]), q_j - 1)\\
&Q[j] <= (shl(Q[j-1]), 0) && \text{if } q_j = 0\\
&QM[j] <= (shl(QM[j-1]), r - 1)\\
&Q[j] <= (shl(QM[j-1]), r - |q_j|) && \text{if } q_j < 0\\
&QM[j] <= (shl(QM[j-1]), (r-1) - |q_j|)
\end{aligned}
$$

For example, $Q[j] <= (shl(Q[j-1]), q_j)$ means that the register Q at iteration j is shifted one digit left and the last digit is loaded with q_j. In QM, the current digit is given by $q_j - 1 \pmod r$.

Register QP is updated by the following rules:

$$
\begin{aligned}
&QP[j] <= (shl(QP[j-1]), 0) && \text{if } q_j = r - 1\\
&QP[j] <= (shl(Q[j-1]), q_j + 1) && \text{if } -1 \le q_j \le r-2\\
&QP[j] <= (shl(QM[j-1]), r - |q_j| + 1) && \text{if } q_j < -1
\end{aligned}
$$

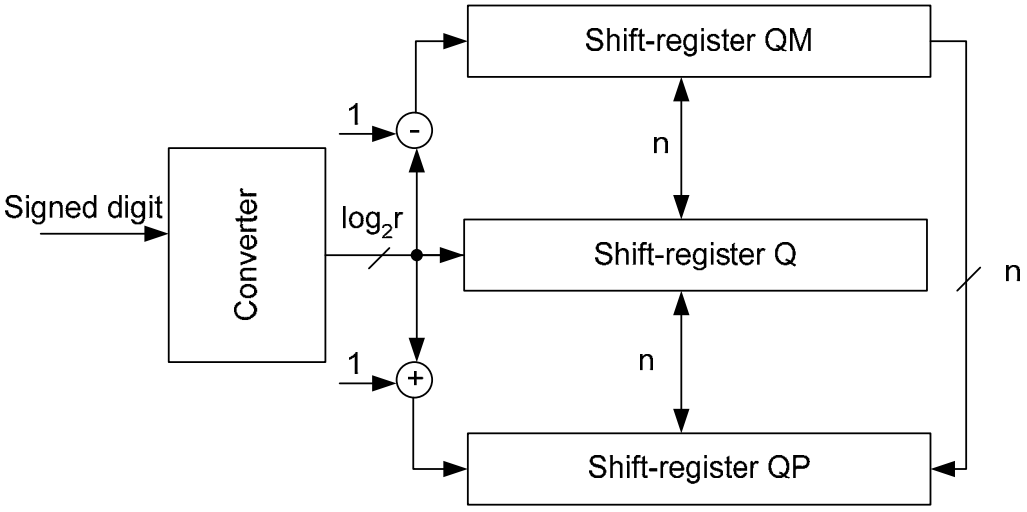

Figure 8 On-the-fly conversion and round unit

Table 1 Value of p in the rounding step

q_m	SIGN	ZERO	p	case
r - 1	0	0	0	1
	0	1	0	1
	1	-	r - 1	2
$0 \le q_m \le r-1$	0	0	$g_m + 1$	2
	0	1	$g_m + G1$	2
	1	-	g_m	2
-1	0	-	0	2
	1	-	r - 1	3
$q_m < -1$	0	0	$g_m + 1$	3
	0	1	$g_m + G1$	3
	1	-	g_m	3

The rounding is performed in the last cycle. First the quotient digit q_m is converted into $g_m = q_m \pmod r$. Then the rounded digit p is computed according to Table 1, where SIGN = 0 if the final residual is positive, ZERO = 1 if it is zero, and G1(guard bit) represents the bit before the least significant in g_m. Two operations are performed in the rounding step:

1. If the remainder is negative, the quotient must be decremented by 1 (in rounding position).

2. To round-to-the-nearest 1 has always to be added in rounding position.

Finally, the quotient is obtained by

$$q = \mathrm{Shl}(QP[m-1], p_t) \quad \text{if case 1}$$
$$q = \mathrm{Shl}(Q[m-1], p_t) \quad \text{if case 2}$$
$$q = \mathrm{Shl}(QM[m-1], p_t) \quad \text{if case 3}$$

where $p_t = [p/2]$.

The on-the-fly conversion for Newton-Raphson approximation part is similar to digit-by-digit part. Each iteration produces one digit (Radix-16) and store in the register. The register is shifted one digit left with insertion of new digit at least-significant position. Since approximation E[j] (or H[j] for combined unit) is in carry-save form, a short addition is needed to convert produced digit to the conventional representation. The digit is calculated in a redundant format with two extra leading bits (for combined unit, one extra bit is sufficient). The extra bits are used to judge if the digit produced by last cycle needs updating, since new iteration may generate carry to the previous digit. If the extra bits are different with the least significant bits of the previous digit, then we know the previous digit has to be updated (plus 1 or 2). The scheme is shown in Figure 9.

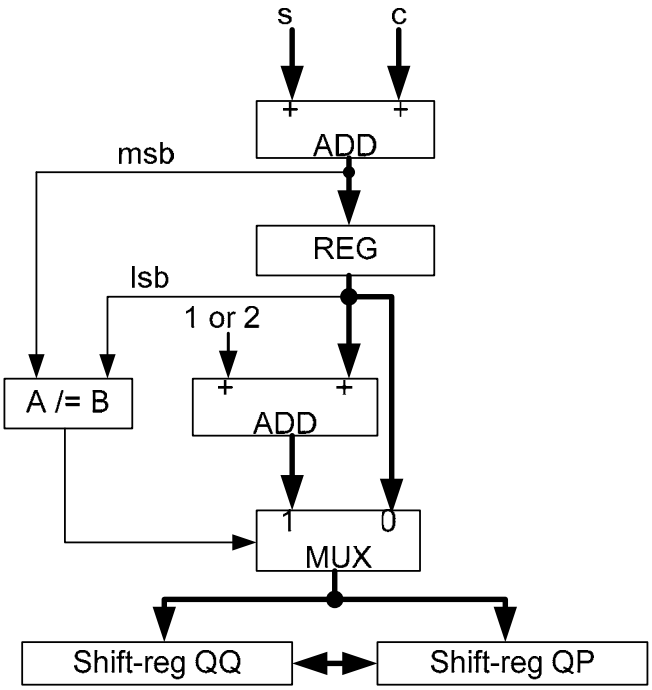

Figure 9 Scheme of on-the-fly conversion for approximation part.

The rounding of approximation part is done using the round-to-nearest scheme. However, a few cases can not be directly rounded. A new method is presented in [1] to obtain correctly rounded result. The method consists of two steps.

1. Determine if result obtained can be directly rounded. The result could be computed with some additional accuracy so that the probability of being able to do this rounding is high.

2. In the few cases, the rounding cannot be performed, then continue with the digit-recurrence until produce the rounding bit and the corresponding final residual.

This scheme leads to the latency increases for critical cases, but these cases have very low probability. Moreover, only little additional hardware is needed since the digit-by-digit hardware is available.

Chapter 3
Architecture

Based on the algorithms of chapter 2, the design of the Architecture can have several alternatives. The design choices mainly depend on several interrelated factors, such as radix, quotient-digit set and representation of the residual. These factors affect the execution time and the complexity of implementation.

High radix reduces the number of iterations. The number of iterations of the algorithm is reduced by a factor k when going from a radix r to a radix r^k. However, increase in radix produces a more complex implementation because of the quotient-digit selection and the generation of the divisor multiples.

The value of the redundancy factor (ρ) for quotient-digit set effects the complexity of the quotient-digit selection function and of the generation of the divisor multiples in an opposite manner: a higher ρ reduces complexity of the selection function but increases complexity of the generation of the divisor multiples [2].

For the representation of the residual, carry-save form has the big advantage that the addition is faster than the conventional two's complement representation. Its disadvantages are that it complicates somewhat the quotient-digit selection and that it increases the number of register bits required to store the residual.

This project uses radix-4 implementation, carry-save representation for the residual and the digit-set {-2, -1, 0, 1, 2}. This choice can achieve low latency with moderate hardware overhead.

3.1 Reciprocal Architecture Design

As indicated in charper 2, the digit-by-digit recurrence expression define as

$$w[j+1] = rw[j] - q_{j+1}\, d$$
$$q_{j+2} = SEL(rw[j+1], d)$$

The recurrences consist five steps [2]:

1. One digit arithmetic left shift of w[j] to produce rw[j].
2. Determination of the quotient digit q_{j+1} by the quotient-digit selection.
3. Generation of the divisor multiple $d \bullet q_{j+1}$.
4. Subtraction of $d \bullet q_{j+1}$ from rw[j].
5. Update of the quotient q[j] to q[j+1] by the on-the-fly conversion.

These steps result in an architecture shown in the left side of Figure 10. A multiplexer is used to select initial value w[0](w[0] = 1/4) and the shifted residual rw[j]. The quotient-digit selection function requires an estimation of rw[j+1]and d (described later). This estimation needs seven most significant bits of rw[j] and three bits of d. Since rw[j] is in carry-save form, a short adder is needed to convert to conventional representation. Because digit q takes values from the digit-set {-2, -1, 0, 1, 2}, generating of the divisor multiple $d \bullet q_{j+1}$ becomes easy. The digit multiplier can be implemented as a 4-1 multiplexers. It makes the multiplication fast and simple. A fast 3-2 carry-save adder is a natural choice for residual recurrence.

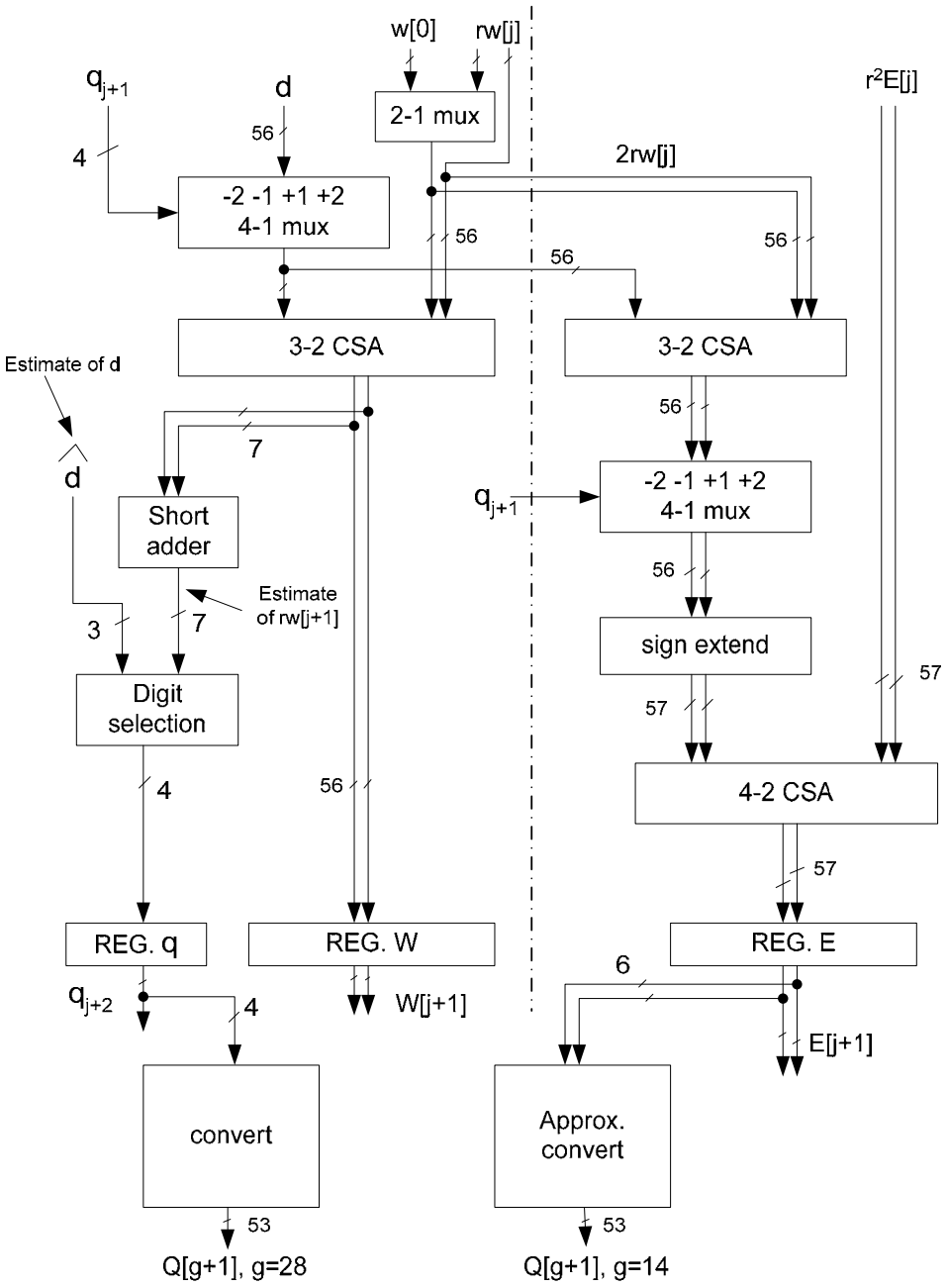

Figure 10 Architecture for reciprocal computation

According to the expression (18), the Newton-Raphson approximation recurrence is

$$E[j+1] = r^2E[j] + q_{j+1}(2rw[j] - q_{j+1}\, d)$$

The recurrences consist following steps:

1. Shift 1 bit left of rw[j]to produce 2rw[j].
2. Subtraction of $d \bullet q_{j+1}$ from 2rw[j].
3. Multiplication of digit q_{j+1} with the result of step 2.
4. Shift 4 bits left (since r = 4) of E[j] to generate $r^2E[j]$.
5. Addition the results of step 3 and step 4 to generate E[j+1].
6. On-the-fly conversion of E[j+1] to the conventional two's complement representation.

The right side of Figure 10 shows the architecture of Newton-Raphson approximation recurrence. Again, using a 3-2 carry-save adder for fast addition of 2rw[j] and $-d \bullet q_{j+1}$. The addition result multiplies digit q_{j+1} using a 4-1 multiplexer as well. But notice that the multiplication here is different with the multiplication in digit-by-digit part. Since the multiplication in digit-by-digit part generate negative $d \bullet q_{j+1}$, and it is possitive in approximation part. Thus the input q_{j+1} to the multiplication in approximation part should be inversed. Also notice that the bit width in approximation part is changed. Value of $-d \bullet q_{j+1}$ and 2rw[j] from digit-by-digit part, being the two inputs to the approximation part are 56 bits. While in approximation part, least 57 bits of E[j] needed for the iterations. It is necessary to do sign extension before the addition of recurrence. Naturally, the addition is performed by a 4-2 carry-save adder [7]. This architecture requires g+1 cycles for reciprocal computation. The first cycle performs the initialization and computes q_1. Cycle 2 to g corresponds to normal iterations and cycle g+1 is to finish the conversion and rounding.

3.2 Square Root Reciprocal Architecture Design

As shown by expression (9), (10), (11), (12), the digit-by-digit recurrences of square root reciprocal consist all five steps of reciprocal recurrences and plus following steps:

1. Update C[j] to C[j+1] by shift right 2 bits.
2. Shift C[j] left 1 bit to produce 2C[j] and multiply with digit $p_{j+1.}$
3. Add the result of step 2 with D[j] to perform the recurrence of D[j].
4. Multiplication to produce $p^2_{j+1}C[j]$ used for w[j] recurrence.

The architecture for the square root reciprocal is shown in Figure 11 (This architecture is also for the combined unit). Similar to reciprocal recurrence, Several 2-1 multiplexers are used for selection between the initial value and the updated value of w[0], C[0], D[0]. Digit multipliers are implemented as 4-1 multiplexers as before. Being different with reciprocal recurrence, there are multiplications by p^2_{j+1}. This multiplication can be simplified to the selection among the multiples 0, 1, 4, and implement as 4-1 multiplexers.

In this design, D[j] and C[j] are represented in conventional two's complement form, but rw[j] is represented in carry-save form. Therefore, a 4-2 carry-save adder is used for residual recurrence, and a fast carry-propagate adder is used to update the value of D[j].

The quotient-digit selection function requires an estimation of rw[j+1]and D[j+1]. Same as reciprocal recurrence, the estimation of rw[j+1] need seven most significant bits, but convert from carry-save form to conventional representation performed by a short adder. The estimate of D[j] is obtained by the addition of the most significant bits of $2p_{j+1}C[j]$ and D[j].

For the Newton-Raphson approximation part, even though the recurrence expression (20) is similar to the reciprocal, the implementation is not straightforward. Because we want to design the architecture to allow the combined implementation of reciprocal and square root reciprocal. We will discuss later, how this architecture is suitable for reciprocal computation. At the moment, the approximation recurrences consist following steps:

1. Shift rw[j] left 1 bit to produce 2rw[j] and multiply with digit $p_{j+1,}$ the multiplication is performed by a 4-1 multiplexer as before.
2. Shift 4 bits left (since r = 4) of H[j] to generate $r^2H[j]$.
3. Add the results of step 1 and step 2, which is performed by a 4-2 carry-save adder.
4. Multiplication to produce $-p^2_{j+1}D[j]/2$. The multiplication by p^2_{j+1} has the same structure as the 4-1 multiplexer (select multiples 0, 1, 4) used in the digit-by-digit recurrence part.
5. Add the result of step 4 and step 5, which is performed by a 3-2 carry-save adder.
6. w[j] multiplies with digit $p_{j+1,}$ the multiplication is performed by a 4-1 multiplexer.
7. Add the result of step 5 and step 6, which is performed by a 4-2 carry-save adder.

The sign of the digit p_{j+1} and the change of bit width should be taken attention as well. The result of the approximation H is converted on-the-fly. The architecture requires g+1 cycles. The first cycle performs the initialization and computes p_1. Cycle 2 to g corresponds to normal iterations and cycle g+1 is to finish the conversion and rounding. Digit-by-digit recurrence and approximation recurrence execute in overlapped manner. This results in 15 cycles for double precision float point computation. In contrast, a conventional digit-by-digit algorithm requires 28 cycles for double precision.

3.3 Combined Unit Architecture

The combined unit architecture is based on the square root reciprocal design. Since the square root reciprocal recurrence is different with reciprocal recurrence. To make it suitable for the reciprocal computation, following steps have to do:

1. Initialize C[0] = 0 for reciprocal computation . As a result, D[j] = d.
2. Reciprocal and square root reciprocal use a single selection function. This will be discussed in next section.
3. Reciprocal approximation recurrence E does not add the term w[j+1]. Therefore, an AND gate is

used to make $p_{j+1} = 0$ before the multiplexer that performs $p_{j+1}w[j]$, then the output of multiplexer will be zero when the reciprocal operation is performed.
4. Recurrence H requires the term $-p^2_{j+1}D[j] / 2$, while the reciprocal recurrence E needs $-p^2_{j+1}D[j]$, therefore, a 2-1 multiplexer is used to select between D[j]/2 and D[j] before multiplication. A control signal OP is used to decide if the operation is reciprocal or square root reciprocal.

Figure 11 shows the full architecture of combined unit.

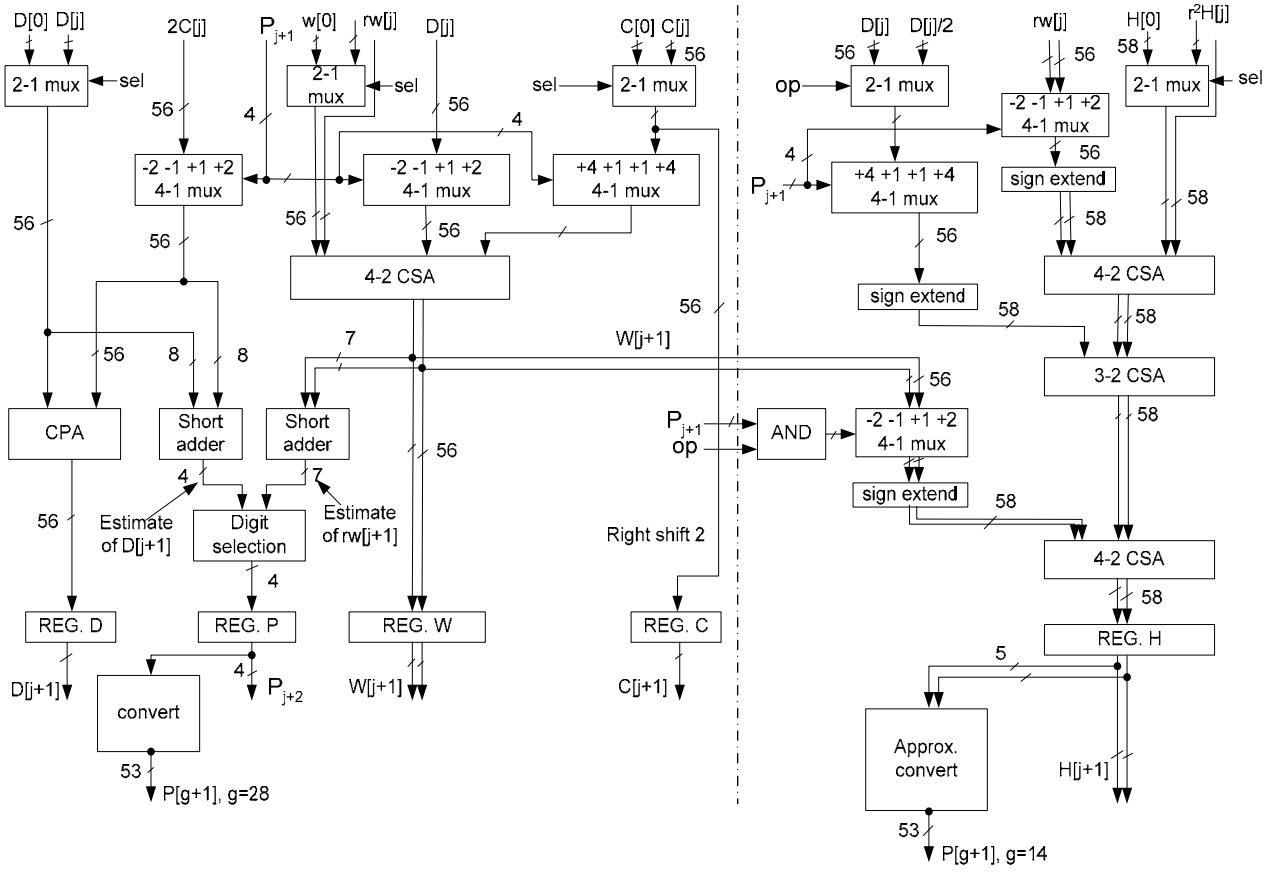

Figure 11. Architecture for reciprocal and square root reciprocal combined unit

3.4 Initialization

This section introduced the first step of implementation, initialization for both reciprocal unit and combined unit.

