Design, Develop and Implement an Efficient Polynomial Divider

Purbayan Deb[1], Anindya Sen[2]

[1,2] *Department of Electronics and Communication Engineering (VLSI), Heritage Institute of Technology, Kolkata, West Bengal*

***Abstract*- Polynomial Division is a most common numerical operation experienced in many filters and similar circuits next to multiplication, addition and subtraction. Due to frequent use of such components in mobile and other communication applications, a fast polynomial division would improve overall speed for many such applications. This project is to design, develop and implement an efficient polynomial divider algorithm, along with the circuit. Next its output performance result is verified using Verilog simulation. A literature survey on the normal division algorithms currently used by ALU's to perform division for large numbers, yielded Booth's algorithm, Restoring and Non-restoring algorithm. Verilog simulation of these algorithms were used to derive efficiency in terms of the timing characteristics, required chip area and power dissipation. Initially, performance analysis of the existing algorithms was done based on the simulated outputs. Later similar analysis with the updated polynomial divider circuit is performed.**

***Keywords*- Division, Polynomial, Booth's algorithm, Restoring algorithm, Non-restoring algorithm, Verilog.**

I. INTRODUCTIONS

Throughout the years, mathematicians and engineers have developed many algorithms to divide numbers. The ALU which is primarily used for division has gone through many changes in its design. One of these changes was in its division algorithms. In a typical computer, an ALU is called upon to do hundreds of division operations per second. Divider circuits are used for various purposes like error correcting codes. So to perform at its peak, the ALU's algorithms need to be as efficient as possible. However, some of these algorithms work better when computing the result of the operation by hand than using a computer and so these algorithms are not efficient in every case [1].

The traditional pen and paper algorithm, when converted to computer algorithm, resulted in the Booth's algorithm, Restoring Division algorithm [5]. Smaller improvements have been made to the restoring division algorithm, which resulted in Non-Restoring Division algorithm and many high radix algorithms, later combinational array divider circuits are also implemented for the division purpose [6]. In many cases, division is performed by taking the inverse of the divider and then multiplying the two numbers.

Division methods are divided in five classes that include iteration, digit recurrence, very high radix, table look up and variable latency. Each of these classes of division is implemented differently in hardware (using multiplication, subtraction, table look up, etc.).

Some algorithms use multiple classes rather than just one in particular. This report focuses on subtraction-based methods, such as restoring and non-restoring division algorithms, to obtain the final answer in a division computation. Digit recurrence algorithm is another division algorithm and they produce one digit of the final quotient per iteration.SRT (Sweeney Robertson and Tocher) division algorithm is very commonly use for digit recurrence purpose [12].

Use of Verilog code, which is a Hardware Descriptive language (HDL) will be done for the simulation purpose of the project.

II. MOTIVATION

Despite the recent improvement in division algorithms, division remains a complex operation and is therefore not implemented in many low cost or low power ALUs. Division can add more complexity to the computations since it can have invalid inputs such as division by zero and can have multiple machine cycles used in it. The time taken for its operation is also high and the circuits get more complex if more precision is needed and so it is avoided by most ALU's [1].

A division operation is an indispensible tool for a high performance system. A common perception of division is that it is an infrequent operation whose implementation need not receive high priority. However, it has been shown that ignoring its implementation can result in significant system performance degradation for many applications. Refining the polynomial division algorithm helps in improving the precision of the result and reduces the delay to obtain output. So implementation of a proper divider circuit is very important in an ALU.

III. OBJECTIVE

The main objective of the project work will be to focus on using the most optimal division algorithm and to design an efficient polynomial divider circuit & simulate its performance.

To find the most optimal algorithm timing characteristics and area report generated by the Xilinx tool for various division algorithms will be compared and analyzed.

IV. Tool used

For this project Verilog, HDL is used under Xilinx ISE Design Suite 13.4 platform.

