ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013

VHDL Implementation of Non Restoring Division Algorithm Using High Speed Adder/Subtractor

Sukhmeet Kaur[1], Suman[2], Manpreet Singh Manna[3], Rajeev Agarwal[4]

M.Tech Student, ECE, SSIET, Derabassi, Punjab, India [1]

Asst. Professor, ECE, SSIET, Derabassi, Punjab, India [2]

Associate Professor, EIE, SLIET (Deemed University) Longowal, Sangrur, India [3]

Assistant Professor, ECE, SGIT Ghaziabad, U.P. India[4]

ABSTRACT: Binary division is basically a procedure to determine how many times the divisor D divides the dividend B thus resulting in the quotient Q. At each step in the process the divisor D either divides B into a group of bits or it does not. The divisor divides a group of bits when the divisor has a value less than or equal to the value of those bits. Therefore, the quotient is either 1 or 0. The division algorithm performs either an addition or subtraction based on the signs of the divisor and the partial remainder. There are number of binary division algorithm like Digit Recurrence Algorithm restoring, non-restoring and SRT Division (Sweeney, Robertson, and Tocher), Multiplicative Algorithm, Approximation Algorithms, CORDIC Algorithm and Continued Product Algorithm. This paper focus on the digit recurrence non restoring division algorithm, Non restoring division algorithm is designed using high speed subtractor and adder. High speed adder and subtractor are used to speed up the operation of division. Designing of this division algorithm is done by using VHDL and simulated using Xilinx ISE 8.1i software has been used and implemented on FPGA xc3s100e-5vq100.

Keywords: Restoring division algorithm, Non-restoring division algorithm, high speed adder, high speed subtractor

I.INTRODUCTION

Computers have evolved rapidly since their creation. However, there is one thing that has not changed: 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 addition and multiplication, division is the least used operation. However, computers will experience performance degradation if division is ignored [I, 2, 3]. A survey by Oberman and Flynn [9] presents the main algorithms used for implementing division in hardware. There are three main classes for hardware-oriented division algorithms: digit recurrenc, functional iteration and table based methods.

Each method has its own advantages [I], 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 than division by convergence [2, 4, 5]. Restoring, non-restoring and SRT dividers are representative algorithms for digit recurrence division. Division is equivalent to repeat subtraction of the divisor from the dividend until the quantity left is smaller in magnitude than the divisor. The number of subtractions is the quotient, and the quantity left is the remainder. This process, if done straight forwardly, is very time consuming. It is substantially speeded if the most significant digits of the divisor and dividend are aligned before the first subtraction, and the divisor then shifted to the right one position whenever the partial remainder become smaller than the divisor before shifting. One shift may be necessary before any subtraction, if the initial alignment makes the divisor larger than the dividend" In binary, at most one subtraction can be made between shifts except as noted below. Two conventional techniques avoid the need to compare the remainder with the divisor after every subtraction. In restoring division, subtraction continues until the sign of the partial remainder changes; the change causes an immediate addition of the divisor and a corresponding decrement of the accumulating quotient, before the right shift. In non restoring division, the sign change causes a shift followed by one or more additions until the sign changes back.

II.RESTORING DIVISION

In the restoring division method the quotient is represented using a non-redundant number system This is the "paper-and-pencil" usual algorithm. Its main characteristic is the full width comparisons required to deduce the new quotient digit[6]. In restoring division, the divisor is shift-positioned and subtracted from the dividend. If subtraction of the divisor produces a negative result at any bit position relative to the dividend, the operation at that bit position is

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013

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. Complete hardware for restoring division is shown in Fig.1.In this figure an n-bit positive divider is loaded into register M and n- bit dividend is loaded into register Q at the start of the operation. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. The result after the last restore operation is the remainder.