Initialization for reciprocal

There are several ways for the initialization as described in [2]. To assure convergence, we initialize:

$$ws[0] = 1/4$$
$$wc[0] = 0$$
$$Q[0] = 0$$
$$E[0] = 0$$

Initialization for combined unit [1]

To assure convergence and have the result within the interval [2,1), the following initializations are used:

For reciprocal computation,

$$Q[0] = 1 \text{ if } d \geq 0.75 \text{ else } 2$$
$$w[0] = 1 - dQ[0]$$
$$E[0] = Q[0](2 - dQ[0])$$

For square root reciprocal computation,

$$P[0] = 1 \text{ if } d \geq 0.5 \text{ else } 2$$
$$w[0] = (1/2)(1 - dP[0]^2)$$
$$D[0] = dP[0]$$
$$C[0] = (1/8)d$$
$$H[0] = (P[0]/2)(3 - dP[0]^2)$$

Table 2 lists the initial value for combined unit implementation. Note that in the initialization formula, d is in decimal format. In practice, d as operand is represented in sign-and-magnitude format. Therefore, shifting is needed when initial values go to implementation.

Table 2 Initialization for combined unit implementation

Reciprocal computation		Square root reciprocal computation	
$d \geq 0.75$	Q[0] = 1 w[0] = 1 - d C[0] = 0 D[0] = d E[0] = 2 - d	$d \geq 0.5$	P[0] = 1 w[0] = (1/2)(1 - d) D[0] = d C[0] = (1/8)d H[0] = (1/2)(3 - d)
$d < 0.75$	Q[0] = 2 w[0] = 1 - 2d C[0] = 0 D[0] = d E[0] = 2(2 - 2d)	$d < 0.5$	P[0] = 2 w[0] = (1/2)(1 - 4d) D[0] = 2d C[0] = (1/8)d H[0] = 3 - 4d

3.5 Selection Function

This section introduced the quotient digit selection function for radix-4 implementation. The residual is in carry-save form, represented by the sum ws and stored-carry wc. The qotient digit set is {-2, -1, 0, 1, 2 }.

Selection function for reciprocal [2]

The quotient-digit selection depends on the truncated carry-save shifted residual $\hat{y}$ and the truncated divisor $\hat{d}$ in terms of selection constants $m_k(i)$ so that

$$q_{j+1} = k \text{ if } m_k(i) \le \hat{y} < m_{k+1}(i)$$

where

- $i = 16\,\hat{d}$: $\hat{d}$ is the divisor truncated to the fourth fractional bit
- $\hat{y}$ is 4w[j] in carry-save form and truncated to the fourth fractional bit.

The selection constants are given by the following table:

Table 3 Selection constants for reciprocal

i	8	9	10	11	12	13	14	15
m_{-2} (i)	12	14	15	16	18	20	20	24
m_1(i)	4	4	4	4	6	6	8	8
m_0(i)	-4	-6	-6	-6	-8	-8	-8	-8
m_{-1}(i)	-13	-15	-16	-18	-20	-20	-22	-24

* real value = shown value/16 (fractional numbers)

Figure 12 shows an example of how the digit q_{j+1} is selected for division computation, naturlly, the method applies to the reciprocal as well. The calculation is based on the recurrence expression (3). We suppose divisor:

$$d = (0.11000101)_2$$

then

$$i = 16(0.1100)_2 = 12$$

```
4ws[0] =   000.10101111
4wc[0] =   000.00000001        ŷ[0] = 10/16     q1 = 1
  -q1d =    11.00111010
-------------------------------------------------------------------------
 ws[1] =     1.10010100
 wc[1] =     0.01010110
-------------------------------------------------------------------------
4ws[1] =   110.01010000
4wc[1] =   001.01011000        ŷ[1] = -6/16     q2 = 0
  -q2d =    00.00000000
-------------------------------------------------------------------------
 ws[2] =     1.00001000
 wc[2] =     0.10100000
-------------------------------------------------------------------------
4ws[2] =   100.00100000
4wc[2] =   010.10000000        ŷ[2] = -22/16    q3 = -2
  -q3d =    01.10001010
-------------------------------------------------------------------------
 w[3] =      0.00101010
```

Figure 12 Example of quotient digit selection.

Selection function for combined unit [3]

The single selection function for both reciprocal and square root reciprocal was developed in [3]. The quotient-digit selection depends on the truncated carry-save shifted residual $\widehat{rw[j]}$ and the truncated D[j], that is

$$p_{j+1} = SEL(\ \widehat{rw[j]},\ \widehat{D[j]}\ )$$

where

- $\widehat{D[j]}$ is D[j] truncated to the fifth fractional bits.
- $\widehat{rw[j]}$ is the seven most significant bits of 4w[j].

The selection constants are given by table 4:

Table 4 Selection constants for reciprocal and square root reciprocal

$D_L(i)$*	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
m_{-1}	-13	-14	-14	-15	-16	-17	-17	-18	-18	-19	-21	-21	-21	-23	-24	-24
m_0	-5	-5	-5	-6	-6	-6	-7	-7	-7	-8	-8	-8	-9	-9	-9	-10
m_1	3	4	4	4	4	4	4	5	7	7	7	7	8	8	8	8
m_2	12	13	14	14	15	15	16	17	18	18	19	19	22	22	22	22

*If D_L < 16/32(> 31/32) saturate to 16/32 (31/32).
*$D_L(i)$ scaled by 32 and m_k scaled by 16.

The selection is performed as follows:

$$p_{j+1} = -2 \quad \text{if} \quad \widehat{4w} < m_{-1}(i)$$

$$p_{j+1} = k\ (-1 \le k \le 1) \quad \text{if} \quad m_k(i) \le \widehat{4w} \le m_{k+1}(i)$$

$$p_{j+1} = 2 \quad \text{if} \quad \widehat{4w} \ge m_2(i)$$

Figure 13 shows an example of how the digit p_{j+1} is selected for square root reciprocal computation. The calculation is based on the recurrence expression (9), (10) and (11). We suppose:

$$d = (0.0101001010000000)_2$$

d < 0.5, then P[0] = 2. As indicated by Table 2 and Table 4, the initialization and digit selection are shown below:

Initaialzation and computation of p_1

ws[0] = 1.1101101100000000 D[0] = 0.1010010100000000 $\widehat{D[j]}$ = 20/32, $\widehat{4w}$ = -10/16

wc[0] = 0.0000000000000000 C[0] = 0.0000101001010000 p_1 = -1, P = $(1.3)_4$

Iteration j = 0

ws[1] = 0.0000011010101111 D[1] = 0.1001000001100000 $\widehat{D[j]}$ = 18/32, $\widehat{4w}$ = 1/16,

wc[1] = 0.0000000000000001 C[1] = 0.0000001010010100 p_2 = 0, P = $(1.30)_4$

Iteration j = 1

ws[2] = 0.0001101010110000 D[2] = 0.1001000001100000 $\widehat{D[j]}$ = 18/32, $\widehat{4w}$ = 6/16,

wc[2] = 0.0000000000010000 C[2] = 0.0000000010100101 p_3 = 1, P = $(1.301)_4$

Iteration j = 2

ws[3] = 1.1101100111000100 D[3] = 0.1001000110101010 $\widehat{D[j]}$= 18/32, $\widehat{4w}$ = -11/16,

wc[3] = 0.0000000000110111 C[3] = 0.0000000000101001 p_4 = -1

Figure 13 Example of square root reciprocal computation

Chapter 4
Implementation and Results

This chapter introduces the implementation procedure and shows the results. The results are evaluated and compared with other implementation schemes.

The design of reciprocal unit and combined unit are described in VHDL. The VHDL descriptions were put into Synopsys for synthesis. The synthesis uses the 0.18 μm standard cell library. From synthesis, the critical path, area and power dissipation are estimated. The synthesis results are summarized in Table 5, Table 6 and Table 7.

The layout was created by using SoC encounter. The gate-level netlist was imported into Encounter and doing Place and Route. After creating the layout, the delays are calculated for each wire (routes). The timing information is saved in .sdf file, and can be imported to the Synopsys again to evaluate the impact of the interconnections on the cycle time. The delay and area after layout are list in table 8.

The implementation procedure can be summarized as following:

1. Write VHDL description of the design.
2. Synthesis by using Synopsys. Save the design in Verilog format.
3. Import the Verilog file into the SoC Encounter for layout.
4. Calculate the delay of the layout and generate the timing information.
5. Read the design in Synopsys and Import the timing information of layout.
6. Analyze the timing information and generate the timing report.

The procedure is illustrated in Figure 14.

Table 5 Critical path of reciprocal unit.

QLATCH	4-1 multiplexer	CSA32	Selection	Critical path (ns)
0.46	0.19	0.17	0.74	1.56

Table 6 Critical path of combined unit

Control	2-1 multiplexer	4-1 multiplexer	CSA42	Short adder	Selection	QLATCH	Critical path (ns)
0.40	0.14	0.09	0.36	0.47	0.40	0.11	1.97

Table 7 Area and power dissipation for reciprocal and combined unit

Unit	Area (μm^2)	Power(mw)
Reciprocal unit	183197.703125	29.3092
Combined unit	328405.000000	41.8629

Table 8 Comparison of delay before and after layout

Unit	Before layout (ns)	After layout (ns)	Difference (%)
reciprocal	1.56	1.68	7.7
Combined unit	1.97	2.42	22.8

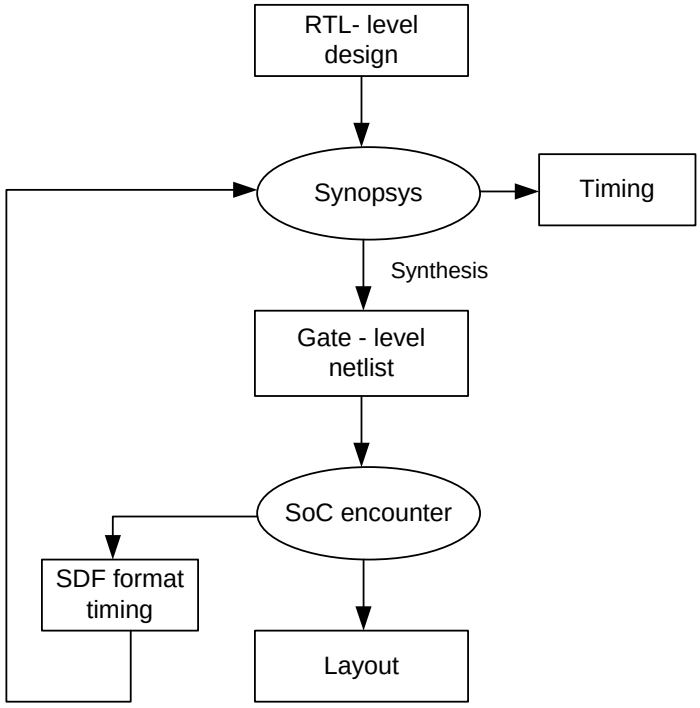

Figure 14 Procedure of Implementation

Table 5 and Table 6 show that for both reciprocal and square root reciprocal operation, the critical path corresponds to the digit-by-digit part. This means, the proposed new schemes have the same cycle time as the corresponding digit-by-digit schemes. But the number of iterations is only half of the digit-by-digit schemes alone. The total delay is reduced roughly by half.

Since the approximation part has roughly the same complexity as the conventional digit-by-digit part, the area of the proposed schemes increase the hardware overhead by nearly 2 with respect to the corresponding digit-by-digit schemes.

For reciprocal, the area-time ratios of higher radix schemes with respect to the radix-4 scheme are described in [2]. Therefore, we can compare the design in proposed scheme with more instances in an area-time space. Figure 15 shows the area-time ratios for digit-by-digit radix-4, radix-16 using overlapped radix-4 and very high radix-512 schemes in comparison with the proposed design.

For the square root reciprocal, there are several implementations using radix-4 and very high radix described in [3] an [5]. A radix-16 implementation with overlapped radix-4 iterations would be significantly more complex, because of the many conditional forms required, so we do not consider it. The scheme and implementation proposed by [3] has good area-delay figures, so take [3] as the

radix-4 design reference. For the very high radix implementation, assume a cost in area 1.5 times (a conservative figure) the cost of a very high radix reciprocal unit with the same cycle time. Figure 15 also shows the ratios corresponding to the square-root reciprocal, using the area and delay of the conventional radix-4 reciprocal unit as a reference. From the estimations, it can be concluded that the proposed designs introduce attractive points in the area-time space.

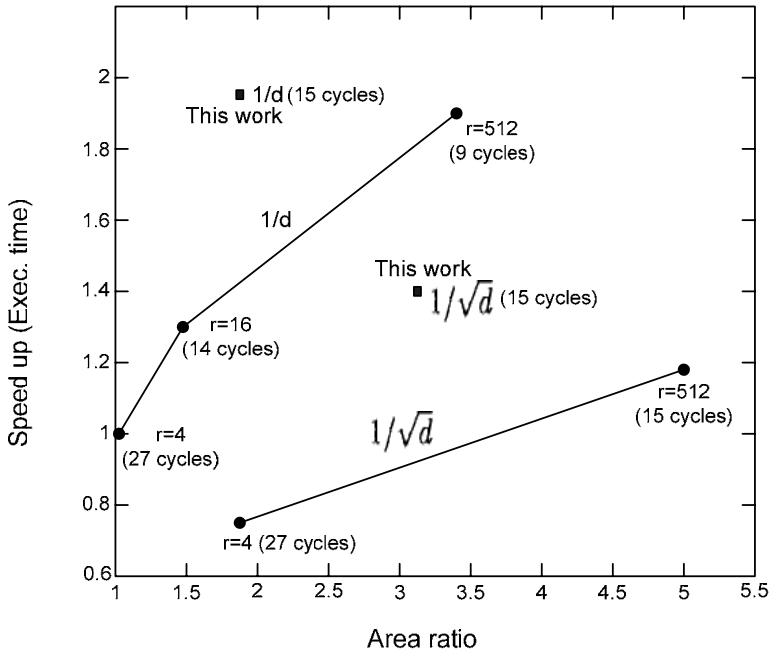

Figure 15 Area-time space.

Chapter5

Conclusions

In this project, the entire process of design a combined unit for reciprocal and square root reciprocal are presented. A new algorithm is used, which combined digit-by-digit algorithm and Newton-Raphson approximation. The approximation part is performed by a digit recurrence using the digit produced by the digit-by-digit part. In this way, two parts can execute in an overlapped manner. Consequently, only half of the iterations are needed as compared to the conventional digit-recurrence algorithm.

For the design of reciprocal unit and combined unit, the radix-4 implementation is used and the residual are represented in carry-save form. This choice can achieve low latency with moderate hardware overhead. To make the architecture suitable for both reciprocal and square root reciprocal computation, a signal quotient digit selection function is used which is developed in [3].

The design was synthesized using the 0.18 μm standard cell library. From synthesis, the critical path, area and power dissipation are estimated. The synthesis results show that the cycle time is as same as that of the conventional digit-by-digit unit. Since the number of iterations is nearly half of the digit-by-digit method alone, the total execution time is almost reduced by half. On the other hand, the addition of the approximation part almost doubles the required area.

Since the proposed design produces four bits per cycle, for reciprocal, it is about 50% faster than a radix-16 implementation with overlapped radix-4 stages with an increase of 20% in area.The evaluation shows that the propose design shows the good figure in area-time space.

References

[1] E. Antelo, P. Montuschi, T. Lang, A. Nannarelli, "Low Latency Digit-Recurrence Reciprocal and Square-Root Reciprocal Algorithm and Architecture", in Proceedings - 17th IEEE Symposium on Computer Arithmetic, ARITH-17 2005. pp. 147-154.

[2] M. Ercegovac and T. Lang, "Digital Arithmetic", Morgan Kaufmann Publishers, 2003.

[3] T. Lang and E. Antelo, "Radix-4 Reciprocal Square-root and Its Combination with Division and Square Root", IEEE Trans. On Comput., vol. 52, no 9, Sept. 2003, pp. 1100-1114.

[4] D. A. Patterson and J. L. Hennessy, "Computer Organization and Design – the hardware/software interface", Morgan Kaufmann Publishers Inc., 2nd edition, 1998.

[5] 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 Trans. On Computers, vol. 47, no. 2, Feb. 1998, pp. 152-161.

[6] N. Takagi, "A Hardware Algorithm for Computing Reciprocal Square Root", in Proc. 15th IEEE Symposium on Computer Arithmetic, pp. 94-100, 2001.

[7] A. Weinberger, "4 – 2 carry-save adder module", IBM Technical Disclosure Bulletin, vol. 23, no. 8, Jan. 1981, pp. 3811 – 3814.

Appendices

A

A.1 VHDL files included in the reciprocal unit design

reciprocal.vhd, this is the top level design.
control.vhd
convert.vhd
convert_app.vhd
cpa.vhd
cr_qcalc.vhd
csa32LSBs.vhd
csa42.vhd
digit_compute.vhd
digit_update2.vhd
extend.vhd
gl_csa32.vhd
gl_dualreg_ld.vhd
gl_mux21.vhd
gl_reg.vhd
mult2.vhd
q_regs.vhd
qds_adder.vhd
qds_table.vhd
qdsel.vhd
qlatch.vhd
quotient.vhd
rounding.vhd
sdet.vhd
shifter.vhd
tb2_reciprocal.vhd

Files used by both reciprocal and combined unit are listed as following, these files are not shown in A.2, but shown in B.2.

cpa.vhd
csa42.vhd
gl_csa32.vhd
gl_dualreg_ld.vhd
gl_mux21.vhd
gl_reg.vhd
mult2.vhd
qlatch.vhd

A.2 VHDL code for reciprocal unit

-- VHDL code for reciprocal unit: RECIPROCAL.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity RECIPROCAL is
      Port (   CLOCK : In    std_logic;
                   D : In    std_logic_vector (52 downto 0);
               RESET : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
end RECIPROCAL;

architecture SCHEMATIC of RECIPROCAL is

   signal       Y1 : std_logic_vector(6 downto 0);
   signal       D3 : std_logic_vector(2 downto 0);
   signal       DD : std_logic_vector(56 downto 0);
   signal       Y2 : std_logic_vector(6 downto 0);
   signal      SW2 : std_logic_vector(56 downto 0);
   signal      SW1 : std_logic_vector(56 downto 0);
   signal       W2 : std_logic_vector(56 downto 0);
   signal       W1 : std_logic_vector(56 downto 0);
   signal       WC : std_logic_vector(56 downto 0);
   signal       WS : std_logic_vector(56 downto 0);
   signal      MXW : std_logic_vector(56 downto 0);
   signal     CSWS : std_logic_vector(55 downto 0);
   signal     CSWC : std_logic_vector(55 downto 0);
   signal    CS4ES : std_logic_vector(56 downto 0);
   signal    CS4EC : std_logic_vector(56 downto 0);
   signal     CS1S : std_logic_vector(55 downto 0);
   signal       DQ : std_logic_vector(52 downto 0);
   signal       EE : std_logic_vector(56 downto 0);
   signal     srws : std_logic_vector(55 downto 0);
   signal     srwc : std_logic_vector(55 downto 0);
   signal     muwc : std_logic_vector(55 downto 0);
   signal     muws : std_logic_vector(55 downto 0);
   signal       EC : std_logic_vector(56 downto 0);
   signal     ES : std_logic_vector(56 downto 0);
   signal     E1 : std_logic_vector(56 downto 0);
   signal     E2 : std_logic_vector(56 downto 0);
   signal     SEC : std_logic_vector(56 downto 0);
   signal     SES : std_logic_vector(56 downto 0);
   signal     SN_12 : std_logic;
   signal     CN_12 : std_logic;
   signal     NM1, NM2, NP1, NP2  : std_logic;
   signal     MXLB : std_logic_vector(56 downto 0);
   signal       XX : std_logic_vector(56 downto 0);
   signal      CS1 : std_logic_vector(56 downto 0);
   signal       QJ : std_logic_vector(3 downto 0);
   signal    ROUND : std_logic;
   signal    DIGIT : std_logic;
   signal     N_20 : std_logic;
   signal     N_23 : std_logic;
   signal     N_24 : std_logic;
   signal     SIGN : std_logic;
   signal     N_12 : std_logic;
   signal carry_ex : std_logic;
   signal       P2 : std_logic;
   signal       P1 : std_logic;
   signal       M1 : std_logic;
   signal       M2 : std_logic;
   signal iM1, iM2, iP1, iP2  : std_logic;
   signal     GND : std_logic;

   component gl_csa32
        GENERIC(n : integer);
        Port (  A : In std_logic_vector (n downto 0);
```

