Radix 4 SRT Division with Quotient Prediction and Operand Scaling

Nishant R Srivastava

Illinois Institute of Technology
Chicago, Illinois 60616, USA

Abstract

SRT division is an efficient method for implementing high radix division circuits. However, as the radix increases the size of a quotient digit selection table increases exponentially. To overcome the limitations of quotient prediction, a method in which a quotient digit is speculated has been proposed. The speculated quotient digit is utilized to update the possible partial remainders while the speculated quotient is corrected. In this paper, instead of using a huge quotient selection table an estimation and correction scheme is used for prediction of quotient digit. The prediction is done in parallel with the calculation of the partial remainder for the quotient predicted earlier thus improving the latency. In addition, since this method tends to consume less area as the radix increases compared to previous methods, it has the ability to improve higher radix implementations for SRT division.

I. Introduction

Many general-purpose microprocessors require an emphasis on functional units that can compute addition, subtraction, multiplication, division, and square root with relatively small number of cycles. While many advancements within these functional units has been placed on improving addition and multiplication, there still remains plenty of advancements that can be made for division and square root. In fact, data based on SpecFP benchmarks has shown that for the slowest division hardware with a latency of 60 cycles or more, a cycle per instruction (CPI) penalty up to 0.50 can be incurred [1]. Consequently, finding a cost-effective implementation for division that can compute a quotient in a relatively small number of cycles is beneficial in many applications.

There are many implementations for computing division and square root, however, the two methods that are most-often utilized are digit-recurrence and multiplicative methods [2]. Digit-recurrence methods, such as SRT division, use subtraction as a fundamental operator to retire a fixed number of quotient digits in every iteration. SRT division is named for three researchers, Sweeney, Robertson [3], and Tocher who developed the algorithm independently at approximately the same time. Atkins [4] published the first major analysis of the SRT algorithm and Ercegovac and Lang [5] developed many of the enhancements, including studies demonstrating trade-offs using several optimizations for static CMOS standard-cells, that are now incorporated in many current designs.

Although SRT division can be effective at providing fast and efficient implementations, performance can be smaller compared to multiplicative-based methods because the latter converges quadratically. Moreover, various techniques for increasing the performance of digit-recurrence division, such as including staging of low-radix stages, using high-radix digit sets, overlapping sections of one stage with another, and prescaling the input operand introduce area-performance trade-offs.

One of the simplest methods for improving digit-recurrence dividers is by increasing the radix. Although increasing the radix reduces the number of iterations necessary to produce a quotient, a more complex implementation of the QST is often required. The simplest way for computing the quotient digit selection is by using a single quotient digit selection table, usually using a ROM or PLA. Unfortunately, the problem with this implementation is that the size of the quotient selection table (QST) increases exponentially with the radix contributing to a high amount of area consumption and slower worst-case delays. Therefore, as the radix increases the complexity of hardware implementation increases for most implementations that utilize simple QSTs mitigating the advantage of the reduction in the number of iterations.

Recently, an improvement in quotient selection was introduced that uses two separate units. One unit estimates an approximate value of quotient and another unit corrects the estimated value and selects the correct partial remainder out of different possible partial remainders calculated using the estimated value of quotient [6]. The disadvantage with this design is that although it reduces the complexity of the QST, the different partial remainders, which are calculated in parallel, may lead to increased area as the radix increases. The partial remainder can be calculated after a correction step but it will almost double the latency since the extra time required for the calculation of correct partial remainder will be added to the latency.

This paper improves upon previous designs by incorporating both quotient prediction and operand scaling to avoid having to estimate the value of the quotient by storing it. In our method, the partial remainder for the quotient selected in an earlier cycle is calculated in parallel with quotient prediction, therefore, eliminating the parallel circuits required for calculating the partial remainder, decreasing the area as the radix increases.

