A New Divide and Conquer Method for Achieving High Speed Division in Hardware

Murali Mohan K.N, Rohini Krishnan, Anshul Kumar & Balakrishnan M
Dept. of Computer Science & Engg.,
Indian Institute of Technology,
New Delhi, INDIA 110016
{ murli, rohini, anshul, mbala}@cse.iitd.ac.in

Abstract

In this paper, we present a new method of performing Division in Hardware and explore different ways of implementing it. This method involves computing a preliminary estimate of the quotient by splitting the Dividend, performing division of each of the parts in parallel and merging them. The estimate is refined iteratively to get the final quotient. This method is significantly fast since it carries out parallel operations to compute the preliminary quotient and makes use of a fast multiplier to refine the result. It is possible to pipeline the execution of the unit yielding further increase in throughput. Speed estimates show that this method yields a much higher throughput than other fast methods, while area and latency are comparable

1. Introduction

The absence of inherent parallelism unlike other arithmetic operations is the reason for the high latency of division operation. There are basically two methods of performing division– $Multiplicative\ and\ Subtractive$ [4]. While multiplicative algorithms begin by using an approximate value and refine it iteratively, the subtractive algorithms return few bits of quotient in each iteration .

The Subtractive methods in their basic form are very compact and easy to realise [11, 2]. Several modifications ranging from usage of higher radix to overlapped implementation have been proposed to increase the speed at the cost of area [11, 8]. However, there is a limit on the maximum achievable throughput.

Multiplicative methods generally use a LookUp Table to store the approximate inverse of the divisor, and refine it using a multiplier in fewer iterations than the Subtractive methods [4]. Although this results in a higher throughput , the size of memory required is very large. Parallelism in terms of refining and returning more number of bits in each iteration has been achieved in some of these methods [13, 14]. However, these methods have very less pipelineability.

In this paper we propose a new divide and conquer approach which consists of mapping the given division into shorter divisions to get a preliminary estimate, followed by multiplicative refinement to yield the quotient.

- The speed is considerably high as the short divisions can be performed in parallel and refinement can be done by a parallel multiplier .
- It is pipelineable and thereby higher throughput than any other contemporary method can be achieved.

The rest of the paper is organised as follows. In section 2, we present basic algorithm. In section 3, we present the simplification and modification of this. Section 4 proposes the architecture for implementation and in section 5, a comparison of our method with other methods is presented.

2 The Basic Algorithm

We consider the division of X by Y to yield the rounded Quotient Q. Let R be the residual ($R = X - QY$). Without loss of generality, we assume that X, Y, Q and R are all n bit unsigned integers. Let k and j be integers such that $j = \lceil n/k \rceil$. Let X be represented in base-2^j system as $\{X_0, X_1 ... X_{k-1}\}$ with each X_i containing i th block of j bits of X starting from the MSB [1]. If $J = 2^j$, $X = \sum_{i=0}^{i=k-1} X_i J^{(k-1-i)}$. Let $Q'_0, Q'_1 .. Q'_{k-1}$ and $R'_0, R'_1, ..., R'_{k-1}$ be obtained by dividing $X_0, X_1 .. X_{k-1}$ by Y. That is, $Q'_i = X_i\ div\ Y$ and $R'_i = X_i\ mod\ Y$. Therefore $X_i = Q'_i Y + R'_i$. If $Q' = \sum_{i=0}^{i=k-1} Q'_i J^{(k-1-i)}$ and $R' = \sum_{i=0}^{i=k-1} R'_i J^{(k-1-i)}$, $X = Q'Y + R'$. Thus Q can be computed from Q' and R' using the following equation

$$Q = \lfloor Q' + R'/Y + 0.5 \rfloor \quad (1)$$

[1] If $jk \neq n$, then we assume that the last $(jk - n)$ bits are zeros

R can also be expressed directly in terms of Q' and R', but it is simpler to compute R as $(X - QY)$ once Q is known. The above formulation leads to a two phase division process

- Compute Q'_i and R'_i in parallel for all i ($Approximate\text{-}Phase$).
- Compute Q and R from Q'_i and R'_i values ($Refine\text{-}Phase$).

Divisions required in the first phase involve only j bit quotients and therefore are simpler than the original division. This approach is interesting only when Y is small. When $Y \geq J$, all Q'_is become zero as $X_i < J$ for all i. However if we replace integer division $X_i \ div \ Y$ and $X_i \ mod \ Y$ by finite precision fractional division, an interesting possibility emerges.

