



Implementation of N-Bit Divider using VHDL

Raghawendra Sharma
M.Tech. Scholar,
S. V. U., Merut, U.P., India
robie2007@gmail.com

Anurag Banoudha
Guide,
Electrical Engg.
iisc.anurag@gmail.com

Abstract-Division is the inverse of multiplication, but it differs from multiplication in many aspects. The major difference is that division is a shift-and-subtract-divisor operation, while multiplication is a shift and-add-multiplicand operation. The results of one subtraction determine the next operation in a division sequence. Thus, division has an inherent serial dependency. This problem does not occur in multiplication, because all summands are generated simultaneously. Next, division is not a deterministic.

SRT dividers are common in modern floating point units. Higher division performance is achieved by retiring more quotient bits in each cycle. Previous research has shown that realistic stages are limited to radix-2 and radix-4. Higher radix dividers are therefore formed by a combination of low radix stages. In this paper, we present an analysis of the n-bit divider and Comparative analysis of different dividers in case of delays and performance. We show the performance and area results for a wide variety of divider architectures and implementations. We conclude that divider performance is only weakly sensitive to reasonable choices of architecture but significantly improved by restoring and non restoring techniques.

Keywords-- Divider, Modelsim, SRT Algorithm, Xilinx

Introduction:

A simple and widely implemented class of division algorithm is digit recurrence. The most common implementation of digit recurrence division in modern microprocessors is SRT division, taking its name from the initials of Sweeney, Robertson [1] and Tocher [2], who developed the algorithm independently at approximately the same time. SRT division uses subtraction as the fundamental operator to retire a fixed number of quotient bits in every iteration. Two fundamental works on SRT division are those of Atkins [3], the first major analysis of SRT algorithms, and Tan [4], a derivation of high-radix SRT division and an analytic method of implementing SRT look-up tables. Ercegovic and Lang [5] provide a comprehensive treatment of the theory of SRT division and square root. Although division is typically an infrequent operation, ignoring its implementation significantly degrades system performance for many applications.

Various techniques have been proposed for increasing division performance, including staging of simple low-radix stages, overlapping sections of one stage with another stage, and prescaling the input operands. All of these methods introduce area-performance tradeoffs. Ercegovic and Lang [5] analyze the tradeoffs of using several of these optimizations in the context of static CMOS standard-cells. Williams [8] presents a self-timed dynamic CMOS divider comprising a ring of five radix-2 stages that incorporates several of these techniques, and he also presents an analysis of the performance and area effects of the architectural components. Prabhu [9] presents the tradeoffs encountered when designing the Sun UltraSparc radix-8 divider.

In contrast to previous works, this paper analyzes in detail the effects of both circuit style and divider architecture on the area and performance of divider implementations. We present the performance results using the technology independent metric of fanout-of-4 inverter delay. We are

therefore able to extrapolate our results to future process technologies. While the discussion here is devoted to division, the theory of square root computation is an extension of the theory of division. Accordingly, most of the analyses presented here can also be applied to the design of square root units.

2 SRT Division:

Performing division requires making a choice of quotient digits starting with the most significant, and progressing to the least significant digits. The quotient digit decision is made as a part of each iteration which recomputes the partial remainder based on the last partial remainder and quotient digit. The complete quotient is accumulated from the equation:

$$Q = \sum_{i=0}^{N-1} q_i r^{-i}$$

where

r is the radix

N is the number of quotient digits calculated

Q is the accumulated quotient result with a precision of $r^{-(n-1)}$

q_i is the quotient digit determined from stage i

Since in binary hardware the full quotient result is easiest to form if it is merely the concatenation of the bits of the individual digits, we set the radix $r = 2^m$ where m is the number of quotient bits determined at each stage.

