Volume 4, Issue 2, February 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Various Algorithmic Approaches for Computer Architecture

Shruti
Department of Electronics and Communication Engineering
Bhagat Phool Singh Mahila Vishwavidyalaya,
Khanpur Kalan- 131305
Sonipat (Haryana), India

Abstract— Computer architecture have allowed the performance of digital computer hardware to continue its exponential growth, despite increasing technological difficulty in speed improvement at the circuit level. This would not have been possible without the basic arithmetic techniques which build modules which are useful in the field of electronics and computer. Modules can be adder, multiplier, divider or an ALU which performs the operation on the operands. To make adder, multiplier, divider etc a designer requires a complete knowledge of arithmetic techniques, how they are implemented on hardware, how an algorithm works, how a hardware perform operation on operands, carry is considered or not. In arithmetic techniques, user friendliness, compactness, simplicity, high performance, low cost, and low power plays a very key role. The bulk of hardware in early digital computers resided in accumulator and other arithmetic/logic circuits. In fact arithmetic circuits are no longer dominant in terms of complexity; registers, memory and memory management, instruction issue logic, and pipelining improves the performance. The future scope of this paper is this kind of algorithmic techniques are further implemented onto the FPGA and also can be used in the field of ASIC.

I. Introduction

In this paper we have discussed basic VLSI arithmetic techniques and their implementations on hardware which is helpful in development of the basic building blocks of an integrated circuit.

Multiplication Technique is realized by k-cycles of shifting and adding. It is used in signal processing and scientific applications.

High Radix Multiplier can handle more than one bit of the multiplier in each cycle.The reduction in cycles can be done by using Booth's Recoding Algorithm.

Division Technique of a 2k-bit dividend by a k-bit divisor can be realized in k-cycles of shifting and subtracting. In this technique we discuss economical but slow, bit-at-a-time designs.

Square Rooting technique is basically used to calculate square root of operands as well as floating point data. The function z is the most important elementary function.

These designs can be implemented on the **Reconfigurable Fabric**. One common use of the this is the prototyping of a piece of hardware that will further be implemented into an ASIC. By using this technique the cost of ASIC can be saved.

II. Multiplication Technique

The multioperand addition process needed for multiplying two k-bit operands can be realized in k-cycles of shifting and adding, with hardware or software control of the loop. We discuss here some basic multiplication scheme which is given below:--

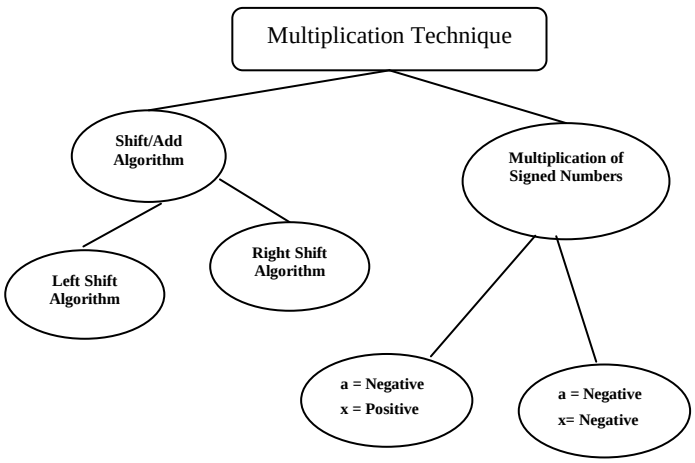

Fig1:- Block Diagram shows the summary of Multiplication Technique

© 2014, IJARCSSE All Rights Reserved

A. Shift/Add Multiplication Algorithm

In this algorithm we assume a is unsigned operand x is multiplier and p is partial fraction. The two numbers a and x are shown at the top. The result is partial product represented by p. In this two types of algorithm are Right Shift Algorithm and Left Shift Algorithm.

A.1 Right Shift Algorithm

In multiplication with right shifts, the partial product terms $x_j.a$ is accumulated from top to bottom:-

$p^{(j+1)} = (p^j + x_j.a.2^k)2^{-1}$ with $p^{(0)}=0$ and $p^{(k)}=p$

a		1	0	1	0	←	1	0	1	0
x							1	0	1	1
$p^{(0)}$		0	0	0	0					
$+x_0a$		1	0	1	0					
$2p^{(1)}$	0	1	0	1	0					
$p^{(1)}$		0	1	0	1	0				
$+x_1a$		1	0	1	0					
$2p^{(2)}$	0	1	1	1	1	0				
$p^{(2)}$		0	1	1	1	1	0			
$+x_2a$		0	0	0	0					
$2p^{(3)}$	0	0	1	1	1	1	0			
$p^{(3)}$		0	0	1	1	1	1	0		
$+x_3a$		1	0	1	0					
$2p^{(4)}$	0	1	1	0	1	1	1	0		
$p^{(4)}$		0	1	1	0	1	1	1	0	

Fig.2 :-- Right Shift Algorithm Technique

Description of Algorithm:--

The right shifts will cause the first partial product to be multiplied by 2^{-k} by the time we are done, we premultiply a by 2^k to offset the effect of right address. Here we are taking an equivalent example of 10 x 11. (Fig.2) The result of 10 x11 is 110 and the $p^{(4)}$ which is the partial product is equal to 0110 1110 which is equivalent to Decimal 110.In this multiplication algorithm firstly write a and x at the top and then initialize the partial product product $p^{(0)}$ to 0 and then add $x_0.a$ .The result would be $2p^{(2)}$ and then again right shift by one bit we get the original $p^{(2)}$ and then add $x_2.a$. same process would repeat again and again till original $p^{(4)}$ get achieved.