```
             B : In std_logic_vector (n downto 0);
             C : In std_logic_vector (n downto 0);
             Cin : In std_logic;
             Z : Out std_logic_vector (n downto 0);
             Y : Out std_logic_vector (n downto 0) );
end component;

component gl_dualreg_ld
  GENERIC(n : integer);
   Port (      AS : In    std_logic_vector (n downto 0);
               AC : In    std_logic_vector (n downto 0);
            RESET : In    std_logic;
            CLOCK : In    std_logic;
             LOAD : In    std_logic;
               ZS : Out   std_logic_vector (n downto 0);
               ZC : Out   std_logic_vector (n downto 0) );
end component;

component QLATCH
   Port (   CLEAR : In    std_logic;
              CLK : In    std_logic;
               I1 : In    std_logic;
               I2 : In    std_logic;
               J1 : In    std_logic;
               J2 : In    std_logic;
             LOAD : In    std_logic;
               M1 : Out   std_logic;
               M2 : Out   std_logic;
               P1 : Out   std_logic;
               P2 : Out   std_logic );
end component;

component MULT2
   GENERIC(n : integer);
   Port (       A : In    std_logic_vector (n downto 0);
                Ap : In    std_logic; -- A(-1)
               M1 : In    std_logic;
               M2 : In    std_logic;
               P1 : In    std_logic;
               P2 : In    std_logic;
             COUT : Out   std_logic;
                Z : Out   std_logic_vector (n downto 0) );
end component;

component CONTROL
   Port (   CLOCK : In    std_logic;
            RESET : In    std_logic;
              CL1 : Out   std_logic;
              DIGIT : Out   std_logic;
              LD1 : Out   std_logic;
              MX1 : Out   std_logic;
            ROUND : Out   std_logic );
end component;

component gl_mux21
     GENERIC(n : integer);
     Port (  A0 : In std_logic_vector (n downto 0);
             A1 : In std_logic_vector (n downto 0);
             SEL : In std_logic;
             Z : Out std_logic_vector (n downto 0) );
end component;

component  csa32LSBs
     GENERIC(n : integer);
     Port (  A : In std_logic_vector (n downto 0);
             B : In std_logic_vector (n downto 0);
             C : In std_logic_vector (n downto 0);
             Cin : In std_logic;
             Cout : Out std_logic;
             Z : Out std_logic_vector (n downto 0);
             Y : Out std_logic_vector (n downto 0) );
end component;

component extend
```

```
        Port (A : in std_logic_vector (55 downto 0);
                  B : in std_logic_vector (55 downto 0);
                  Y : out std_logic_vector (56 downto 0);
              Z : out std_logic_vector (56 downto 0));
   end component;

   component csa42
        GENERIC(n : integer);
        Port (  A : In std_logic_vector (n downto 0);
                B : In std_logic_vector (n downto 0);
                C : In std_logic_vector (n downto 0);
                D : In std_logic_vector (n downto 0);
                SCin : In std_logic;
                CCin : In std_logic;
                Z : Out std_logic_vector (n downto 0);
                Y : Out std_logic_vector (n downto 0) );
   end component;

   component QDSEL
      Port (      A1 : In    std_logic_vector (6 downto 0);
                  A2 : In    std_logic_vector (6 downto 0);
                   D : In    std_logic_vector (2 downto 0);
                  M1 : Out   std_logic;
                  M2 : Out   std_logic;
                  P1 : Out   std_logic;
                  P2 : Out   std_logic );
   end component;

   component SDET
      Port (       A : In    std_logic_vector (56 downto 0);
                   B : In    std_logic_vector (56 downto 0);
                   Z : Out   std_logic );
   end component;
   component CONVERT
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               DIGIT : In    std_logic;
                  M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
               ROUND : In    std_logic;
                SIGN : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
   end component;

   component shifter
      Port ( A : In    std_logic_vector (56 downto 0);
             B : Out   std_logic_vector (55 downto 0));
   end component;

   component CONVERT_app
      Port (CLEAR : In    std_logic;
              CLK : In    std_logic;
            round : In    std_logic;
               E1 : In    std_logic_vector(5 downto 0);
               E2 : In    std_logic_vector(5 downto 0);
                Q : Out   std_logic_vector(52 downto 0) );
   end component;

begin
----------Digit-by-digit part----------
   GND <='0';
   XX(56 downto 52) <= "00001";--++
   XX(51 downto 0) <=(others => '0');
   D3(2 downto 0)<=D(51 downto 49);
   DD(56)<='0'; DD(55)<='0'; DD(54 downto 2)<=D(52 downto 0);
   DD(1)<='0'; DD(0)<='0';

   I_CTRL : CONTROL
      Port Map ( CLOCK=>CLOCK, RESET=>RESET, CL1=>N_23, DIGIT=>DIGIT,
                 LD1=>N_24, MX1=>N_20, ROUND=>ROUND );
   -- shift 2 left
   MXW(56 downto 2)<=W1(54 downto 0) ; MXW(1)<='0'; MXW(0)<='0';
```

```
  SW2(56 downto 2)<=W2(54 downto 0) ; SW2(1)<='0'; SW2(0)<='0';

  I_MUXm : gl_mux21 Generic Map(n=>8)
     Port Map ( A0=>MXW(56 downto 48), SEL=>N_20,
                A1=>XX(56 downto 48),
                Z=>SW1(56 downto 48) );

  I_MUXl : gl_mux21 Generic Map(n=>47)
     Port Map ( A0=>MXW(47 downto 0), SEL=>N_20,
                A1=>XX(47 downto 0),
                Z=>SW1(47 downto 0) );

  I_MULTm : MULT2 Generic Map(n=>8)
     Port Map ( A=>DD(56 downto 48), Ap=>DD(47),
                M1=>M1, M2=>M2, P1=>P1, P2=>P2, COUT=>N_12,
                Z=>CS1(56 downto 48) );
  I_MULTl : MULT2 Generic Map(n=>47)
     Port Map ( A=>DD(47 downto 0), Ap=>GND,
                   M1=>M1, M2=>M2, P1=>P1, P2=>P2, COUT=>N_12,
                Z=>CS1(47 downto 0) );

  I_CSAm : gl_csa32 Generic Map(n=>8)
     Port Map ( A=>SW1(56 downto 48),
                B=>SW2(56 downto 48), CIN=>carry_ex,
                C=>CS1(56 downto 48),
                Y=>WC(56 downto 48),
                Z=>WS(56 downto 48) );
  I_CSAl : csa32LSBs Generic Map(n=>47)
     Port Map ( A=>SW1(47 downto 0),
                B=>SW2(47 downto 0), CIN=>N_12,
                C=>CS1(47 downto 0), Cout=>carry_ex,
                Y=>WC(47 downto 0),
                Z=>WS(47 downto 0) );
----------Digit-recurrence linear approximation part----------
--shift 1 left to generate 2rw[j]----
   srws(55 downto 1)<=w1(54 downto 0); srws(0)<='0';
   srwc(55 downto 1)<=w2(54 downto 0); srwc(0)<='0';
 --shift 4 left to generate 16E[j]
   SES(56 downto 4)<=E1(52 downto 0); SES(3 downto 0)<= "0000";
   SEC(56 downto 4)<=E2(52 downto 0); SEC(3 downto 0)<= "0000";

 I_shifter : shifter
  Port Map ( A=>CS1,
             B=>CS1S);

 I_CSA32 : gl_csa32 Generic Map(n=>55)
     Port Map ( A=>srws,
                B=>srwc, CIN=>N_12,
                C=>CS1S,
                Y=>muwc(55 downto 0),
                Z=>muws(55 downto 0));
     NM2<=P2;
     NM1<=P1;
     NP1<=M1;
     NP2<=M2;

  I_MULTws : MULT2 Generic Map(n=>55)
     Port Map ( A=>muws, Ap=>GND,
                    M1=>NM1, M2=>NM2, P1=>NP1, P2=>NP2,
                    COUT=>SN_12,
                Z=>CSWS);

  I_MULTwc : MULT2 Generic Map(n=>55)
     Port Map ( A=>muwc, Ap=>GND,
                    M1=>NM1, M2=>NM2, P1=>NP1, P2=>NP2,
                    COUT=>CN_12,
                Z=>CSWC);

  I_extend : extend
     Port Map (A => CSWS,
               B => CSWC,
               Y => CS4EC,
                   Z => CS4ES);
```

```
     I_CSA42 : csa42 Generic Map(n=>56)
      Port Map ( A=>SES,
                 B=>SEC,
                 SCin=>SN_12,
                CCin=>CN_12,
                 C=>CS4ES,
                 D=>CS4EC,
                 Y=>EC,
                 Z=>ES);

    I_regE : gl_dualreg_ld Generic Map(n=>56)
      Port Map ( AS=>ES, AC=>EC,
                 RESET=>N_23, CLOCK=>CLOCK, LOAD=>N_24,
                 ZS=>E1, ZC=>E2);
    I_convert_app :CONVERT_APP
      Port Map (CLEAR=>N_23, CLK=>CLOCK,
                E1=>E1(56 downto 51), E2=>E2(56 downto 51),
                ROUND=>ROUND, --SIGN=>SIGN,
                Q(52 downto 0)=>Q(52 downto 0)  );
   ----------End of linear approximation part----------
   Y1(6 downto 0)<=Ws(55 downto 49);
   Y2(6 downto 0)<=Wc(55 downto 49);

   I_regWm : gl_dualreg_ld Generic Map(n=>10)
      Port Map ( AS=>WS(56 downto 46), AC=>WC(56 downto 46),
                 RESET=>N_23, CLOCK=>CLOCK, LOAD=>N_24,
                 ZS=>W1(56 downto 46), ZC=>W2(56 downto 46) );
   I_regWl : gl_dualreg_ld Generic Map(n=>45)
      Port Map ( AS=>WS(45 downto 0), AC=>WC(45 downto 0),
                 RESET=>N_23, CLOCK=>CLOCK, LOAD=>N_24,
                 ZS=>W1(45 downto 0), ZC=>W2(45 downto 0) );

   I_SEL : QDSEL
      Port Map ( A1(6 downto 0)=>Y1(6 downto 0),
                 A2(6 downto 0)=>Y2(6 downto 0),
                 D(2 downto 0)=>D3(2 downto 0), M1=>iM1, M2=>iM2,
                 P1=>iP1, P2=>iP2 );
   I_regQ : QLATCH
      Port Map ( CLEAR=>reset, CLK=>CLOCK, LOAD=>N_24,
                     I1=>iM1, I2=>iM2, J1=>iP1, J2=>iP2,
                     M1=>M1,  M2=>M2,  P1=>P1,  P2=>P2 );
   I_SDET : SDET
      Port Map ( A(56 downto 0)=>W2(56 downto 0),
                 B(56 downto 0)=>W1(56 downto 0), Z=>SIGN );
   I_7 : CONVERT
      Port Map ( CLEAR=>N_23, CLK=>CLOCK, DIGIT=>DIGIT, M1=>M1,
                 M2=>M2, P1=>P1, P2=>P2, ROUND=>ROUND, SIGN=>SIGN,
                 Q(52 downto 0)=>DQ(52 downto 0) );
  end SCHEMATIC;

configuration CFG_RECIPROCAL_SCHEMATIC of RECIPROCAL is

   for SCHEMATIC
      for I_shifter: shifter
         use configuration WORK.CFG_shifter_BEHAVIORAL;
      end for;
           for I_SDET: SDET
         use configuration WORK.CFG_SDET_BEHAVIORAL;
      end for;
      for I_CTRL: CONTROL
         use configuration WORK.CFG_CONTROL_BEHAVIORAL;
      end for;
      for I_MUXm, I_MUXl : gl_mux21
         use configuration WORK.CFG_gl_mux21_BEHAVIORAL;
      end for;
      for I_regQ: QLATCH
         use configuration WORK.CFG_QLATCH_BEHAVIORAL;
      end for;
      for I_extend: extend
         use configuration WORK.CFG_extend_BEHAVIORAL;
      end for;
      for I_CSA42 : csa42
         use configuration WORK.CFG_csa42_BEHAVIORAL;
      end for;
```

```
      for I_regWm, I_regWl, I_regE: gl_dualreg_ld
         use configuration WORK.CFG_gl_dualreg_ld_BEHAVIORAL;
      end for;
      for I_CSAl: csa32LSBs
         use configuration WORK.CFG_csa32LSBs_BEHAVIORAL;
      end for;
      for I_CSAm, I_CSA32 : gl_csa32
         use configuration WORK.CFG_GL_CSA32_BEHAVIORAL;
      end for;
      for I_SEL: QDSEL
         use configuration WORK.CFG_QDSEL_SCHEMATIC;
      end for;
      for I_MULTm, I_MULTl, I_MULTws, I_MULTwc: MULT2
         use configuration WORK.CFG_MULT2_BEHAVIORAL;
      end for;
      for I_7: convert
         use configuration WORK.CFG_convert_SCHEMATIC;
      end for;
      for I_convert_app: convert_app
         use configuration WORK.CFG_convert_app_SCHEMATIC;
      end for;
   end for;

end CFG_RECIPROCAL_SCHEMATIC;
```

-- VHDL code for reciprocal unit: control.vhd

```
library IEEE;

   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity CONTROL is
        Port ( CLOCK : In    std_logic;
               RESET : In    std_logic;
                 CL1 : Out   std_logic;
               DIGIT : Out   std_logic;
                 LD1 : Out   std_logic;
                 MX1 : Out   std_logic;
               ROUND : Out   std_logic);
end CONTROL;

architecture BEHAVIORAL of CONTROL is

   begin
   process(reset,clock)
   variable state : integer range 0 to 17;

     begin

        if ( reset = '0' ) then
          CL1 <= '0';
          LD1 <= '0';
          DIGIT <= '0' ;
          ROUND <= '1' ;
          MX1 <= '1' ;
          state := 0;

        elsif ((clock'EVENT) AND ( clock = '1' )) then

             if( 0 = state ) then
              ROUND <= '0' ;
                  CL1 <= '1';
                    LD1 <= '1';
                state  := 2 ;

             elsif( 1 = state ) then
                    CL1 <= '1';
                ROUND <= '0' ;
                state  := 2 ;

             elsif( 2 = state ) then
                 DIGIT <= '1' ;
```

```
                    MX1 <= '0' ;
                state  := 3 ;

            elsif( 17 = state ) then
                DIGIT <= '0' ;
                ROUND <= '1' ;
                    MX1 <= '1' ;
                    state  := 1 ;

            else
                state  := state + 1 ;
            end if;
        end if;
    end process;
end BEHAVIORAL;

configuration CFG_CONTROL_BEHAVIORAL of CONTROL is
   for BEHAVIORAL

   end for;

end CFG_CONTROL_BEHAVIORAL;
```

-- VHDL code for reciprocal unit: convert.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity CONVERT is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               DIGIT : In    std_logic;
                  M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
               ROUND : In    std_logic;
                SIGN : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
end CONVERT;

architecture SCHEMATIC of CONVERT is

   signal       QK : std_logic_vector(1 downto 0);
   signal       QN : std_logic_vector(55 downto 0);
   signal       QQ : std_logic_vector(55 downto 0);
   signal       QM : std_logic_vector(1 downto 0);
   signal       QP : std_logic_vector(55 downto 0);
   signal     QSIG : std_logic;
   signal      N_2 : std_logic;

   component CR_QCALC
      Port (      M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
                  QK : Out   std_logic_vector (1 downto 0);
                  QM : Out   std_logic_vector (1 downto 0);
                QSIG : Out   std_logic;
                ZERO : Out   std_logic );
   end component;

   component ROUNDING
      Port (     CLK : In    std_logic;
                  QN : In    std_logic_vector (55 downto 0);
                  QP : In    std_logic_vector (55 downto 0);
                  QQ : In    std_logic_vector (55 downto 0);
               ROUND : In    std_logic;
                SIGN : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
   end component;
```

```
   component QUOTIENT
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
                DET0 : In    std_logic;
               DIGIT : In    std_logic;
                  QK : In    std_logic_vector (1 downto 0);
                  QM : In    std_logic_vector (1 downto 0);
                QSIG : In    std_logic;
               ROUND : In    std_logic;
                  QN : InOut std_logic_vector (55 downto 0);
                  QP : InOut std_logic_vector (55 downto 0);
                  QQ : InOut std_logic_vector (55 downto 0) );
   end component;

begin

   I_10 : CR_QCALC
      Port Map ( M1=>M1, M2=>M2, P1=>P1, P2=>P2,
                 QK(1 downto 0)=>QK(1 downto 0),
                 QM(1 downto 0)=>QM(1 downto 0), QSIG=>QSIG, ZERO=>N_2 );
   I_7 : ROUNDING
      Port Map ( CLK=>CLK, QN(55 downto 0)=>QN(55 downto 0),
                 QP(55 downto 0)=>QP(55 downto 0),
                 QQ(55 downto 0)=>QQ(55 downto 0), ROUND=>ROUND,
                 SIGN=>SIGN, Q(52 downto 0)=>Q(52 downto 0) );
   I_8 : QUOTIENT
      Port Map ( CLEAR=>CLEAR, CLK=>CLK, DET0=>N_2, DIGIT=>DIGIT,
                 QK(1 downto 0)=>QK(1 downto 0),
                 QM(1 downto 0)=>QM(1 downto 0), QSIG=>QSIG,
                 ROUND=>ROUND, QN(55 downto 0)=>QN(55 downto 0),
                 QP(55 downto 0)=>QP(55 downto 0),
                 QQ(55 downto 0)=>QQ(55 downto 0) );

end SCHEMATIC;

configuration CFG_CONVERT_SCHEMATIC of CONVERT is

   for SCHEMATIC
      for I_10: CR_QCALC
         use configuration WORK.CFG_CR_QCALC_BEHAVIORAL;
      end for;
      for I_7: ROUNDING
         use configuration WORK.CFG_ROUNDING_BEHAVIORAL;
      end for;
      for I_8: QUOTIENT
         use configuration WORK.CFG_QUOTIENT_BEHAVIORAL;
      end for;
   end for;

end CFG_CONVERT_SCHEMATIC;
```

-- VHDL code for reciprocal unit: convert_app.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity CONVERT_APP is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               round : In    std_logic;
                  E1 : In    std_logic_vector(5 downto 0);
                  E2 : In    std_logic_vector(5 downto 0);
                   Q : Out   std_logic_vector(52 downto 0) );
end CONVERT_APP;

architecture SCHEMATIC of CONVERT_APP is

   CONSTANT n : integer := 5;
   signal        A,QS : std_logic_vector(3 downto 0);
   signal           T : std_logic;
   signal          QQ : std_logic_vector(55 downto 0);
   signal       B, Bn : std_logic_vector(n downto 0);
   signal     SIXTEEN : std_logic_vector(1 downto 0);
```

```
   component Q_regs
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               round : In    std_logic;
                   S : In    std_logic_vector (3 downto 0);
                   T : In    std_logic;
                  QK : In    std_logic_vector (3 downto 0);
                   Q : InOut   std_logic_vector (55 downto 0) );
   end component;

   component gl_reg is
        GENERIC(n : integer);
        Port (  A : In std_logic_vector (n downto 0);
                CLOCK : In std_logic;
                RESET : In std_logic;
                Z : Out std_logic_vector (n downto 0) );
   end component;

   component digit_update2 is
        GENERIC(n : integer);
        Port (  B : In std_logic_vector(n downto 0);
                QMS : In std_logic_vector(1 downto 0);
                A : Out std_logic_vector(3 downto 0);
                T : Out std_logic);
   end component;

   component digit_compute is
        GENERIC(n : integer);
        Port (  E1 : In std_logic_vector(n downto 0);
                E2 : In std_logic_vector(n downto 0);
                SS : Out std_logic_vector (3 downto 0);
                QK : Out std_logic_vector (n downto 0);
           SIXTEEN : Out std_logic_vector(1 downto 0) );
   end component;

begin

   I_1 : digit_compute Generic Map(n=>5)
         Port Map ( E1=>E1, E2=>E2,SS =>QS,
                         QK=>BN, SIXTEEN=>SIXTEEN );

   REG_B: gl_reg Generic Map(n=>5)
        Port Map ( A=>BN, CLOCK=>CLK, RESET=>CLEAR, Z=>B );

   I_2 : digit_update2 Generic Map(n=>5)
        Port Map ( B=>B, QMS=>SIXTEEN, A=>A, T=>T );

   I_8 : Q_regs
        Port Map ( CLEAR=>CLEAR, CLK=>CLK,round=>round, S=>QS,
                               T=>T, QK(3 downto 0)=>A(3 downto 0), Q=>QQ );

