Abstract—Design of a 16-bit divider is a significant issue in high-speed computing. In its place of find the correct quotient digit, an estimated quotient digit is first ventured. The speculated quotient digit is used to simultaneously compute the two possible partial remainders for the next step while the quotient digit is being corrected. This process does not affect the overall speed. Since the circuit can be implemented with simple gate structures, the proposed divider offers fast speed operation. We will develop a 16-bit Divider. The circuits are using non-restoring method to obtain quotient (root) bits. The quotient (root) value in each iterative step is kept in binary form, the result of an on-the-fly redundant binary to binary conversion. The partial remainders (radicands) are in redundant binary representations. The iterative core is a redundant binary and binary subtraction featuring a simple implementation, the same complexity as a carry-save adder. It is a carry propagation-free subtraction, and its speed is independent of the data width of the subtrahend. The quotient (root) bit selection logic inputs three leading digits of partial remainders (radicands).

Keywords- SRT, n BIT, HSA, HSS.

I. Introduction

The main purpose of computers is to do the arithmetic to run programs and applications. Basically, computers handle lots of numbers based on the three basic arithmetic operations of addition, multiplication and division. Compared to other arithmetic operation, division is the least used operation. However, computers will experience performance degradation if division is ignored. A survey by Oberman and Flynn presents the main algorithms used for implementing division in hardware.

There are three main classes for hardware-oriented division algorithms.

i. Digit Recurrence
ii. Functional Iteration
iii. Table Based Methods.

Each method has its own advantages, however digit recurrence division is most common algorithm for division and square root in many floating point units, since it is simple and lower in complexity.

II. Division Algorithm

A division algorithm is an algorithm which is employed by digital circuit designs and software. Division algorithms fall into two main categories: slow division and fast division. Slow division algorithms produce one digit of the final quotient per iteration. Examples of slow division include restoring, non-performing restoring, non-restoring, and SRT division. Fast division methods start with a close approximation to the final quotient and produce twice as many digits of the final quotient on each iteration. Newton-Raphson and Goldschmidt fall into this category.

Discussion will refer to the form $Q = N/D$ where $Q =$ Quotient
$N =$ Numerator (dividend)
$D =$ Denominator (division)

A sequential n-bit divider will use n-clock pulses in order to produces the result (Quotient: Remainder). On the other hand will a "true parallel" combinational division-circuit typically use 3 to 10 times more logic.

![Fig. 1: 16-Bit Divider](image)

III. Restoring Division

In the restoring division method (RDM) the quotient is represented using a non-redundant number system. Its main characteristic is the full width comparisons required to deduce the new quotient digit. In RDM, the divisor is shift-positioned and subtracted from the
if subtraction of the divisor produces a negative result at any bit position relative to the dividend, the operation at that bit position is unsuccessful, and a 0 is placed in the corresponding location of the quotient. The divisor is added back (restored) to the result of the division operation, then the next highest bit of the dividend is shifted into the left bit position of the result. As each bit of the dividend is shifted from right to left, the quotient is built up from left to right. After n shifts, where n represents the number of bits in the dividend, the division operation is complete. Restoring division algorithm is very similar to manually performing long division.

IV. Non Restoring Division Algorithm

Non-restoring Division Algorithm comes from the restoring division. The restoring algorithm calculates the remainder by successively subtracting the shifted denominator from the numerator until the remainder is in the appropriate range. The operation in each step depends on the result of the previous step. Non-restoring division has a quotient digit set of \{1, -1\} instead of the conventional binary digit set. By the non-restoring division approach, we find the -1 of the quotient bit can be simply set to 0, and the quotient is the actual quotient that we want to find.

V. SRT Division

The name of the SRT division stands for Dura W. Sweeney, James E. Robertson, and Keith D. Tocher, independently proposed a fast algorithm for 2’s complement numbers that use the technique of shifting over zeros for division. To divide two n-bit numbers X and Y, the operands are loaded into the Q and B registers, respectively, and the A register is set to 0. The SRT division algorithm is as follows:-

1. If register B has k leading zeros when expressed using n bits, shift all the registers k positions left.
2. The following steps are repeated n times: If the top three bits of the A register are equal, shift the A_Q registers one position left, and set Q0 = 0.
   If the top three bits of the A register are not equal and A is negative, shift the A_Q registers one position left, set Q0 = -1 (1), and add B to A.