A.2 Left Shift Algorithm

In this algorithm the terms $x_j.a$ are added up from bottom to top and the technique based formula is

$p^{(j+1)} = 2p^{(j)} + x_{k-j-1}.a$ with $p^{(0)}=0$ and $p^{(k)}= p$

Let us take an example of 10 X 11 = 110 and the result obtained from left shift algorithm is 0110 1110 which is notation of 110 in decimal as shown in Fig.4.

a					1	0	1	0	
x					1	0	1	1	
$p^{(0)}$					0	0	0	0	
$2p^{(0)}$				0	0	0	0	0	
$+x_3a$					1	0	1	0	
$p^{(1)}$				0	1	0	1	0	
$2p^{(1)}$			0	1	0	1	0	0	
$+x_2a$					0	0	0	0	
$p^{(2)}$			0	1	0	1	0	0	
$2p^{(2)}$		0	1	0	1	0	0	0	
$+x_1a$					1	0	1	0	
$p^{(3)}$		0	1	1	0	0	1	0	
$2p^{(3)}$	0	1	1	0	0	1	0	0	
$+x_0a$					1	0	1	0	
$p^{(4)}$	0	1	1	0	1	1	1	0	

Fig.4:- Left Shift Algorithm

Procedure of Left Shift Algorithm :--

In left shift algorithm firstly we write a and x at the top and then initially first partial product term i.e. $p^{(0)}$ is set to 0 and then left shift of the digit is done to achieve $2p^{(0)}$ and addition of $x_3.a$ is performed accordingly because as per technique x_j are added from bottom to top and basically the left shift of the number is done to achieve original number for further process in the technique**.**

B. Multiplication of Signed Numbers

An indirect multiplication scheme is quite efficient for 1's compliment numbers but involves too much overhead for 2's complement representation. It is preferable to use a direct multiplication algorithm for such numbers. In signed multiplication technique the carry would be neglected.A four digit number with one extra bit of sign would become 5 digit

© 2014, IJARCSSE All Rights Reserved

number. If the extra bit attached to the number is 1 then the number is negative but if the bit is 0 then number is positive. We would discuss here two cases:--

B.1 **When the multiplicand (a) is negative and the multiplier (x) is positive.:--**

Let us take an example in which a is 10110 (1 represents that no. is negative) and x is 01011 (0 represents that no. is positive). In the example -10X11 = -110 . The result is 1110010010 which is the notation of -110 (-512+256+128+16+2) as shown in fig.6. In this algorithm negative number is always put at the top. If we put negative number at the bottom we would have to take the 2's complement of the number in the end when we would multiply with sign bit. In this carry is not considered.

a		1	0	1	1	0			
x		0	1	0	1	1			
$p^{(0)}$		0	0	0	0	0			
$+x_0a$		1	0	1	1	0			
$2p^{(1)}$	1	1	0	1	1	0			
$p^{(1)}$		1	1	0	1	1	0		
$+x_1a$		1	0	1	1	0			
$2p^{(2)}$	1	1	0	0	0	1	0		
$p^{(2)}$		1	1	0	0	0	1	0	
$+x_2a$		0	0	0	0	0			
$2p^{(3)}$	1	1	1	0	0	0	1	0	
$p^{(3)}$		1	1	1	0	0	0	1	0
$+x_3a$		1	0	1	1	0			
$2p^{(4)}$	1	1	0	0	1	0	0	1	0
$p^{(4)}$		1	1	0	0	1	0	0	1	0
$+x_4a$		0	0	0	0	0			
$2p^{(5)}$	1	1	1	0	0	1	0	0	1	0
$p^{(5)}$		1	1	1	0	0	1	0	0	1	0

Fig.6:-- Multiplication of Signed Numbers Algorithm
(a is –ve and x is +ve)

B.2 When the multiplicand (a) is negative and multiplier is also negative:--

Let us take an example in which a is 10110 (-10) and x is 10101 (-11) . The result is (-10) X (-11) = 110 and the result obtained by the algorithm is (64+32+8+4+2) = 110.

a		1	0	1	1	0			
x		1	0	1	0	1			
$p^{(0)}$		0	0	0	0	0			
$+x_0a$		1	0	1	1	0			
$2p^{(1)}$	1	1	0	1	1	0			
$p^{(1)}$		1	1	0	1	1	0		
$+x_1a$		0	0	0	0	0			
$2p^{(2)}$	1	1	1	0	1	1	0		
$p^{(2)}$		1	1	1	0	1	1	0	
$+x_2a$		1	0	1	1	0			
$2p^{(3)}$	1	1	0	0	1	1	1	0	
$p^{(3)}$		1	1	0	0	1	1	1	0
$+x_3a$		0	0	0	0	0			
$2p^{(4)}$	1	1	1	0	0	1	1	1	0
$p^{(4)}$		1	1	1	0	0	1	1	1	0
$+(-x_4a)$		0	1	0	1	0			
$2p^{(5)}$	0	0	0	1	1	0	1	1	1	0
$p^{(5)}$		0	0	0	1	1	0	1	1	1	0

Fig.7:- Multiplication of Signed Numbers Algorithm
(a and x both are –ve)

Description of Technique :--

© 2014, IJARCSSE All Rights Reserved

Firstly write the a and x at the top and initialize the $p^{(0)}$ to zero and add $x_0.a$. The result would be $2p^{(1)}$ with extra 1 which represents sign. Evaluate $p^{(1)}$ by shifting one digit right and add $x_1.a$.Then calculate $p^{(2)}$ by shifting one digit right and add $x_2.a$. Result is $2p^{(3)}$ and evaluate $p^{(3)}$ and add $x_3.a$. Now the Result achieved is $2p^{(4)}$ and then evaluate $p^{(4)}$.Addition of $-x_4.a$ is done with $p^{(4)}$. It is done so to add the two's complement of $x_4.a$ because x is a negative number which is multiplier. When we multiply the last digit of a negative no to the multiplicand then it can not be multiply as done before. We have to add a negative product. Now Evaluate $p^{(5)}$ which is the required result. In this carry out is not neglected.