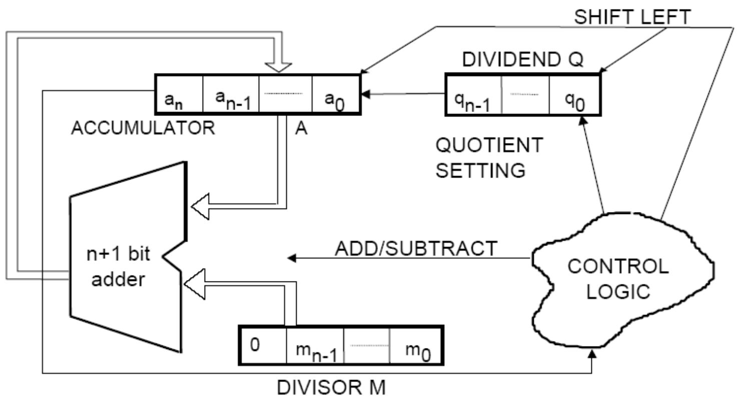


Fig. 1 Hardware Design of Restoring Division Algorithm.

Restoring division algorithm is very similar to manually performing long division. Algorithm for restoring division is mentioned below along with flowchart shown in Fig.2.

- Set Count to 0 and put 0 in A register.
- Start loop for n times
- Shift A & Q left one binary position
- Subtract M from a, placing the answer back in A
- if the sign of A < 0, set Q0 to 0 and add M back to A (restore A);
- otherwise, set q0 to 1
- Check for count, when count = n-1 then stop the loop.

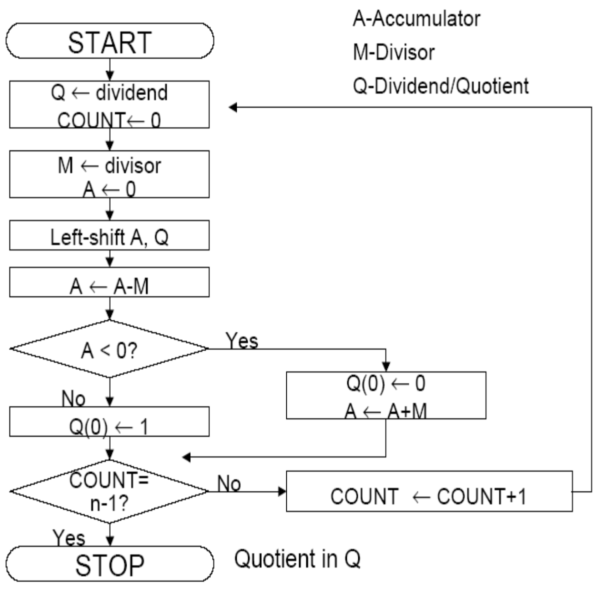


Fig. 2 Flow chart of Restoring Division Algorithm.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013

III.NON-RESTORING DIVISION

Non-restoring Division Algorithm (NrDA) 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 { I, - I} instead of the conventional binary digit set [7,9]. 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 [8]. We dismantle Q into bits.
Algorithm for restoring division is mentioned below along with flowchart shown in Fig.3.
1. Subtract the divisor from the most significant bit (MSB) of the dividend.
2. “Bring down” the next MSB of the divisor and append it to the result of step 1.
3. Check the sign for the result of step 2. If the result of step 2 is positive:
    a. Set the next MSB of the quotient to 1.
    b. Subtract the divisor from the result to produce a new result.
If the result from step 2 is negative:
    a. Set the next MSB of the quotient to 0.
    b. Add the divisor to the result to produce a new result.
4. Repeat steps 2 and 3 until all bits of the quotient are determined.