V. Literature survey

A. Booth's Algorithm

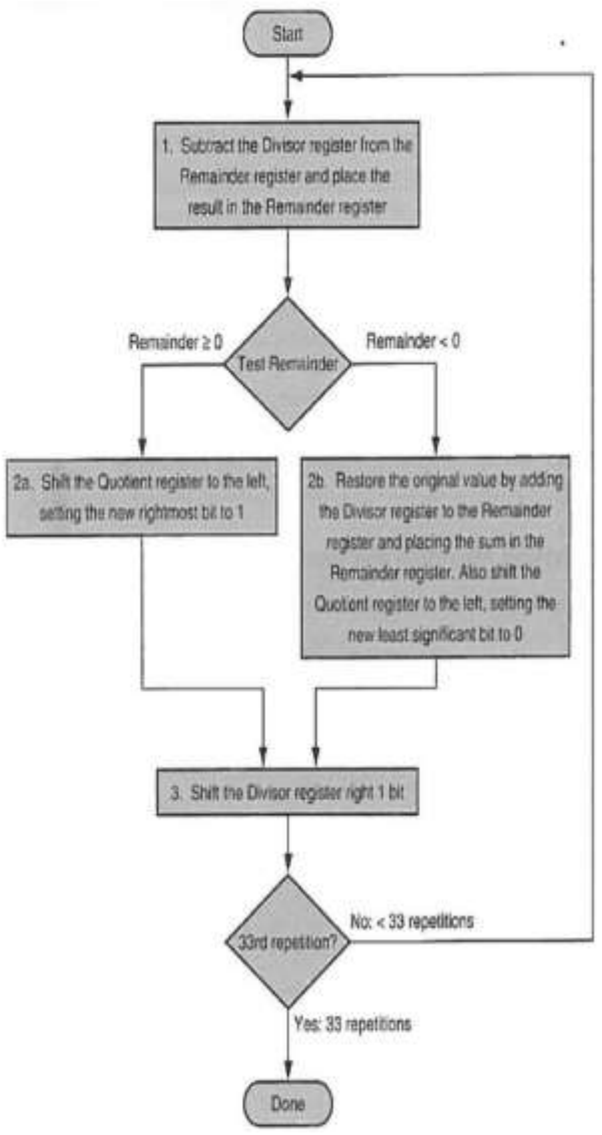

Start
1. Subtract the Divisor register from the Remainder register and place the result in the Remainder register
Remainder ≥ 0
Test Remainder
Remainder < 0
2a. Shift the Quotient register to the left, setting the new rightmost bit to 1
2b. Restore the original value by adding the Divisor register to the Remainder register and placing the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0
3. Shift the Divisor register right 1 bit
33rd repetition?
No: < 33 repetitions
Yes: 33 repetitions
Done

Fig.1 Flowchart of booth's algorithm for division

Division is done by doing shifts and subtractions. Dividing a number of 2n bits by a number of n bits results in a quotient of up to 2n bits and a remainder of up to n bits.

At start, the n bits divisor is shifted to the left, while n 0's are added to its right. This way the dividend and the divisor are 2n bits long.

At each step (repeating the following n+1 time), subtract the divisor from the dividend. If the result is non-negative, Shift the quotient left and place 1 in the new place.

Else, Shift the quotient left and place 0.

Restore the dividend by adding the divisor to it. Shift the divisor to the right [2].

At start, the dividend occupies the right half of the remainder register. The left half of the reminder register is full with zeros. Shift reminder left 1 position.

At each step, the control subtracts the divisor from the left half of the remainder register, putting there the result. If the remainder is negative, it restores it. Then, instead of shifting the divisor to the right, it shifts the remainder to the left and inserts 0 or 1, according to the sign of the remainder. 0 if the sign bit is 1 and 1 if the sign bit is 0.

At the end, the remainder register contains the quotient in its right half and the remainder in its left half [3].

The following example (7/2) will illustrate the division process of booth's algorithm clearly.

TABLE I

Shows Division of 0111 / 0010 by Booth's Algorithm

Itera-tion	**Step**	**Quotient**	**Divisor**	**Remainder**
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem=Rem-Div	0000	0010 0000	1110 0111
	2b: Rem<0=>+Div, sll Q,Q_0=0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem=Rem-Div	0000	0001 0000	1111 0111
	2b: Rem<0=>+Div, sll Q, Q_0=0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem=Rem-Div	0000	0000 1000	1111 1111
	2b: Rem<0=>+Div, sll Q, Q_0=0	0000	0000 1000	0000 0111

	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem=Rem-Div	0000	0000 0100	0000 0011
	2a: Rem≥0=> sll Q, Q_0=1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem=Rem-Div	0001	0000 0010	0000 0001
	2a: Rem≥0=> sll Q, Q_0=1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

B. Restoring Division Algorithm

Digital Recurrence algorithms use subtractive methods to calculate quotients one digit per iteration. Restoring division algorithm is based on the digital recurrence algorithm [1].

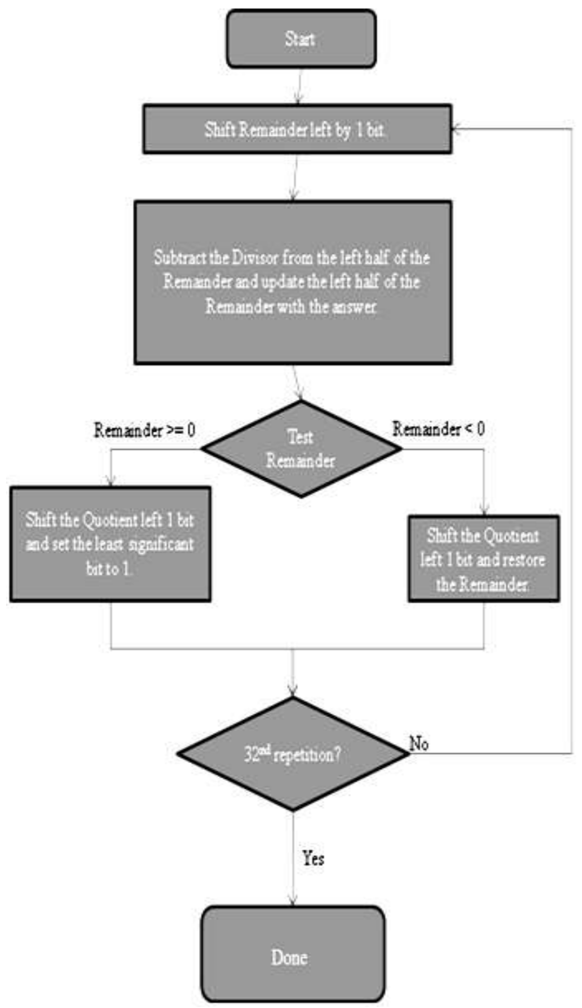

Start
Shift Remainder left by 1 bit
Subtract the Divisor from the left half of the Remainder and update the left half of the Remainder with the answer
Remainder >= 0
Test Remainder
Remainder < 0
Shift the Quotient left 1 bit and set the least significant bit to 1.
Shift the Quotient left 1 bit and restore the Remainder.
32nd repetition?
No
Yes
Done