III. **Booth's Recoding Algorithm**

The radix-2 multiplication consists of a sequence of shifts and adds. When 0 is added to the cumulative partial product in a step, the addition operation can be skipped together. The table shown below is the base of booth's recoding algorithm:--

x_i	x_{i-1}	y_i	Explanation
0	0	0	No string of 1s in sight
0	1	1	End of string of 1s in *x*
1	0	-1	Beginning of string of 1s in *x*
1	1	0	Continuation of string of 1s in *x*

Fig.8:- Table of Booth's Recoding Algorithm

Let us take an example in which a is 10110 and x is 10101. As shown in fig.9. We have to calculate y that is booth's recoded element first by multiplying two consecutive digits as per table. Firstly assume 0 and now
X_0 is 1 and multiply it with 0 it would be (1X0= (-1)) .
Similarly other elements can be calculated.

```
a                1  0  1  1  0
x                1  0  1  0  1     Multiplier
y              -1  1 -1  1 -1     Booth-recoded
==============================
     p(0)              0  0  0  0  0
     +y0a              0  1  0  1  0
     ------------------------------
     2p(1)        0  0  1  0  1  0
     p(1)            0  0  1  0  1   0
     +y1a            1  0  1  1  0
     ------------------------------
     2p(2)        1  1  1  0  1  1   0
     p(2)            1  1  1  0  1   1  0
     +y2a            0  1  0  1  0
  2p(3)        0  0  0  1  1  1   1  0
  p(3)            0  0  0  1  1   1  1  0
  +y3a            1  0  1  1  0
-------------------------------------------
  2p(4)        1  1  1  0  0  1   1  1  0
  p(4)            1  1  1  0  0   1  1  1  0
  y4a             0  1  0  1  0
-------------------------------------------
  2p(5)        0  0  0  1  1  0   1  1  1  0
  p(5)            0  0  0  1  1   0  1  1  1  0
```

Fig.9:- Booth's Recoding Algorithm

Description of Booth's Recoded Algorithm:--

a) Firstly write a and x at the top and y and then initialize $p^{(0)}$ and add (y_0 .a) but the first digit of number y is (-1) so the multiplication product is considered as (-1.a) that is (-a) i.e.01010. Now this is equivalent to (y_0 .a) and add to $p^{(0)}$ then result would be 2 $p^{(1)}$ with extra bit zero. Now evaluate $p^{(1)}$ and add (y_1 .a). Result would be 2 $p^{(2)}$ with extra bit 1. Now find $p^{(2)}$ and add (y_2 .a). Now the product (y_2 .a) is added to get 2 $p^{(3)}$ .Now evaluate $p^{(3)}$ by shifting one digit towards right side and add (y_3 .a). The Result achieved is 2 $p^{(4)}$ with sign bit 1. Find $p^{(4)}$ and add (y_4 .a). as the value of y_4 is (-1) so the value would be the 2's compliment of a instead of a and add to the partial product. Now the result in the form of $2p^{(5)}$ and evaluate the final result $p^{(5)}$. This is the equivalent example of -10X-11= 110 and the result obtained by this Booth's Recoding algorithm is 00011 01110 which is 64+32+8+4+2 equal to 110.

IV. **Division Technique**

The division of a 2k-bit dividend by a k-bit divisor can be realized in k cycle of shifting and subtracting .We are discussing here some basic division techniques:--

© 2014, IJARCSSE All Rights Reserved

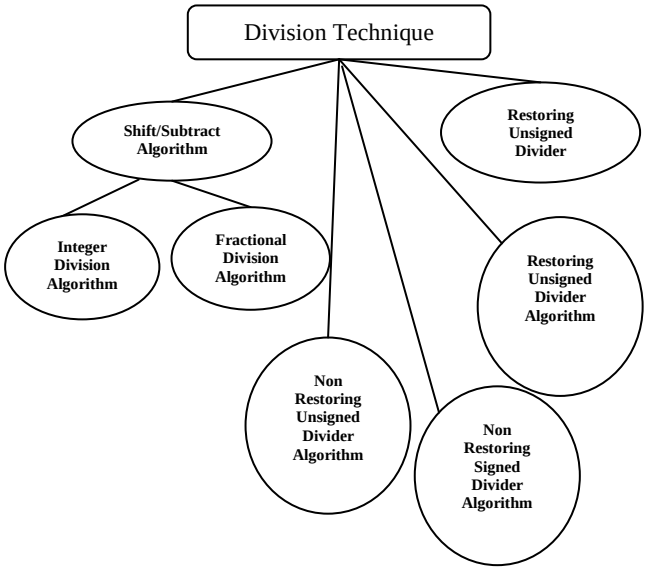

Fig 10:- Block Diagram Summary of Division Technique

A. Shift/Subtract Division Algorithm

In this algorithm d denotes divisor and q as quotient while s is remainder which is(z-(dxq)).Sequential division is performed by repeated subtractions. The partial remainder is initialized to $s^{(0)}$=z. The sequential division algorithm with left shifts:-

$s^{(j)} = 2s^{(j-1)} - q_{k-j} (2^k d)$ with $s^{(0)} = z$ and $s^{(k)} = 2^k.s$

A.1 Integer Division Algorithm

In this we take an example of 117 divided by 10. Quotient is 11 and the Remainder is 7 and the same result is obtained by using Integer Algorithm. Evaluation of Algorithm is as shown below in Fig. 11.

z 117	0 1 1 1 0 1 0 1
$2^4 d$ 10	1 0 1 0
$s^{(0)}$	0 1 1 1 0 1 0 1
$2s^{(0)}$	0 1 1 1 0 1 0 1
$-q_3 2^4 d$	1 0 1 0 $\{q_3 = 1\}$
$s^{(1)}$	0 1 0 0 1 0 1
$2s^{(1)}$	0 1 0 0 1 0 1
$-q_2 2^4 d$	0 0 0 0 $\{q_2 = 0\}$
$s^{(2)}$	1 0 0 1 0 1
$2s^{(2)}$	1 0 0 1 0 1
$-q_1 2^4 d$	1 0 1 0 $\{q_1 = 1\}$
$s^{(3)}$	1 0 0 0 1
$2s^{(3)}$	1 0 0 0 1
$-q_0 2^4 d$	1 0 1 0 $\{q_0 = 1\}$
$s^{(4)}$	0 1 1 1
s 7	0 1 1 1
q 11	1 0 1 1