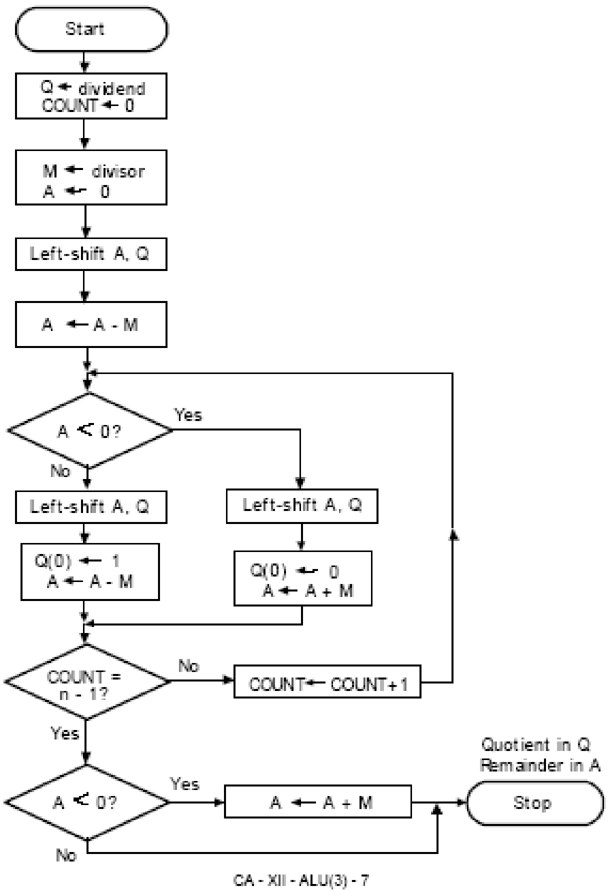


Fig. 3 Flowchart of Restoring Division Algorithm.

IV.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

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013

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 depends on intermediate carry depends only on input bits.[10]

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 with be generated at that bit, or if a carry will be propagated through that bit. Logic diagram of full-adder with carry generation and carry propagation is shown in Fig.4.

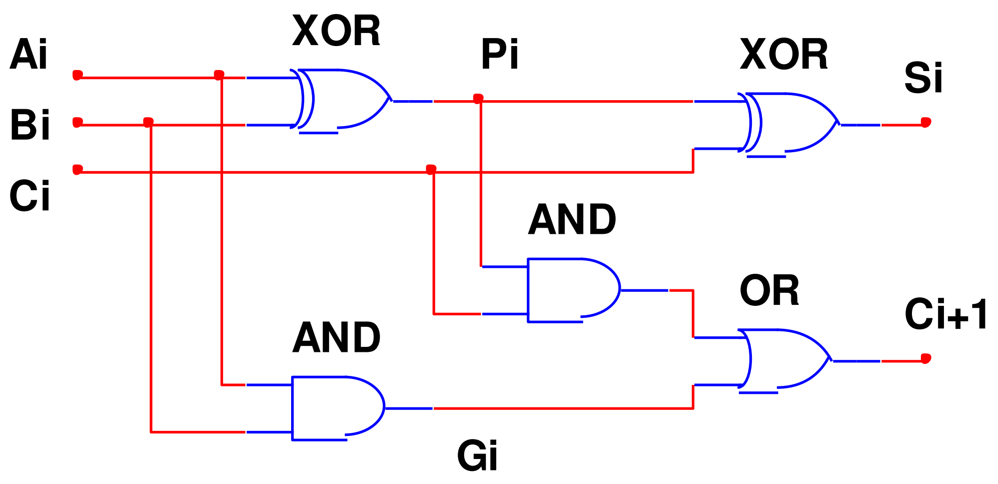


Fig. 3 Logic diagram of Full-adder with Carry Generation and Carry Propagation

From above Fig.3, we can write two functions : Carry generate and Carry propagate

Pi = Ai XOR Bi
Gi = Ai AND Bi

The Pi is equal to binary 1 when both A and B are binary 1, regardless of the value of the incoming carry to the stage Ci,

Where Ai & Bi = Inputs
Ci = Carry input
Gi = Carry generation
Pi = Carry Propagation
Si = Sum Output
Ci+1 = Carry Output