Fig.2 Flowchart for restoring division algorithm

Restoring division follows the same method as the pen and paper long division algorithm. In the long division algorithm, the divisor is compared to the left digits of the dividend. If the divisor is bigger than the dividend numbers being compared, then a 0 in appended to the quotient and divisor is shifted to the right to compare with bigger dividend digits. If the divisor is smaller than the dividend, then the divisor being compared is subtracted from the dividend and the result is stored as remainder, while the number of times the divisor can go into the dividend is appended to the quotient [4].

During the next loop, the dividend and the remainder need to be appended together to form the new dividend. This process is repeated until the dividend cannot be divided further by the divisor. The same process is applied in the Restoring division algorithm.

To decide whether the divisor is bigger than the dividend bits it is being compared to, it subtracts the divisor from the dividend bits and stores the result in remainder field. If the divisor is bigger than the dividend bits, then the result will be negative.

If the result is negative, then the remainder is wrong and it must be “restored” to the previous value and a 0 must be appended to the quotient before the divisor is shifted to the right (or dividend shifted to the left) and subtraction is tried again.

If the result is positive, then divisor is bigger than the dividend bits being compared to and the result is valid. Therefore, a 1 is appended to the quotient.

Below the algorithm for restoring division method is explained with an example by dividing 0011 1101 (61), by 01010 (10) and how it works.

Following the Restoring division algorithm discussed above, the first step is to shift the quotient 1 bit to the left.

Second, subtract the divisor from the left half of the dividend and update the left half of the quotient with the answer. This new dividend value now has the remainder and rest of the quotient appended together.

To avoid destroying the initial dividend value, the dividend can be copied into the remainder register and use remainder register as the dividend value.

If the new dividend is less than zero, then shift the quotient left 1 bit and restore the previous value of the dividend. Otherwise, shift the quotient left 1 bit and set the least significant bit to 1. Repeat for procedure 4 times to evaluate all bits of the dividend [5].

At the end of the procedure, the quotient value will be the remainder.

Dividend z: 0011 1101 (61),
Divisor d: 01010 (10)
Quotient : 0110 (6),
Remainder : 0001 (1)

TABLE II

Division of 00111101 / 01010 by Restoring Method

Iteration	P	Q
1	Shift z left once: 0111101 Subtract d from left half of z: 11111101 Result is negative. Restore to previous value: 0111101	0
2	Shift z left once: 111101 Subtracting d from left half of z: 010101 Result is positive. New z is 010101	01
3	Shift z left once: 1010 1 Subtracting d from left half of z: 00001 Result is positive. New z is 00001	011
4	Shift z left once: 0001 Subtracting d from left half of z: 10111 Result is negative: Restore to previous value: 0001	0110

C. Non-Restoring Division Algorithm

The non-restoring divide does not "restore" the remainder to the correct value but leaves it incorrect until the next cycle [1]. In the restoring divide algorithm, if we had restored the partial remainder to its correct value, we would proceed with the next shift and trial subtraction getting the result. Instead, because we used the incorrect partial remainder, a shift and trial subtraction would yields result which is not the intended. However, an addition would do the trick.

The non-restoring algorithm can result in a negative remainder, which is incorrect. Therefore, a correction step is needed to obtain the correct remainder. The algorithm to perform non-restoring division is as follows:

Non-restoring divide algorithm [6]:

i. Shift remainder left 1 bit.
ii. If remainder is negative, add divisor to the left half of
iii. the remainder. Shift quotient left 1 bit.
iv. If remainder is positive, subtract divisor from the left
v. half of the remainder. Shift quotient left 1 bit and add 1.
vi. Repeat for number of bits in divisor.
vii. Correction step: If remainder is negative, add divisor to the remainder to obtain the correct value.