Fig.11:- Integer Division Algorithm

Description of Integer Division Algorithm:--

Firstly write the z and d on the top. We would write 2^4.d on the top instead of d because it is shifted towards right by 5 bits. Initially we assume z<d Initially take $s^{(0)}$ equals to z i.e. 0111 0101. Now calculate $2s^{(0)}$ by shifting one bit towards left i.e. 01110 101. Now check that $2s^{(0)}$ is greater than or less than d. 1110 < 1010 (14>10) digits are greater than d so quotient q would be equal to 1.

if digits are less than d so quotient would become equal to 0. Now subtract $q_3 2^4 d$ with $2s^{(0)}$. as q_3 is 1 and 2^4.d is 1010 so the result obtained would be $s^{(1)}$ i.e. 0100101. Now we would calculate $2s^{(1)}$ i.e. 0100101 by shifting one bit towards left and now again we have to check the condition. "Whether 01001> 1010". when we subtract the term $q_2 2^4 d$ with $2s^{(1)}$ (as q_2 is 0 so whole term would be 0000) we get $s^{(2)} = 2s^{(1)}$ i.e. 100101. Now evaluate $2s^{(2)}$ by shifting one bit left and check the condition .Now subtract the factor $q_1 2^4 d$ to achieve $s^{(3)}$. Repeat the process till we get the result in the form of $s^{(4)}$

A.2 Fractional Division Algorithm

© 2014, IJARCSSE All Rights Reserved

This division algorithm is useful to divide fractional numbers. Actually the position of the decimal remains as such till the end. When we shift the digits it crosses the decimal . We would write d_{frac} instead of shifted d because due to decimal it is already shifted.

z_{frac}		.	0	1	1	1	0 1 0 1	
d_{frac}		.	1	0	1	0		
$s^{(0)}$		.	0	1	1	1	0 1 0 1	
$2s^{(0)}$	0	.	1	1	1	0	1 0 1	
$-q_{-1}d$		.	1	0	1	0		$\{q_{-1}=1\}$
$s^{(1)}$		.	0	1	0	0	1 0 1	
$2s^{(1)}$	0	.	1	0	0	1	0 1	
$-q_{-2}d$		.	0	0	0	0		$\{q_{-2}=0\}$
$s^{(2)}$		.	1	0	0	1	0 1	
$2s^{(2)}$	1	.	0	0	1	0	1	
$-q_{-3}d$		.	1	0	1	0		$\{q_{-3}=1\}$
$s^{(3)}$		.	1	0	0	0	1	
$2s^{(3)}$	1	.	0	0	0	1		
$-q_{-4}d$		.	1	0	1	0		$\{q_{-4}=1\}$
$s^{(4)}$		.	0	1	1	1		
s_{frac}	0	.	0	0	0	0	0 1 1 1	
q_{frac}		.	1	0	1	1		

Fig.12:- Fractional Division Algorithm

Description of Fractional Division Algorithm:-

Initially write z_{frac} and d_{frac} and $s^{(0)}$ would be initially equal to z_{frac} and evaluate $2s^{(0)}$ by shifting one bit left.Now check the condition whether 0.1110 is greater than .1010 or not. We found that .14> .10 so q_{-1}=1. Subtract the factor q_{-1}d to get $s^{(1)}$ and then shift by one bit and again check the condition of greater than or less than according to which quotient is decided. This process would be repeated again and again till $s^{(4)}$ is achieved.

B. Restoring Unsigned Divider Algorithm

It is called restoring because when carry is zero it restores the previous value instead of obtaining new value. In this we would perform addition instead of subtraction. The algorithm Pictorial Representation is as shown below in Fig.13:-

z		0	1	1	1	0 1 0 1	
2^4d	0	1	0	1	0		
-2^4d	1	0	1	1	0		
$s^{(0)}$	0	0	1	1	1	0 1 0 1	
$2s^{(0)}$	0	1	1	1	0	1 0 1	
$+(-2^4d)$	1	0	1	1	0		
$s^{(1)}$	0	0	1	0	0	1 0 1	
$2s^{(1)}$	0	1	0	0	1	0 1	
$+(-2^4d)$	1	0	1	1	0		
$s^{(2)}$	1	1	1	1	1	0 1	
$s^{(2)}=2s^{(1)}$	0	1	0	0	1	0 1	
$2s^{(2)}$	1	0	0	1	0	1	
$+(-2^4d)$	1	0	1	1	0		
$s^{(3)}$	0	1	0	0	0	1	
$2s^{(3)}$	1	0	0	0	1		
$+(-2^4d)$	1	0	1	1	0		
$s^{(4)}$	0	0	1	1	1		
s						0 1 1 1	
q						1 0 1 1	

Fig.13:- Restoring Unsigned divider Algorithm

Description of Restoring Unsigned Divider:-

Write the z and 2^4d and (-2^4d).Now $s^{(0)}$ will remain equal to z . Shift the number by one bit towards left and add (-2^4d) to it. We get $s^{(1)}$ with carry 1 .If the carry is found the quotient must be 1 otherwise 0 and if the quotient is zero then previous value would be stored in partial remainder instead of new value. As above carry occurs so that q_3=1. As quotient is 1 means that new value would be stored so shift one but towards left and $2s^{(1)}$. Now again add the compliment factor to it we get $s^{(2)}$ as a result of addition with no carry. No carry obtained indicates that quotient must be 0 and value is restored to previous one. i.e. q_2=0 and $s^{(2)}=2s^{(1)}$. (Restoring). Now repeat the addition with complimented factor and again check carry. And then apply carry condition and restoring condition till $s^{(4)}$ is achieved.