In irredundant division, the quotient digits are in the set $\{0, \dots, r-1\}$, and the full quotient has only a single valid representation since each digit position in the quotient has only a single correct possibility. Unfortunately, determining the correct digit at each position requires comparison of the entire partial remainder, and this means that the entire partial remainder must be computed before making each quotient digit selection. This computation requires a complete carry propagation along the length of the partial remainder before each quotient digit may be selected [10]. These irredundant division schemes are much slower than multiplication because multiplication does not require such a carry propagation in order to compute partial results.

A complete carry propagation in each iteration can be avoided by making the set of valid quotient digits redundant by including both positive and negative integers. In this method, the divisor and dividend must be normalized to the same binary range, and the valid quotient digits for a maximum quotient digit p are in the set $\{-p, \dots, 0, \dots, p\}$ which is symmetric about, and includes, zero. The quotient digit chosen at each stage in the division determines the operation computing the next partial remainder according to the equation:

$$R_{i+1} = rR_i - Dq_i$$

where

R_i is the partial remainder output from stage i

D is the Divisor

and the sequence is initialized with

$$rR_0 = \text{the Dividend}$$

3. Methodology:

To avoid the delay of the carry propagation, the following applet uses a stack of borrow-save "BS" adders/subtractors. The "tail" cell, variant of the "SC" and "AS" cells, is controlled by two bits and executes one of the three following operations:

- an addition : $R_{j-1} = R_j + 2^{j-1} * D$**
- a subtraction : $R_{j-1} = R_j - 2^{j-1} * D$**
- an identity : $R_{j-1} = R_j$**

This operation is selected according to the sign of the partial remainders R_j . To always know precisely this sign would require the examination of all the remainder's digits. It suffices to check only three. Moreover, the position of the three digits is known: the least significant one is aligned with the most significant non-zero bits of D . To nail down this digit position, D is "normalized", that is the position of its first '1' bit is fixed. Let us call this position 0 and accordingly D 's first bit d_0 . Thus $d_0 = '1'$. For an n -bit divider, $2^{n-2} < D < 2^{n-1}$.

The vertical arrow "view" next to the button displays digits or bits or moves the decimal point right after the most significant digit of D as with a mantissa or after the least significant digit as with an integer.

A "conditional adder/subtractor" outputs S from one of the three following equations:

- if $q = '-1'$ then $S = R + D$;**
- if $q = '0'$ then $S = R$;**
- if $q = '1'$ then $S = R - D$;**

Each "tail" cell carries out a one-bit addition/subtraction. The carry is not propagated to the "tail" cell at left but fed down directly to the "tail" cell below (next row). The delay is independent from the number of digits.

The "conditional adder/subtractor" function is abstracted by its transfer function called "Robertson's diagram". To converge the division imposes moreover that $-2 * D \leq R \leq 2 * D$. If $-D \leq R \leq 2$ then S has two possible values.