Let div^p and mod^p respectively denote the quotient and remainder of division with p fraction bits defined as follows:$A \ div^p \ B$=$(2^p A \ div \ B)2^{-p}$ and $A \ mod^p \ B$=$(2^p A \ mod \ B)2^{-p}$.

This can be considered as integer division with prescaling of the dividend by 2^p and postscaling of the quotient and the remainder by 2^{-p}. As p increases $A \ div^p \ B$ becomes closer to real division A/B and $Amod^pB$ approaches 0. We can alternatively define $A \ mod^p B$ as $[A - (A \ div^p \ B)B]$. This allows us to extend the definitions of div^p and mod^p to the dividends which are not integers. We can derive the following recurrence relations in terms of these operators which give an iterative algorithm to compute Q. Let $Q'_i = X_i \ div^p \ Y$ and $R'_i = X_i \ mod^p \ Y$. Note that X_i can still be expressed as $Q'_i Y + R'_i$. Therefore equation(1) is still valid.

$$R''_i = \begin{cases} (R''_{i-1} J + R'_i) \ mod^p \ Y & : 0 < i < k \\ R'_0 & : i = 0 \end{cases}$$

$$\Delta Q'_i = \begin{cases} (R''_{i-1} J + R'_i) \ div^p \ Y & : 0 < i < k \\ 0 & : i = 0 \end{cases}$$

$$Q = \lfloor \sum_{i=0}^{i=k-1} (Q'_i + \Delta Q'_i) J^{(k-1-i)} + 0.5 \rfloor \qquad (2)$$

The algorithm is as follows: [2]

Basic Algorithm

$Compute \ R'_i. \ Q'_i \ and \ \Delta Q'_0;$
$Q = Q'_0;$
$for \ i = 1 \ to \ (k-1) \ \{$
    $Compute \ R''_i \ and \ \Delta Q'_i;$
    $Q = QJ + Q'_i + \Delta Q'_i;$
$\}$
$Q = \lfloor Q \ + \ R''_{k-1}/Y + 0.5 \rfloor;$
$R = X - QY;$

[2]For integer division, it is necessary to compute Q. For floating point division, X and Y will be normalised and it is necessary to compute $Q_f = 2^n Q$ correctly rounded

Each iteration involves a division, making this algorithm inefficient. In the next section, we show that by proper choice of p, a value sufficiently close to Q can be computed without division.

3 Modified Algorithm

3.1 Eliminating the Division

Let $J_q = J \ div^p \ Y$ and $J_r = J \ mod^p \ Y$.It is evident from (2) that elimination of division is possible if J_r as well as R'_i are very small compared to Y. The computation of quotients with fractional bits makes this possible. Division-free approximation of $\Delta Q'_i$ and R''_i can be expressed as follows.

$$\begin{aligned} \Delta\overline{Q}'_0 &= 0; \overline{R}''_0 = R'_0; \\ \Delta\overline{Q}'_i &= \overline{R}''_{i-1} J_q \qquad : 0 < i < k; \qquad (3) \\ \overline{R}''_i &= \overline{R}''_{i-1} J_r + R'_i \qquad : 0 < i < k; \end{aligned}$$

$$\overline{Q} = \lfloor \sum_{i=0}^{k-1} (Q'_i + \Delta\overline{Q}'_i) J^{k-1-i} \rfloor \qquad (4)$$

$\overline{Q}$ is an approximation of Q and can be computed iteratively . Theorem-1 helps in computing the difference between Q and $\overline{Q}$

Theorem 1

$$\sum_{l=0}^{i} Y \Delta\overline{Q}'_l J^{k-1-l} + \overline{R}''_i J^{k-1-i} + \sum_{l=i+1}^{k-1} R'_l J^{k-1-l} = R'$$

for $\quad 0 \leq i < k$

This can proved by Mathematical Induction [7]. Setting $i = (k-1)$ in Theorem-1,

$$R' = Y \sum_{l=0}^{l=k-1} \Delta\overline{Q}'_l J^{(k-1-l)} + \overline{R}''_{k-1} \qquad (5)$$

From (1), (5) and (6), the error in quotient is due to the presence of $\overline{R}''_{k-1}$.