C. Non Restoring Unsigned Division Algorithm

In this we store the difference in the partial remainder register.This leads to the partial remainder being temporarily incorrect hence the name is Non Restoring. The main concept of this division technique is if during addition carry is found then quotient would be 1 and the term (-2^4d) is added and if carry is not present then quotient is zero and the term (2^4d) is get added. The term containing negative sign is complement of the positive one. Firstly write z, (2^4d), (2^4d) with extra sign bit representing negative and positive. Initially $s^{(0)}$ would equal to z. and add the factor (-2^4d) to it. When Carry

© 2014, IJARCSSE All Rights Reserved

1 is obtained so set q_3 =1 and add the factor (-2^4d) and perform the addition. We found that no carry is obtained so q_2=0 and the factor (2^4d) is being added. Again perform shifting and adding accordingly with factor which depends on quotient. We found s $^{(4)}$ 0 0111. As it contains an extra bit 0 that means it is a positive no. and also it has been shifted towards left by 4 bits from its original position so shift it towards right by 4 bit to achieve original remainder. The non restoring division technique example is shown in the Fig.14:--

z		0 1 1 1	0 1 0 1
2^4d	0	1 0 1 0	
-2^4d	1	0 1 1 0	
$s^{(0)}$	0	0 1 1 1	0 1 0 1
$2s^{(0)}$	0	1 1 1 0	1 0 1
$+(-2^4d)$	1	0 1 1 0	
$s^{(1)}$	0	0 1 0 0	1 0 1
$2s^{(1)}$	0	1 0 0 1	0 1
$+(-2^4d)$	1	0 1 1 0	
$s^{(2)}$	1	1 1 1 1	0 1
$2s^{(2)}$	1	1 1 1 0	1
$+2^4d$	0	1 0 1 0	
$s^{(3)}$	0	1 0 0 0	1
$2s^{(3)}$	1	0 0 0 1	
$+(-2^4d)$	1	0 1 1 0	
$s^{(4)}$	0	0 1 1 1	
s			0 1 1 1
q			1 0 1 1

Fig.14:- Non Restoring Unsigned Divider

D. Non Restoring Signed Division Algorithm

In this technique both operands are negative. The main concept of this is "When the partial remainder is achieved then the sign of remainder and divisor is checked". If the sign of partial remainder and the sign of divisor is not same then quotient would be set to (-1) and addition would perform. But if the sign of partial remainder and the divisor is same then quotient would be 1 and subtraction would perform. Fig.15 shows how this algorithm perform division of signed operands.

z		0 0 1 0	0 0 0 1
2^4d	1	1 0 0 1	
-2^4d	0	0 1 1 1	
$s^{(0)}$	0	0 0 1 0	0 0 0 1
$2s^{(0)}$	0	0 1 0 0	0 0 1
$+2^4d$	1	1 0 0 1	
$s^{(1)}$	1	1 1 0 1	0 0 1
$2s^{(1)}$	1	1 0 1 0	0 1
$+(-2^4d)$	0	0 1 1 1	
$s^{(2)}$	0	0 0 0 1	0 1
$2s^{(2)}$	0	0 0 1 0	1
$+2^4d$	1	1 0 0 1	
$s^{(3)}$	1	1 0 1 1	1
$2s^{(3)}$	1	0 1 1 1	
$+(-2^4d)$	0	0 1 1 1	
$s^{(4)}$	1	1 1 1 0	
$+(-2^4d)$	0	0 1 1 1	
$s^{(4)}$	0	0 1 0 1	
s			0 1 0 1
q			-1 $1-1$ 1

Fig. 15:- Non Restoring Signed Divider Algorithm

Description of Non Restoring Signed Divider:-

Firstly write s, (2^4d) , (-2^4d). Initially $s^{(0)}$ is same as z. Now shift by one bit towards left . check the sign of s and d. here sign of s is positive and the sign of d is negative. Both are different so set q_3= (-1) and add the factor (2^4d) with partial remainder.We get result of this addition in the form of $s^{(1)}$ .Again shift by one bit towards left and also check the condition of sign. The sign of $s^{(1)}$ and d are same i.e. negative so set the q_2 to 1 and subtraction would perform.

Why the result obtained from non restoring and restoring algorithm is same???

The main concept in this algorithm is "To check the condition and decide whether the restoring of result is required or not??"

Now in the first graph the value 148 is restored as such upto one level because the value 148 is less than the previous value i.e.234 so no shifting would be performed and Restoring of old value would be done. But at next level multiply by factor 2 and result obtained is 296 then repeat the process of subtracting 160 and get 136 and multiply it by 2 and result obtained is 272 and after that both the graphs are same again and the final result obtained is 112 .

Now in the second graph the result comes 148 then no condition is checked and subtraction of factor 160 is done and after that it is shifted by a factor of 2 means multiplied by 2 and then factor 160 is added to the partial remainder value. Now multiply by a factor by 2 and we got the result and then again add 160 and then multiply by 2 and we get the result 272 and same process would be repeated again and again and we got the result 112 which is same as found by restoring technique algorithm. In this algorithm the concept of incorrect value comes and always at a next level a new value is stored not the

© 2014, IJARCSSE All Rights Reserved

old one. It is to be noted that we have subtracted the factor first 160 and then addition of 160 is done that's why the value at which 160 is added and subtracted would remain as such and we perform the operations again on the same value that's why we got the same result. The graphical notations are shown in Fig.16 and Fig.17 with particular values at the sharp edges.