Since, quotient selection and calculation of the partial re-

978-3-9810801-2-4/DATE07 © 2007 EDAA

mainder takes place in parallel, the latency is also improved. In simple division methods, the critical path corresponds to the sum of delays of the quotient digit selection plus the selection of divisor multiple plus the redundant adder (CSA or signed digit) used for calculation of partial remainder plus loading of the registers. The critical path is reduced by prediction of quotient digit, which operates in parallel with the computation of partial remainder. In this case, the critical path becomes the maximum of quotient digit selection plus the register loading and of division multiple selection plus the adder plus the register loading.

By pre-scaling the operands close to unity, which is required in the suggested implementation, the complexity of quotient selection further reduces. Consequently, the proposed method has the ability to enhance area and speed for architectures that support digit recurrence division. Power savings can be also obtained with proper modifications to this architecture [7]. This scheme is proposed for higher radices, however, for simplicity, it is implemented for radix-4.

This paper is organized as follows: Section 2 discusses the background material and Section 3 discusses the implementation. Section 4 presents the results in a TSMC 0.25 μm feature size [8]. Finally, Section 5 presents our conclusions.

II. Background Information

Digit-recurrence algorithms produce a fixed number of result bits in each iteration determined by the radix of implementation. Higher radices reduce the number of iterations to complete the operation, but increase the cycle time and the complexity of the circuit. The division algorithm, $x = q \cdot d + rem$, is implemented by the residual recurrence

$$w[j+1] = r \cdot w[j] - q_{j+1} \cdot d; \quad j = 0, 1, \ldots, m-1 \quad (1)$$

where $q_{j+1} \in \{a, a+1, \ldots, 1, 0, 1, \ldots, a-1, a\}$ with initial value $w[0] = x$, where r is the radix, x is the dividend, d is the divisor, and q_{j+1} is the quotient digit at the j-th iteration. The quotient is computed as

$$q_j = \frac{x}{d} = \sum_{q-1}^{m'} q_j r^{-j} \quad (2)$$

where m' is the number of iterations needed to produce the $n+1$ bits of the representation. Both d and x are normalized in $[0.5, 1)$, however, it is common to normalize the operand between $[1, 2)$ for IEEE-754 implementations [9]. The quotient digit is in Signed-Digit (SD) representation $a, \ldots, 1, 0, 1, \ldots, a$ and the residual $w[j]$ is stored in carry-save form (wS and wC) [10].

The quotient digit is determined, at each iteration, by a selection function

$$q_{j+1} = sel(dd, \hat{y}) \quad (3)$$

where dd is the divisor d truncated after the d-th fractional bit and

$$\hat{y} = rwS_t + rwC_t \quad (4)$$

where rwS_t and rwC_t refer to the carry-save representation of the shifted residual truncated after t fractional bits. The quotient digit is selected such that

$$-\frac{a}{r-1} \cdot d < w[j] < \frac{a}{r-1} \cdot d \quad (5)$$

However, a correction step is required at the end if the final residual is negative.

Rounding is also implemented according to the IEEE-754 by adding a unit in the last place (ulp) to the b-th position where b is the actual number of bits required to represent the quotient. Moreover, to perform this correction and rounding, the sign of the final residual is computed and logic computes whether it is zero or not (i.e. necessary for the round-to-nearest-even scheme). That is, the signed-digit representation of the quotient must be converted to the conventional representation in two's complement. A conversion block performs the conversion from the SD quotient using on-the-fly conversion [11]. This conversion is faster than what is done conventionally because no carry-propagate addition (CPA) is needed. The rounding unit requires the remainder to determine the sign of the final residual. The rounding unit also contains the sign-zero-detection block (SZD) which produces the signal that detects if the final residual is zero [11]. The number of bits in the recurrence depends on the radix r and on the redundancy factor $k = [a/(r-1)]$.