  Q(52 downto 0)<= QQ(52 downto 0);


end SCHEMATIC;

configuration CFG_CONVERT_APP_SCHEMATIC of CONVERT_APP is

   for SCHEMATIC
      for I_8: Q_regs
         use configuration WORK.CFG_Q_regs_BEHAVIORAL;
      end for;
      for REG_B: gl_reg
         use configuration WORK.CFG_gl_reg_BEHAVIORAL;
      end for;
      for I_1: digit_compute
         use configuration WORK.CFG_digit_compute_BEHAVIORAL;
      end for;
      for I_2: digit_update2
         use configuration WORK.CFG_digit_update2_BEHAVIORAL;
      end for;
   end for;

end CFG_CONVERT_APP_SCHEMATIC;
```

-- VHDL for reciprocal unit: cr_qcalc.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity CR_QCALC is
      Port (      M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
                  QK : Out   std_logic_vector (1 downto 0);
                  QM : Out   std_logic_vector (1 downto 0);
                QSIG : Out   std_logic;
                ZERO : Out   std_logic );
end CR_QCALC;

architecture BEHAVIORAL of CR_QCALC is

   begin
    process(M2,M1,P1,P2)
     begin

              if ( M2 = '1' ) then
                 qk <= "10" ;
                 qm <= "01" ;
                 qsig <= '0' ;
                 zero <= '0' ;

              elsif ( M1 = '1' ) then
                 qk <= "01" ;
                 qm <= "00" ;
                 qsig <= '0' ;
                 zero <= '0' ;

              elsif ( P1 = '1' ) then
                 qk <= "11" ;
                 qm <= "10" ;
                 qsig <= '1' ;
                 zero <= '0' ;

              elsif ( P2 = '1' ) then
                 qk <= "10" ;
                 qm <= "01" ;
                 qsig <= '1' ;
                 zero <= '0' ;

        else           -- qk = 0
                 qk <= "00" ;
                 qm <= "11" ;
                 qsig <= '0' ;
                 zero <= '1' ;
              end if;

   end process;

end BEHAVIORAL;

configuration CFG_CR_QCALC_BEHAVIORAL of CR_QCALC is
   for BEHAVIORAL

   end for;

end CFG_CR_QCALC_BEHAVIORAL;
```

-- VHDL for reciprocal unit: csa32LSBs.vhd

```
library IEEE;
               use IEEE.std_logic_1164.all;

entity csa32LSBs is
              GENERIC(n : integer);
              Port (       A : In std_logic_vector (n downto 0);
```

```
                           B : In std_logic_vector (n downto 0);
                           C : In std_logic_vector (n downto 0);
                         Cin : In std_logic;
                         Cout : Out std_logic;
                           Z : Out std_logic_vector (n downto 0);
                           Y : Out std_logic_vector (n downto 0) );
end csa32LSBs;

architecture BEHAVIORAL of csa32LSBs is

 begin
process(A, B, C, Cin)
              variable p : std_logic_vector (n downto 0) ;
              variable g : std_logic_vector (n downto 0) ;
              variable i : integer;
begin

for i in 0 to n loop
              p(i) := A(i) XOR B(i) ;
              g(i) := A(i) AND B(i) ;
end loop;

-- CARRY ----------------------------------
Y(0) <= Cin;
for i in 0 to n-1 loop
              Y(i+1) <= g(i) OR (c(i) AND p(i));
end loop;
Cout <= g(n) OR (c(n) AND p(n));

-- SUM ------------------------------------
for i in 0 to n loop
              Z(i) <= p(i) XOR c(i);
end loop;

end process;
end BEHAVIORAL;

configuration CFG_csa32LSBs_BEHAVIORAL of csa32LSBs is
               for BEHAVIORAL
               end for;
end CFG_csa32LSBs_BEHAVIORAL;
```

-- VHDL for reciprocal unit: digit_compute.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity digit_compute is
              GENERIC(n : integer); -- input bits
        Port (  E1 : In std_logic_vector(n downto 0);
                E2 : In std_logic_vector(n downto 0);
                SS : Out std_logic_vector (3 downto 0);
                QK : Out std_logic_vector (n downto 0);
                SIXTEEN : Out std_logic_vector (1 downto 0));
end digit_compute;

architecture BEHAVIORAL of digit_compute is

begin

process(E1, E2)
              variable p1 : std_logic_vector (n downto 0) ;
              variable p2 : std_logic_vector (n downto 0) ;
              variable  p : std_logic_vector (n downto 0) ;
              variable  g : std_logic_vector (n downto 0) ;
              variable  c : std_logic_vector (n downto 0) ;
              variable  s : std_logic_vector (n downto 0) ;
              variable a, b : std_logic;
              variable i : integer;

begin

c(0) := '0';
```

```
for i in 0 to n loop
        p(i) := E1(i) XOR E2(i) ;
        g(i) := E1(i) AND E2(i) ;
end loop;

-- CARRY (ripple) -------------------------
for i in 0 to n-1 loop
        c(i+1) := g(i) OR (c(i) AND p(i));
end loop;

-- SUM ------------------------------------
for i in 0 to n loop
  S(i) := p(i) XOR c(i);
end loop;
SIXTEEN <=s(5 downto 4);
qk <=S(5 downto 0);
SS <=S(3 downto 0);
end process;

end BEHAVIORAL;

configuration CFG_digit_compute_BEHAVIORAL of digit_compute is
   for BEHAVIORAL
   end for;
end CFG_digit_compute_BEHAVIORAL;
```

-- VHDL for reciprocal unit: digit_update2.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

entity digit_update2 is
    GENERIC(n : integer);
        Port (  B : In std_logic_vector(n downto 0);
                qms : In std_logic_vector(1 downto 0);
                A : Out std_logic_vector(3 downto 0);
                T : Out std_logic);--_vector(1 downto 0) );
end digit_update2;

architecture BEHAVIORAL of digit_update2 is

   begin
   process(B,qms)

     begin
         if(qms(0) /= B(0)) then
            A(3 downto 0) <= B(3 downto 0) + "0001";--+1
            T <= '1';
         elsif (qms(1) /= B(1)) then
            A(3 downto 0) <= B(3 downto 0) + "0010";--+2
            T <= '1';
         else
            A(3 downto 0) <= B(3 downto 0);
            T <= '0';
         end if;
   end process;

end BEHAVIORAL;

configuration CFG_digit_update2_BEHAVIORAL of digit_update2 is
   for BEHAVIORAL
   end for;
end CFG_digit_update2_BEHAVIORAL;
```

-- VHDL for reciprocal unit extend.vhd

```
library IEEE;
              use IEEE.std_logic_1164.all;

entity extend is
             Port (A : in std_logic_vector (55 downto 0);
```

```
                  B : in std_logic_vector (55 downto 0);
                  Y : out std_logic_vector (56 downto 0);
                Z : out std_logic_vector (56 downto 0));
end extend;

architecture BEHAVIORAL of extend is

 begin

   Z(55 downto 0)<= A(55 downto 0);
   Y(55 downto 0)<= B(55 downto 0);

   process(A, B)
   begin

       if (A(55) = '1' and B(55) = '1') then
           Y(56) <= '1';
           Z(56) <= '0';
       else
           if(A(55) = '1') then
              Z(56) <= '1';
           else
              Z(56) <= '0';
           end if;

           if(B(55) = '1') then
              Y(56) <= '1';
           else
              Y(56) <= '0';
           end if;
       end if;
   end process;

end BEHAVIORAL;

configuration CFG_extend_BEHAVIORAL of extend is
              for BEHAVIORAL
              end for;
end CFG_extend_BEHAVIORAL;
```

-- VHDL for reciprocal unit: q_regs.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

entity q_regs is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               round : In    std_logic;
                   S : In    std_logic_vector (3 downto 0);
                   T : In    std_logic;--_vector(1 downto 0);
                  QK : In    std_logic_vector (3 downto 0);
                   Q : InOut   std_logic_vector (55 downto 0) );
end q_regs;

architecture BEHAVIORAL of q_regs is

   begin
    process(clear,clk)
     variable qq, qn, qp : std_logic_vector (55 downto 0) ;
     variable qplus1, qminus1 : std_logic_vector (3 downto 0) ;
     variable carry: std_logic;
     variable i: integer;
     begin

       if ( clear = '0' ) then
          for i in 0 to 55 loop
              q(i) <= '0';
              qq(i) := '0';
              qn(i) := '0';
              qp(i) := '0';
```

```
         end loop;
       elsif ((clk = '1') AND (clk'EVENT)) then

         if((t ='1') and qk = "0000" )then
            for i in 55 downto 4 loop
              qq(i) := qp(i-4);
                 qn(i) := qp(i-4);
                 qp(i) := qp(i-4);
             end loop;
         elsif ((t ='0') and qk = "1111" )then
            for i in 55 downto 4 loop
             qq(i) := qq(i-4);
             qn(i) := qq(i-4);
             qp(i) := qp(i-4);
            end loop;
         else
              for i in 55 downto 4 loop
             qq(i) := qq(i-4);
             qn(i) := qq(i-4);
             qp(i) := qq(i-4);
           end loop;
         end if;
                  --insert last digit
                  qplus1  := qk + "0001";
                  qminus1 := qk + "1111";
                  qq(3 downto 0) := qk;
                  qn(3 downto 0) := qminus1;
                  qp(3 downto 0) := qplus1;

                  ------rounding-------
                  if(round = '1') then
                     carry := s(3) and '1';
                     if(carry = '1') then
                        Q <= qp;
                     else
                       Q <= qq;
                    end if;
       end if;
    end if;--end if clock
    end process;

end BEHAVIORAL;

configuration CFG_q_regs_BEHAVIORAL of q_regs is
   for BEHAVIORAL
   end for;

end CFG_q_regs_BEHAVIORAL;
```

-- VHDL code for reciprocal unit qds_adder.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity QDS_ADDER is
      Port (      A1 : In    std_logic_vector (6 downto 0);
                  A2 : In    std_logic_vector (6 downto 0);
                   Y : Out   std_logic_vector (6 downto 0) );
end QDS_ADDER;

architecture BEHAVIORAL of QDS_ADDER is

begin

  process(A1, A2)
             variable g0,p0 : std_logic_vector (1 downto 0) ;
             variable g1,p1 : std_logic_vector (1 downto 0) ;
             variable g2,p2 : std_logic_vector (1 downto 0) ;
             variable g3,p3 : std_logic;
             variable   c   : std_logic_vector (4 downto 0) ;
             variable  cc   : std_logic_vector (1 downto 0) ;
             variable i,j,k,l : integer;
```

```
      begin

              cc(0) := '0';

--------------------------------- 1st level ----------------------
              j := 0 ;
              g0(0) := A1(0+j) AND A2(0+j); p0(0) := A1(0+j) OR A2(0+j);
              g1(0) := A1(1+j) AND A2(1+j); p1(0) := A1(1+j) OR A2(1+j);
              g2(0) := A1(2+j) AND A2(2+j); p2(0) := A1(2+j) OR A2(2+j);
              g3    := A1(3+j) AND A2(3+j); p3    := A1(3+j) OR A2(3+j);
              j := 4 ;
              g0(1) := A1(0+j) AND A2(0+j); p0(1) := A1(0+j) OR A2(0+j);
              g1(1) := A1(1+j) AND A2(1+j); p1(1) := A1(1+j) OR A2(1+j);
              g2(1) := A1(2+j) AND A2(2+j); p2(1) := A1(2+j) OR A2(2+j);
--------------------------------- 2nd level ----------------------
           k := 0;
              cc(1) := g3 OR (g2(k) AND p3 ) OR (g1(k) AND p2(k) AND p3 )
                                 OR (g0(k) AND p1(k) AND p2(k) AND p3   ) ;

-- CARRY ------------------------- 1st level ----------------------
         k := 0;
              c(0) := cc(k);
              c(1) := g0(k) OR (c(0) AND p0(k));
              c(2) := g1(k) OR (g0(k) AND p1(k)) OR (c(0) AND p0(k) AND p1(k));
              c(3) := g2(k) OR (g1(k) AND p2(k)) OR (g0(k) AND p1(k) AND p2(k))
                                 OR (c(0) AND p0(k) AND p1(k) AND p2(k));
              j := k*4 ;
              for i in 0 to 3 loop
                Y(i+j)  <= A1(i+j) XOR A2(i+j) XOR c(i) ;
              end loop;

         k := 1;
              c(0) := cc(k);
              c(1) := g0(k) OR (c(0) AND p0(k));
              c(2) := g1(k) OR (g0(k) AND p1(k)) OR (c(0) AND p0(k) AND p1(k));
              j := k*4 ;
              for i in 0 to 2 loop
               Y(i+j)  <= A1(i+j) XOR A2(i+j) XOR c(i) ;
              end loop;

     end process;

end BEHAVIORAL;

configuration CFG_QDS_ADDER_BEHAVIORAL of QDS_ADDER is
   for BEHAVIORAL

   end for;

end CFG_QDS_ADDER_BEHAVIORAL;
```

-- VHDL code for reciprocal unit: QDS_TABLE.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;

entity QDS_TABLE is
      Port (       D : In    std_logic_vector (2 downto 0);
                   Y : In    std_logic_vector (6 downto 0);
                  M1 : Out   std_logic;
                  M2 : Out   std_logic;
                  P1 : Out   std_logic;
                  P2 : Out   std_logic );
end QDS_TABLE;

architecture BEHAVIORAL of QDS_TABLE is

   begin

  process(Y, D)
    TYPE table_type IS ARRAY(0 to 3) OF std_logic_vector (6 downto 0);
```

```
CONSTANT sel_fun0 : table_type := (
         "0001100", "0000100", "1111100", "1110011" );--12,4,-4,-13
CONSTANT sel_fun1 : table_type := (
         "0001110", "0000100", "1111010", "1110001" );
CONSTANT sel_fun2 : table_type := (
         "0001111", "0000100", "1111010", "1110000" );--15,4,-6,-16
CONSTANT sel_fun3 : table_type := (
         "0010000", "0000100", "1111010", "1101110" );--16,4,-6,-18
CONSTANT sel_fun4 : table_type := (
         "0010010", "0000110", "1111000", "1101100" );
CONSTANT sel_fun5 : table_type := (
         "0010100", "0000110", "1111000", "1101100" );
CONSTANT sel_fun6 : table_type := (
         "0010100", "0001000", "1111000", "1101010" );
CONSTANT sel_fun7 : table_type := (
         "0011000", "0001000", "1111000", "1101000" );

begin

          -- Selection function
    if ( D = "000" ) then
              if ( y(6) = '0' ) then   ------- y is positive
                        if ( y >= sel_fun0(0) ) then
                           M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        elsif ( y < sel_fun0(1) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                        end if;
              else                      ------- y is negative
                        if ( y < sel_fun0(3) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                        elsif ( y >= sel_fun0(2) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                        end if;
              end if;
    elsif ( D = "001" ) then
              if ( y(6) = '0' ) then   ------- y is positive
                        if ( y >= sel_fun1(0) ) then
                           M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        elsif ( y < sel_fun1(1) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                        end if;
              else                      ------- y is negative
                        if ( y < sel_fun1(3) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                        elsif ( y >= sel_fun1(2) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                        end if;
              end if;
    elsif ( D = "010" ) then
              if ( y(6) = '0' ) then   ------- y is positive
                        if ( y >= sel_fun2(0) ) then
                           M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        elsif ( y < sel_fun2(1) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                        end if;
              else                      ------- y is negative
                        if ( y < sel_fun2(3) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                        elsif ( y >= sel_fun2(2) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                        end if;
              end if;
```

```
elsif ( D = "011" ) then
          if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_fun3(0) ) then
                       M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_fun3(1) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
          else                      ------- y is negative
                    if ( y < sel_fun3(3) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_fun3(2) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
          end if;
elsif ( D = "100" ) then
          if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_fun4(0) ) then
                       M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_fun4(1) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
          else                      ------- y is negative
                    if ( y < sel_fun4(3) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_fun4(2) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
          end if;
elsif ( D = "101" ) then
          if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_fun5(0) ) then
                       M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_fun5(1) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
          else                      ------- y is negative
                    if ( y < sel_fun5(3) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_fun5(2) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
          end if;
elsif ( D = "110" ) then
          if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_fun6(0) ) then
                       M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_fun6(1) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
          else                      ------- y is negative
                    if ( y < sel_fun6(3) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_fun6(2) ) then
                       M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                       M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
          end if;
elsif ( D = "111" ) then
          if ( y(6) = '0' ) then   ------- y is positive
```

```
                        if ( y >= sel_fun7(0) ) then
                           M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        elsif ( y < sel_fun7(1) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                        end if;
                else                        ------- y is negative
                        if ( y < sel_fun7(3) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                        elsif ( y >= sel_fun7(2) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                        end if;
                end if;
        end if;

  end process;

end BEHAVIORAL;

configuration CFG_QDS_TABLE_BEHAVIORAL of QDS_TABLE is
   for BEHAVIORAL

   end for;

end CFG_QDS_TABLE_BEHAVIORAL;
```

-- VHDL code for reciprocal unit: QDSEL.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;

entity QDSEL is
      Port (      A1 : In    std_logic_vector (6 downto 0);
                  A2 : In    std_logic_vector (6 downto 0);
                   D : In    std_logic_vector (2 downto 0);
                  M1 : Out   std_logic;
                  M2 : Out   std_logic;
                  P1 : Out   std_logic;
                  P2 : Out   std_logic );
end QDSEL;

architecture SCHEMATIC of QDSEL is

   signal        Y : std_logic_vector(6 downto 0);

   component QDS_TABLE
      Port (       D : In    std_logic_vector (2 downto 0);
                   Y : In    std_logic_vector (6 downto 0);
                  M1 : Out   std_logic;
                  M2 : Out   std_logic;
                  P1 : Out   std_logic;
                  P2 : Out   std_logic );
   end component;

   component QDS_ADDER
      Port (      A1 : In    std_logic_vector (6 downto 0);
                  A2 : In    std_logic_vector (6 downto 0);
                   Y : Out   std_logic_vector (6 downto 0) );
   end component;

begin

   I_1 : QDS_TABLE
      Port Map ( D(2 downto 0)=>D(2 downto 0),
                 Y(6 downto 0)=>Y(6 downto 0), M1=>M1, M2=>M2, P1=>P1,
                 P2=>P2 );
   I_2 : QDS_ADDER
      Port Map ( A1(6 downto 0)=>A1(6 downto 0),
```

```
                A2(6 downto 0)=>A2(6 downto 0),
                Y(6 downto 0)=>Y(6 downto 0) );

end SCHEMATIC;

configuration CFG_QDSEL_SCHEMATIC of QDSEL is

   for SCHEMATIC
      for I_1: QDS_TABLE
         use configuration WORK.CFG_QDS_TABLE_BEHAVIORAL;
      end for;
      for I_2: QDS_ADDER
         use configuration WORK.CFG_QDS_ADDER_BEHAVIORAL;
      end for;
   end for;

end CFG_QDSEL_SCHEMATIC;
```

-- VHDL code for reciprocal unit: QUOTIENT.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;

entity QUOTIENT is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
                DET0 : In    std_logic;
               DIGIT : In    std_logic;
                  QK : In    std_logic_vector (1 downto 0);
                  QM : In    std_logic_vector (1 downto 0);
                QSIG : In    std_logic;
               ROUND : In    std_logic;
                  QN : InOut std_logic_vector (55 downto 0);
                  QP : InOut std_logic_vector (55 downto 0);
                  QQ : InOut std_logic_vector (55 downto 0) );
end QUOTIENT;

architecture BEHAVIORAL of QUOTIENT is

   begin
    process(clear,clk)
              variable flag : std_logic;
              variable last : std_logic;
     begin

       if ( clear = '0' ) then
          for i in 0 to 55 loop
              qq(i) <= '0';
              qn(i) <= '0';
              qp(i) <= '0';
          end loop;

       elsif ((clk = '1') AND (clk'EVENT)) then
          if ( digit = '1' ) then
                if ( det0 = '1' ) then
                        for i in 55 downto 2 loop
                          qq(i) <= qq(i-2);
                          qn(i) <= qn(i-2);
                        end loop;
                elsif ( qsig = '0') then
                        for i in 55 downto 2 loop
                          qq(i) <= qq(i-2);
                          qn(i) <= qq(i-2);
                        end loop;
                elsif ( qsig = '1') then
                        for i in 55 downto 2 loop
                          qq(i) <= qn(i-2);
                          qn(i) <= qn(i-2);
                        end loop;
                end if;
                for i in 0 to 1 loop
```

```
                  qq(i) <= qk(i);
                  qn(i) <= qm(i);
                end loop;
-- --------------------- addition for variable rounding --------------------

                           qp(1) <= qk(1) xor qk(0);
                           qp(0) <= not qk(0);

                           last := qq(52) or qq(51);
                           if (last = '1') then
                             if ( qq(52) = '0' ) then -- round in last position

                               if (( qk = "10" ) and ( QSIG = '1') ) then
                                        for i in 55 downto 2 loop
                                          qp(i) <= qn(i-2);
                                        end loop;
                               else
                                        for i in 55 downto 2 loop
                                          qp(i) <= qq(i-2);
                                        end loop;
                               end if;

                             else -- round last-1 position

                               if ( ( qk = "10" ) AND ( QSIG = '0') ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qp(i-2);
                           end loop;
                                  elsif ( ( QSIG = '1') ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qq(i-2);
                           end loop;
                      else
                           -- nop
                      end if;
                                end if;

                             else -- it is not the last iteration

                      if ( ( qk = "11" ) ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qp(i-2);
                           end loop;
                      elsif ( ( qk = "10" ) AND ( QSIG = '1') ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qn(i-2);
                           end loop;
                      else
                           for i in 55 downto 2 loop
                             qp(i) <= qq(i-2);
                           end loop;
                      end if;