The output SUM and CARRY can be expressed as

Si = Pi XOR Ci
Ci+1 = Gi + PiCi

For 4-bit binary addition, we can write the Carry propagate output as follows :-

Pi = Ai XOR Bi
P0 = A0 XOR B0
P1 = A1 XOR B1
P2 = A2 XOR B2
P3 = A3 XOR B3

For 4-bit binary addition, we can write the Carry generate output as follows :-

Gi = Ai AND Bi
G0 = A0 AND B0

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013

G1 = A1 AND B1
G2 = A2 AND B2
G3 = A3 AND B3

For 4-bit binary addition, we can write the SUM output as follows :-

Si = Pi XOR Ci
S0 = P0 XOR C0
S1 = P1 XOR C1
S2 = P2 XOR C2
S3 = P3 XOR C3

For 4-bit binary addition, we can write the Carry output as follows :-
Ci+1 = Gi + PiCi
C0 = Carry in
C1 = Go + P0C0
C2 = G1 + P1C1 = G1 + P1(Go + P0C0) = G1 + P1Go + P1P0C0
C3 = G2 + P2C2 = G2 + P2(G1 + P1Go + P1P0C0) = G2 + P2G1 + P2P1G0+P2P1P0C0
C4 = G3 + P3C3 = G3 + P3(G2 + P2G1 + P2P1G0+P2P1P0C0)
= G3 + P3G2 + P3P2G1+P3P2P1G0+ P3P2P1P0C0

In above calculation, we can see that C1, C2, C3 & C4 are depends only on Co (Carry in), not on previous carry. There is not much difference between high speed adder and subtractor circuit. Only difference is 2nd input of the adder will inverted and then added using adder circuit. So we can use same circuit but with slightly modification as shown in Fig.5 diagram.

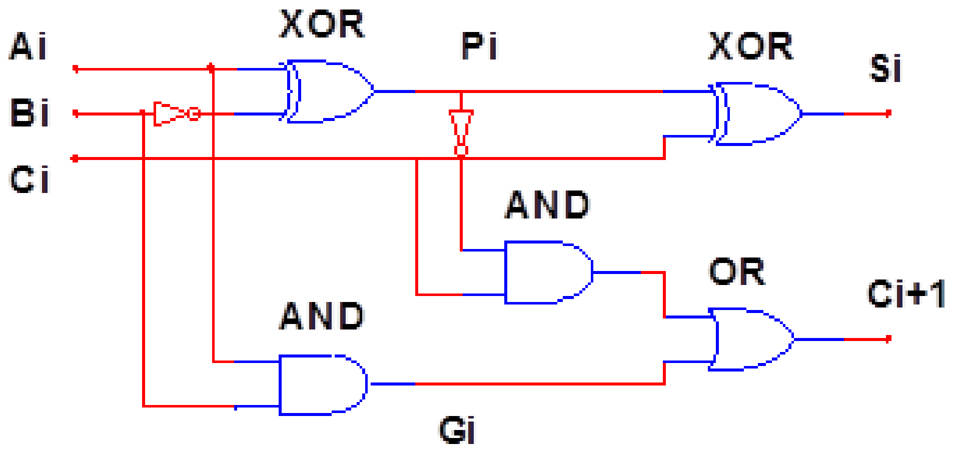


Fig. 5 Logic Diagram of Full-Subtractor with Carry Generation and Carry Propagation

V.STEP BY STEP CALCULATION OF DIVISION OF 29 BY 11.

In this example we take 29 as dividend and store this value in register Dnd but in binary form, 9 as divisor and store its binary value in register D. We have taken two other registers AD and ZD for calculation and temporary storage of result as shown in Fig 6.