$$\overline{R}''_{k-1} = \sum_{l=0}^{k-1} R'_l J_r^{k-1-l} \qquad (6)$$

It can be seen that R'_l is bounded by $(Y-1)2^{-p}$. Upper bound on J_r is different from that of R'_l since J_r is obtained from division of $(j+1)$ bit J by Y unlike R'_ls, which need j bit by n bit division. $\overline{R}''_{k-1}$ is required to be less than Y to minimise $(Q - \overline{Q})$. Moreover, when $1 \leq Y \leq 2$, all the R'_is, $\Delta Q'_i$s and $\overline{R}''_i$s are zeros. Thus, $\overline{R}''_{k-1}$ is non-zero only

for $Y > 2$. Denoting $(Y-1)_{max}2^{-p}$ by q and $J_{r,max}$ by γ, upper bound on $\overline{R}''_{k-1}$ can be obtained as follows:

$$\overline{R}''_{k-1,max} = q\sum_{l=0}^{k-1}\gamma^{k-1-l} = \frac{q(1-\gamma^k)}{1-\gamma} \quad (7)$$

$$\overline{R}''_{k-1,max} < 3 \quad (8)$$

By proper choice of p, the above condition can be satisfied, and thus the division in the $Refine\text{-}Phase$ can be expressed as a series of multiplications followed by rounding-off of the Quotient at the end.

3.2 Simplification of Multiplication

Due to the iterative nature of $Refine\text{-}Phase$, any simplification of computation of $\overline{R}''_i$ results in a significant improvement in the overall performance. One possibility is to perform multiplication using few bits of $\overline{R}''_{i-1}$ say β. Let $\overline{R}''_{i,a} = (\overline{R}''_{i-1} div^{\beta}\ 1)$ and $\overline{R}''_{i,b} = (\overline{R}''_{i-1} mod^{\beta}\ 1)$ [3]

$$\begin{aligned} \Delta\overline{Q}'_i &= \overline{R}''_{i,a}J_q \\ \overline{R}''_i &= \overline{R}''_{i,a}J_r + \overline{R}''_{i,b}J + R'_i \\ &\quad \text{with } \overline{R}''_0 = R'_0 \text{ and } \Delta Q'_0 = 0 \end{aligned} \quad (9)$$

Thus it can be observed that the theorem still holds and the error is still due to $\overline{R}''_{k-1}$ but the value of $\overline{R}''_{k-1}$ is different.

From (7), the Upper bound on $\overline{R}''_{k-1}$

$$\overline{R}''_{k-1,max} < \frac{q(1-\gamma^k)+2^{j-\beta}(1-\gamma^{k-1})}{1-\gamma} \quad (10)$$

When $j < \beta$, value of $\overline{R}''_{k-1}$ is almost the same as that given by (8). With $\beta > j$, it is possible to reduce the size of the multiplier without much increase in error.

3.3 Rounding off the Quotient after iterative algorithm

With prescaling, the quotient $\overline{Q}$ obtained at the end of $Refine\text{-}Phase$ has bits in the fractional part. In the case of floating point division, it is necessary to return $Q_f = 2^n\overline{Q}$ correctly rounded. Based on the sign of $\overline{R}''_{k-1}$ as well as the fractional bits of $\overline{Q}$, any of the standard rounding schemes recommended in [1] can be implemented. The availability of the residual makes it possible to roundoff the result easily unlike $Multiplicative\text{-}Methods$ [10]. This is a disadvantage of the $Multiplicative\text{-}Methods$ and these methods have to compute the remainder from the values X, Y and quotient (obtained at the end of the division). This results in an additional iteration [10, 12] and proliferation of area due to additional circuitry. Similarly, for integer division, $\overline{Q}$ can be rounded to get Q. The roundoff phase is dependent on the implementation of the $Approximate\text{-}Phase$ and $Refine\text{-}Phase$ and is explained in Section 4.3. The Modified Algorithm can be stated as follows:

Modified Algorithm

$Compute\ R'_i.\ Q'_i\ and\ \Delta Q'_0;$
$\overline{Q} = Q'_0;$
$for\ i = 1\ to\ (k-1)\ \{$
    $Compute\ \overline{R}''_i\ and\ \Delta\overline{Q}'_i;$
    $\overline{Q} = \overline{Q}J + Q'_i + \Delta\overline{Q}'_i;$
$\}$
$Perform\ rounding\ of\ \overline{Q};$

4 Architecture

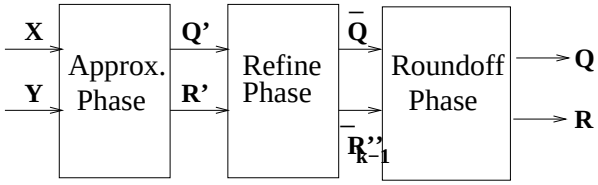

Figure 1. Block Diagram of the system

We describe the architecture of the Modified Algorithm in this part. From the equations, it is evident that we need two separate units, one for $Approximate\text{-}Phase$ and the other for $Refine\text{-}Phase$. The detailed diagram of the system is shown in the Fig. 1.