                             end if;
---------------------- end addition for variable rounding ----------------

               else
            for i in 0 to 55 loop
               qq(i) <= '0';
               qn(i) <= '0';
               qp(i) <= '0';
            end loop;
           end if;
       end if;

    end process;

end BEHAVIORAL;

configuration CFG_QUOTIENT_BEHAVIORAL of QUOTIENT is
   for BEHAVIORAL

   end for;
```

```
end CFG_QUOTIENT_BEHAVIORAL;
```

-- VHDL code for reciprocal: rounding.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity ROUNDING is
      Port (     CLK : In    std_logic;
                  QN : In    std_logic_vector (55 downto 0);
                  QP : In    std_logic_vector (55 downto 0);
                  QQ : In    std_logic_vector (55 downto 0);
               ROUND : In    std_logic;
                SIGN : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
end ROUNDING;

architecture BEHAVIORAL of ROUNDING is

   begin
    process(clk)
    variable signn : std_logic ;
    variable p : std_logic_vector (2 downto 0) ;
    variable mqq,mqn,mqp : std_logic_vector (55 downto 0) ;

     begin
      if (( clk = '1' ) AND (clk'EVENT)) then
        if ( round = '1' ) then
             p := "000";
             mqq := QQ;
             mqn := Qn;
             mqp := Qp;
             signn := SIGN;

        if ( qq(54) = '0')  then
                if ( signn = '0' ) then
                           -- add 1
                           p(0) := NOT qq(0);
                           p(1) := qq(0) XOR qq(1);
                           p(2) := qq(0) AND qq(1);
                else
                           p(0) := qq(0);
                           p(1) := qq(1);
                           p(2) := '0';
                end if;

                if ( p >= "100" ) then
                   for i in 0 to 52 loop
                      Q(i) <= qp(i+1);
                   end loop;
           else
                   for i in 1 to 52 loop
                      Q(i) <= qq(i+1);
                   end loop;
                              Q(0) <= p(1);
           end if;

-- new part introduced for variable rounding
             else -- qq(54) = '1'

                if ( signn = '0' ) then
                           -- add 2
                           p(0) := qq(0);
                           p(1) := NOT qq(1);
                           p(2) := qq(1);
                else
                           -- subtract 1 add 2 => add 1
                           p(0) := NOT qq(0);
                           p(1) := qq(0) XOR qq(1);
                           p(2) := qq(0) AND qq(1);
                end if;
```

```
                if ( p >= "100" ) then
                   for i in 0 to 52 loop
                      Q(i) <= qp(i+2);
                   end loop;
           elsif (( qq(55) = '0')  or
              (( qq(1) = '1') and ( qq(0) = '1') and ( signn = '0' ))) then
                   for i in 0 to 52 loop
                      Q(i) <= qq(i+2);
                   end loop;
           else
                   for i in 0 to 52 loop
                      Q(i) <= qn(i+2);
                   end loop;
           end if;
         end if; -- close qq(54) = '0'
        end if; -- close round = '1'
        end if;  -- close CLK

     end process;

end BEHAVIORAL;

configuration CFG_ROUNDING_BEHAVIORAL of ROUNDING is
   for BEHAVIORAL

   end for;

end CFG_ROUNDING_BEHAVIORAL;
```

-- VHDL code for reciprocal unit: sdet.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity SDET is
      Port (       A : In    std_logic_vector (56 downto 0);
                   B : In    std_logic_vector (56 downto 0);
                   Z : Out   std_logic );
end SDET;

architecture BEHAVIORAL of SDET is

begin
process(a,b)
variable d : std_logic;
begin

d := '0';
for i in 0 to 55 loop
  d := ( A(i) AND B(i) ) OR ( A(i) AND D ) OR ( D AND B(i) ) ;
end loop;
Z <= A(56) XOR B(56) XOR D ;

end process;

end BEHAVIORAL;

configuration CFG_SDET_BEHAVIORAL of SDET is
   for BEHAVIORAL

   end for;

end CFG_SDET_BEHAVIORAL;

-- VHDL code for reciprocal shifter.vhd

library IEEE;
   use IEEE.std_logic_1164.all;

entity shifter is
      Port (  A : In    std_logic_vector (56 downto 0);
              B : Out   std_logic_vector (55 downto 0) );
```

```
end shifter;

architecture BEHAVIORAL of shifter is

   begin
    process(A)
       begin
        if ( A(56) = '1' ) then
        B <= '1' & A(56 downto 2);
        else
        B <= '0' & A(56 downto 2);
        end if;
   end process;
end BEHAVIORAL;
configuration CFG_shifter_BEHAVIORAL of shifter is
   for BEHAVIORAL
   end for;
end CFG_shifter_BEHAVIORAL;
```

-- VHDL code for reciprocal: tb2_reciprocal.vhd

```
library IEEE;
             use IEEE.std_logic_1164.all;
             use STD.textio.all;
             use IEEE.std_logic_textio.all;
             use IEEE.std_logic_signed.all;

entity E is
end E;

architecture A of E is

   signal    CLOCK : std_logic;
   signal        D : std_logic_vector (52 downto 0);
   signal    RESET : std_logic;
   signal        Q : std_logic_vector (52 downto 0);

   component RECIPROCAL
      Port (   CLOCK : In    std_logic;
                   D : In    std_logic_vector (52 downto 0);
               RESET : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
   end component;

begin
   UUT : RECIPROCAL
      Port Map ( CLOCK, D, RESET, Q );

   TB : block
   begin
   process
             CONSTANT NLOOPS : integer := 17;
             --CONSTANT NLOOPS : integer := 30;
             file cmdfile: TEXT;                       -- Define the file 'handle'
             variable line_in,line_out: Line; -- Line buffers
             variable good: boolean;     -- Status of the read operations
             variable A: std_logic_vector(55 downto 0);
             variable S: std_logic_vector(55 downto 0);
             variable Z: std_logic_vector(55 downto 0);
             variable ERR: std_logic_vector(55 downto 0);
             variable c : integer;

             begin


  c := 1;
  FILE_OPEN(cmdfile,"testvecs1.in",READ_MODE);

-- ------------------------------------------------------------------------
  loop
    if endfile(cmdfile) then  -- Check EOF
             assert false
                report "End of file encountered; exiting."
```

```
               severity NOTE;
            exit;
    end if;

  reset <= '0';
  clock <= '1'; wait for 5 ns;
  reset <= '1';
  clock <= '0'; wait for 5 ns;


    readline(cmdfile,line_in);     -- Read a line from the file
    next when line_in'length = 0;  -- Skip empty lines

    hread(line_in,A,good);         -- Read the X argument as hex value
    assert good report "Text I/O read error" severity ERROR;

    hread(line_in,S,good);         -- Read the Q argument as hex value
    assert good report "Text I/O read error" severity ERROR;

    D(52 downto 0)  <= A(52 downto 0);

  for i in 0 to NLOOPS loop
        clock <= '1'; wait for 5 ns;
        clock <= '0'; wait for 5 ns;
  end loop;

    Z(55) := '0'; Z(54) := '0'; Z(53) := '0';
    Z(52 downto 0)  := Q(52 downto 0);
    ERR := S-Z;

    write(line_out, string'("Test "));
    write(line_out,c);
    if (Z = S) then
        write(line_out, string'(" passed: "));
              write(line_out, string'(" 1/ "));
              hwrite(line_out,A,RIGHT,14);

              --hwrite(line_out,B,RIGHT,14);
              write(line_out, string'(" -> "));
              hwrite(line_out,Z,RIGHT,14);
    else
        write(line_out, string'(" FAILED: "));
              write(line_out, string'(" 1/ "));
              hwrite(line_out,A,RIGHT,14);

              --hwrite(line_out,B,RIGHT,14);
              write(line_out, string'(" -> "));
              hwrite(line_out,Z,RIGHT,14);
        write(line_out, string'(" <> "));
              hwrite(line_out,S,RIGHT,14);
              writeline(OUTPUT,line_out);
        write(line_out, string'("Test "));
        write(line_out,c);
              write(line_out, string'("                       ERR "));
              hwrite(line_out,ERR,RIGHT,14);
    end if;
    writeline(OUTPUT,line_out);     -- write the message
    assert (Z = S) report "Z does not match in pattern " severity error;
    c := c + 1;
  end loop;
write(line_out, string'("--------- END SIMULATION  ------------------"));
writeline(OUTPUT,line_out);
-- =============================================================
  clock <= '1'; wait for 1000 ns;
  assert q = "11111111111111111111111111111111111111111111111111111111"
    report "--------- END SIMULATION  ------------------ " severity error;
-- =============================================================


   end process;
 end block;
end A;

configuration CFG_TB_RECIPROCAL_BEHAVIORAL of E is
```

```
   for A
      for UUT : RECIPROCAL
         use configuration WORK.CFG_RECIPROCAL_SCHEMATIC;
      end for;
      for TB
      end for;
   end for;
end CFG_TB_RECIPROCAL_BEHAVIORAL;
```

Appendix B

B.1 VHDL files included in the combined unit design:

radix4.vhd
control.vhd
convert.vhd
convert_app.vhd
cpa.vhd
cr_qcalc.vhd
csa42.vhd
digit_compute.vhd
digit_update2.vhd
gl_csa32.vhd
gl_dualreg_ld.vhd
gl_mux21.vhd
gl_reg.vhd
mult2.vhd
q_regs.vhd
qlatch.vhd
quotient.vhd
rounding.vhd
sdet.vhd
extendsc.vhd
extend.vhd
init.vhd
mult4.vhd
mux2.vhd
qdsel-beh.vhd
tb2_radix4.vhd

B.2 VHDL code for combined unit design

-- File Name : radix4.vhd, this is the top level design.
-- Combined unit of reciprocal and square-root reciprocal

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity radix4 is
      Port (   CLOCK : In    std_logic;
                   D : In    std_logic_vector (52 downto 0);
               RESET : In    std_logic;
                  ED : In    std_logic;
                  OP : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
end radix4;

architecture SCHEMATIC of radix4 is

   signal SRWS,SRWC,PDJ,WSEX,WCEX,SWC :std_logic_vector(55 downto 0);
   signal SWS,SDJ,DMU4,H0,ND :std_logic_vector(55 downto 0);

   signal PDJ32,CS42S,CS42C,SHS,SHC : std_logic_vector(57 downto 0);
   signal ADD42S,ADD42C,EXH0,SH42S : std_logic_vector(57 downto 0);
   signal HS32,HC32,HS42,HC42 : std_logic_vector(57 downto 0);
   signal HS,HC,H1,H2 : std_logic_vector(57 downto 0);
   signal MS,PNM1,PNM2,PNP1,PNP2,N_S42 : std_logic;
   signal N_C42,N_C32,N_S42e,N_C42e : std_logic;
   signal      C0,C0_1,C0_2,C0_RP,C0_SQ : std_logic_vector(55 downto 0);
   signal      DQ : std_logic_vector(52 downto 0);
   signal      SC1 : std_logic_vector(55 downto 0);
   signal      SCJ : std_logic_vector(55 downto 0);
   signal      MXW : std_logic_vector(55 downto 0);
   signal       DD : std_logic_vector(55 downto 0);
   signal      SW2 : std_logic_vector(55 downto 0);
   signal      SW1 : std_logic_vector(55 downto 0);
   signal       W2 : std_logic_vector(55 downto 0);
   signal       W1, W0 : std_logic_vector(55 downto 0);
   signal       WC : std_logic_vector(55 downto 0);
   signal       WS : std_logic_vector(55 downto 0);
   signal       SD, NSD : std_logic_vector(55 downto 0);
   signal     CTMU : std_logic_vector(55 downto 0);
   signal    CJREG : std_logic_vector(55 downto 0);
   signal     DTMU : std_logic_vector(55 downto 0);
   signal      CSC : std_logic_vector(55 downto 0);
   signal      CSD : std_logic_vector(55 downto 0);
   signal     CPAC : std_logic_vector(55 downto 0);
   signal       DJ : std_logic_vector(55 downto 0);
   signal       C1 : std_logic_vector(55 downto 0);
   signal   D1, D0 : std_logic_vector(55 downto 0);
   signal       DT : std_logic_vector(7 downto 0);
   signal      DL1 : std_logic_vector(7 downto 0);
   signal      DL2 : std_logic_vector(7 downto 0);
   signal       YT : std_logic_vector(6 downto 0);
   signal       Y1 : std_logic_vector(6 downto 0);
   signal       Y2 : std_logic_vector(6 downto 0);
   signal      P,pzero, pini : std_logic_vector(3 downto 0);
   signal      N_C, N_w0, N_h0 : std_logic;
   signal      N_D, sign : std_logic;
   signal    N_CPA : std_logic;
   signal    ROUND : std_logic;
   signal    DIGIT : std_logic;
   signal     N_20 : std_logic;
   signal     N_23 : std_logic;
   signal     N_24 : std_logic;
   signal       P2 : std_logic;
   signal       P1 : std_logic;
   signal       M1 : std_logic;
   signal       M2 : std_logic;
```

```
 signal iM1, iM2, iP1, iP2  : std_logic;
 signal NM1, NM2, NP1, NP2  : std_logic;
 signal     GND : std_logic;

component init
    Port (    D : In std_logic_vector (52 downto 0);
               ED : In std_logic;
               OP : In std_logic;
               M  : In std_logic;
               ZP : Out std_logic_vector (3 downto 0);
               ZW : Out std_logic_vector (55 downto 0);
               ZH : Out std_logic_vector (55 downto 0) );
 end component;

 component gl_dualreg_ld
   GENERIC(n : integer);
    Port (      AS : In    std_logic_vector (n downto 0);
                AC : In    std_logic_vector (n downto 0);
             RESET : In    std_logic;
             CLOCK : In    std_logic;
              LOAD : In    std_logic;
                ZS : Out   std_logic_vector (n downto 0);
                ZC : Out   std_logic_vector (n downto 0) );
 end component;

 component QLATCH
    Port (   CLEAR : In    std_logic;
               CLK : In    std_logic;
                I1 : In    std_logic;
                I2 : In    std_logic;
                J1 : In    std_logic;
                J2 : In    std_logic;
              LOAD : In    std_logic;
                M1 : Out   std_logic;
                M2 : Out   std_logic;
                P1 : Out   std_logic;
                P2 : Out   std_logic );
 end component;


 component MULT2
    GENERIC(n : integer);
    Port (       A : In    std_logic_vector (n downto 0);
                Ap : In    std_logic; -- A(-1)
                M1 : In    std_logic;
                M2 : In    std_logic;
                P1 : In    std_logic;
                P2 : In    std_logic;
              COUT : Out   std_logic;
                 Z : Out   std_logic_vector (n downto 0) );
 end component;

 component MULT4
    GENERIC(n : integer);
    Port (A : In    std_logic_vector (n downto 0);
          M1 : In    std_logic;
          M2 : In    std_logic;
          P1 : In    std_logic;
          P2 : In    std_logic;
          Cout : out    std_logic;
          Z : Out   std_logic_vector (n downto 0) );
 end component;

component csa42
      GENERIC(n : integer);
      Port (  A : In std_logic_vector (n downto 0);
                  B : In std_logic_vector (n downto 0);
                  C : In std_logic_vector (n downto 0);
                  D : In std_logic_vector (n downto 0);
                  SCin : In std_logic;
                  CCin : In std_logic;
                  NCin : In std_logic;
                  Z : Out std_logic_vector (n downto 0);
                  Y : Out std_logic_vector (n downto 0));
```

```
end component;

component CONTROL
   Port (   CLOCK : In    std_logic;
            RESET : In    std_logic;
              CL1 : Out   std_logic;
            DIGIT : Out   std_logic;
              LD1 : Out   std_logic;
              MX1 : Out   std_logic;
            ROUND : Out   std_logic );
end component;

component gl_mux21
     GENERIC(n : integer);
     Port (  A0 : In std_logic_vector (n downto 0);
             A1 : In std_logic_vector (n downto 0);
             SEL : In std_logic;
             Z : Out std_logic_vector (n downto 0) );
end component;

component mux2
     GENERIC(n : integer);
     Port (  A0 : In std_logic_vector (n downto 0);
             A1 : In std_logic_vector (n downto 0);
             SEL : In std_logic;
             Cout : Out std_logic;
             Z : Out std_logic_vector (n downto 0) );
end component;

component QDSEL
   Port (Y : In    std_logic_vector (6 downto 0);
         D : In    std_logic_vector (3 downto 0);
        M1 : Out   std_logic;
        M2 : Out   std_logic;
        P1 : Out   std_logic;
        P2 : Out   std_logic);
end component;

component cpa
   GENERIC(n : integer);
   Port ( A1 : In    std_logic_vector (n downto 0);
          A2 : In    std_logic_vector (n downto 0);
          Cin : in  std_logic;
          Z : Out   std_logic_vector (n downto 0));
end component;

component gl_csa32
     GENERIC(n : integer);
     Port (  A : In std_logic_vector (n downto 0);
             B : In std_logic_vector (n downto 0);
             C : In std_logic_vector (n downto 0);
             Cin : In std_logic;
             Z : Out std_logic_vector (n downto 0);
             Y : Out std_logic_vector (n downto 0) );
end component;

component extendsc
     Port (A : in std_logic_vector (55 downto 0);
                 B : in std_logic_vector (55 downto 0);
                 Y : out std_logic_vector (57 downto 0);
           Z : out std_logic_vector (57 downto 0));
end component;

component extend
     Port (A : in std_logic_vector (55 downto 0);
           Z : out std_logic_vector (57 downto 0));
end component;

component SDET
   Port (    A : In    std_logic_vector (55 downto 0);
             B : In    std_logic_vector (55 downto 0);
             Z : Out   std_logic );
end component;
```

```
  component CONVERT
     Port (   CLEAR : In    std_logic;
                CLK : In    std_logic;
              DIGIT : In    std_logic;
                 M1 : In    std_logic;
                 M2 : In    std_logic;
                 P1 : In    std_logic;
                 P2 : In    std_logic;
              ROUND : In    std_logic;
               SIGN : In    std_logic;
                  Q : Out   std_logic_vector (52 downto 0) );
  end component;

  component CONVERT_APP
      Port ( CLEAR : In    std_logic;
                CLK : In    std_logic;
             round : In    std_logic;
                E1 : In    std_logic_vector(4 downto 0);
                E2 : In    std_logic_vector(4 downto 0);
                 Q : Out   std_logic_vector(52 downto 0) );
  end component;

begin

GND <='0';
MS<=D(51);
C0_RP(55 downto 0) <= (others => '0');--C0=0 reciprocal computation
C0_2(55 downto 49)<="0000000"; C0_2(48 downto 0)<=D(52 downto 4);--C[0]=(1/16)*d
C0_1(55)<='0'; C0_1(54 downto 0)<=C0_2(55 downto 1);--C[0]=(1/32)*d
DD(55 downto 53) <= "000"; DD(52 downto 0) <= D(52 downto 0);
SD(55 downto 52)<="0000"; SD(51 downto 0)<=D(52 downto 1);--D[0]=d*1/2
MXW(55 downto 2)<=W1(53 downto 0) ; MXW(1)<='0'; MXW(0)<='0';
SW2(55 downto 2)<=W2(53 downto 0) ; SW2(1)<='0'; SW2(0)<='0';

 I_CTRL : CONTROL
      Port Map ( CLOCK=>CLOCK, RESET=>RESET, CL1=>N_23, DIGIT=>DIGIT,
                 LD1=>N_24, MX1=>N_20, ROUND=>ROUND );

   I_MUXsq : gl_mux21 Generic Map(n=>55)--C0 for square-root reciprocal
      Port Map ( A0=>C0_1, --C0=d/32
                 SEL=>ED,
                 A1=>C0_2,--C0=d/16
                 Z=>C0_SQ);

   I_MUXc0 : gl_mux21 Generic Map(n=>55)--C0
      Port Map ( A0=>C0_RP, --C0=0
                 SEL=>OP,
                 A1=>C0_SQ,
                 Z=>C0);

   I_init : init
      Port Map(              D => D,
                       ED => ED,
                             OP => OP,
                             M  => MS,
                             ZP => PZERO,
                             ZW => W0,
                             ZH => H0);

   I_MUXw : mux2 Generic Map(n=>55)
      Port Map ( A0=>MXW(55 downto 0),
                 SEL=>N_20,
                 A1=>W0(55 downto 0),--++
                 Cout=>N_w0,
                 Z=>SW1(55 downto 0));

   I_MUXc : gl_mux21 Generic Map(n=>55)
      Port Map ( A0=>C1(55 downto 0),
                 SEL=>N_20,
                 A1=>C0(55 downto 0),
                 Z=>CTMU(55 downto 0));

   I_MUXd : gl_mux21 Generic Map(n=>55)--D0
      Port Map ( A0=>D1(55 downto 0),
```

```
              SEL=>N_20,
              A1=>SD(55 downto 0),
              Z=>DTMU(55 downto 0) );

I_MULTc : MULT4 Generic Map(n=>55)
   Port Map ( A=>CTMU(55 downto 0),
                          M1=>M1, M2=>M2, P1=>P1, P2=>P2,
              Cout=>N_C,
              Z=>CSC(55 downto 0) );

I_MULTd : MULT2 Generic Map(n=>55)
   Port Map ( A=>DTMU(55 downto 0), Ap=>GND,
                          M1=>M1, M2=>M2, P1=>P1, P2=>P2, COUT=>N_D,
              Z=>CSD(55 downto 0) );

I_CSA42q : csa42 Generic Map(n=>55)
   Port Map (A =>SW1,
                         B =>SW2,
                         C =>CSC,
                         D =>CSD,
                         SCin => N_C,
                         CCin => N_D,
                         NCin => N_w0,
                         Z => WS,
                         Y => WC );
I_regRW : gl_dualreg_ld Generic Map(n=>55)
   Port Map ( AS=>WS(55 downto 0), AC=>WC(55 downto 0),
              RESET=>N_23, CLOCK=>CLOCK, LOAD=>N_24,
              ZS=>W1(55 downto 0), ZC=>W2(55 downto 0) );

SCJ(55 downto 54)<="00"; SCJ(53 downto 0)<=CTMU(55 downto 2); --Cj+1=Cj/4
SC1(55 downto 1)<=CTMU(54 downto 0) ; SC1(0)<='0';

NM2<=P2;
NM1<=P1;
NP1<=M1;
NP2<=M2;
pini(3)<=M2;
pini(2)<=M1;
pini(1)<=p1;
pini(0)<=p2;

I_MULT2c : MULT2 Generic Map(n=>55)
   Port Map ( A=>SC1(55 downto 0), Ap=>GND,
                          M1=>NM1, M2=>NM2, P1=>NP1, P2=>NP2, COUT=>N_CPA,
              Z=>CPAC(55 downto 0) );

  I_CPAd : cpa Generic Map(n=>55)
   Port Map ( A1=>DTMU,
              A2=>CPAC,
              Cin=>N_CPA,
              Z=>DJ);

 I_MUXcj : gl_mux21 Generic Map(n=>55)
   Port Map ( A0=>SCJ,
              SEL=>N_20,
              A1=>CTMU,
              Z=>CJREG);


I_regCD : gl_dualreg_ld Generic Map(n=>55)
   Port Map ( AS=>CJREG(55 downto 0), AC=>DJ(55 downto 0),
              RESET=>N_23, CLOCK=>CLOCK, LOAD=>N_24,
              ZS=>C1(55 downto 0), ZC=>D1(55 downto 0) );

Y1(6 downto 0)<=WS(52 downto 46);
Y2(6 downto 0)<=WC(52 downto 46);
DL1(7 downto 0)<=DTMU(51 downto 44);
DL2(7 downto 0)<=CPAC(51 downto 44);

I_CPAdt : cpa Generic Map(n=>7)
   Port Map ( A1=>DL1,
              A2=>DL2,
              Cin=>GND,
```

```
               Z=>DT);

  I_CPAyt : cpa Generic Map(n=>6)
     Port Map ( A1=>Y1,
                A2=>Y2,
                Cin=>GND,
                Z=>YT);

  I_SEL : QDSEL
     Port Map ( Y=>YT(6 downto 0),
                D=>DT(6 downto 3),
                M1=>iM1, M2=>iM2,
                P1=>iP1, P2=>iP2 );
  I_regQ : QLATCH
     Port Map ( CLEAR=>reset, CLK=>CLOCK, LOAD=>N_24,
                I1=>iM1, I2=>iM2, J1=>iP1, J2=>iP2,
                M1=>M1,  M2=>M2,  P1=>P1,  P2=>P2 );

  I_MUXp : gl_mux21 Generic Map(n=>3)
       Port Map ( A0=>pini,
                  SEL=>N_20,
                  A1=>pzero,
                  Z=>p);

  I_SDET : SDET
     Port Map ( A(55 downto 0)=>W2(55 downto 0),
                B(55 downto 0)=>W1(55 downto 0), Z=>SIGN );

  I_7 : CONVERT
     Port Map ( CLEAR=>N_23, CLK=>CLOCK, DIGIT=>DIGIT, M1=>p(2),
                M2=>p(3), P1=>p(1), P2=>p(0), ROUND=>ROUND, SIGN=>SIGN,
                Q(52 downto 0)=>DQ(52 downto 0) );

-------Approximation part----------------------
PNM1<= OP AND NM1;
PNM2<= OP AND NM2;
PNP1<= OP AND NP1;
PNP2<= OP AND NP2;
ND<= NOT DD;
NSD<= NOT SD;
SDJ(55)<='0'; SDJ(54 downto 0)<=DTMU(55 downto 1);--DJ/2
--shift 1 left to generate 2rw[j]----
   sws(55 downto 1)<=sw1(54 downto 0); sws(0)<='0';
   swc(55 downto 1)<=sw2(54 downto 0); swc(0)<='0';
--shift 4 left to generate 16H[j]
   SHS(57 downto 4)<=H1(53 downto 0); SHS(3 downto 0)<= "0000";
   SHC(57 downto 4)<=H2(53 downto 0); SHC(3 downto 0)<= "0000";

   I_MULTs42 : MULT2 Generic Map(n=>55)
     Port Map ( A=>SWS, Ap=>GND,--++
                           M1=>NM1, M2=>NM2, P1=>NP1, P2=>NP2, COUT=>N_s42,
                Z=>srws(55 downto 0) );

   I_MULTc42 : MULT2 Generic Map(n=>55)
     Port Map ( A=>SWC, Ap=>GND,--++
                           M1=>NM1, M2=>NM2, P1=>NP1, P2=>NP2, COUT=>N_c42,
                Z=>srwc(55 downto 0) );

     I_extendsc1 : extendsc
     Port Map (A =>SRWS(55 downto 0) ,
               B =>SRWC(55 downto 0) ,
               Y =>CS42C(57 downto 0) ,
                         Z =>CS42S(57 downto 0));

 EXH0(57 downto 56)<= "00"; EXH0(55 downto 0)<= H0(55 downto 0);

     I_MUXhs : mux2 Generic Map(n=>57)
     Port Map ( A0=>SHS,
                SEL=>N_20,
                A1=>EXH0,
                Cout=>N_h0,
                Z=>SH42S);

    I_CSA42b : csa42 Generic Map(n=>57)
```

```
      Port Map (A =>CS42S,
                           B =>CS42C,
                           C =>SH42S,
                           D =>SHC,
                           SCin =>N_s42 ,
                           CCin =>N_c42 ,
                           NCin => N_h0,
                           Z =>HS32 ,
                           Y =>HC32 );

     I_MUXdj : gl_mux21 Generic Map(n=>55)
      Port Map ( A0=>DTMU, --DJ
                 SEL=>OP,
                 A1=>SDJ,--DJ/2
                 Z=>DMU4);

      I_MULT4d : MULT4 Generic Map(n=>55)
      Port Map ( A=>DMU4,
                            M1=>M1, M2=>M2, P1=>P1, P2=>P2,
                 Cout=>N_C32,
                 Z=>PDJ );

      I_extend : extend
      Port Map (A =>PDJ(55 downto 0),
                           Z =>PDJ32(57 downto 0));

      I_CSA32 : gl_csa32 Generic Map(n=>57)
      Port Map ( A=>PDJ32,
                 B=>HS32, CIN=>N_C32,
                 C=>HC32,
                 Y=>HC42,
                 Z=>HS42);

      I_MULTs42e : MULT2 Generic Map(n=>55)
      Port Map ( A=>WS, Ap=>GND,
                 M1=>PNM1, M2=>PNM2, P1=>PNP1, P2=>PNP2, COUT=>N_s42e,
                 Z=>WSEX );

     I_MULTc42e : MULT2 Generic Map(n=>55)
      Port Map ( A=>WC, Ap=>GND,
                     M1=>PNM1, M2=>PNM2, P1=>PNP1, P2=>PNP2, COUT=>N_c42e,-
                 Z=>WCEX );

      I_extendsc2 : extendsc
      Port Map (A =>WSEX(55 downto 0) ,
                B =>WCEX(55 downto 0) ,
                Y =>ADD42C(57 downto 0) ,
                           Z =>ADD42S(57 downto 0));

      I_CSA42E : csa42 Generic Map(n=>57)
      Port Map (A =>HS42,
                           B =>HC42,
                           C =>ADD42S,
                           D =>ADD42C,
                           SCin =>N_s42E ,
                           CCin =>N_c42E ,
                           NCin => GND,
                           Z =>HS ,
                           Y =>HC );

      I_regH : gl_dualreg_ld Generic Map(n=>57)
      Port Map ( AS=>HS, AC=>HC,
                 RESET=>N_23, CLOCK=>CLOCK, LOAD=>N_24,
                 ZS=>H1, ZC=>H2);
      I_APP : CONVERT_APP
      Port Map ( CLEAR=>N_23, CLK=>CLOCK, E1=>HS(56 downto 52),
                 E2=>HC(56 downto 52), ROUND=>ROUND, --SIGN=>SIGN,
                 Q(52 downto 0)=>Q(52 downto 0) );

----------End of linear approximation part----------

end SCHEMATIC;

configuration CFG_radix4_SCHEMATIC of radix4 is
```

```
   for SCHEMATIC
     for I_init: init
         use configuration WORK.CFG_init_BEHAVIORAL;
      end for;
      for I_cpad, I_cpadt, I_cpayt: cpa
         use configuration WORK.CFG_cpa_BEHAVIORAL;
      end for;
      for I_CTRL: CONTROL
         use configuration WORK.CFG_CONTROL_BEHAVIORAL;
      end for;
      for I_MUXc,I_MUXd,I_MUXdj,I_MUXcj,I_MUXc0,I_MUXsq,I_MUXp : gl_mux21
         use configuration WORK.CFG_gl_mux21_BEHAVIORAL;
      end for;
      for I_MUXw,I_MUXhs: mux2
         use configuration WORK.CFG_mux2_BEHAVIORAL;
      end for;
      for I_regQ: QLATCH
         use configuration WORK.CFG_QLATCH_BEHAVIORAL;
      end for;
      for I_regrW, I_regCD, I_regH: gl_dualreg_ld
         use configuration WORK.CFG_gl_dualreg_ld_BEHAVIORAL;
      end for;
      for I_CSA42q, I_CSA42E: csa42
         use configuration WORK.CFG_CSA42_BEHAVIORAL;
      end for;
      for I_SEL: QDSEL
         use configuration WORK.CFG_QDSEL_BEHAVIORAL;
      end for;
      for I_MULTd,I_MULT2c,I_MULTs42,I_MULTc42,I_MULTs42e,I_MULTc42e: MULT2
         use configuration WORK.CFG_MULT2_BEHAVIORAL;
      end for;
      for I_MULTc, I_MULT4d : MULT4
         use configuration WORK.CFG_MULT4_BEHAVIORAL;
      end for;
      for I_CSA32 : gl_csa32
         use configuration WORK.CFG_GL_CSA32_BEHAVIORAL;
      end for;
      for I_extendsc2, I_extendsc1: extendsc
         use configuration WORK.CFG_extendsc_BEHAVIORAL;
      end for;
      for I_extend: extend
         use configuration WORK.CFG_extend_BEHAVIORAL;
      end for;
            for I_SDET: SDET
         use configuration WORK.CFG_SDET_BEHAVIORAL;
      end for;
      for I_7: CONVERT
         use configuration WORK.CFG_CONVERT_SCHEMATIC;
      end for;
      for I_APP: CONVERT_APP
         use configuration WORK.CFG_CONVERT_APP_SCHEMATIC;
      end for;
   end for;

end CFG_radix4_SCHEMATIC;
```