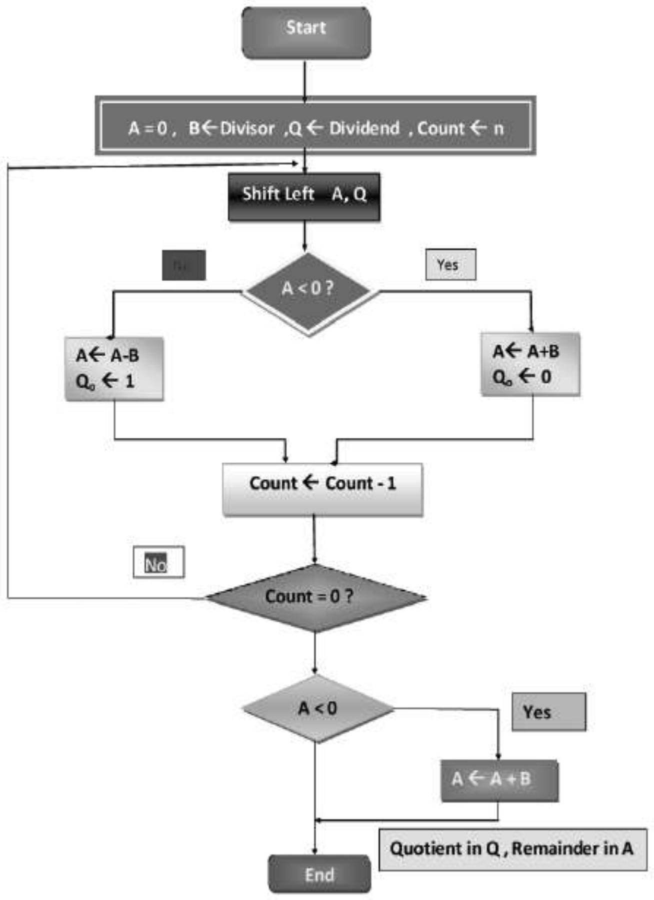


Fig.3 Flowchart for non-restoring division algorithm

Here, the dividend is not restored after each unsuccessful subtraction operation as was in restoring algorithm.

Instead, the following logic is followed:

If the current remainder is positive,

Then $Q_0 = 1$.

Next operation will be shift and subtract

Else (remainder negative),

Then $Q_0 = 0$.

Next operation will be shift and add.

Example: 10 / 4

Quotient = 2, Remainder = 2, A = 0

Q = Dividend = 10 = 1010

B = Divisor = 4 =0100 ,

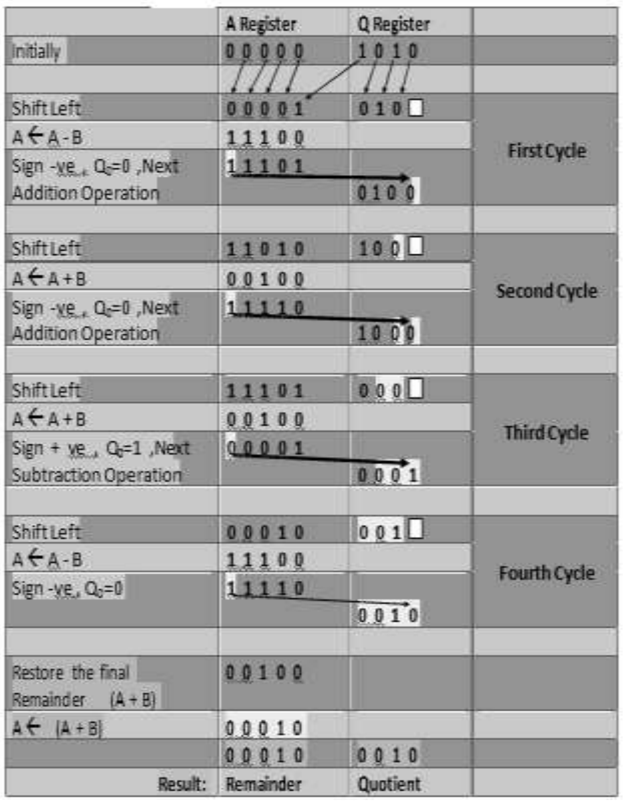
- B means 2'sComplement of B = 4

B =00100

11 011	1's Complement of B
+ 1	addition of 1 to 1's Complement of B
11100	2's Complement of B

If Q (Divisor) register contains 4 bits then Contents of A register is 4+1=5 bits

TABLE III
Division of 1010 / 0100 by Non-Restoring Method



	A Register	Q Register	
Initially	0 0 0 0 0	1 0 1 0	
Shift Left	0 0 0 0 1	0 1 0 □	First Cycle
A ← A - B	1 1 1 0 0		
Sign -ve , $Q_0=0$,Next Addition Operation	1 1 1 0 1	0 1 0 0	
Shift Left	1 1 0 1 0	1 0 0 □	Second Cycle
A ← A + B	0 0 1 0 0		
Sign -ve , $Q_0=0$,Next Addition Operation	1 1 1 1 0	1 0 0 0	
Shift Left	1 1 1 0 1	0 0 0 □	Third Cycle
A ← A + B	0 0 1 0 0		
Sign + ve , $Q_0=1$,Next Subtraction Operation	0 0 0 0 1	0 0 0 1	
Shift Left	0 0 0 1 0	0 0 1 □	Fourth Cycle
A ← A - B	1 1 1 0 0		
Sign -ve , $Q_0=0$	1 1 1 1 0	0 0 1 0	
Restore the final Remainder (A + B)	0 0 1 0 0		
A ← (A + B)	0 0 0 1 0		
	0 0 0 1 0	0 0 1 0	
Result:	Remainder	Quotient	

D. *LFSR (Linear Feedback Shift Registers)*

Linear Feedback Shift Registers (LFSR) can be used for polynomial division, among its other uses.

Two example LFSRs are shown in Fig4 & Fig5. Note that both these structures use D-type flip-flops and linear logic elements (EOR gates) to realize LFSRs [8]. The basic difference in these two structures being the circuit of Fig4 uses linear elements interspersed between the flip-flops whereas the circuit of Fig5 has no linear element appearing between the flip-flops instead the linear elements appear only in the feedback path. It is for this reason the realization of Fig4 is called internal-EOR LFSR and the realization of Fig5 is called external EOR LFSR. A equivalence exists between the two structures in the sense that knowing the properties of the first structure one can deduce the properties of the second structure.