In order to simplify the quotient selection, it is important to guarantee a bounded next residual. This is accomplished according to a containment condition that selects the proper interval for q_{j+1} and $|wj| \leq k \cdot d$ Therefore, the upper limit and lower limit, according to our containment condition for the partial remainder, P, is

$$P_{max} = (k+q) \cdot d \quad (6)$$

and

$$P_{min} = (-k+q) \cdot d \quad (7)$$

The values of P_{ju} and P_{jl} denote the upper and lower limits of P over the divisor range $[D_{min}, D_{max})$ for $q = j$

$$P_{ju} = (k+j) \cdot d \quad (8)$$

and

$$P_{jl} = (-k+j) \cdot d \quad (9)$$

assuming $d_{min} \leq d \leq d_{max}$

The representation of the residual can be represented in either nonredundant or redundant form. The redundant form has the advantage that the addition/subtraction in the recurrence can be utilized using carry-free addition, however, this complicates the QST. On the other hand, the use of a digit set that is redundant influences the complexity of the QST by creating possible selection intervals that are redundant. Therefore, a second requirement for the selection interval is the continuity condition which states that for any value of $r \cdot w[j]$, it must be possible to select some value from a digit set for the quotient digit between its range [5]. The overlap that exists between two consecutive values of a quotient digit given as

$$P_{j,j+1} = (-k+j+1) \cdot d \leq P \leq (k+j) \cdot d \quad (10)$$

Since the QST contains regions which must obey the containment and continuity conditions, the partial remainder is examined to determine the proper quotient based on the divisor. Although this can lead to large table sizes, the utilization of a

redundant digit simplifies the QST by selecting truncated comparison multiples. Unfortunately, even if a low amount of precision is employed in the truncated comparison multiples, the QST can still be large.

Recently, it has been proposed that instead of selecting the correct quotient digit q, an estimate is utilized such that the recurrence divider stores both values associated with the containment condition, and then, during the subsequent cycle, the proper containment is selected [6]. That is, the actual quotient digit is expressed as $q = q^{\#} + q^{*}$, where the correction value $q^{*} = 0, 1, \cdots, \gamma$. In other words, $q \in q^{\#}, q^{\#}+1, ..., q^{\#}+\gamma$. Therefore the upper and lower limits for P assuming $q^{\#} = j$ are

$$\begin{aligned} P_{max} &= (k + q_{max}) \cdot d = (k + q^{\#} + \gamma) \cdot d \\ &= (k + j + \gamma) \cdot d = P_{(j+\gamma)u} \end{aligned} \tag{11}$$

$$\begin{aligned} P_{min} &= (-k + q_{min}) \cdot d = (-k + q^{\#}) \cdot d \\ &= (-k + j) \cdot d = P_{jl} \end{aligned} \tag{12}$$

Therefore their overlap region $P_{j,j+1}$ for $q^{\#} = j$ and $q^{\#} = j + 1$ is

$$P_{j,j+1} = (-k + j + 1) \cdot d \leq P \leq (k + j + \gamma) \cdot d \tag{13}$$

and $d_{min} \leq d \leq d_{max}$

Normally, the QST implementation of the containment condition, assuming C is some constant, is

$$(-k + j + 1) \cdot d_{max} \leq C \leq (k + j + 1) \cdot d_{min} \tag{14}$$

making the selection of q independent of d and depending only on P. Wey and Wang's impelementation alter the QST after estimating the quotient digit $q^{\#}$ [6]. Consequently, the actual quotient digit is selected from $\left(q^{\#}, q^{\#}+1, \ldots, q^{\#}+\gamma\right)$. With this enhancement, the overlap region for $q = q^{\#}$ and $q = q^{\#} + 1$ is

$$(-k + q^{\#} + 1) \cdot d \leq P \leq (k + q^{\#}) \cdot d \tag{15}$$

or

$$(-k + 1) \cdot d \leq P - q^{\#} \cdot d \leq k \cdot d \tag{16}$$

Therefore, assuming the adjusted partial remainder is $P^{*} = P - q^{\#}d$, the equation can be rewritten as

$$(-k + 1) \cdot d \leq P^{*} \leq k \cdot d \tag{17}$$

Comparing it with overlap equation of conventional QST approach shows that P^{*} is included in overlap region (P_{ju}, P_{jl}). Hence the correction value q^{*} is determined by the comparison constant derived from this overlap region. Therefore, the value of the actual subtraction to get the residual is selected using this q^{*}. Therefore, the final value of quotient is calculated as

$$q = q^{\#} + q^{*} \tag{18}$$

In hardware, this implementation creates the remainders for $q^{\#}, q^{\#}+1, q^{\#}+2, \cdots, q^{\#}+\gamma$ that are calculated in parallel and then the correct partial remainder is selected depending on value of q^{*}. Since the QST gets decomposed into a bipartite table, it makes the QST smaller. Although this implementation method makes the QST smaller, the hardware used for calculating the remainder increases depending upon the increase in radix value and the estimation limit. Therefore, this method can become disadvantageous for higher radices and higher estimation limits.