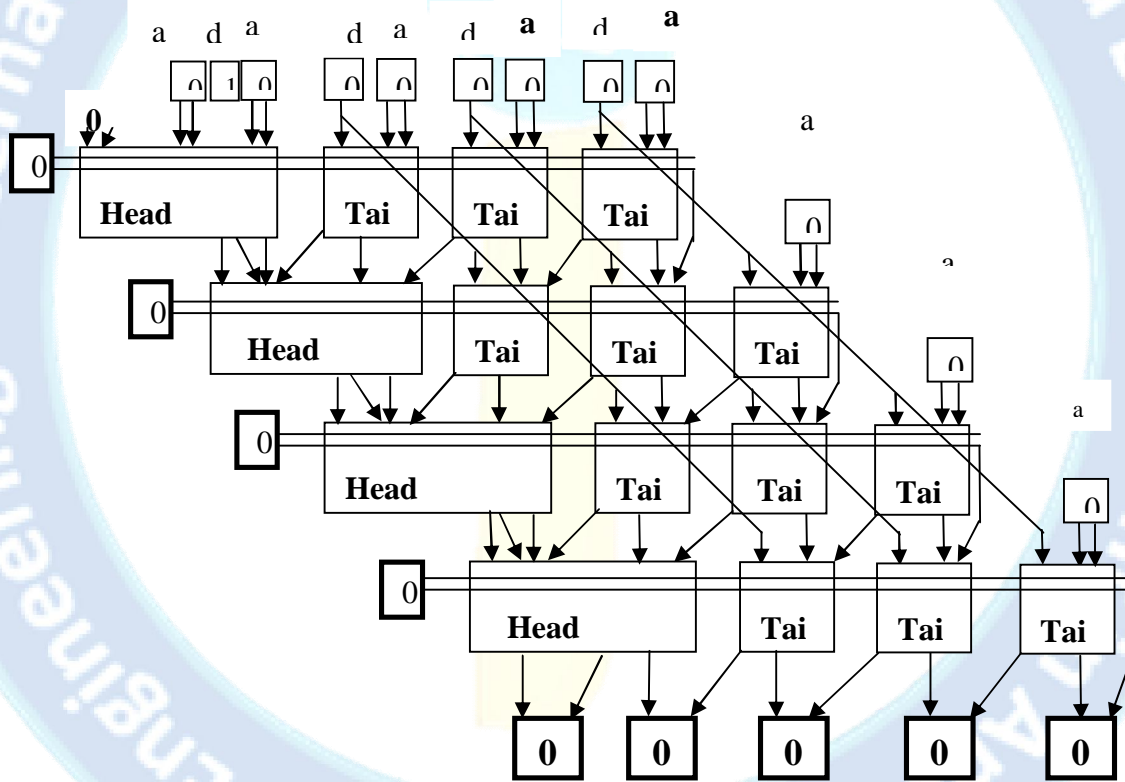


Fig. 1. SRT divider

A. SRT division with divider range reduction:

The previous division is simple because the first bit d_0 of the divider D is always '1'. It may be even further simplified if the two first bits d_0 and d_1 of the divider D are reduced to "1 0" thanks to the operation.

$$\text{if } d_0 \text{ then } \{ D' = D * 3/4 ; A' = A * 3/4 \} \text{ else } \{ D' = D ; A' = A \}$$

This multiplication of both A and D by the same constant does not affect the quotient Q , but on the other hand the

final remainder R is also multiplied. For an n -bit divider, $2^{n-1} < D < 2^n$.

Let $r_0 + r_1 * 0.5$ be the "head" cell input value.

- if > 0.5 then { $s_1 = -1.5$; $q = '+1'$; }**
- if $= 0.5$ then { $s_1 = 0$; $q = '-0'$; }**
- if $= 0$ then { $s_1 = 0$; $q = '+0'$; } or { $s_0 = -0.5$; $q = '-0'$; }**
- if $= -0.5$ then { $s_1 = -0.5$; $q = '+0'$; }**
- if < -0.5 then { $s_1 = +1$; $q = '-1'$; }**

Here the difference between the two 0 representations for q : '+0' and '-0' matters.

B. Quotient converter:

The quotient Q is in "BS" redundant notation. The conversion into a conventional binary representation is obtained thanks to an adder (in fact a subtractor). For the subtraction, '0' gives 'P', '1' gives 'G' et '-1' gives 'K'. Since the digits qj are obtained sequentially, most significant digit first, the conversion can be carried out in parallel with the quotient digits selection by the "head" cells.

Let "Ratio" be the "head" cell and "BK" cell delays ratio. The higher this ratio, the more delay available thus the simpler the converter. But the higher this ratio, the less delay is gained by the concurrence of the converter.

4. Result and Discussion:

In case of 16-bit divider we took 16-bit dividend is "0000000010101010" and 16-bit divisor is "0000000000000010" then the remainder is "0000000000000000". In fig 2 we have shown the simulation result for 16-bit divider.