The circuit in Fig4 is better for its high-speed operation since they don't have multiple adder combination propagation delays [9].

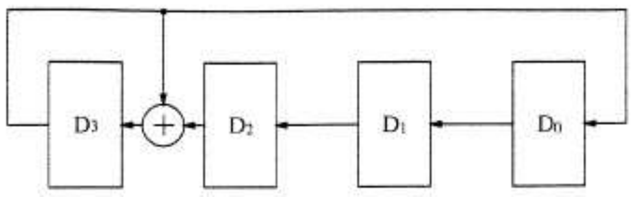


Fig.4 Internal EOR type LFSR

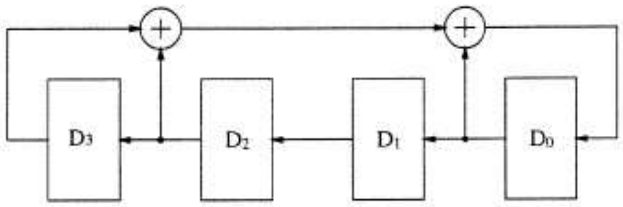


Fig.5 External EOR type LFSR

$\oplus \rightarrow$ Represents EOR gate

Polynomial arithmetic can be performed using this LFSR circuit. The LFSR here does the shifting operation at each clock cycle and keeps on generating new patterns based on the input values. One thing to be noted here during polynomial division is that all these polynomials are not usually written with minus signs, but they could be, therefore a coefficient of –1 is equivalent to coefficient of 1 and polynomials with co-efficient other than 1 cannot be used as in digital domain we have only 0 & 1[10].

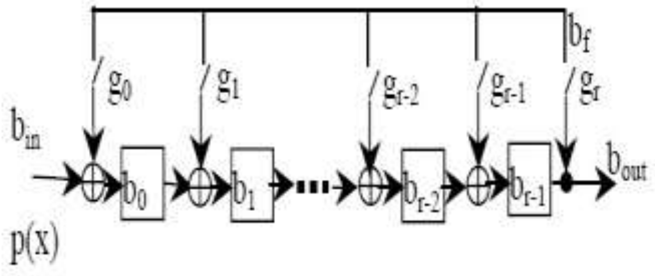


Fig.6 A basic polynomial division register

VI. METHODOLOGY

A. *Preliminary Work*

In this project firstly comparison of the restoring and non restoring algorithms for division are done and the results are obtained.

The comparison is on the basis of area analysis, timing, delay and power consumed. The algorithms are implemented using Verilog code and done the comparison.

I have done the coding in Xilinx ISE Design Suite 13.4 tool.

B. Proposed Implementation in Polynomial Divider Circuit

Steps to be followed for implementation of polynomial division-

i. Firstly the circuit is built on basis of the denominator given.
ii. The input is given serially in the circuit.
iii. The output of each stage is calculated at each clock cycle.
iv. The process continues until all the bits of input are given.
v. The quotient is calculated from the outputs of the final register.
vi. Finally, the remainder is calculated from the output of all the registers in the final clock cycle.

Below two examples are given to explain the process of polynomial division-

i) $(X^3+X^2+1)/(X^2+1)$

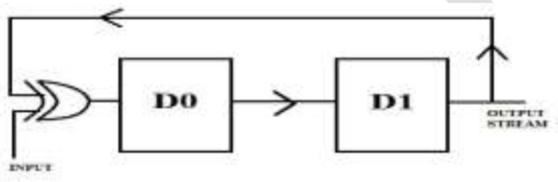


Fig.7 Circuit for $(X^3+X^2+1)/(X^2+1)$

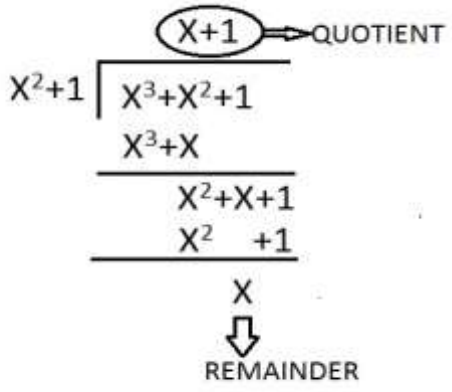


Fig.8 Division using long division method

TABLE IV
LFSR Stages Corresponding to the Input Applied

INPUT	D0	D1	OUTPUT STREAM
1	0	0	0
1	1	0	00
0	1	1	001
1	1	1	0011
	0	1	

Here the final output stream is 0011.

So the quotient is $0*X^3 + 0*X^2 + 1*X^1 + 1*X^0 = X+1$

The value of D0 and D1 in final cycle is 0 & 1 respectively.