-- VHDL code for combined unit: control.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity CONTROL is
      Port (   CLOCK : In    std_logic;
               RESET : In    std_logic;
                 CL1 : Out   std_logic;
               DIGIT : Out   std_logic;
                 LD1 : Out   std_logic;
                 MX1 : Out   std_logic;
               ROUND : Out   std_logic );
end CONTROL;

architecture BEHAVIORAL of CONTROL is
```

```
   begin
   process(reset,clock)
   variable state : integer range 0 to 15;
    begin

        if ( reset = '0' ) then
          CL1 <= '0';
          LD1 <= '0';
          DIGIT <= '0' ;
          ROUND <= '1' ;
          MX1 <= '1' ;
          state := 0;

        elsif ((clock'EVENT) AND ( clock = '1' )) then

                            if( 0 = state ) then
                                DIGIT <= '1' ;
                          ROUND <= '0' ;
                                CL1 <= '1';
                             LD1 <= '1';
                          state  := 2 ;

                            elsif( 1 = state ) then
                                DIGIT <= '1' ;
                                CL1 <= '1';
                          ROUND <= '0' ;
                          MX1 <= '1' ;
                          state  := 2 ;

                            elsif( 2 = state ) then
                                MX1 <= '0' ;
                          state  := 3 ;

                            elsif( 15 = state ) then
                          DIGIT <= '0' ;
                          ROUND <= '1' ;
                          state  := 1 ;
                            else
                          state  := state + 1 ;
             end if;
        end if;

    end process;

end BEHAVIORAL;

configuration CFG_CONTROL_BEHAVIORAL of CONTROL is
   for BEHAVIORAL

   end for;

end CFG_CONTROL_BEHAVIORAL;
```

-- VHDL code for combined unit: convert.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity CONVERT is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               DIGIT : In    std_logic;
                  M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
               ROUND : In    std_logic;
                SIGN : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
end CONVERT;
```

```
architecture SCHEMATIC of CONVERT is

   signal       QK : std_logic_vector(1 downto 0);
   signal       QN : std_logic_vector(55 downto 0);
   signal       QQ : std_logic_vector(55 downto 0);
   signal       QM : std_logic_vector(1 downto 0);
   signal       QP : std_logic_vector(55 downto 0);
   signal     QSIG : std_logic;
   signal      N_2 : std_logic;

   component CR_QCALC
      Port (      M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
                  QK : Out   std_logic_vector (1 downto 0);
                  QM : Out   std_logic_vector (1 downto 0);
                QSIG : Out   std_logic;
                ZERO : Out   std_logic );
   end component;

   component ROUNDING
      Port (     CLK : In    std_logic;
                  QN : In    std_logic_vector (55 downto 0);
                  QP : In    std_logic_vector (55 downto 0);
                  QQ : In    std_logic_vector (55 downto 0);
               ROUND : In    std_logic;
                SIGN : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
   end component;

   component QUOTIENT
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
                DET0 : In    std_logic;
               DIGIT : In    std_logic;
                  QK : In    std_logic_vector (1 downto 0);
                  QM : In    std_logic_vector (1 downto 0);
                QSIG : In    std_logic;
               ROUND : In    std_logic;
                  QN : InOut std_logic_vector (55 downto 0);
                  QP : InOut std_logic_vector (55 downto 0);
                  QQ : InOut std_logic_vector (55 downto 0) );
   end component;

begin

   I_10 : CR_QCALC
      Port Map ( M1=>M1, M2=>M2, P1=>P1, P2=>P2,
                 QK(1 downto 0)=>QK(1 downto 0),
                 QM(1 downto 0)=>QM(1 downto 0), QSIG=>QSIG, ZERO=>N_2 );
   I_7 : ROUNDING
      Port Map ( CLK=>CLK, QN(55 downto 0)=>QN(55 downto 0),
                 QP(55 downto 0)=>QP(55 downto 0),
                 QQ(55 downto 0)=>QQ(55 downto 0), ROUND=>ROUND,
                 SIGN=>SIGN, Q(52 downto 0)=>Q(52 downto 0) );
   I_8 : QUOTIENT
      Port Map ( CLEAR=>CLEAR, CLK=>CLK, DET0=>N_2, DIGIT=>DIGIT,
                 QK(1 downto 0)=>QK(1 downto 0),
                 QM(1 downto 0)=>QM(1 downto 0), QSIG=>QSIG,
                 ROUND=>ROUND, QN(55 downto 0)=>QN(55 downto 0),
                 QP(55 downto 0)=>QP(55 downto 0),
                 QQ(55 downto 0)=>QQ(55 downto 0) );

end SCHEMATIC;

configuration CFG_CONVERT_SCHEMATIC of CONVERT is

   for SCHEMATIC
      for I_10: CR_QCALC
         use configuration WORK.CFG_CR_QCALC_BEHAVIORAL;
      end for;
      for I_7: ROUNDING
         use configuration WORK.CFG_ROUNDING_BEHAVIORAL;
```

```
      end for;
      for I_8: QUOTIENT
         use configuration WORK.CFG_QUOTIENT_BEHAVIORAL;
      end for;
   end for;

end CFG_CONVERT_SCHEMATIC;
```

-- VHDL code for combined unit: convert_app.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity CONVERT_APP is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               round : In    std_logic;
                  E1 : In    std_logic_vector(4 downto 0);
                  E2 : In    std_logic_vector(4 downto 0);
                   Q : Out   std_logic_vector(52 downto 0) );
end CONVERT_APP;

architecture SCHEMATIC of CONVERT_APP is

   CONSTANT n : integer := 4;
   signal     A,QS : std_logic_vector(3 downto 0);
   signal        T : std_logic;
   signal       QQ : std_logic_vector(55 downto 0);
   signal    B, Bn : std_logic_vector(n downto 0);
   signal  SIXTEEN : std_logic;

   component Q_regs
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               round : In    std_logic;
                   S : In    std_logic_vector (3 downto 0);
                   T : In    std_logic;
                  QK : In    std_logic_vector (3 downto 0);
                   Q : InOut   std_logic_vector (55 downto 0) );
   end component;

     component gl_reg is
        GENERIC(n : integer);
        Port (  A : In std_logic_vector (n downto 0);
                CLOCK : In std_logic;
                RESET : In std_logic;
                Z : Out std_logic_vector (n downto 0) );
   end component;

   component digit_update2 is
       GENERIC(n : integer);
        Port (  B : In std_logic_vector(n downto 0);
                QMS : In std_logic;
                A : Out std_logic_vector(3 downto 0);
                T : Out std_logic);
   end component;

   component digit_compute is
        GENERIC(n : integer);
        Port (  E1 : In std_logic_vector(n downto 0);
                E2 : In std_logic_vector(n downto 0);
                SS : Out std_logic_vector (3 downto 0);
                QK : Out std_logic_vector (n downto 0);
           SIXTEEN : Out std_logic--_vector(1 downto 0)
           );
   end component;

begin

   I_1 : digit_compute Generic Map(n=>4)
      Port Map ( E1=>E1, E2=>E2, SS =>QS,
                 QK=>BN, SIXTEEN=>SIXTEEN );

   REG_B: gl_reg Generic Map(n=>4)
```

```
        Port Map ( A=>BN, CLOCK=>CLK, RESET=>CLEAR, Z=>B );

   I_2 : digit_update2 Generic Map(n=>4)
     Port Map ( B=>B, QMS=>SIXTEEN, A=>A, T=>T );

   I_8 : Q_regs
      Port Map ( CLEAR=>CLEAR, CLK=>CLK,round=>round, S=>QS,
                               T=>T, QK(3 downto 0)=>A(3 downto 0), Q=>QQ );

  Q(52 downto 0)<= QQ(52 downto 0);

end SCHEMATIC;

configuration CFG_CONVERT_APP_SCHEMATIC of CONVERT_APP is

   for SCHEMATIC
      for I_8: Q_regs
         use configuration WORK.CFG_Q_regs_BEHAVIORAL;
      end for;
      for REG_B: gl_reg
         use configuration WORK.CFG_gl_reg_BEHAVIORAL;
      end for;
      for I_1: digit_compute
         use configuration WORK.CFG_digit_compute_BEHAVIORAL;
      end for;
      for I_2: digit_update2
         use configuration WORK.CFG_digit_update2_BEHAVIORAL;
      end for;
   end for;

end CFG_CONVERT_APP_SCHEMATIC;
```

-- VHDL for combined unit: cr_qcalc.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity CR_QCALC is
      Port (      M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
                  QK : Out   std_logic_vector (1 downto 0);
                  QM : Out   std_logic_vector (1 downto 0);
                QSIG : Out   std_logic;
                ZERO : Out   std_logic );
end CR_QCALC;

architecture BEHAVIORAL of CR_QCALC is

   begin
    process(M2,M1,P1,P2)
     begin

             if ( M2 = '1' ) then
                qk <= "10" ;
                qm <= "01" ;
                qsig <= '0' ;
                zero <= '0' ;

             elsif ( M1 = '1' ) then
                qk <= "01" ;
                qm <= "00" ;
                qsig <= '0' ;
                zero <= '0' ;

             elsif ( P1 = '1' ) then
                qk <= "11" ;
                qm <= "10" ;
                qsig <= '1' ;
                zero <= '0' ;
```

```
              elsif ( P2 = '1' ) then
                 qk <= "10" ;
                 qm <= "01" ;
                 qsig <= '1' ;
                 zero <= '0' ;

        else           -- qk = 0
                 qk <= "00" ;
                 qm <= "11" ;
                 qsig <= '0' ;
                 zero <= '1' ;
              end if;

   end process;

end BEHAVIORAL;

configuration CFG_CR_QCALC_BEHAVIORAL of CR_QCALC is
   for BEHAVIORAL

   end for;

end CFG_CR_QCALC_BEHAVIORAL;
```

-- VHDL code for combined unit csa42.vhd

```
library IEEE;
               use IEEE.std_logic_1164.all;

entity csa42 is
              GENERIC(n : integer);
              Port (       A : In std_logic_vector (n downto 0);
                           B : In std_logic_vector (n downto 0);
                           C : In std_logic_vector (n downto 0);
                           D : In std_logic_vector (n downto 0);
                           SCin : In std_logic;
                           CCin : In std_logic;
                           NCin : In std_logic;
                           Z : Out std_logic_vector (n downto 0);
                           Y : Out std_logic_vector (n downto 0) );
end csa42;

architecture BEHAVIORAL of csa42 is

 begin

process(A, B, C, D, SCin, CCin, NCin)
              variable pp : std_logic_vector (n downto 0) ;
              variable t : std_logic_vector (n downto 0) ;
              variable sum, carry : std_logic;
              variable i : integer;
begin

sum := (SCin xor CCin) xor NCin;
carry := (SCin and CCin) or (SCin and NCin) or (CCin and NCin);

t(0) := sum;
for i in 0 to n-1 loop
      t(i+1) := (A(i) AND B(i)) OR (A(i) AND C(i)) OR (C(i) AND B(i));
end loop;

for i in 0 to n loop
      pp(i) := (A(i) XOR B(i)) XOR C(i) ;
end loop;

-- CARRY -----------------------------------
Y(0) <= carry;
for i in 0 to n-1 loop

      Y(i+1) <= (D(i) AND t(i)) OR (D(i) AND pp(i)) OR (pp(i) AND t(i));

end loop;

-- SUM -------------------------------------
```

```
for i in 0 to n loop
             Z(i) <= pp(i) XOR (t(i) XOR D(i));
end loop;

end process;
end BEHAVIORAL;

configuration CFG_csa42_BEHAVIORAL of csa42 is
              for BEHAVIORAL
              end for;
end CFG_csa42_BEHAVIORAL;
```