4.1 Approximate Phase

This Phase can be implemented by either Lookup tables or Dividers. But the constraints imposed by (8) make it prohibitive to use lookup tables on account of huge area requirement. Hence, dividers based on $Subtractive\text{-}methods$ are to be used for this purpose. Also, J_q is required only in the $Refine\text{-}Phase$ and therefore the whole of $Approximate\text{-}Phase$ can be used to compute it. This eliminates the need for an LUT at the cost of an extra divider which is justified since the area of the divider used for this is less than that of an LUT as the LUT needs to store the inverses for vast ranges of the divisor.

4.1.1 Simplifying the computation to obtain fractional bits

Let η be a minimum power of 2 which is greater than Y and $lz = [n - \log_2(\eta)]$. Considering integer division to obtain j bit quotient Q_i

[3] It means that $\overline{R}''_{i,a}$ contains the top β bits of $\overline{R}''_{i-1}$ and $\overline{R}''_{i,b}$=$\overline{R}''_{i-1} - \overline{R}''_{i,a}$

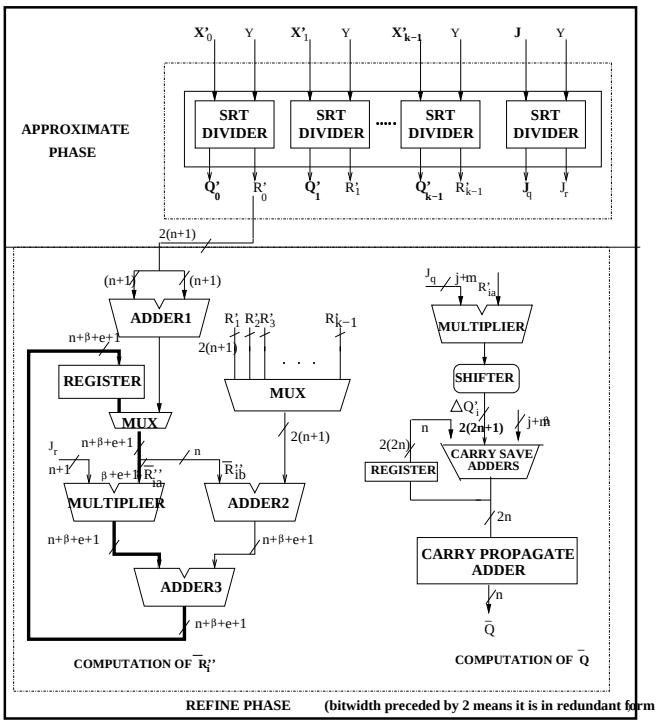

Figure 2. Implementation of the Phases of Modified Algorithm

$$\mid Q_i \mid_{max} = \frac{X_{i,max}}{Y_{min}} = \frac{2^j - 1}{\eta/2} < 2^{j-[(n-1)-lz]} \qquad (11)$$

The topmost $[(n-1) - lz]$ bits of Q_i are zeros and need not be computed. Therefore, it is meaningless to perform integer division yielding j bits. Instead, if X_i is scaled up by $2^{(n-1-lz)}$, all the bits obtained are meaningful. The quotient and the remainder so obtained have to be postscaled by division with $2^{(n-1-lz)}$ to get Q_i's and R_i's . With this,

$$R'_{i,max} = q = \frac{(Y-1)_{max}}{2^{(n-1-lz)}} = \frac{2^{(n-lz)} - 1}{2^{(n-1-lz)}} < 2 \qquad (12)$$

Since every fractional bit obtained halves the upperbound on $R'_{i,max}$, prescaling by 2^{n-1-lz} reduces the upper bound on R'_i ie q. With extra m fractional bits,

$$q < 2^{n-1-lz-m} \qquad (13)$$

4.1.2 SRT Dividers

We use SRT dividers to obtain Q_i's because of the possibility of achieving the desired area-speed tradeoff by a proper choice of Radix [4, 11, 2, 8]. The important properties of SRT division are given in this section. We assume that [P, A], B and Q are the registers used to store Partial Remainder, the Dividend and the Quotient respectively. The msb of P is denoted by p_0