So, remainder is $0*X^0 + 1*X^1 = X$

Here any numerator can be applied and the output result will be obtained using the same denominator.

ii) $(X^7+X^6+X^5+X^4+X^2+1)/(X^5+X^4+X^2+1)$

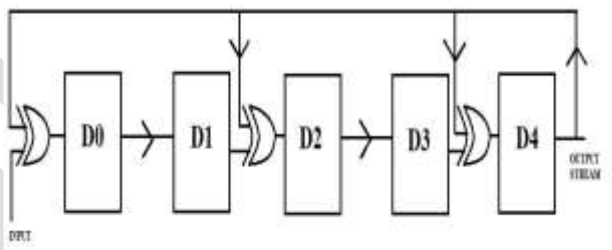


Fig.9 Circuit for $(X^7+X^6+X^5+X^4+X^2+1)/(X^5+X^4+X^2+1)$

TABLE V
LFSR Stages Corresponding to the Input Applied

INPUT	D0	D1	D2	D3	D4	OUTPUT STREAM
1	0	0	0	0	0	0
1	1	0	0	0	0	00
1	1	1	0	0	0	000
1	1	1	1	0	0	0000
0	1	1	1	1	0	00000
1	0	1	1	1	1	000001
0	0	0	0	1	0	000010
1	0	0	0	0	1	0000101
	0	0	1	0	1	

Here the final output stream is 0000101.

So the quotient is

$0*X^6 + 0*X^5 + 0*X^4 + 0*X^3 + 1*X^2 + 0*X^1 + 1*X^0 = X^2+1$

The value of D0,D1,D2,D3,D4 in final cycle is 0,0,1,0,1 respectively.

So, remainder is

$0*X^0 + 0*X^1 + 1*X^2 + 0*X^3 + 1*X^4 = X^2+X^4$.

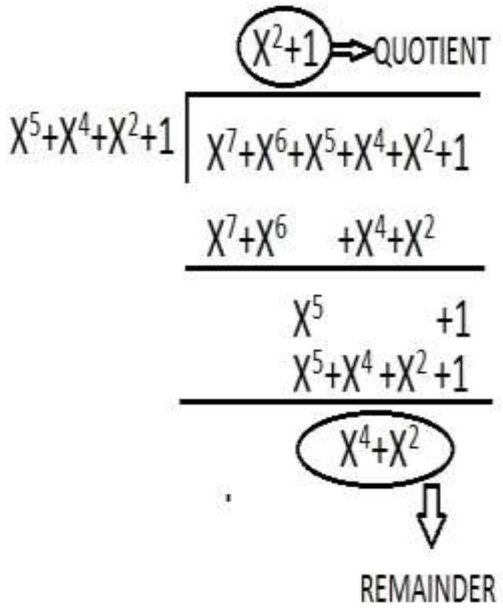


Fig.10 Division using long division method

These are simulated using the Xilinx tool and the results are compared.

The max power of the denominator can be 9312 as this is the number of Flip Flops supported by the Xilinx tool.

This division operation can also be performed using multiplication of LFSR. But that will be a much more time consuming process.

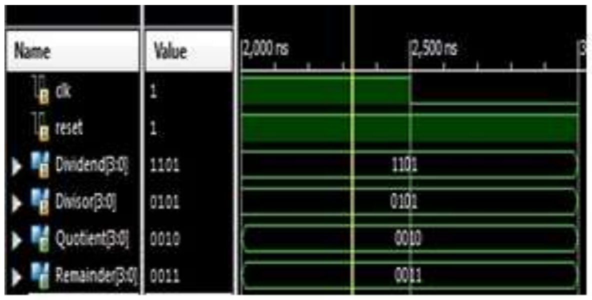
Example- $X^2/X = X$

This will be the process if it is done normally using polynomial division method.

But if multiplication is used it will be, $X^2 * X^{-1} = X$

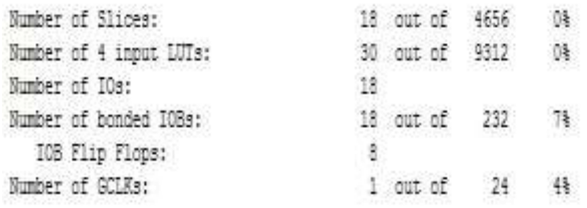
So the number of operations needed to be performed in this method will be more and hence more time will be consumed [11].

VII. RESULT



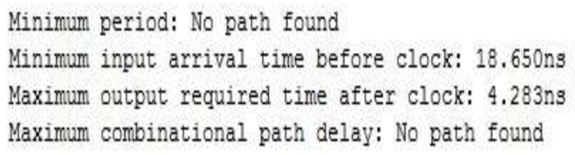
Name	Value
clk	1
reset	1
Dividend[3:0]	1101
Divisor[3:0]	0101
Quotient[3:0]	0010
Remainder[3:0]	0011

Fig.11 Division of 13 by 5 using restoring division



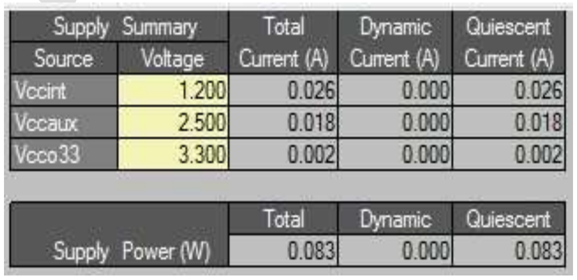
```
Number of Slices:               18  out of   4656    0%
Number of 4 input LUTs:         30  out of   9312    0%
Number of IOs:                  18
Number of bonded IOBs:          18  out of    232    7%
   IOB Flip Flops:               8
Number of GCLKs:                 1  out of     24    4%
```