III. Proposed Implementation

This paper enhances the previous technique by utilizing prediction of the quotient digit. Using prediction of the quotient digit allows the simultaneous computation of the quotient digit and of the residual. In this implementation, since the overlap region is largest close to $d = 1$, it is convenient to restrict the divisor close to 1 by prescaling the divisor. By pre-scaling the operands, the complexity of calculation is reduced since the overlap region in the P-D plot is maximum near unity. In addition, since the truncated bits of the partial remainder are used for estimation and the partial remainder from this estimate is used for correction, the error induced must be absorbed in the overlap [12], [13], [14] . To reduce the complexity of prediction unit the divisor D is neglected in correction part, since the divisor has been pre-scaled close to unity.

For this implementation, there exists an overlapping region for $q = j$ and $q = j + 1$ given as

$$P_{j,j+1} = (-k + j + 1) \cdot d \leq P \leq (k + j + 1) \cdot d \tag{19}$$

where, k=redundancy factor. The redundant form has the advantage that the addition/subtraction is completed using a $carry\text{-}free$ adder. A carry-save adder (CSA) as opposed to a SD adder is implemented, however, either could be utilized. The disadvantage to using $carry\text{-}free$ addition is that is complicates the quotient-digit selection and introduces some error. The error introduced by using a CSA is

$$\epsilon_{max} - \epsilon_{min} = 2 \cdot r \cdot a \cdot |(1 - d)| \tag{20}$$

where a represents the redundant digit set for quotient $q_j = \{a, a+1, \ldots, 1, 0, 1, \ldots, a\}$

The proposed design absorbs the error in the overlap section. Therefore, the new containment condition becomes

$$\begin{aligned} (k + j + 1) \cdot d - (-k + j + 1) \cdot d &\geq \epsilon_{max} - \epsilon_{min} \\ 2 \cdot k \cdot d &\geq 2 \cdot r \cdot a \cdot |(1 - d)| \\ 2 \cdot k \cdot d + 2 \cdot r \cdot a \cdot d &\geq 2 \cdot r \cdot a \end{aligned} \tag{21}$$

hence,

$$\frac{2ra}{2ra - 2k} > d \geq \frac{2ra}{2ra + 2k} \tag{22}$$

where, $k = a/(r-1)$ is redundancy factor

Therefore, the divisor d is scaled to $D = m \cdot d$ where m is the scaling factor such that, $1 - \alpha \leq |D| \leq 1 + \beta$, α and β are chosen so as to have a simple scaling implementation [14]. The factor m can be determined as an approximation of $1/d$ [15] such that

$$1 - \frac{r-2}{4 \cdot r \cdot (r-1)} < m \cdot d < 1 + \frac{r-2}{4 \cdot r \cdot (r-1)} \tag{23}$$

The suggested scaling for our implementation is shown in Table 1 which gives an attractive implementation $\frac{63}{64} \leq |D| \leq$

d	m	D
[14/16, 1)	9/8=1+1/8	[252/256, 288/256)
[13/16, 14/16)	10/8=1+1/4	[260/256, 280/256)
[12/16, 13/16)	11/8=1+1/4+1/8	[264/256, 286/256)
[11/16, 12/16)	12/8=1+1/2	[264/256, 288/256)
[10/16, 11/16)	13/8=1+1/2+1/8	[260/256, 286/256)
[9/16, 10/16)	14/8=1+1/2+1/4	[252/256, 280/256)
[8/16, 9/16)	16/8=1+1	[256/256, 288/256)