To speed-up division operations, number of iteration steps can be reduced by using a higher radix. Increasing the radix of SRT division to \(r=2^m\) allows the generation of \(m\) quotient bits every step. In this manner, the number of required iterations reduces to \([n/m]\), where n is the width of the input operands in bits. However, high-radix division increases the complexity of quotient bit selection and remainder updates, which eliminates the advantage of reduced number of iteration steps.
To eliminate this problem, pre-scaling and prediction methods are used. With pre-scaling both dividend and divisor are scaled preserving the quotient. Prediction, involves the selection of new quotient digit is overlapped with the update of the remainder

2.) Choice of Quotient Digit Set:-

For a given choice of radix r, some range of digits is decided upon for the allowed values of the quotient in each iteration. The simplest case is where, for radix r, there are exactly r allowed values of the quotient. However, to increase the performance of the algorithm, a redundant digit set is used. This allows a quotient digit to be selected based upon an approximation of the partial remainder, permitting the use of a redundant remainder representation. Small errors in the quotient due to the remainder approximation are corrected in later iterations. Such a digit set is composed of symmetric signed-digit consecutive Integers, where the maximum digit is ‘a’. The digit set is made redundant by having more than r digits in the set. By using a larger number of allowed quotient digits, the complexity and latency of the quotient selection function is reduced. However, choosing a smaller number of allowed digits for the quotient simplifies the generation of the multiple of the divisor. Specifically, for radix 2, the digit set is (-1,0,1). For radix 4, there are two typical choices for the digit set: minimally redundant (-2,-1,0,1,2) and maximally redundant (-3,-2,-1,0,1,2,3). The quotient selection logic for a maximally-redundant radix 4 digit set is about 20% faster and 50% smaller than for a minimally redundant digit set.

3.) Choice of Remainder Representation:-

The partial remainder also can be represented in two different forms, either redundant or non-redundant. Each iteration of the algorithm requires a subtraction to compute the next partial remainder. If this partial remainder is in a non-redundant form, then this operation requires a time-consuming full-width carry-propagate-adder, increasing the cycle time. Therefore, the partial remainder is typically stored in redundant form so that a fast carry-free adder, such as a carry-save adder (CSA), can be used in the partial remainder calculation.

High speed adder/subtractor:-

The addition of binary numbers in parallel implies that all the bits of the augends and the addend are available for computation at the same time. In a parallel adder, the carry output of each stage is connected to the carry input of the next higher order stage (ripple carry). Therefore, the sum and carry outputs of any stage can’t be produced until the input carry occurs. This leads to a time delay in the addition process. This delay is known as “carry propagation delay”.

Since each bit of the sum output depends on the value of input carry, the value of “Si” in any given stage in the adder will be in its steady state final value only after the input carry to that stage has been propagated. The following methods are used to improve the speed of adders:

1. One method for reducing the carry propagation delay time is to employ faster gates with reduced delays.

2. The most widely used technique employs the principle of “Look Ahead Carry”. This method utilizes logic gates to look at the lower-order bits of the augends and addend to see if a higher order carry is to be generated. It uses two functions: - Carry generate and Carry propagate. The final carry will not depend on intermediate carry depends only on input bits. The carry-look ahead is a fast adder designed to minimize the delay caused by carry propagation in basic adders. It utilizes the fact that, at each bit position in the addition, it can be determined if a carry will be generated at that bit, or if a carry will be propagated through that bit.

VI. RESULT AND CONCLUSION

Result: In case of 16-bit divider we took 16-bit dividend is “0000000010101010” and 16-bit divisor is “0000000000000010” then the reminder is “0000000000000000”. In case of 8-bit divider we took 8-bit dividend is “10101010” and 8-bit divisor is “00000010” then the reminder is “0000000000”. In case of 4-bit divider we took 4-bit dividend is “1010” and 4-bit divisor is “0010” then the reminder is “00000000”. In case of n-bit divider we took as an example 11-bit dividend is “010101010100” and 11-bit divisor is “000000000010” then the reminder is “0000000000000000”.

Conclusion:

We have shown how to divide n-bits by implement n-Bit divider. In this work we also compare Results of 16-bit divider, 8-bit Divider, 4-bit Divider, and n-Bit Divider. And we found that the logic delay and rout delays are almost same. So in place of use 16-bit divider, 8-bit Divider, 4-bit Divider… e.t.c. we can use n-bit divider. It will be reduce the cost, low power consumption and less area. In modern VLSI implementations, these tradeoffs directly affect the time and space required since custom designs use only the required number of bits.

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 1.507 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 everywhere. 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 then 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 defiantly 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