(i) The Partial Remainder in the case of integer division involving X and Y (both n bits long) remains bounded between -2^{j-1} and $2^{j-1} - 1$ in Radix-2 SRT methods and between -ρY and ρY in higher radices with ρ representing the redundancy factor [4, 11, 2, 8]It is independent of the number of iterations.
(ii) Every iteration of SRT Division returns $\log_2(Radix)$ number of quotient bits. Therefore with every additional iteration, the upper bound on $\mid X - YQ \mid$ reduces by a factor of the Radix and the Quotient tends to become more accurate.

4.1.3 Implementation of $Approximate\text{-}Phase$

In the case of SRT Division of X_i by Y with prescaling by 2^p, q is about half of that given by (13) if the remainder is not restored at the end of $Approximation\text{-}Phase$ Since J requires one bit more than X_i, $\gamma = J_{r,max} = 2q$. Given the values of j and k, and using (8) and (10), it is possible to calculate the values of β and p. We set $p = (n-1-lz),\ m = 1$ and $\beta > (j+2)$. Therefore, $q \leq \frac{1}{2}$ with Radix-2 and slightly greater with other radices. From 4.1.2(ii), it is evident that extra m fractional bits can be obtained by additional $\lceil m/\log_2(Radix) \rceil = 1$ iteration. Thus $Approximate\text{-}Phase$ requires $(k+1)$ SRT Dividers to compute the $(j+m)$ bits of J_q and the Q_i's. Since the additions to compute partial remainder happen every iteration, carry save adders are used to minimise iteration delay. The quotient bits are not in binary format and we need use a radix-to-binary converter as in [9].

4.2 Refine Phase

The $Refine\text{-}Phase$ consists of $(k-1)$ iterations of computation of $\overline{R}_i''$, $\Delta\overline{Q}_i'$ and refinement of $\overline{Q}$ followed by correction of $\overline{Q}$ and R. Since $\overline{R}_{k-1}''$ can be greater than 1, the registers required to hold $\overline{R}_i''$ need additional e bits where $e = \lceil \log_2(\mid \overline{R}_{k-1}'' \mid_{max}) \rceil$. Computation of $\overline{R}_i''$ requires a Register, multiplier ($(n+1)$bit by $(\beta + e + 1)$ bit), adders and muxes as shown in the Fig. 2. The delay of this phase can be reduced significantly by using 4:2 and 3:2 CSAs (Carry Save Adders) [6]. Also by retaining only $\overline{R}_{i,a}''$ in nonredundant binary format and using redundant representation for other variables like $\overline{R}_{i,b}''$, delay is further reduced. Since, only $\overline{R}_{i-1,a}''$ is used to compute the $\overline{R}_i''$, the delay involved in the summation to obtain $\overline{R}_i''$ in every iteration can be reduced by using a CPA (carry propagate adder) to sum up the bits of $R_{i,a}''$ only. Since successive iterations of

$Refine\text{-}Phase$ depend upon the value of $\overline{R}_i''$ computed in the previous phase, we have reduced the delay of updation of $\overline{Q}$ with respect to this. The critical path is highlighted in the figure. The adders can be incorporated inside the CSA tree of the multipliers. Computation of $\Delta\overline{Q}_i'$ and assimilation into $\overline{Q}$ requires a multiplier ($(j+m)$b by $(\beta+e+1)$b), shifter and adders. The usage of 3:2 and 4:2 CSAs reduces the delay of this computation.

4.3 Roundoff Phase

Due to prescaling, the number of bits of $\overline{Q}$ computed is always more than n. Also,

$$\mid \overline{Q} - (X\ div^{\infty} Y) \mid = \frac{\overline{R}_{k-1}''}{Y} < \frac{\overline{R}_{k-1,max}''}{\eta/2}$$

For both integer division and floating point division, it is necessary to observe the fractional bits of $\overline{Q}$ along with the sign of $\overline{R}_{k-1}''$. The correction required can be $\pm$ulp (ie. lsb) if $\overline{R}_{k-1}'' \leq 1$. This procedure is significantly different from the procedures adopted to roundoff the quotient computed in the case of other algorithms. The roundoff in the case of integer and floating point division differs in the fact that quotient can be midway between 2 integers unlike in floating point division [12]. Implementation of hardware to Roundoff can be done by simple combinational logic.