TABLE I

SELECTION OF SCALING FACTOR FOR $r = 4$

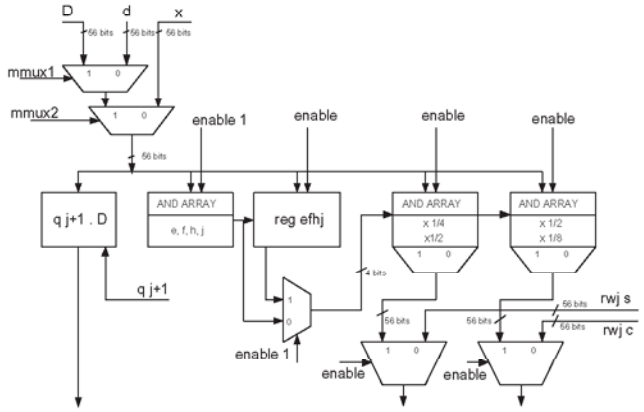

Fig. 1. Scaling Unit

$\frac{9}{8}$ [12]. Since the scaling is only in terms of multiples of two, it can be simplified with combinational logic defined as $m = 1 + e \cdot \frac{1}{8} + f \cdot \frac{1}{4} + h \cdot \frac{1}{2} + j \cdot \frac{1}{2}$. The binary variables e, f, h and j are determined by combinational logic utilizing of $D[55 : 49]$, the first 7 bits of a divisor and the sign bit, $D[55] = S$.

The scaling of operands is done in a sequence. First, the scaling block scales the divisor d to $D = m.d$ in one pass through a scaling block and valuc of D is storcd in a rcgister. The value of m is also stored in a register. Then the block scales the dividend to $X = m \cdot d$. In further cycles the scaling block is disabled and qD is generated along with $rwjs$ and $rwjc$, the sum and carry parts of the partial remainder, respectively. The output components of scaling blocks are passed through a three input CSA to get the required partial remainder. The implementation of the scaling unit is shown in Figure 1.

This research utilizes a QST lookup and modifying it so that it is easier to store or compute using combinational logic [6]. Without the modified QST, a staircase function is usually created which can be area intensive. However, with this method, by partitioning the QST into separate, smaller tables, the selection is simpler. A simplified diagram of this method is shown in Figure 2. In this figure, QS, QH, and QC are the modified QSTs. In addition, the dashed line represents the baseline logic that is repeated depending on the radix. For $radix = 4$, only two baseline modules are needed, however, for $radix = 16$, four baseline modules would be required including larger multiplexors and registers [16].

Block QS generates the correction value, q^* from the computed partial remainder, P^0. The correction value q^* is determined by the comparison constants in the overlap region (i.e., $(-k+1) \cdot D \leq P^* \leq k \cdot D$) and the overlap region is expressed as

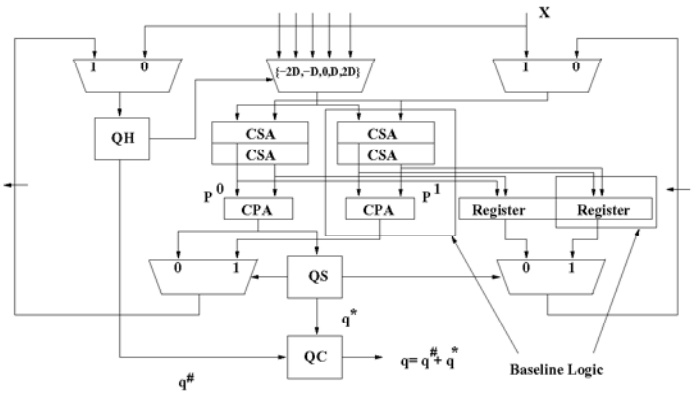

Fig. 2. Block diagram of Radix-4 divider [6].

j	Cj	[D j-1 , D j)
1	000.011	[0.11101, 1.00010)
2	000.100	[1.00011,1.00110)

TABLE II

COMPARISON CONSTANT FOR q^*

$$\left[1 - \frac{a}{r-1}\right] \cdot D \leq P \leq \left[\frac{a}{r-1}\right] \cdot D \tag{24}$$