-- VHDL for combined unit: digit_compute.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity digit_compute is
             GENERIC(n : integer); -- input bits
        Port (  E1 : In std_logic_vector(n downto 0);
                        E2 : In std_logic_vector(n downto 0);
                        SS : Out std_logic_vector (3 downto 0);
                QK : Out std_logic_vector (n downto 0);
                 SIXTEEN : Out std_logic--_vector (1 downto 0));
end digit_compute;

architecture BEHAVIORAL of digit_compute is

begin

process(E1, E2)
             variable p1 : std_logic_vector (n downto 0) ;
             variable p2 : std_logic_vector (n downto 0) ;
             variable p : std_logic_vector (n downto 0) ;
             variable g : std_logic_vector (n downto 0) ;
             variable c : std_logic_vector (n downto 0) ;
             variable s : std_logic_vector (n downto 0) ;
             variable a, b : std_logic;
             variable i : integer;

begin

c(0) := '0';
for i in 0 to n loop
        p(i) := E1(i) XOR E2(i) ;
        g(i) := E1(i) AND E2(i) ;
end loop;

-- CARRY (ripple) --------------------------
for i in 0 to n-1 loop
        c(i+1) := g(i) OR (c(i) AND p(i));
end loop;

-- SUM -------------------------------------
for i in 0 to n loop
  S(i) := p(i) XOR c(i);
end loop;
SIXTEEN <= s(4);
qk <=S(4 downto 0);
SS <=S(3 downto 0);
end process;

end BEHAVIORAL;

configuration CFG_digit_compute_BEHAVIORAL of digit_compute is
   for BEHAVIORAL
   end for;
end CFG_digit_compute_BEHAVIORAL;
```

-- VHDL for combined unit: digit_update2.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

entity digit_update2 is
    GENERIC(n : integer);
        Port (  B : In std_logic_vector(n downto 0);
                qms : In std_logic;
                A : Out std_logic_vector(3 downto 0);
                T : Out std_logic));
end digit_update2;

architecture BEHAVIORAL of digit_update2 is

   begin
       process(B,qms)

         begin

         if(qms /= B(0)) then
               A(3 downto 0) <= B(3 downto 0) + "0001";
               T <= '1';
         else
               A(3 downto 0) <= B(3 downto 0);
               T <= '0';
         end if;
       end process;

end BEHAVIORAL;

configuration CFG_digit_update2_BEHAVIORAL of digit_update2 is
   for BEHAVIORAL
   end for;
end CFG_digit_update2_BEHAVIORAL;
```

-- VHDL for combined unit: extend.vhd

```
library IEEE;
               use IEEE.std_logic_1164.all;

entity extend is
             Port (A : in std_logic_vector (55 downto 0);
                   Z : out std_logic_vector (57 downto 0));
end extend;

architecture BEHAVIORAL of extend is

 begin

   Z(55 downto 0)<= A(55 downto 0);

   process(A)
   begin

           if(A(55) = '1') then
              Z(57 downto 56) <= "11";
           else
              Z(57 downto 56) <= "00";
           end if;

   end process;

end BEHAVIORAL;

configuration CFG_extend_BEHAVIORAL of extend is
               for BEHAVIORAL
               end for;
end CFG_extend_BEHAVIORAL;
```

-- VHDL for combined unit: extendsc.vhd

```
library IEEE;
              use IEEE.std_logic_1164.all;

entity extendsc is
             Port (A : in std_logic_vector (55 downto 0);
                   B : in std_logic_vector (55 downto 0);
                   Y : out std_logic_vector (57 downto 0);
                    Z : out std_logic_vector (57 downto 0));
end extendsc;

architecture BEHAVIORAL of extendsc is

 begin

   Z(55 downto 0)<= A(55 downto 0);
   Y(55 downto 0)<= B(55 downto 0);

   process(A, B)
   begin

       if (A(55) = '1' and B(55) = '1') then
           Y(57 downto 56) <= "11";
           Z(57 downto 56) <= "00";
       else
           if(A(55) = '1') then
              Z(57 downto 56) <= "11";
           else
              Z(57 downto 56) <= "00";
           end if;
           if(B(55) = '1') then
           Y(57 downto 56) <= "11";
           else
           Y(57 downto 56) <= "00";
           end if;
       end if;
   end process;

end BEHAVIORAL;

configuration CFG_extendsc_BEHAVIORAL of extendsc is
              for BEHAVIORAL
              end for;
end CFG_extendsc_BEHAVIORAL;
```

-- VHDL for combined unit: gl_csa32.vhd

```
library IEEE;
              use IEEE.std_logic_1164.all;

entity gl_csa32 is
             GENERIC(n : integer);
             Port (       A : In std_logic_vector (n downto 0);
                          B : In std_logic_vector (n downto 0);
                          C : In std_logic_vector (n downto 0);
                          Cin : In std_logic;
                          Z : Out std_logic_vector (n downto 0);
                          Y : Out std_logic_vector (n downto 0) );
end gl_csa32;

architecture BEHAVIORAL of gl_csa32 is

 begin
process(A, B, C, Cin)
             variable p : std_logic_vector (n downto 0) ;
             variable g : std_logic_vector (n downto 0) ;
             variable i : integer;
begin
```

```
for i in 0 to n loop
             p(i) := A(i) XOR B(i) ;
             g(i) := A(i) AND B(i) ;
end loop;

-- CARRY ----------------------------------
Y(0) <= Cin;
for i in 0 to n-1 loop
             Y(i+1) <= g(i) OR (c(i) AND p(i));
end loop;

-- SUM ------------------------------------
for i in 0 to n loop
             Z(i) <= p(i) XOR c(i);
end loop;

end process;
end BEHAVIORAL;

configuration CFG_gl_csa32_BEHAVIORAL of gl_csa32 is
              for BEHAVIORAL
              end for;
end CFG_gl_csa32_BEHAVIORAL;
```

-- VHDL for combined unit: gl_dualreg_ld.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity gl_dualreg_ld is
     GENERIC(n : integer);
      Port (      AS : In    std_logic_vector (n downto 0);
                  AC : In    std_logic_vector (n downto 0);
                RESET : In    std_logic;
                CLOCK : In    std_logic;
                 LOAD : In    std_logic;
                   ZS : Out   std_logic_vector (n downto 0);
                   ZC : Out   std_logic_vector (n downto 0) );
end gl_dualreg_ld;

architecture BEHAVIORAL of gl_dualreg_ld is

begin

  process(reset,clock)
      begin

      if ( reset = '0' ) then
             ZS <= (others => '0');
             ZC <= (others => '0');

      elsif (( clock = '1' ) and (clock'EVENT)) then
           if ( load = '1' ) then
              ZS <= AS ;
              ZC <= AC ;
           end if;
      end if;

  end process;

end BEHAVIORAL;

configuration CFG_gl_dualreg_ld_BEHAVIORAL of gl_dualreg_ld is
   for BEHAVIORAL
   end for;
end CFG_gl_dualreg_ld_BEHAVIORAL;
```

-- VHDL for combined unit: gl_mux21.vhd

```
library IEEE;
              use IEEE.std_logic_1164.all;

entity gl_mux21 is
```

```
              GENERIC(n : integer);
              Port (        A0 : In std_logic_vector (n downto 0);
                            A1 : In std_logic_vector (n downto 0);
                            SEL : In std_logic;
                            Z : Out std_logic_vector (n downto 0) );
end gl_mux21;

architecture BEHAVIORAL of gl_mux21 is

 begin
   process(A0, A1, SEL)
   begin
     if ( SEL = '0' ) then
               Z <= A0;
     else
               Z <= A1;
     end if;
   end process;

end BEHAVIORAL;

configuration CFG_gl_mux21_BEHAVIORAL of gl_mux21 is
               for BEHAVIORAL
               end for;
end CFG_gl_mux21_BEHAVIORAL;
```

-- VHDL for combined unit: gl_reg.vhd

```
library IEEE;
               use IEEE.std_logic_1164.all;

entity gl_reg is
              GENERIC(n : integer);
              Port (        A : In std_logic_vector (n downto 0);
                            CLOCK : In std_logic;
                            RESET : In std_logic;
                            Z : Out std_logic_vector (n downto 0) );
end gl_reg;

architecture BEHAVIORAL of gl_reg is

 begin
   process(clock, reset)
   begin
     if ( reset = '0' ) then
               Z <= (others => '0');
     elsif (( clock = '1' ) and (clock'EVENT)) then
               Z <= A;
     end if;
   end process;

end BEHAVIORAL;

configuration CFG_gl_reg_BEHAVIORAL of gl_reg is
               for BEHAVIORAL
               end for;
end CFG_gl_reg_BEHAVIORAL;
```

-- VHDL for combined unit: init.vhd

```
library IEEE;
               use IEEE.std_logic_1164.all;

entity init is
              Port (        D : In std_logic_vector (52 downto 0);
                            ED : In std_logic;
                            OP : In std_logic;
                            M  : In std_logic;
                            ZP : Out std_logic_vector (3 downto 0);
                            ZW : Out std_logic_vector (55 downto 0);
                            ZH : Out std_logic_vector (55 downto 0) );
end init;
```

```
architecture BEHAVIORAL of init is

 begin

   process(D, ED, OP, M)
   variable HFD, DF: std_logic_vector (55 downto 0);
   begin

     HFD(55 downto 52) := "0000"; HFD(51 downto 0) := D(52 downto 1);--d/2
     DF(55 downto 51) := "00000"; DF(50 downto 0):=D(52 downto 2);--d/4

     if (OP = '0') then--reciprocal operation
        if (M = '1') then --d>=0.75, E[0]= 2-(d/2), W[0]=1-(d/2)
           ZW(55 downto 52) <= HFD(55 downto 52);
           ZW(51 downto 0) <= NOT HFD(51 downto 0);
           ZH(55 downto 53) <= HFD(55 downto 53);
           ZH(52 downto 0) <= NOT HFD(52 downto 0);
           ZP <= "0100";
        else --d<0.75, E[0]= 4-2d, W[0]=1-d
           ZW(55 downto 52) <= "1111"; ZW(51 downto 0) <= NOT D(51 downto 0);
           ZH(55 downto 54) <= "00"; ZH(53 downto 1) <= NOT D(52 downto 0); ZH(0) <= '1';
           ZP <= "1000";
         end if;
      else --square-root reciprocal operation
         if (ED = '1') then --d>=0.5, H[0]= 3/2 -(d/4), W[0]=1/2 -(d/4)
            ZW(55 downto 51) <= DF(55 downto 51);
            ZW(50 downto 0) <= NOT DF(50 downto 0);
            ZH(55 downto 51) <= "00010"; ZH(50 downto 0) <= NOT DF(50 downto 0);
            ZP <= "0100";
          else --d<0.5, H[0]= 3 - d, W[0]=1/2 -(d/2)
            ZW(55 downto 51) <= "11111";
            ZW(50 downto 0) <= NOT HFD(50 downto 0);
            ZH(55 downto 52) <= "0001"; ZH(51 downto 0) <= NOT D(51 downto 0);
            ZP <= "1000";
          end if;
      end if;
   end process;

end BEHAVIORAL;

configuration CFG_init_BEHAVIORAL of init is
               for BEHAVIORAL
               end for;
end CFG_init_BEHAVIORAL;
```

-- VHDL for combined unit: mult2.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;

entity MULT2 is
      GENERIC(n : integer);
      Port (       A : In    std_logic_vector (n downto 0);
                  Ap : In    std_logic; -- A(-1)
                  M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
                COUT : Out   std_logic;
                   Z : Out   std_logic_vector (n downto 0) );
end MULT2;

architecture BEHAVIORAL of MULT2 is

   begin
    process (M1, M2, P1, P2, A, Ap )
     variable    pd : std_logic;

      begin

        cout <= M1 or M2 ;
```

```
        Z(0) <= ( M2 AND NOT(Ap) ) OR ( M1 AND NOT(A(0)) ) OR
                ( P1 AND A(0) )    OR ( P2 AND Ap ) ;
        for i in 1 to n loop
            pd := A(i-1);
            Z(i) <= ( M2 AND NOT(pd) ) OR ( M1 AND NOT(A(i)) ) OR
                    ( P1 AND A(i) )    OR ( P2 AND pd ) ;
        end loop;

      end process;

end BEHAVIORAL;

configuration CFG_MULT2_BEHAVIORAL of MULT2 is
   for BEHAVIORAL

   end for;

end CFG_MULT2_BEHAVIORAL;
```

-- VHDL for combined unit: mult4.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity MULT4 is
      GENERIC(n : integer);
      Port (      A : In    std_logic_vector (n downto 0);
                  M1 : In    std_logic;
                  M2 : In    std_logic;
                  P1 : In    std_logic;
                  P2 : In    std_logic;
                  Cout : out    std_logic;
                  Z    : Out   std_logic_vector (n downto 0) );
end MULT4;

architecture BEHAVIORAL of MULT4 is

   begin

    process (M1, M2, P1, P2, A)
      variable p: std_logic_vector (3 downto 0);
      begin
      p:= m2 & m1 & p1 & p2;
      case p is
           WHEN "0001" =>
                Z(n downto 2) <= NOT(A(n-2 downto 0)); Z(1) <='1'; Z(0) <='1';
                Cout <= '1';
           WHEN "0010" =>
                Z <= NOT(A);
                Cout <= '1';
           WHEN "0100" =>
                Z <= NOT(A);
                Cout <= '1';
           WHEN "1000" =>
                Z(n downto 2) <= NOT(A(n-2 downto 0)); Z(1) <='1'; Z(0) <='1';
                Cout <= '1';
           WHEN OTHERS =>
                Z(n downto 0) <=(others => '0');
                Cout <= '0';
      end case;
      end process;



end BEHAVIORAL;

configuration CFG_MULT4_BEHAVIORAL of MULT4 is
   for BEHAVIORAL

   end for;

end CFG_MULT4_BEHAVIORAL;
```

-- VHDL for combined unit: mux2.vhd

```
library IEEE;
              use IEEE.std_logic_1164.all;

entity mux2 is
             GENERIC(n : integer);
             Port (       A0 : In std_logic_vector (n downto 0);
                          A1 : In std_logic_vector (n downto 0);
                          SEL : In std_logic;
                          Cout : Out std_logic;
                          Z : Out std_logic_vector (n downto 0) );
end mux2;

architecture BEHAVIORAL of mux2 is

 begin
   process(A0, A1, SEL)
   begin
     if ( SEL = '0' ) then
               Z <= A0;
               Cout <= '0';
     else
               Z <= A1;
               Cout <= '1';
     end if;
   end process;

end BEHAVIORAL;

configuration CFG_mux2_BEHAVIORAL of mux2 is
              for BEHAVIORAL
              end for;
end CFG_mux2_BEHAVIORAL;
```

-- VHDL for combined unit: q_regs.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

entity q_regs is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
               round : In    std_logic;
                   T : In    std_logic;--_vector(1 downto 0);
                  QK : In    std_logic_vector (3 downto 0);
                   Q : InOut   std_logic_vector (55 downto 0) );
end q_regs;

architecture BEHAVIORAL of q_regs is

   begin
    process(clear,clk)
     variable qq, qn, qp : std_logic_vector (55 downto 0) ;
     variable qplus1, qminus1 : std_logic_vector (3 downto 0) ;
     variable carry: std_logic;
     variable i: integer;
     begin

       if ( clear = '0' ) then
          for i in 0 to 55 loop
              q(i) <= '0';
              qq(i) := '0';
              qn(i) := '0';
              qp(i) := '0';
          end loop;
       elsif ((clk = '1') AND (clk'EVENT)) then

         if((t ='1') and qk = "0000" )then
```

```
             for i in 55 downto 4 loop
            qq(i) := qp(i-4);
            qn(i) := qp(i-4);
             qp(i) := qp(i-4);
          end loop;
       elsif ((t ='0') and qk = "1111" )then
          for i in 55 downto 4 loop
            qq(i) := qq(i-4);
            qn(i) := qq(i-4);
            qp(i) := qp(i-4);
           end loop;
       else
        for i in 55 downto 4 loop
           qq(i) := qq(i-4);
           qn(i) := qq(i-4);
           qp(i) := qq(i-4);
        end loop;
      end if;
                --insert last digit
                qplus1  := qk + "0001";
                qminus1 := qk + "1111";
                qq(3 downto 0) := qk;
                qn(3 downto 0) := qminus1;
                qp(3 downto 0) := qplus1;

                ------rounding-------
                if(round = '1') then
                   carry := s(3) and '1';
                   if(carry = '1') then
                      Q <= qp;
                   else
                     Q <= qq;
                  end if;
               end if;
    end if;--end if clock
    end process;

end BEHAVIORAL;

configuration CFG_q_regs_BEHAVIORAL of q_regs is
   for BEHAVIORAL
   end for;

end CFG_q_regs_BEHAVIORAL;
```

-- VHDL for combined unit: QDSEL.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
entity QDSEL is
      Port (       Y : In    std_logic_vector (6 downto 0);
                   D : In    std_logic_vector (3 downto 0);
                  M1 : Out   std_logic;
                  M2 : Out   std_logic;
                  P1 : Out   std_logic;
                  P2 : Out   std_logic );
end QDSEL;

architecture BEHAVIORAL of QDSEL is

   begin

  process(Y, D)
    TYPE table_type IS ARRAY(0 to 3) OF std_logic_vector (6 downto 0);
--  m_2=12, m_1=3, m_0=-5, m_-1=-13
    CONSTANT sel_fun0  : table_type := (
                                       "0001100", "0000011", "1111011", "1110011" );
--  m_2=13, m_1=4, m_0=-5, m_-1=-14
    CONSTANT sel_fun1  : table_type := (
                                       "0001101", "0000100", "1111011", "1110010" );
--  m_2=14, m_1=4, m_0=-5, m_-1=-14
    CONSTANT sel_fun2  : table_type := (
                                       "0001110", "0000100", "1111011", "1110010" );
```

```
--  m_2=14, m_1=4, m_0=-6, m_-1=-15
    CONSTANT sel_fun3  : table_type := (
                                       "0001110", "0000100", "1111010", "1110001" );
--  m_2=15, m_1=4, m_0=-6, m_-1=-16
    CONSTANT sel_fun4  : table_type := (
                                       "0001111", "0000100", "1111010", "1110000" );
--  m_2=15, m_1=4, m_0=-6, m_-1=-17
    CONSTANT sel_fun5  : table_type := (
                                       "0001111", "0000100", "1111010", "1101111" );
--  m_2=16, m_1=4, m_0=-7, m_-1=-18
    CONSTANT sel_fun6  : table_type := (
                                       "0010000", "0000100", "1111001", "1101110" );
--  m_2=17, m_1=5, m_0=-7, m_-1=-18
    CONSTANT sel_fun7  : table_type := (
                                       "0010001", "0000101", "1111001", "1101110" );
--  m_2=17, m_1=5, m_0=-7, m_-1=-19
    CONSTANT sel_fun8  : table_type := (
                                       "0010001", "0000101", "1111001", "1101101" );
--  m_2=18, m_1=7, m_0=-8, m_-1=-19
    CONSTANT sel_fun9  : table_type := (
                                       "0010010", "0000111", "1111000", "1101101" );
--  m_2=19, m_1=7, m_0=-8, m_-1=-20
    CONSTANT sel_funa  : table_type := (
                                       "0010011", "0000111", "1111000", "1101100" );
--  m_2=19, m_1=6, m_0=-8, m_-1=-21
    CONSTANT sel_funb  : table_type := (
                                       "0010011", "0000110", "1111000", "1101011" );
--  m_2=20, m_1=5, m_0=-9, m_-1=-23
    CONSTANT sel_func  : table_type := (
                                       "0010100", "0000101", "1110111", "1101001" );
--  m_2=21, m_1=6, m_0=-9, m_-1=-23
    CONSTANT sel_fund  : table_type := (
                                       "0010101", "0000110", "1110111", "1101001" );
--  m_2=21, m_1=8, m_0=-9, m_-1=-24
    CONSTANT sel_fune  : table_type := (
                                       "0010101", "0001000", "1110111", "1101000" );
--  m_2=22, m_1=8, m_0=-10, m_-1=-25
    CONSTANT sel_funf  : table_type := (
                                       "0010110", "0001000", "1110110", "1100111" );
    variable i : integer;


    begin

-- Selection function =====================================================
        if ( D = "0000" ) then
                 if ( y(6) = '0' ) then   ------- y is positive
                          if ( y >= sel_fun0(0) ) then
                             M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                          elsif ( y < sel_fun0(1) ) then
                             M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                          else
                             M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                          end if;
                 else                      ------- y is negative
                          if ( y < sel_fun0(3) ) then
                             M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                          elsif ( y >= sel_fun0(2) ) then
                             M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                          else
                             M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                          end if;
                 end if;
        elsif ( D = "0001" ) then
                 if ( y(6) = '0' ) then   ------- y is positive
                          if ( y >= sel_fun1(0) ) then
                             M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                          elsif ( y < sel_fun1(1) ) then
                             M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                          else
                             M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                          end if;
                 else                      ------- y is negative
                          if ( y < sel_fun1(3) ) then
```