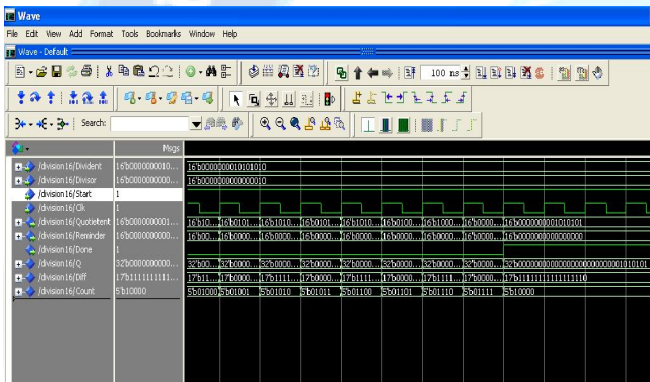


Fig 2. Simulation Result for 16 Bit Divider

In case of 8-bit divider we took 8-bit dividend is "10101010" and 8-bit divisor is "00000010" then the remainder is "00000000". In fig 3 we have shown the simulation result for 8-bit divider.

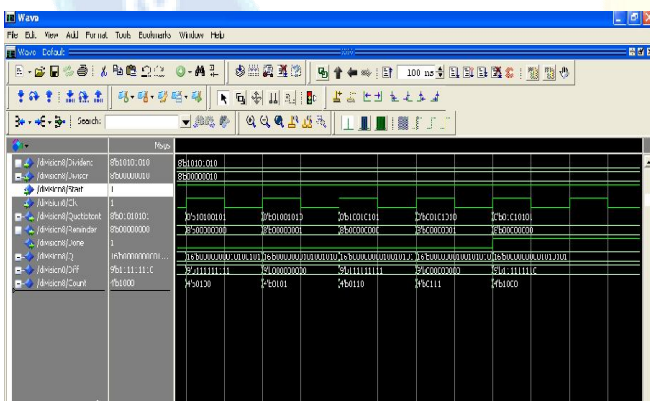


Fig. 3. Simulation Result for 8 Bit Divider

In case of 4-bit divider we took 4-bit dividend is "1010" and 4-bit divisor is "0010" then the remainder is "0000". In fig 4 we have shown the simulation result for 4-bit divider.

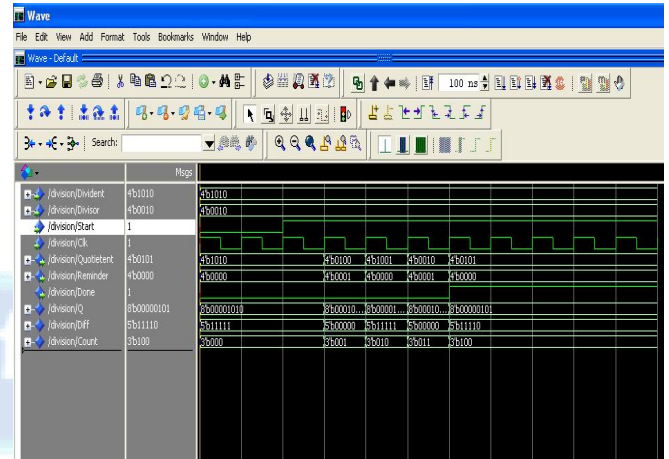


Fig. 4. Simulation Result for 4 Bit Divider

In case of n-bit divider we took as an example 11-bit dividend is "01010101010" and 11-bit divisor is "0000000010" then the remainder is "0000000000". In fig 5 we have shown the simulation result for n-bit divider.