INPUTS FOR DIVISION:

DND (Dividend) = 000011101 = (01DH) = (+29)

D (Divisor) = 000001011 = (00BH) = (+11)

OUTPUTS FOR DIVISION:

Remainder (10 bits) = 0000000111= (007H)

Quotient (9 bits) = 000000011= (002H)

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013

STEPS			AD-Register	DND-Register (Dividend)
Initially		AD	0000000000	000011101(01D$_H$)
STEP1	1bit Left Shift	AD1	0000000000	0 0 0 1 1 1 0 1
	Sub D	-D	1111110101	
	Set DND$_0$	ZD1	1111110101	0 0 0 1 1 1 0 1 0
STEP2	1bit Left Shift	AD2	1111101010	0 0 1 1 1 0 1 0
	Add D	+D	0000001011	
	Set DND$_0$	ZD2	1111110101	0 0 1 1 1 0 1 0 0
STEP3	1bit Left Shift	AD3	1111101010	0 1 1 1 0 1 0 0
	Add D	+D	0000001011	
	Set DND$_0$	ZD3	1111110101	0 1 1 1 0 1 0 0 0
STEP4	1bit Left Shift	AD4	1111101010	1 1 1 0 1 0 0 0
	Add D	+D	0000001011	
	Set DND$_0$	ZD4	1111110101	1 1 1 0 1 0 0 0 0
STEP5	1bit Left Shift	AD5	1111101011	1 1 0 1 0 0 0 0
	Add D	+D	0000001011	
	Set DND$_0$	ZD5	1111110110	1 1 0 1 0 0 0 0 0
STEP6	1bit Left Shift	AD6	1111101101	1 0 1 0 0 0 0 0
	Add D	+D	0000001011	
	Set DND$_0$	ZD6	1111111000	1 0 1 0 0 0 0 0 0
STEP7	1bit Left Shift	AD7	1111110001	0 1 0 0 0 0 0 0
	Add D	+D	0000001011	
	Set DND$_0$	ZD7	1111111100	0 1 0 0 0 0 0 0 0
STEP8	1bit Left Shift	AD8	1111111000	1 0 0 0 0 0 0 0
	Add D	+D	0000001011	
	Set DND$_0$	ZD8	0000000011	1 0 0 0 0 0 0 0 1
STEP9	1bit Left Shift	AD9	0000000111	0 0 0 0 0 0 0 1
	Sub D	-D	1111110101	
	Set DND$_0$	ZD9	1111111100	0 0 0 0 0 0 0 1 0
	In final step when			
	ZD9$_0$ = 1, Add D		0000000111	0 0 0 0 0 0 0 1 1
	ZD9$_0$ = 0, Ignore			
RESULT			0000000111	0 0 0 0 0 0 0 1 1

Fig. 6 Step by Step Result of Division of 29 by 11.

VI.SIMULATION RESULT

The entire architecture is modelled using VHDL. The coding is done on Xilinx ISE 8.1i on Spartan 3 using target device: XC3s100e-5vq100at speed grade of -5. Simulation can be done using ModelSim SE 6.3f simulator. The simulation result for DIV_HSA is shown on Fig. 7 and Fig.8 respectively. Fig. 7 shows the top RTL view of the division circuit. In this figure we can see that division hardware has two inputs and two outputs. Divisor is stored in register D and dividend is stored in register Dnd. Quotient and reminder of division algorithm is stored in register quotient and reminder register respectively.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013

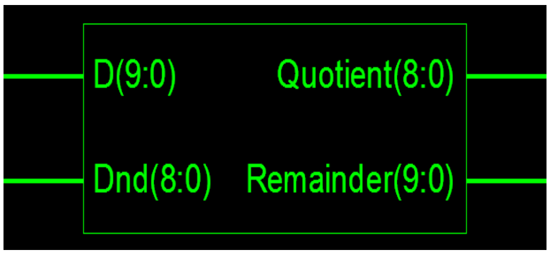


Fig. 7 Top RTL View of Non Restoring Division Algorithm.