The comparison constants can then be systematically generated as follows assuming $r = 2^m$ and $a = r/2 = 2^{m-1}$ The range of divider D is equally partitioned into regions $[D_j, D_{j+1})$, where

$$D_j = D_{min} + \frac{D_{max} - D_{min}}{a} \cdot j \tag{25}$$

and $a = r/2 = 2^{m-1}$ For example, if $r = 4$ with the scaled divisor in the range $0.923076923 \leq D \leq 1.2$, it produces the value

$$D_j = 0.923076923 + 0.138461539 \cdot j \tag{26}$$

In order to separate the QST into a separate QS block, it is necessary to make sure that the partial remainder obeys the containment condition. Therefore, for each constant C_j, it must satisfy the following condition

$$\begin{gathered}\left[\frac{a}{r-1}\right] \cdot D_j \leq j \leq \left[1 - \frac{a}{r-1}\right] \cdot D_{j-1} \\ \left[\frac{2^{m-1}-1}{2^m-1}\right] \cdot D_j \leq C_j \leq \left[\frac{2^{m-1}}{2^m-1}\right] \cdot D_{j-1}\end{gathered} \tag{27}$$

For $r = 4$ and $m = 2$, this produces $1/3 \cdot Dj \leq C_j \leq 2/3 \cdot D_{j-1}$. The comparison constants utilized in this paper for $r = 4$ and $a = 2$ are shown in Table 2.

On the other hand, block QH generates the estimated value, $q^{\#}$. The truncated partial remainder is used for estimation of

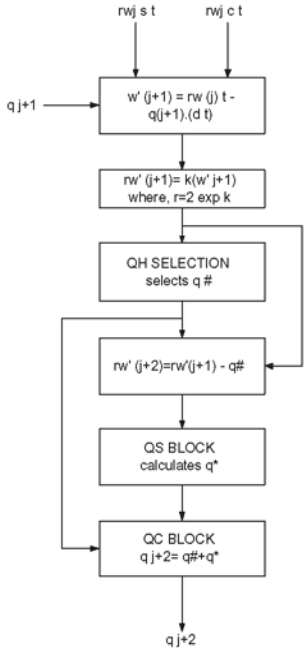

Fig. 3. Quotient Prediction Scheme

$q^{\#}$ in QH block and the approximate truncated partial remainder for next iteration is calculated, however, the divisor, D, is neglected since it was scaled near unity.

$$rw'_{(j+2)} = rw_{(j+1)} - q^{\#} \tag{28}$$

The correction value q^* from the computed partial remainder $rw'_{(j+2)}$ is calculated in QS. This partial remainder lies in the overlap region and is used to find the correction value q^*. The final predicted value is calculated by $q_{j+2} = q^{\#} + q^*$ in the QC block.

A block diagram of the quotient prediction scheme including QS, QH, and QC is given in Figure 3. In this block diagram, $rwjst$ and $rwjct$= sum and carry parts of the partial remainder truncated to t bits, respectively. Initially, the truncated partial remainder rw_{j+1} is calculated using

$$rw'_{j+1} = rwjs_t + rwjc_t - q_{j+1} \cdot D_t \tag{29}$$

After this the truncated partial remainder is computed, QS and QH are utilized to select the proper quotient digit. Block QC utilizes the estimated quotient digit $q^{\#}$ and the correction value q^* to produce the actual quotient digit q_{j+2} as shown in Table 3.

IV. VLSI Implementation

The overall implementation is shown in the Figure 3. Initially, in the first clock cycle the operand scaling block selects the divisor d and scales it to $D = d \cdot m$, where m is the multiplying factor. Because the scaling is produced in carry-save form, it must be sent through a carry-propagate adder (CPA). In our case, we utilized a carry-lookahead adder for the CPA. Although this logic is area-intensive it only needs to be computed once at the beginning of the divide operation.

The scaled operands are passed through a 3-2 CSA that is also used in the recurrence steps to give the partial remainder.