Let t_{apx}, t_{ref} and t_{rnd} denote the delays of $Approximate$, $Refine$ and $Roundoff$ phases respectively. Also, let t_{apxi} and t_{refi} represent the delay of each iteration of $Approximate\text{-}Phase$ and $Refine\text{-}Phase$ respectively. Then the Execution time E is given by $E = t_{apx} + t_{ref} + t_{rnd}$. t_{ref} can be written as $(k-1)t_{refi}$

Due to the pipelining, Throughput-Rate increases considerably and is given by $T^{-1} = \text{Max}\ \{t_{apx}, t_{ref}\}$.

5 Evaluation and Comparison

We compare our method with the methods mentioned in [14, 5, 3] as well as with Newton Raphson/Goldschmidt methods for 53-bit division. These methods have been chosen since they are among the fastest of the reported methods. The results for SRT methods are also shown. The works in [5, 3] have reported the area and iteration delays of components wrt full adder area (A_{FA}) and delay (t_{FA}) respectively and we use this for our estimation. These values are given in Table 1. In this estimate, the effect of technology, area, interconnect and clock-skew are not estimated and we assume that the clock rate adapts exactly to the iteration time of the implementation as in [14, 5].

Of the methods given in Table(2), only the proposed and VHR methods can be pipelined [4]. Rest of the methods can be pipelined only by loop unrolling. Since the hardware replication associated with loop unrolling leads to large increase in area without proportional increase in throughput, Loop unrolling is not considered for comparison.

Table 1. Component areas and delays with respect to full adder [5]

Component	Rad-2 SRT	Rad-4 SRT	Rad-16 SRT	Computation of $\overline{R}_i''$ and Q updation	Roundoff Phase
A_{FA}	250	250	500	$\simeq$ 600 each	$\simeq$ 500
t_{FA}	5	6	8	10	5

5.1 Delay

The estimated values of the delays are given in Table 2. Based on [5], t_{refi}=$10t_{FA}$ and $t_{rnd} \approx 5t_{FA}$. The details are given in [7]

With $k = 4$ and using Radix 16 SRT for approximate phase the delay is computed as follows: $j = \lceil 53/4 \rceil = 14$. However with Radix, 16 we need to have a value which is a multiple of 4. Hence we choose $j + m = 16$. We set $\beta = 14 + 1$ to satisfy (8) . The conversion of R_0' from redundant to non redundant format involves usage of a $\beta + e + 1$ bit CPA whose delay is given by $3.5t_{FA}$. $t_{apx} = 8X4 + 3.5 = 35.5t_{FA}$. $E = 35.5 + 10X3 + 5 = 70.5t_{FA}$

Several Techniques which can be used to reduce the latency are proposed here.

- If the $\overline{R}_i''$, $\overline{R}_{i,a}''$ and $\overline{R}_{i,b}''$ are represented in residual notation, then t_{refi} reduces by $2t_{FA}$
- Using Hybrid Overlap or Overlapped Quotient selection and Remainder formation configuration [8] for the SRT dividers used to compute Q_0' and J_q , we reduce the t_{apxi} by t_{FA}.
- Prescaling by $(n - lz)$ instead of $(n - 1 - lz)$ and loading SRT Dividers with $X_i - Y$ instead of X_i produces the effect of an additional iteration of division and speeds up $Approximate\text{-}Phase$.

By combining the above methods, the delay can be reduced to $57.5t_{FA}$.

[4] Pipelining can be achieved in the case of VHR Methods by splitting the execution into prescaling (which includes computation of scaling factor followed by scaling of dividend and divisor) and Digit selection

Table 2. Timing and Area Estimates based on [5]