Fig 8. Shows the final RTL view of the Division algorithm. in this diagram we can see that there are various block and that are performing various operation like calculation of partial division and calculation of remainder etc.

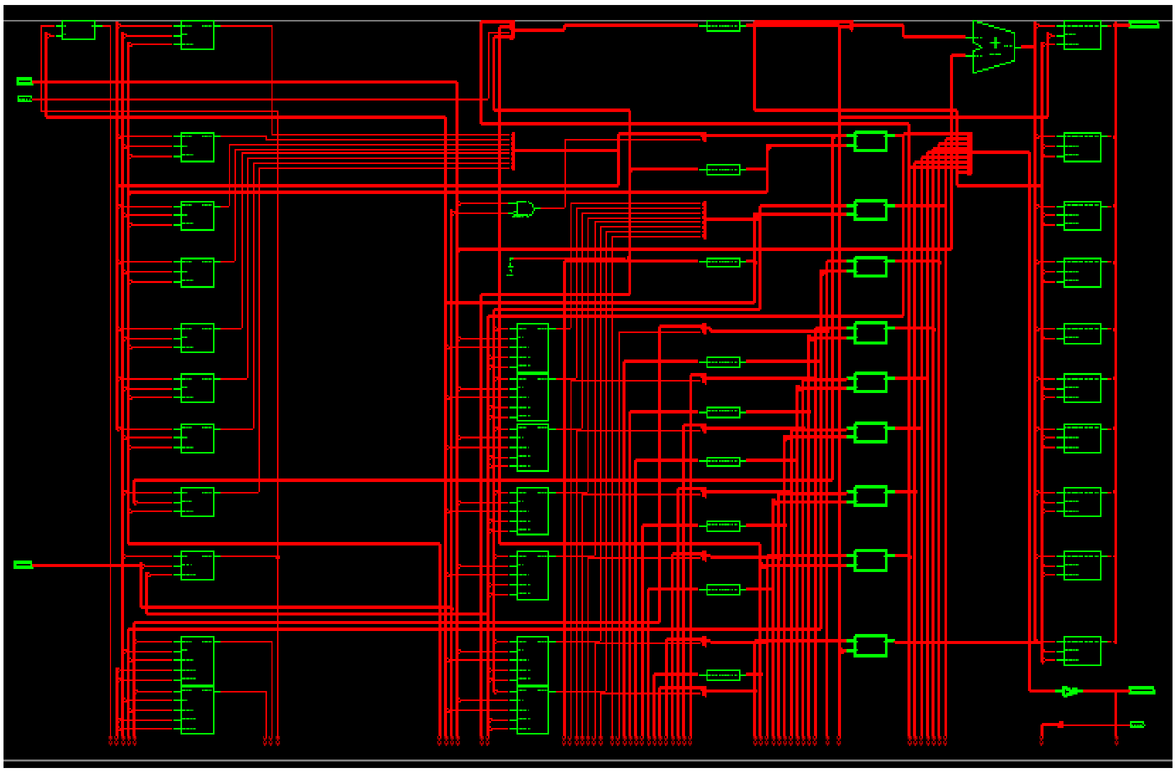
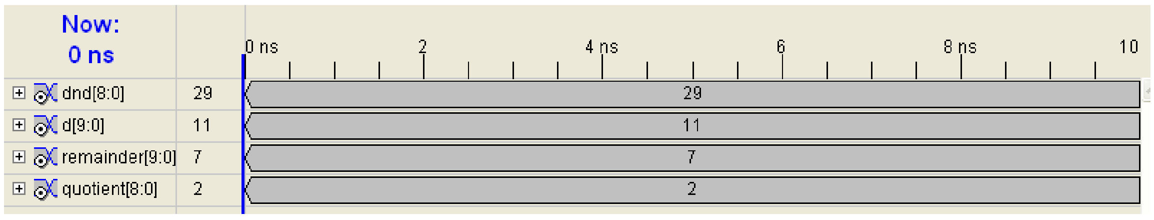


Fig. 8 RTL View of Non Restoring Division Algorithm.