```
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun1(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "0010" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun2(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun2(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                    ------- y is negative
                if ( y < sel_fun2(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun2(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "0011" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun3(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun3(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                    ------- y is negative
                if ( y < sel_fun3(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun3(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "0100" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun4(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun4(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                    ------- y is negative
                if ( y < sel_fun4(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun4(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "0101" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun5(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun5(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                    ------- y is negative
                if ( y < sel_fun5(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun5(2) ) then
```

```
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "0110" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun6(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun6(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                     ------- y is negative
                if ( y < sel_fun6(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun6(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "0111" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun7(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun7(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                     ------- y is negative
                if ( y < sel_fun7(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun7(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "1000" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun8(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun8(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                     ------- y is negative
                if ( y < sel_fun8(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun8(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                end if;
        end if;
elsif ( D = "1001" ) then
        if ( y(6) = '0' ) then   ------- y is positive
                if ( y >= sel_fun9(0) ) then
                    M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                elsif ( y < sel_fun9(1) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
                    M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                end if;
        else                     ------- y is negative
                if ( y < sel_fun9(3) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                elsif ( y >= sel_fun9(2) ) then
                    M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                else
```

```
                        M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
            end if;
    elsif ( D = "1010" ) then
            if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_funA(0) ) then
                        M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_funA(1) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
            else                      ------- y is negative
                    if ( y < sel_funA(3) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_funA(2) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
            end if;
    elsif ( D = "1011" ) then
            if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_funB(0) ) then
                        M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_funB(1) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
            else                      ------- y is negative
                    if ( y < sel_funB(3) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_funB(2) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
            end if;
    elsif ( D = "1100" ) then
            if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_funC(0) ) then
                        M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_funC(1) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
            else                      ------- y is negative
                    if ( y < sel_funC(3) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_funC(2) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
            end if;
    elsif ( D = "1101" ) then
            if ( y(6) = '0' ) then   ------- y is positive
                    if ( y >= sel_funD(0) ) then
                        M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    elsif ( y < sel_funD(1) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                    end if;
            else                      ------- y is negative
                    if ( y < sel_funD(3) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                    elsif ( y >= sel_funD(2) ) then
                        M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                    else
                        M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                    end if;
```

```
                end if;
        elsif ( D = "1110" ) then
                if ( y(6) = '0' ) then   ------- y is positive
                        if ( y >= sel_funE(0) ) then
                           M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        elsif ( y < sel_funE(1) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                        end if;
                else                     ------- y is negative
                        if ( y < sel_funE(3) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                        elsif ( y >= sel_funE(2) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                        end if;
                end if;
        elsif ( D = "1111" ) then
                if ( y(6) = '0' ) then   ------- y is positive
                        if ( y >= sel_funF(0) ) then
                           M2 <= '1'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        elsif ( y < sel_funF(1) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '1'; P1 <= '0'; P2 <= '0';
                        end if;
                else                     ------- y is negative
                        if ( y < sel_funF(3) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '1';
                        elsif ( y >= sel_funF(2) ) then
                           M2 <= '0'; M1 <= '0'; P1 <= '0'; P2 <= '0';
                        else
                           M2 <= '0'; M1 <= '0'; P1 <= '1'; P2 <= '0';
                        end if;
                end if;
        end if;

  end process;

end BEHAVIORAL;

configuration CFG_QDSEL_BEHAVIORAL of QDSEL is
   for BEHAVIORAL

   end for;

end CFG_QDSEL_BEHAVIORAL;
```

-- VHDL for combined unit: QLATCH.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity QLATCH is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
                  I1 : In    std_logic;
                  I2 : In    std_logic;
                  J1 : In    std_logic;
                  J2 : In    std_logic;
                LOAD : In    std_logic;
                  M1 : Out   std_logic;
                  M2 : Out   std_logic;
                  P1 : Out   std_logic;
                  P2 : Out   std_logic );
end QLATCH;

architecture BEHAVIORAL of QLATCH is

   begin
```

```
     process(clear,clk)
      begin

      if ( clear = '0' ) then
              M2 <= '0';
              M1 <= '0';
              P2 <= '0';
              P1 <= '0'; -- must be 1

        elsif (( clk = '1' ) and (clk'EVENT)) then
           if ( load = '1' ) then
              M2 <= I2 ;
              M1 <= I1 ;
              P2 <= J2 ;
              P1 <= J1 ;
           end if;
      end if;

     end process;

end BEHAVIORAL;

configuration CFG_QLATCH_BEHAVIORAL of QLATCH is
   for BEHAVIORAL

   end for;

end CFG_QLATCH_BEHAVIORAL;
```

-- VHDL code for combined unit: QUOTIENT.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;

entity QUOTIENT is
      Port (   CLEAR : In    std_logic;
                 CLK : In    std_logic;
                DET0 : In    std_logic;
               DIGIT : In    std_logic;
                  QK : In    std_logic_vector (1 downto 0);
                  QM : In    std_logic_vector (1 downto 0);
                QSIG : In    std_logic;
               ROUND : In    std_logic;
                  QN : InOut std_logic_vector (55 downto 0);
                  QP : InOut std_logic_vector (55 downto 0);
                  QQ : InOut std_logic_vector (55 downto 0) );
end QUOTIENT;

architecture BEHAVIORAL of QUOTIENT is

   begin
    process(clear,clk)
              variable flag : std_logic;
              variable last : std_logic;
     begin

       if ( clear = '0' ) then
          for i in 0 to 55 loop
              qq(i) <= '0';
              qn(i) <= '0';
              qp(i) <= '0';
          end loop;

       elsif ((clk = '1') AND (clk'EVENT)) then
          if ( digit = '1' ) then
                if ( det0 = '1' ) then
                        for i in 55 downto 2 loop
                          qq(i) <= qq(i-2);
                          qn(i) <= qn(i-2);
                        end loop;
```

```
                elsif ( qsig = '0') then
                        for i in 55 downto 2 loop
                          qq(i) <= qq(i-2);
                          qn(i) <= qq(i-2);
                        end loop;
                elsif ( qsig = '1') then
                        for i in 55 downto 2 loop
                          qq(i) <= qn(i-2);
                          qn(i) <= qn(i-2);
                        end loop;
                end if;
                for i in 0 to 1 loop
                  qq(i) <= qk(i);
                  qn(i) <= qm(i);
                end loop;
-- --------------------- addition for variable rounding --------------------

                           qp(1) <= qk(1) xor qk(0);
                           qp(0) <= not qk(0);

                           last := qq(52) or qq(51);
                           if (last = '1') then
                             if ( qq(52) = '0' ) then -- round in last position

                               if (( qk = "10" ) and ( QSIG = '1') ) then
                                         for i in 55 downto 2 loop
                                           qp(i) <= qn(i-2);
                                         end loop;
                               else
                                         for i in 55 downto 2 loop
                                           qp(i) <= qq(i-2);
                                         end loop;
                               end if;

                             else -- round last-1 position

                               if ( ( qk = "10" ) AND ( QSIG = '0') ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qp(i-2);
                           end loop;
                               elsif ( ( QSIG = '1') ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qq(i-2);
                           end loop;
                       else
                           -- nop
                       end if;
                                end if;

                              else -- it is not the last iteration

                       if ( ( qk = "11" ) ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qp(i-2);
                           end loop;
                       elsif ( ( qk = "10" ) AND ( QSIG = '1') ) then
                           for i in 55 downto 2 loop
                             qp(i) <= qn(i-2);
                           end loop;
                       else
                           for i in 55 downto 2 loop
                             qp(i) <= qq(i-2);
                           end loop;
                       end if;

                              end if;
---------------------- end addition for variable rounding ----------------

                    else
                for i in 0 to 55 loop
                   qq(i) <= '0';
                   qn(i) <= '0';
                   qp(i) <= '0';
                end loop;
```

```
         end if;
       end if;

    end process;

end BEHAVIORAL;

configuration CFG_QUOTIENT_BEHAVIORAL of QUOTIENT is
   for BEHAVIORAL

   end for;

end CFG_QUOTIENT_BEHAVIORAL;
```

-- VHDL for combined unit: rounding.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

entity ROUNDING is
      Port (     CLK : In    std_logic;
                  QN : In    std_logic_vector (55 downto 0);
                  QP : In    std_logic_vector (55 downto 0);
                  QQ : In    std_logic_vector (55 downto 0);
               ROUND : In    std_logic;
                SIGN : In    std_logic;
                   Q : Out   std_logic_vector (52 downto 0) );
end ROUNDING;

architecture BEHAVIORAL of ROUNDING is

   begin
    process(clk)
    variable signn : std_logic ;
    variable p : std_logic_vector (2 downto 0) ;
    variable mqq,mqn,mqp : std_logic_vector (55 downto 0) ;

     begin
      if (( clk = '1' ) AND (clk'EVENT)) then
        if ( round = '1' ) then
             p := "000";
             mqq := QQ;
             mqn := Qn;
             mqp := Qp;
             signn := SIGN;

        if ( qq(54) = '0')  then
                if ( signn = '0' ) then
                          -- add 1
                          p(0) := NOT qq(0);
                          p(1) := qq(0) XOR qq(1);
                          p(2) := qq(0) AND qq(1);
                else
                          p(0) := qq(0);
                          p(1) := qq(1);
                          p(2) := '0';
                end if;

                if ( p >= "100" ) then
                   for i in 0 to 52 loop
                      Q(i) <= qp(i+1);
                   end loop;
           else
                   for i in 1 to 52 loop
                      Q(i) <= qq(i+1);
                   end loop;
                             Q(0) <= p(1);
           end if;

-- new part introduced for variable rounding
              else -- qq(54) = '1'
```

```
                if ( signn = '0' ) then
                          -- add 2
                          p(0) := qq(0);
                          p(1) := NOT qq(1);
                          p(2) := qq(1);
                else
                          -- subtract 1 add 2 => add 1
                          p(0) := NOT qq(0);
                          p(1) := qq(0) XOR qq(1);
                          p(2) := qq(0) AND qq(1);
                end if;

                if ( p >= "100" ) then
                  for i in 0 to 52 loop
                     Q(i) <= qp(i+2);
                  end loop;
           elsif (( qq(55) = '0')  or
              (( qq(1) = '1') and ( qq(0) = '1') and ( signn = '0' ))) then
                  for i in 0 to 52 loop
                     Q(i) <= qq(i+2);
                  end loop;
           else
                  for i in 0 to 52 loop
                     Q(i) <= qn(i+2);
                  end loop;
           end if;
         end if; -- close qq(54) = '0'
       end if; -- close round = '1'
       end if;  -- close CLK

    end process;

end BEHAVIORAL;

configuration CFG_ROUNDING_BEHAVIORAL of ROUNDING is
   for BEHAVIORAL

   end for;

end CFG_ROUNDING_BEHAVIORAL;
```

-- VHDL code for combined unit: SDET.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;

entity SDET is
      Port (       A : In    std_logic_vector (55 downto 0);
                   B : In    std_logic_vector (55 downto 0);
                   Z : Out   std_logic );
end SDET;

architecture BEHAVIORAL of SDET is

begin
process(a,b)
variable    d : std_logic;
begin

d := '0';
for i in 0 to 54 loop
  d := ( A(i) AND B(i) ) OR ( A(i) AND D ) OR ( D AND B(i) ) ;
end loop;
Z <= A(55) XOR B(55) XOR D ;

end process;

end BEHAVIORAL;

configuration CFG_SDET_BEHAVIORAL of SDET is
   for BEHAVIORAL
   end for;

end CFG_SDET_BEHAVIORAL;
```

-- VHDL code for combined unit: tb2_radix4.vhd

```
library IEEE;
              use IEEE.std_logic_1164.all;
              use STD.textio.all;
              use IEEE.std_logic_textio.all;
              use IEEE.std_logic_signed.all;

entity E is
end E;

architecture A of E is

   signal    CLOCK : std_logic;
   signal        D : std_logic_vector (52 downto 0);
   signal    RESET : std_logic;
   signal       OP : std_logic;
   signal       ED : std_logic;
   signal        Q : std_logic_vector (52 downto 0);

   component radix4
      Port (   CLOCK : In    std_logic;
                   D : In    std_logic_vector (52 downto 0);
               RESET : In    std_logic;
               OP    : In    std_logic;
                  ED : std_logic;--++
                   Q : Out   std_logic_vector (52 downto 0) );
   end component;

begin
   UUT : radix4
      Port Map ( CLOCK, D, RESET, OP, ED, Q );

   TB : block
   begin
   process
              CONSTANT NLOOPS : integer := 15;
              --CONSTANT NLOOPS : integer := 29;
              file cmdfile1, cmdfile2, cmdfile: TEXT;                -- Define the file 'handle'
              variable line_in,line_out: Line; -- Line buffers
              variable good: boolean;     -- Status of the read operations
              variable A: std_logic_vector(55 downto 0);
              variable S: std_logic_vector(55 downto 0);
              variable Z: std_logic_vector(55 downto 0);
              variable ERR: std_logic_vector(55 downto 0);
              variable c : integer;

              begin

  op <='0';--Reciprocal computation
  FILE_OPEN(cmdfile1,"testvecs1.in",READ_MODE);
  c := 1;
  ------------------------------------------------------------------------
  write(line_out, string'("--------- RECIPROCAL SIMULATION START ------------"));
  writeline(OUTPUT,line_out);
  loop
    if endfile(cmdfile1) then  -- Check EOF
              assert false
                 report "End of file encountered; exiting."
                 severity NOTE;
              exit;
    end if;

  reset <= '0';
  clock <= '1'; wait for 5 ns;
  reset <= '1';
  clock <= '0'; wait for 5 ns;


    readline(cmdfile1,line_in);     -- Read a line from the file
    next when line_in'length = 0;  -- Skip empty lines
```

```
    hread(line_in,A,good);         -- Read the X argument as hex value
    assert good report "Text I/O read error" severity ERROR;


    hread(line_in,S,good);         -- Read the Q argument as hex value
    assert good report "Text I/O read error" severity ERROR;

    D(52 downto 0)  <= A(52 downto 0);

  for i in 0 to NLOOPS loop
        clock <= '1'; wait for 5 ns;
        clock <= '0'; wait for 5 ns;
  end loop;

    Z(55) := '0'; Z(54) := '0'; Z(53) := '0';
    Z(52 downto 0)  := Q(52 downto 0);
    ERR := S-Z;

    write(line_out, string'("Test "));
    write(line_out,c);
    if (Z = S) then
        write(line_out, string'(" passed: "));
              write(line_out, string'(" 1/ "));
              hwrite(line_out,A,RIGHT,14);

              --hwrite(line_out,B,RIGHT,14);
              write(line_out, string'(" -> "));
              hwrite(line_out,Z,RIGHT,14);
    else
        write(line_out, string'(" FAILED: "));
              write(line_out, string'(" 1/ "));
              hwrite(line_out,A,RIGHT,14);

              --hwrite(line_out,B,RIGHT,14);
              write(line_out, string'(" -> "));
              hwrite(line_out,Z,RIGHT,14);
        write(line_out, string'(" <> "));
              hwrite(line_out,S,RIGHT,14);
              writeline(OUTPUT,line_out);
        write(line_out, string'("Test "));
        write(line_out,c);
              write(line_out, string'("                        ERR "));
              hwrite(line_out,ERR,RIGHT,14);
    end if;
    writeline(OUTPUT,line_out);     -- write the message
    assert (Z = S) report "Z does not match in pattern " severity error;
    c := c + 1;
  end loop;
write(line_out, string'("--------- END RECIPROCAL SIMULATION ------------"));
writeline(OUTPUT,line_out);
-- ------------------------------------------------------------------------
op <='1'; --Square-root reciprocal computation
ED <='1';--D > 0.5
FILE_OPEN(cmdfile,"testvecs3.in",READ_MODE);--++
c := 1;
write(line_out, string'("---------SQUARE-ROOT RECIPROCAL SIMULATION START ------------"));
writeline(OUTPUT,line_out);
  ------------------------------------------------------------------------
  loop
    if endfile(cmdfile) then  -- Check EOF
              assert false
                 report "End of file encountered; exiting."
                 severity NOTE;
              exit;
    end if;

  reset <= '0';
  clock <= '1'; wait for 5 ns;
  reset <= '1';
  clock <= '0'; wait for 5 ns;


    readline(cmdfile,line_in);     -- Read a line from the file
    next when line_in'length = 0;  -- Skip empty lines
```

```
  hread(line_in,A,good);         -- Read the X argument as hex value
  assert good report "Text I/O read error" severity ERROR;


  hread(line_in,S,good);         -- Read the Q argument as hex value
  assert good report "Text I/O read error" severity ERROR;

  D(52 downto 0)  <= A(52 downto 0);

for i in 0 to NLOOPS loop
      clock <= '1'; wait for 5 ns;
      clock <= '0'; wait for 5 ns;
end loop;

  Z(55) := '0'; Z(54) := '0'; Z(53) := '0';
  Z(52 downto 0)  := Q(52 downto 0);
  ERR := S-Z;

  write(line_out, string'("Test "));
  write(line_out,c);
  if (Z = S) then
      write(line_out, string'(" passed: "));
            write(line_out, string'(" 1/sqrt( "));
            hwrite(line_out,A,RIGHT,14);
            write(line_out, string'(" )"));

            --hwrite(line_out,B,RIGHT,14);
            write(line_out, string'(" -> "));
            hwrite(line_out,Z,RIGHT,14);
  else
      write(line_out, string'(" FAILED: "));
            write(line_out, string'(" 1/sqrt( "));
            hwrite(line_out,A,RIGHT,14);
            write(line_out, string'(" )"));

            --hwrite(line_out,B,RIGHT,14);
            write(line_out, string'(" -> "));
            hwrite(line_out,Z,RIGHT,14);
      write(line_out, string'(" <> "));
            hwrite(line_out,S,RIGHT,14);
            writeline(OUTPUT,line_out);
      write(line_out, string'("Test "));
      write(line_out,c);
            write(line_out, string'("                       ERR "));
            hwrite(line_out,ERR,RIGHT,14);
  end if;
  writeline(OUTPUT,line_out);     -- write the message
  assert (Z = S) report "Z does not match in pattern " severity error;
  c := c + 1;
end loop;

ED <='0';--D < 0.5
FILE_OPEN(cmdfile2,"testvecs2.in",READ_MODE);--++
c := 1;
------------------------------------------------------------------------
loop
  if endfile(cmdfile2) then  -- Check EOF
            assert false
               report "End of file encountered; exiting."
               severity NOTE;
            exit;
  end if;

reset <= '0';
clock <= '1'; wait for 5 ns;
reset <= '1';
clock <= '0'; wait for 5 ns;


  readline(cmdfile2,line_in);     -- Read a line from the file
  next when line_in'length = 0;  -- Skip empty lines

  hread(line_in,A,good);         -- Read the X argument as hex value
```

```
    assert good report "Text I/O read error" severity ERROR;


    hread(line_in,S,good);         -- Read the Q argument as hex value
    assert good report "Text I/O read error" severity ERROR;

    D(52 downto 0)  <= A(52 downto 0);

  for i in 0 to NLOOPS loop
        clock <= '1'; wait for 5 ns;
        clock <= '0'; wait for 5 ns;
  end loop;

    Z(55) := '0'; Z(54) := '0'; Z(53) := '0';
    Z(52 downto 0)  := Q(52 downto 0);
    ERR := S-Z;

    write(line_out, string'("Test "));
    write(line_out,c);
    if (Z = S) then
        write(line_out, string'(" passed: "));
              write(line_out, string'(" 1/sqrt( "));
              hwrite(line_out,A,RIGHT,14);
              write(line_out, string'(" )"));

              --hwrite(line_out,B,RIGHT,14);
              write(line_out, string'(" -> "));
              hwrite(line_out,Z,RIGHT,14);
    else
        write(line_out, string'(" FAILED: "));
              write(line_out, string'(" 1/sqrt( "));
              hwrite(line_out,A,RIGHT,14);
              write(line_out, string'(" )"));

              --hwrite(line_out,B,RIGHT,14);
              write(line_out, string'(" -> "));
              hwrite(line_out,Z,RIGHT,14);
        write(line_out, string'(" <> "));
              hwrite(line_out,S,RIGHT,14);
              writeline(OUTPUT,line_out);
        write(line_out, string'("Test "));
        write(line_out,c);
              write(line_out, string'("                        ERR "));
              hwrite(line_out,ERR,RIGHT,14);
    end if;
    writeline(OUTPUT,line_out);     -- write the message
    assert (Z = S) report "Z does not match in pattern " severity error;
    c := c + 1;
  end loop;
write(line_out, string'("--------- END SQUARE-ROOT RECIPROCAL SIMULATION ------------"));
writeline(OUTPUT,line_out);
-- ================================================================
  clock <= '1'; wait for 1000 ns;
  assert q = "1111111111111111111111111111111111111111111111111111111111111111"
    report "--------- END SIMULATION  ------------------ " severity error;
-- ================================================================
   end process;
 end block;
end A;

configuration CFG_TB_radix4_BEHAVIORAL of E is
   for A
      for UUT : radix4
         use configuration WORK.CFG_radix4_SCHEMATIC;
      end for;

      for TB
      end for;

   end for;
end CFG_TB_radix4_BEHAVIORAL;
```