Method	Type	E,T^{-1},Area t_{FA}, t_{FA}, A_{FA}	AT wrt Rad-2 SRT
SRT	Radix-2	270,270,300	1
	Radix 4	170,170,300	0.629
	Radix 16	120,120,550	0.815
Wong's Method [14, 3]		90,90,6000	6.667
Newton--Raphson and Goldschmidt	1 MAC	100,100,3900	4.333
	1 Multiplier + 1 MAC	70,70,6200	5.358
VHR			
	Low Area	72,54,2650	1.766
	High Speed	62,44,4450	2.417
Boosted VHR			
	Low Area	72,54,2550	1.700
	High Speed	62,44,3650	1.893
Proposed			
k=4, Radix-16	Low Area	70.5, 35.5, 3200	1.402
	High Speed	57.5, 28.5, 3900	1.372
k=3, Radix-16	Low Area	65.5, 39.5, 2900	1.414
	High Speed	58.5, 35.5, 3600	1.578
k=2, Radix-16	Low Area	76.5, 59.5, 3000	2.204
	High Speed	65.5, 52.5, 3600	2.333
k=3, Radix-4	Low Area	83.5, 57.5, 2600	1.846
	High Speed	71.5, 48.5, 3300	1.976
k=4, Radix-4	Low Area	80.5, 45.5, 2700	1.517
	High Speed	67.5, 38, 3400	1.616
k=5, Radix-4	Low Area	84, 39.5, 2800	1.365
	High Speed	70.5, 35.5, 3500	1.534

5.2 Area

The areas of basic components relative to A_{FA} are given in Table(1). The total area is the sum of the area of $(k+1)$ dividers and the components used in $Refine\text{-}Phase$.

A comparison of area needed for different methods using these components is given in Table(2). The details are given in [7]. Since Q_0' and J_r are required in the first iteration of $Refine\text{-}Phase$, and the other R_i's and Q_i's are not needed, low area implementations like Radix-4 can be used for these divisions. It is possible to reduce the area by $400A_{FA}$ with a slight reduction in the throughput as computation of the other R_i's overlaps with the computation of $\Delta\overline{Q}_1'$.

From the table, it can be seen that the proposed method is the fastest and also has the best throughput among the low latency methods. Also, the Area is comparable to these methods. Interms of **A-T** product, it is the best among low latency methods.

6 Conclusions

It has been thus shown that by mapping division into several short parallel divisions, a very good estimate of the quotient is obtained without using lookup tables. Refinement using high-speed multiplier and adders results in quotient in a very short time. The method based on this approach is very fast and has the highest throughput whe compared with the fastest methods reported so far.

References

[1] *"IEEE Standard for Binary Floating Point Arithmetic"*, ANSI/IEEE Std 754-1985.

[2] D. Goldberg. Computer arithmetic. Appendix A of *"Computer Organization: A Quantitative Approach"*, JL Hennessey and DH Patterson Morgan Kauffman publishers.

[3] M. Ercegovac et al. "Very High Radix Division with Prescaling and Selection by Rounding", *IEEE Trans. Computers*, vol. 43(8):909–917, August 1994.

[4] P.Soderquist et al. "An Area/Performance comparison of Subtractive and Multiplicative divide/square root implementations", *Proc. 12th IEEE Symp. Computer Arithmetic*, pages 132–139, July 1995.

[5] P.Montusci et al. "Boosting Very-High Radix Division with Prescaling and Selection by Rounding", *IEEE Trans. Computers*, Vol.50(1):13–27, January 2001.

[6] Gary.W.Bewick. "Fast Multiplication: Algorithms and Implementation", *PhD Thesis*, Department of Electrical Engineering:Stanford University, February 1994.

[7] Murali Mohan et al. Extended Report *"A New Method To Achieve Pipelined Division"* http://www.cse.iitd.ac.in/∼vls00003/divider.html

[8] O. Harris and M. Horowitz. "SRT Division:Architecture and Implementations", *Proc. 13th IEEE Symp. on Computer Arithmetic*, pages 18–25, July 1997.

[9] M.Ercegovac and T.Lang. "On the fly conversion of redundant into conventional representations", *IEEE Trans. Computers*, C-36(7):885–887, July 1987.

[10] Oberman and Flynn. "Fast IEEE Rounding for Division by Functional Iteration", *Technical Report No. CSL-TR-96-700*, page Computer Systems Laboratory, Stanford University July 1996.

[11] Obermann and Flynn. "Division Algorithms and Implementations", *IEEE Trans. Computers*, 46(8):833–854, August 1997.

[12] P.W.Markenstein. "Computation of Elementary Function on the IBM RISC System/6000 Processor", *IBM Journal of R & D*, pages 111–119, January 1990.

[13] E. Schwarz and M. Flynn. "Parallel High-Radix Nonrestoring Division", *IEEE Trans. Computers*, 42(10):1234–1246, October 1993.

[14] D. Wong and M. Flynn. "Fast Division using Accurate Quotient Approximations to reduce the number of iterations", *IEEE Trans. Computers*, 41(6):981–995, August 1992.