$q^{\#}$	q^*	q_{j+2}
1	1	2
1	0	1
0	1	1
0	0	0
-1	1	0
-1	0	-1
-2	1	-1
-2	0	-2

TABLE III
FINAL SPECULATED VALUE OF q_{j+2}

D is loaded in a register. In the second clock, the divisor x is scaled to $X = x \cdot m$ and stored in the register for the partial remainder and used as the first partial remainder for prediction of quotient. The predicted quotient is stored in the low power 'on the fly conversion' unit which converts the sign-digit quotient to conventional representation [11]. Sign-zero detection block is used for rounding the final value to nearest even [7].

Delay and area estimates were made to compare estimated QST version [6] to the scaled version in this paper. Although previous implementations implemented each cycle in parallel, the implementation is designed to perform the division by recurrence serially as in most digit-recurrence dividers [2]. The dividend and divisor are initially assume to be 56 bits and the $radix = 4$. Although $radix = 4$ is compared in this paper, there would be obvious benefits to using this method for high radices since it would have less baseline logic as shown in Figure 2. In other words, the area should scale linearly as opposed to exponentially.

The implementations were coded at the RTL level using the Verilog hardware descriptive language. The design is synthesized using Synopsys design compiler and layout is generated using Silicon Ensemble for TSMC 0.25 μm SCMOS DEEP submicron technology [8]. All designs were extensively simulated with at least $10,000$ vectors and compared versus a golden file.

In order to achieve a good delay model, the gate and net delay are extracted using Cadence's HyperExtract and Synopsys' Primetime. The output obtained from HyperExtract is back-annotated into Primetime to obtain the critical path delay. The results are presented in Table 4. The area is expressed in terms of mm^2 and the number of cells is the number of gates utilized from synthesis. Results indicate a 24.93% increase in area and a 12.68% improvement in delay. The area increases in our design because of the area consumed by the full-length carry-propagate adder. Since the CPA utilized in the implementation without prediction is much smaller than the design prposed in this paper, the adder consumes a large portion of the design. If an adder that has a better balance between delay and area was utilized, the area comparisons would be comparable since the proposed design has less registers. Moreover, this design could provide benefits for high radices since it does not incur exponentially increasing area provided the scaling can be done efficiently. On the other hand, since the input operand is scaled in our implementation it consumes one additional cycle. The de-

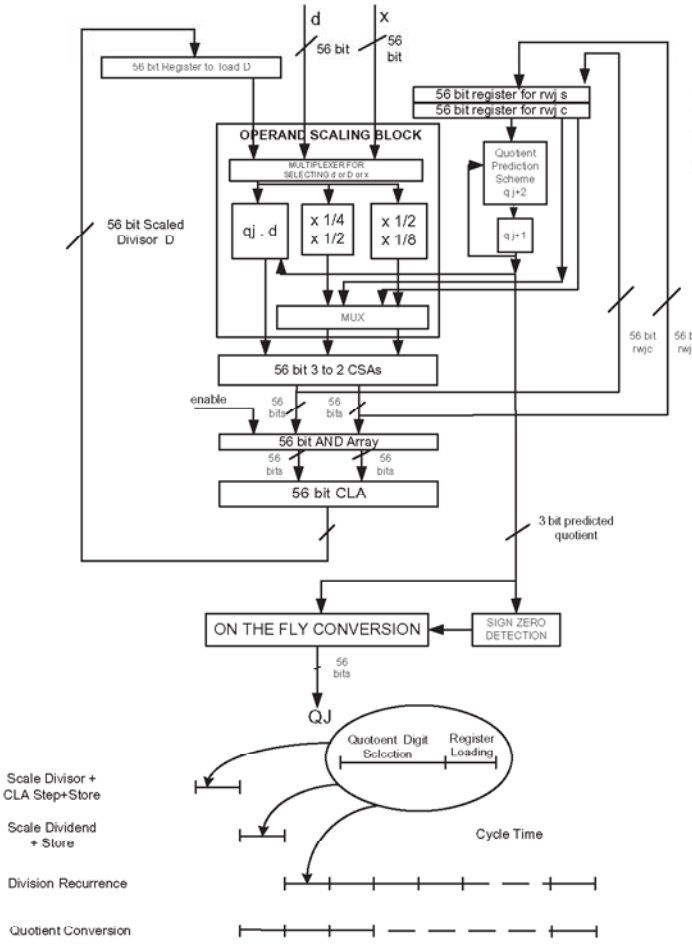

Fig. 4. Overall implementation

Quotient Selection	Number of Cells	Area (mm^2)	Critical Path (ns)	Latency (cycles)
Scaling and Prediction	5,240	0.373	6.2	31
Without prediction, with estimation and correction	3,695	0.280	7.1	30