Fig.12 Area analysis for restoring division



```
Minimum period: No path found
Minimum input arrival time before clock: 18.650ns
Maximum output required time after clock: 4.283ns
Maximum combinational path delay: No path found
```

Fig.13 Timing analysis for restoring division

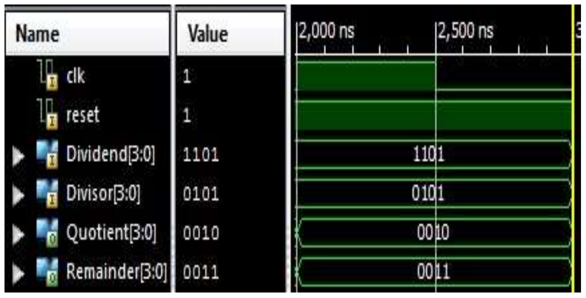


Supply Source	Summary Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.026	0.000	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco33	3.300	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.083	0.000	0.083

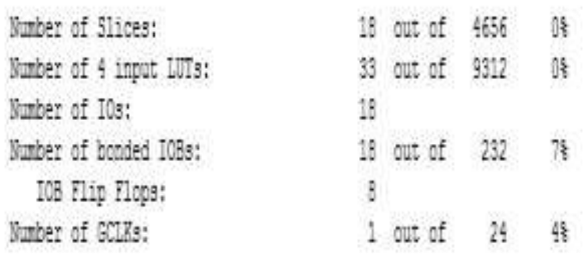
Fig.14 Power analysis for restoring division

Fig 11, Fig 12, Fig 13, Fig 14 represents the waveform of output of the division, area analysis, timing analysis and power analysis respectively.



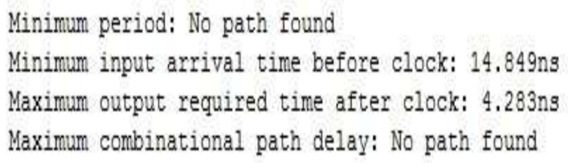
Name	Value
clk	1
reset	1
Dividend[3:0]	1101
Divisor[3:0]	0101
Quotient[3:0]	0010
Remainder[3:0]	0011

Fig.15 Division of 13 by 5 using non-restoring division



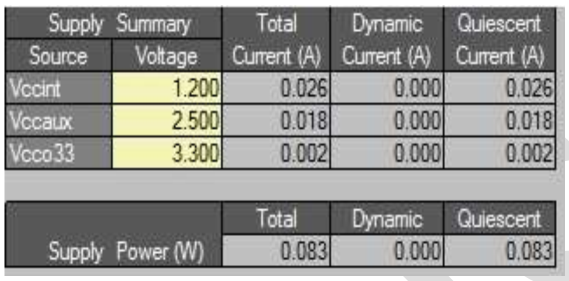
```
Number of Slices:                18  out of   4656    0%
Number of 4 input LUTs:          33  out of   9312    0%
Number of IOs:                   18
Number of bonded IOBs:           18  out of    232    7%
   IOB Flip Flops:                8
Number of GCLKs:                  1  out of     24    4%
```

Fig.16 Area analysis for non-restoring division



```
Minimum period: No path found
Minimum input arrival time before clock: 14.849ns
Maximum output required time after clock: 4.283ns
Maximum combinational path delay: No path found
```

Fig.17 Timing analysis for non-restoring division



Supply Source	Summary Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.026	0.000	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco33	3.300	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.083	0.000	0.083

Fig.18 Power analysis for non-restoring division

Fig 15, Fig 16, Fig 17, Fig 18 represents the waveform of output of the division, area analysis, timing analysis and power analysis respectively.

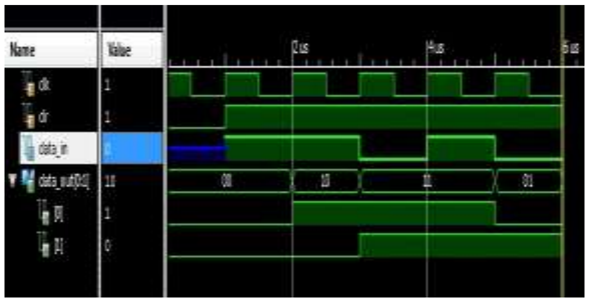
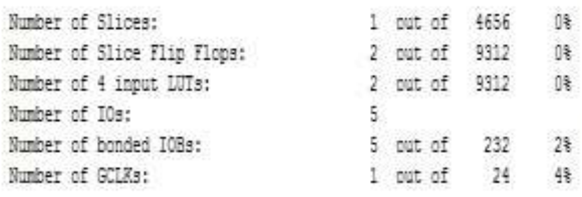
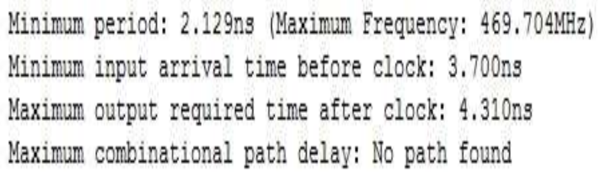