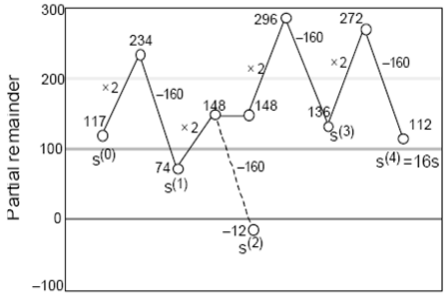

Fig.16:- Restoring Divider

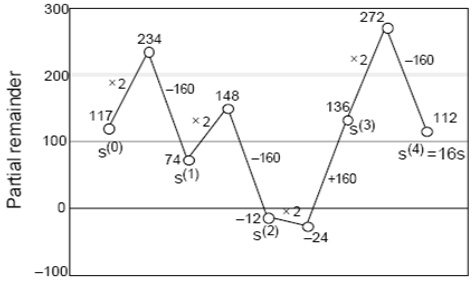

Fig.17:- Non Restoring Divider

V. **Square Rooting Technique**

In this algorithm the z is radicand, q is square root and s is remainder $(z- q^2)$.We are discussing here two techniques of square rooting techniques.

A. Restoring Shift/Subtract Square Rooting Algorithm

The Square rooting can be formulated as a sequence of shift and subtract operations. The operands which are used for square root operation in this algorithm are fractional instead of integers. To choose a next square root digit q_{-j} from the set (0,1), we perform a trial subtraction of $2q^{(j-1)}+ 2^{-j}$. The preceeding algorithm is similar to restoring division so called restoring square rooting. In this algorithm the previous value is restored if the condition is not satisfied .

		Root digit	Partial root
z (radicand = 118/64)	0 1 . 1 1 0 1 1 0		
$s^{(0)} = z - 1$	0 0 0 . 1 1 0 1 1 0	$q_0 = 1$	1.
$2s^{(0)}$	0 0 1 . 1 0 1 1 0 0		
$-[2 \times (1.)+2^{-1}]$	1 0 . 1		
$s^{(1)}$	1 1 1 . 0 0 1 1 0 0	$q_{-1} = 0$	1.0
$s^{(1)} = 2s^{(0)}$ Restore	0 0 1 . 1 0 1 1 0 0		
$2s^{(1)}$	0 1 1 . 0 1 1 0 0 0		
$-[2 \times (1.0)+2^{-2}]$	1 0 . 0 1		
$s^{(2)}$	0 0 1 . 0 0 1 0 0 0	$q_{-2} = 1$	1.01
$2s^{(2)}$	0 1 0 . 0 1 0 0 0 0		
$-[2 \times (1.01)+2^{-3}]$	1 0 . 1 0 1		
$s^{(3)}$	1 1 1 . 1 0 1 0 0 0	$q_{-3} = 0$	1.010
$s^{(3)} = 2s^{(2)}$ Restore	0 1 0 . 0 1 0 0 0 0		
2s(3)	1 0 0 . 1 0 0 0 0 0		
$-[2 \times (1.010)+2^{-4}]$	1 0 . 1 0 0 1		
$s^{(4)}$	0 0 1 . 1 1 1 1 0 0	$q_{-4} = 1$	1.0101
$2s^{(4)}$	0 1 1 . 1 1 1 0 0 0		
$-[2 \times (1.0101)+2^{-5}]$	1 0 . 1 0 1 0 1		
$s^{(5)}$	0 0 1 . 0 0 1 1 1 0	$q_{-5} = 1$	1.01011
$2s^{(5)}$	0 1 0 . 0 1 1 1 0 0		
$-[2\times(1.01011)+2^{-6}]$	1 0 . 1 0 1 1 0 1		
$s^{(6)}$	1 1 1 . 1 0 1 1 1 1	$q_{-6} = 0$	1.010110
$s^{(6)} = 2s^{(5)}$ Restore	0 1 0 . 0 1 1 1 0 0		
s (remainder = 156/64)	0 . 0 0 0 0 0 1 0 0 1 1 1 0 0		
q (root = 86/64)	1 . 0 1 0 1 1 0		
		q_{-7} = 1, so round up	

Fig.18:-- Restoring Square Rooting Algorithm

Description of Restoring Square Rooting Algorithm:--

Firstly we write z at the top and then subtract z by 1 i.e. z-1 which is equivalent to $s^{(0)}$ and initially assume q_0=1 and $q^{(0)}$=1. $s^{(0)}$= z-1. Now shift the number by one digit towards left and subtract the factor ($-(2x(1.) + 2^{-1})$) .

q_{-1} can be decided by the first digit (from the left) of $s^{(1)}$ and $q^{(1)}$= 1.(q_{-1}) actually the first digit's complement of the $s^{(1)}$ is q_{-1} and q_{-1}=0 and $q^{(1)}$= 1.0. As q_{-1}=0 so partial remainder would restore the previous value instead of new one which

© 2014, IJARCSSE All Rights Reserved

indicates $s^{(1)} = 2s^{(0)}$.Now evaluate $2s^{(1)}$ by shifting one bit towards left $2s^{(1)}$ .The factor which is to be subtracted from $2s^{(1)}$ is $(-(2x(1.0) + 2^{-2}))$

$s^{(2)}$= 011.011000 – 10.01 = 001.001000. As the first digit from left is 0 so q_{-2}=1 and $q^{(2)}$= 1.01,As q_{-2}=1 so partial remainder would store new value instead of old one. Now shift one bit of $s^{(2)}$ towards left side and it would be equivalent to $2s^{(2)}$ and then subtract the factor

($-(2x(1.01) + 2^{-3})$) and the result would be $s^{(3)}$. The first bit is 1 so q_{-3}=0. As q_{-3}=0 so partial remainder would restore its previous value instead of new one. This process of restoring would proceed further according to the first bit of the previous partial remainder.

The partial remainder is achieved but in the form of $s^{(6)}$. Evaluate s. The quotient would be the group of q_0 to q_{-6} in ascending order i.e. 1.010110.