TABLE IV

COMPARISON OF DIFFERENT IMPLEMENTATIONS

lay estimates were based off parasitically extracted layout and, therefore, reflect rough estimates of the layout. Further reductions in delay could be achieved with hand-optimized custom layouts.

V. Conclusion

The method suggested in this paper presents a design in which the operands are first scaled and then used for predicting quotient digit using speculation and correction. The result shows an improvement in delay of approximately 13% per cycle over previous implementations. However, the implementation shows an increase in the area compared to the previous radix-4 implementation, however, this design can provide benefits for high radices since it does not require additional registers for storing the partial remainder estimates. In addition, utilizing efficient adders that are more conservative with area might show that the method illustrated in this paper might have similar area consumption compared to previous implementations. By modifying the initial containment and continuity equations, this paper also shows that quotient prediction and operand scaling can provide a fast implementation for SRT dividers.

References

[1] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions*, vol. 46, no. 2, pp. 154–161, 1997.

[2] P. Soderquist and M. Leeser, "Division and square root: Choosing the right implementation," *IEEE Micro*, vol. 17, no. 4, pp. 56–66, 1997.

[3] J. E. Robertson, "A New Class of Digital Division Methods," *IRE Transactions on Electronic Computers*, vol. EC-7, pp. 218–222, 1958.

[4] D. E. Atkins, "Higher Radix Division Using Estimates of the Divisor and Partial Remainder," *IEEE Transactions on Computer*, vol. C-17, pp. 925–934, 1968.

[5] M. D. Ercegovac and T. Lang, *Division and Square-root: Digit-Recurrence Algoritms and Implementations*. Boston: Kluwer Academic Publishers, 1994.

[6] C.-L. Wey, T.-H. Tzu, and Y. Chun, "High-radix division with speculation of quotient digits," in *International Conference of Computer Design (ICCD), Austin,Texas*, pp. 479–482, 1995.

[7] A. Nannarelli and T. Lang, "Low-power divider," *IEEE Transactions on Computers*, vol. 48, pp. 2–14, Jan 1999.

[8] J. B. Sulistyo and D. S. Ha, "Developing Standard Cells for TSMC 0.25um Technology under MOSIS DEEP Rules," Tech. Rep. VISC-2002-01, Virginia Tech, 2002.

[9] Standards Committee of the IEEE Computer Society, *IEEE Standard 754 for Binary Floating Point Arithmetic*. IEEE Press, August 1985.

[10] A. Avizienis, "Signed-Digit Number Representations for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers*, vol. 10, pp. 389–400, 1961.

[11] M. D. Ercegovac and T. Lang, "On-the-fly Rounding," *IEEE Transactions on Computer*, vol. C-41, no. 12, pp. 1497–1503, 1992.

[12] M. D. Ercegovac and T. Lang, "Simple radix-4 division with divisor scaling," tech. rep., UCLA, 1987.

[13] M. D. Ercegovac and T. Lang, "Implementation of fast radix-4 division with operand scaling," *IEEE Transactions*, vol. 39, pp. 1204–1208, Nov 1990.

[14] M. D. Ercegovac, T. Lang, and R. Modiri, "Implementation of fast radix-4 division with operand scaling," *IEEE Transactions*, vol. 37, no. 11, pp. 486–489, 1998.

[15] P. Montuschi and T. Lang, "Boosting very high radix division with prescaling and selection by rounding," *IEEE Transactions on Computers*, vol. 50, pp. 909–918, Jan 2001.

[16] C.-L. Wey and C. P. Wang, "A high performance modular embedded rom architecture," in *IEE Proc., Comput. Digital Tech.*, pp. 275–281, July 2000.