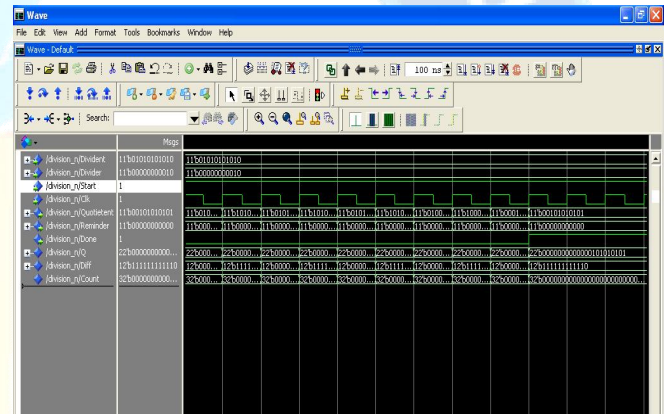


Fig. 5. Simulation Result for n Bit Divider

Table 1. Result Comparison

Divider	Logic Delay (ns)	Route Delay (ns)	Total Delay (ns)
4-Bit Divider	5.753	1.755	7.508
8-Bit Divider	5.753	1.332	7.085
16-Bit Divider	5.753	1.566	7.319
n-Bit Divider	6.302	2.601	8.903

5. Conclusion:

In 4-bit divider the logic delay is 5.753 ns and route delay is 1.755 ns then the total delay for 4-bit divider is 7.508 ns. In case of 8-bit divider the logic delay is 5.753 ns and route delay is 1.332 ns then the total delay for 8-bit divider is 7.085 ns. In case of 16-bit divider the logic delay is 5.753 ns and route delay is 1.566 ns then the total delay for 16-bit divider is 7.319 ns. In case of n-bit divider the logic delay is 6.302 ns and route delay is 2.601 ns then the total delay for n-bit divider is 8.903 ns. If we look simply then we can say that the total delay of n-bit divider is greater than rest of Dividers but we can not decide which divider is required is



where. So that if we required three different dividers like 4, 8, 16-bit dividers then total delay will be 21.912 ns. That is why we use n-bit divider in place of these different dividers. Then the delay will be just 8.903 ns. This delay is clearly less than from these all these dividers. Here we are talking about just three dividers but practically we need more bit dividers and more dividers.

So if we did not take n-bit divider then we have to design many more different dividers for different bit. They will definitely take more time, area, power and also cost. But in VLSI we have to save all of these. Then the result is n-bit divider is taking less power, less area and reduce the cost.

References:

- [1] J. E. Robertson, "A new class of digital division methods," IRE Trans. Electronic Computers, vol. EC-7, pp. 218–222, Sept. 1958.
- [2] K. D. Tocher, "Techniques of multiplication and division for automatic binary computers," Quart. J. Mech. Appl. Math., vol. 11, pt. 3, pp. 364–384, 1958.
- [3] D. E. Atkins, "Higher-radix division using estimates of the divisor and partial remainders," IEEE Trans. Computers, vol. C-17, no. 10, Oct. 1968.
- [4] K. G. Tan, "The theory and implementation of highradix division," in Proc. 4th IEEE Symp. Computer Arithmetic, pp. 154–163, June 1978.
- [5] M. D. Ercegovic and T. Lang, Division and Square Root: Digit-Recurrence Algorithms and Implementations, Kluwer Academic Publishers, 1994.
- [6] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," IEEE Trans. Computers, vol. 46, no. 2, pp. 154–161, Feb. 1997.
- [7] S. F. Oberman and M. J. Flynn, "Division algorithms and implementations," to appear in IEEE Trans. Computers, 1997.
- [8] T. E. Williams and M. A. Horowitz, "A zero-overhead self-timed 160-ns 54-b CMOS divider," IEEE J. Solid-State Circuits, vol. 26, no. 11, pp. 1651–1661, Nov. 1991.
- [9] J. A. Prabhu and G. B. Zyner, "167 MHz radix-8 floating point divide and square root using overlapped radix-2 stages," in Proc. 12th IEEE Symp. Computer Arithmetic, pp. 155–162, July 1995.
- [10] Raghavendra Sharma, "Implementation of CORDIC based Processor using VHDL" International Journal of Research and Development in Applied Science and Engineering, Volume 2, Issue 1, March 2012.