B. Non Restoring Square Rooting Algorithm

In the non restoring square rooting algorithm no restoring condition is used. A concept of incorrect value is used. Because in this incorrect value of the partial remainder is taken so that this algorithm is called non restoring square rooting algorithm. In this root digits are chosen from the set (-1,1) and the resulting root is converted to binary format as sown in Fig.19

		Root digit	Partial root
z (radicand = 118/64)	0 1 . 1 1 0 1 1 0		
$s^{(0)} = z - 1$	0 0 0 . 1 1 0 1 1 0	$q_0 = 1$	1.
$2s^{(0)}$	0 0 1 . 1 0 1 1 0 0	$q_{-1} = 1$	1.1
$-[2 \times (1.)+2^{-1}]$	1 0 . 1		
$s^{(1)}$	1 1 1 . 0 0 1 1 0 0	$q_{-2} = -1$	1.01
$2s^{(1)}$	1 1 0 . 0 1 1 0 0 0		
$+[2 \times (1.1)-2^{-2}]$	1 0 . 1 1		
$s^{(2)}$	0 0 1 . 0 0 1 0 0 0	$q_{-3} = 1$	1.011
$2s^{(2)}$	0 1 0 . 0 1 0 0 0 0		
$-[2 \times (1.01)+2^{-3}]$	1 0 . 1 0 1		
$s^{(3)}$	1 1 1 . 1 0 1 0 0 0	$q_{-4} = -1$	1.0101
$2s(3)$	1 1 1 . 0 1 0 0 0 0		
$+[2 \times (1.011)-2^{-4}]$	1 0 . 1 0 1 1		
$s^{(4)}$	0 0 1 . 1 1 1 1 0 0	$q_{-5} = 1$	1.01011
$2s^{(4)}$	0 1 1 . 1 1 1 0 0 0		
$-[2 \times (1.0101)+2^{-5}]$	1 0 . 1 0 1 0 1		
$s^{(5)}$	0 0 1 . 0 0 1 1 1 0	$q_{-6} = 1$	1.010111
$2s^{(5)}$	0 1 0 . 0 1 1 1 0 0		
$-[2\times(1.01011)+2^{-6}]$	1 0 . 1 0 1 1 0 1		
$s^{(6)}$	1 1 1 . 1 0 1 1 1 1	Negative;	(–17/64)
$+[2\times(1.01011)-2^{-6}]$	1 0 . 1 0 1 1 0 1	Correct	
$s^{(6)}$ Corrected	0 1 0 . 0 1 1 1 0 0		(156/64)
s (remainder = 156/64)	0 . 0 0 0 0 1 0 0 1 1 1 0 0		($156/64^2$)
q (binary)	1 . 0 1 0 1 1 1		(87/64)
q (corrected binary)	1 . 0 1 0 1 1 0		(86/64)

Fig. 19:-- Non Restoring Square Rooting Algorithm

Description of Non Restoring Square Root Algorithm:--

Write z (radicand) at the top and then the process of the algorithm is similar to the restoring one except the restoring condition. In this algorithm no condition is checked. In this quotient would be -1 instead of 0. In this there is a concept of incorrect value. Everytime a new value would be stored in partial product register instead of previous value.

VI. **Conclusion**

We have studied the different basic techniques which are helpful in development of a digital design. We studied the arithmetic techniques like multiplication, division and many more. Some techniques improve the speed while some saves the area. A VLSI designer always think about area constraint first and then about power consumption and Speed. In multiplication technique basically two types of algorithms are found. right shift algorithm and left shift algorithm. We always prefer right shift algorithm because in this algorithm k-bit adder is used to perform operation on the k-bit operands while 2k-bit adder is required to perform operation on operands when we use left shift algorithm. When we use 2k-bit adder then naturally area would be increased in comparison with right shift algorithm so right shift algorithm is preferred to save area. After examining shift/add multiplication schemes and their various implementations, we note that there are two ways two speed up the underlying multioperand addition: one is reducing the number of operands to be added leads to high radix multipliers like booth recoding multiplier and other one is devising hardware multioperand adders that minimize the latency and maximize the throughput.

When we deal with negative numbers we always put one extra bit to the number which shows the number is negative. But if extra bit 1 is attached to the number it makes it negative. In multiplication technique Shift and addition is performed but in division shift and subtraction is performed.In hardware realization we take adder not subtractor because adder development is easy then subtractor one. When during division we want to perform subtraction we take the complement of the number and at the other end 1 is supplied to the adder and it would perform subtraction in the form of addition. As per rules 2's complement of a number is equivalent to the positive number. We can also perform square root

© 2014, IJARCSSE All Rights Reserved

operation on a number by using algorithm of square rooting which also deal with fractional numbers. The hardware designs can be implemented or burned on a FPGA for the development of an integrated circuit.

References

[1] D. Bindel, J. Demmel, W. Kahan, and O. Marques. On computing Givens rotations reliably and efficiently ACM Transactions on Mathematical Software, 28(2):206–238, 2002.

[2] S.R. Dicker et al. Cbm observations with the Jodrell Bank - iac interferometer at 33 Ghz. Mon. Not. R. Astron. Soc., 00:1–12, 2000.

[3] M. D. Ercegovac and Muller, J.-M. Complex division with prescaling of operands Proc. IEEE ASAP03, 2003.

[4] M. D. Ercegovac and Lang, T. Digital Arithmetic. Morgan Kaufmann, 2004.

[5] A. Li, D.B. Sharp, and B.J. Forbes. Improving the high frequency content of the input signal in acoustic pulse reflectometry. In Proc. of the International Symposium on Musical Acoustics, pages 391–394, 2001.

[6] X. Li et al. Design, implementation and testing of extended and mixed precision BLAS. ACM Transactions on Mathematical Software, 28(2):152–205, 2002.

[7] R. D. Mcilhenny. Complex Number On-line Arithmetic for Reconfigurable Hardware: Algorithms, Implementations, and Applications. PhD thesis, University of California at Los Angeles, 2002.