Fig 9. Shows the simulation result of 29/11. We can see that 29 is stored in register Dnd which is dividend and 11 is stored in register D which is divisor. Both value are in decimal. After simulation of 29/11 quotient is stored in quotient and reminder is stored in reminder register this we can see in fig. 8.



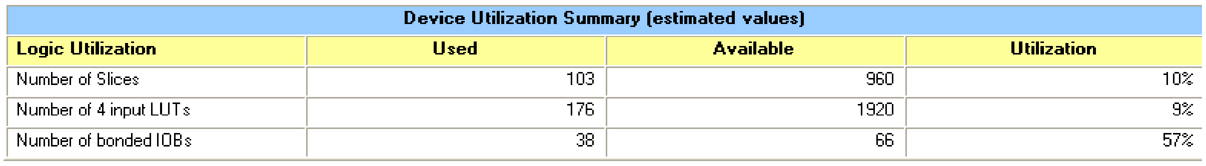
dnd[8:0]	29	29
d[9:0]	11	11
remainder[9:0]	7	7
quotient[8:0]	2	2

Fig. 9 Output Waveforms

Design summary which is generated after the simulation of the program is shown by Fig 10 from this summary we can see that total hardware required for implementation of the division program in FPGA.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 7, July 2013



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	103	960	10%
Number of 4 input LUTs	176	1920	9%
Number of bonded IOBs	38	66	57%

Fig. 10 Design Summary of Non Restoring Division Algorithm.

VII.CONCLUSION

Traditionally dividers have been avoided by DSP algorithm designers due to the complexity and cost of the hardware implementation. This paper presents that the non restoring division program is designed using VHDL and implemented on FPGA, which results the total utilization of Hardware as reflected in fig 10. It has been proved that its hardware implementation is not only easy but also cost effective.

REFERNCES

[1] S. F. Obennan and M. 1. Flynn, "Design issues in division and other floating-point operations," IEEE Transactions on Computers, vol. 46, pp. 154-161, 1997.
[2] Inwook Kong, "Modified Improved Algorithms and Hardware Designs for Division by Convergence," Doctoral dissertation. University of Texas at Austin, 2009.
[3] Peter Soderquist and Miriam Leeser, "Division and square root: choosing the right implementation," IEEE Micro, vol. 17, no. 4, pp. 56-66, 1997.
[4] Milos D. Ercegovac and Tomas Lang, Division and Square Root: Digit Recurrence Algorithms and Implementations, Boston: KJuwer Academic Publishers, 1994.
[5] S. F. Obennan and M. J. Flynn, "Division algorithms and implementations," IEEE Transactions on Computers, vol. 46, pp. 833854, 1997.
[6] Nicolas Boullis and Arnaud Tisserand, “On digit-recurrence division algorithms for self-timed circuits”, in Research Report published at Institut National De Recherche En Informatique Et En Automatique, France, 2012.
[7] Kihwan Jun and Earl E. Swartzlander, “Modified Non-restoring Division Algorithm with Improved Delay Profile and Error Correction” Signals, Systems and Computers (ASILOMAR), pp. 1460-1464, 2012.
[8] Jen-Shiun Chiang, Eugene Lai and Jun-Yao Liao ,”A Radix-2 Non-Restoring 32-b/32-b Ring Divider with Asynchronous Control Scheme” Tamkang Journal of Science and Engineering, vol. 2, No. 1 pp. 37-43 ,1999
[9] S. Oberman and M. Flynn, “Division algorithms and implementations,” IEEE Transactions on Computers 46, pp. 833–854,1997.
[10] Jagannath Samanta, Mousam Halder, Bishnu Prasad De” Performance Analysis of High Speed Low Power Carry Look-Ahead Adder Using Different Logic Styles” International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6, pp. 330–336 2013