[8] J. A. Pineiro, Algorithms and Architectures for Elementary Function Computation, PhD Dissertation, Department of Electronics and Computation, University of Santiago de Compostela, 2003.

[9] G. Vandersteen et al. Comparison of arithmetic functions with respect to Boolean circuits. In 58th ARFTG Conference Digest RF Measurements for a Wireless World, 2001.

[10] Ercegovac, M.D., and Muller, J.M.: 'Design of a complex divider', Proc. SPIE on Advanced Signal Processing Algorithms, Architectures, and Implementations XIV, October 2004, 5559, pp. 51–59

[11] Weaver, B.J., Zakharov, Y.V., and Tozer, T.C.: 'Multiplication-free division of complex numbers'. 6th IMA Conf. on Mathematics in Signal Processing, Cirencester, UK, December 2004, pp. 211–214

[12] Zakharov, Y.V., and Tozer, T.C.: 'Multiplication-free iterative algorithm for LS problem', Electron. Lett., 2004, 40, (9), pp. 567–569

[13] Solomentsev, E.D. (2001), "Complex number", in Hazewinkel, Michiel, *Encyclopaedia of Mathematics*, Kluwer Academic Publishers, ISBN 978-1556080104

[14] [DEM2002] G. De Micheli, R. Ernst, and W. Wolf (eds.), Readings in Hardware/Software Co-Design, Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[15] Xilinx Inc, PicoBlaze 8-bit Microcontroller for Virtex-E and Spartan-II/IIE Devices, application note XAPP213 (v2.0), Dec. 2002; http://www.xilinx.com

[16] [GIL2003] W. J. Gilbert and W. K. Nicholson, Modern Algebra with Applications, John Wiley & Sons, Hoboken, NJ, 2003.

[17] [ROS2000] K. H. Rosen (Editor-in-Chief), Handbook of Discrete and Combinatorial Mathematics, CRC Press, Boca Raton, FL, 2000.

[18] [VZG2003] J. von zur Gathen and J. Gerhard, Modern Computer Algebra, Cambridge University Press, Cambridge, UK, 2003.

[19] [XSA2002] XSA Board V1.1, V1.2 User Manual, June 2002; http://www.xess.com.

[20] [OBE1964] S. F. Oberman and M. Flynn, Advanced Computer Arithmetic Design, Wiley- Interscience, Hoboken, NJ, 2001.

[21] [PAR1999] Behrooz Parhami, Computer Arithmetic, Algorithms and Hardware Designs, Oxford University Press, New York, 1999.

[22] Synthesis of Arithmetic Circuts (FPGA , ASIC and Embedded Systems) By Jean- Pierre Deschamps , Gery Jean Antoine Bioul , Gustavo D. Sutter.

[23] Koren, I., Computer Arithmetic Algorithms, Prentice – Hall, 1993

[24] Research Paper " FPGA implementation of multiplication based floating-point divide and square root algorithms: In: Proceedings of the 14th IEEE Symposium on Computer Arithmetic, 1999, pp. 96–10

[25] A. Edelman, The mathematics of the Pentium division bug. SIAM Rev. 39(1):54–67 (1997).

[26] M. D. Ercegovac, A higher radix division with simple selection of quotient digits. In Proceedings of the 6th IEEE Symposium Computer Arithmetic, July 1983, pp. 94–98.

[27] M. D. Ercegovac and T. Lang, On-the-fly conversion of redundant into conventional representations. IEEE Trans. Comput., 36(7): 895–897 (1987).

[28] M. D. Ercegovac and T. Lang, Simple radix-4 division with operands scaling. IEEE Trans. Comput., 39(9): 1204–1207 (1990).

[29] Xilinx XUP Virtex II Pro Development System', website referred http://www.xilinx.com/univ/xupv2p.html.

[30] M. D. Ercegovac and Lang, T. Digital Arithmetic. Morgan Kaufmann, 2004.

[31] R.D. Mcilhenny. Complex Number On-line Arithmetic for Reconfigurable Hardware: Algorithms, Implementations, and Applications. PhD thesis, University of California at Los Angeles, 2002.

[32] M. D. Ercegovac, Lang, T., and Montuschi, P. Very-high radix division with prescaling and rounding. IEEE Transactions on Computers, 43(8):909–918, 1994.

© 2014, IJARCSSE All Rights Reserved

[33] A. Pineiro, Algorithms and Architectures for Elementary Function Computation, PhD Dissertation, Department of Electronics and Computation, University of Santiago de Compostela, 2003.
[34] Parhami , B., " Analysis of the look table size for square Rooting ," Proc. 33rd Asilomar Conference . signals, systems , and computers , pp.1327-1330, October 1999.
[35] Schwarz, E. M. and M.J. Flyn," Hardware starting Approximation method and its application to the Square Root operation," IEEE trans. Computers, vol.45, No.12, pp. 1356-1369, 1996.
[36] Montuschi, P., and m. Mezzalama ," Survey of Square Rooting algorithm" Proc. IEEE., Pt.E, Vol.137, pp. 31-40, 1990.
[37] Computer Arithmetic (algorithms and hardware designs) By Behrooz Parhami, Oxford University Express.
[39] Wayne Wolf "Reconfigurable Computing"The Morgan Kaufmann Series in Systems on Silicon, Elsevier, 2008.
[40] M. D. Ercegovac and T. Lang, Digital Arithmetic, Morgan Kaufmann, San Francisco, CA, 2004.
[41] G. Even, P.-M. Seidel, and W. E. Ferguson, A parametric error analysis of Goldschmidt's division algorithm. In: Proceedings of the 16th IEEE Symposium on Computer Arithmetic, June 2003.
[42] B. Parhami, Computer Arithmetic, Algorithms and Hardware Designs, Oxford University Press, New York, 1999.

© 2014, IJARCSSE All Rights Reserved