Fig.19 $(X^3+X^2+1)/(X^2+1)$ using LFSR



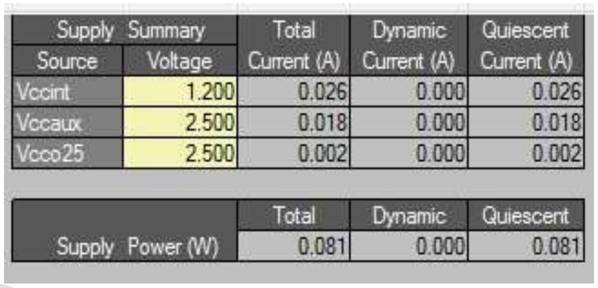
```
Number of Slices:                 1  out of   4656    0%
Number of Slice Flip Flops:       2  out of   9312    0%
Number of 4 input LUTs:           2  out of   9312    0%
Number of IOs:                    5
Number of bonded IOBs:            5  out of    232    2%
Number of GCLKs:                  1  out of     24    4%
```

Fig.20 Area analysis for polynomial division of $(X^3+X^2+1)/(X^2+1)$



```
Minimum period: 2.129ns (Maximum Frequency: 469.704MHz)
Minimum input arrival time before clock: 3.700ns
Maximum output required time after clock: 4.310ns
Maximum combinational path delay: No path found
```

Fig.21 Timing analysis for polynomial division of $(X^3+X^2+1)/(X^2+1)$



Supply Source	Summary Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.026	0.000	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.081	0.000	0.081

Fig.22 Power analysis for polynomial division of $(X^3+X^2+1)/(X^2+1)$

Fig 19, Fig 20, Fig 21, Fig 22 represents the waveform of output of the division, area analysis, timing analysis and power analysis respectively.

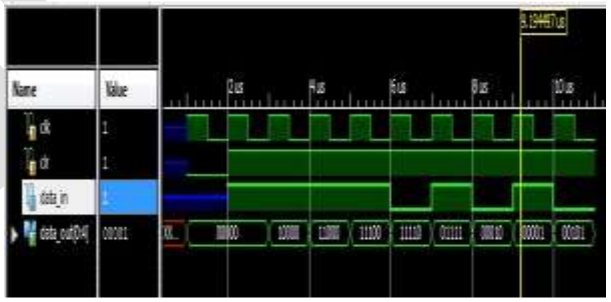
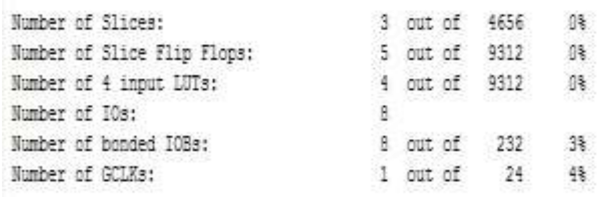
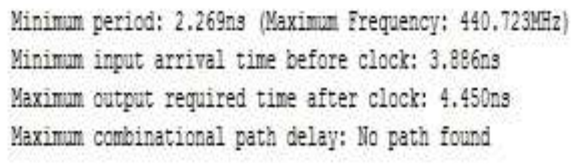


Fig23 $(X^7+X^6+X^5+X^4+X^2+1)/(X^5+X^4+X^2+1)$ using LFSR



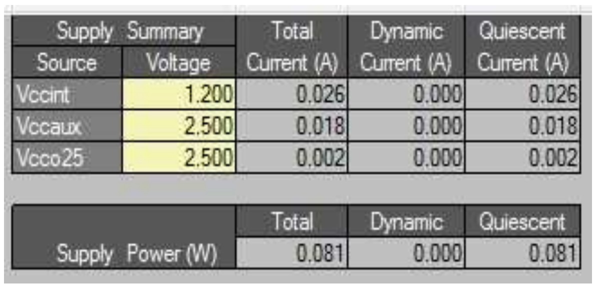
```
Number of Slices:                 3  out of   4656    0%
Number of Slice Flip Flops:       5  out of   9312    0%
Number of 4 input LUTs:           4  out of   9312    0%
Number of IOs:                    8
Number of bonded IOBs:            8  out of    232    3%
Number of GCLKs:                  1  out of     24    4%
```

Fig.24 Area analysis for polynomial division of $(X^7+X^6+X^5+X^4+X^2+1)/(X^5+X^4+X^2+1)$



```
Minimum period: 2.269ns (Maximum Frequency: 440.723MHz)
Minimum input arrival time before clock: 3.886ns
Maximum output required time after clock: 4.450ns
Maximum combinational path delay: No path found
```

Fig.25 Timing analysis for polynomial division of $(X^7+X^6+X^5+X^4+X^2+1)/(X^5+X^4+X^2+1)$



Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.026	0.000	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.081	0.000	0.081

Fig.26 Power analysis for polynomial division of $(X^7+X^6+X^5+X^4+X^2+1)/(X^5+X^4+X^2+1)$

Fig 23, Fig 24, Fig 25, Fig 26 represents the waveform of output of the division, area analysis, timing analysis and power analysis respectively.

TABLE VI
Comparison of Timing, Area and Power of Restoring, Non- Restoring and Polynomial Division

	Restoring Division	Non-Restoring Division	Polynomial Division	
	1101 / 101	1101 / 101	$(X^3+X^2+1)/(X^2+1)$	$(X^7+X^6+X^5+X^4+X^2+1)/(X^5+X^4+X^2+1)$
Timing Analysis	Maximum input arrival time after clock-18.650ns Maximum output required time after clock-4.283ns	Maximum input arrival time after clock-14.849ns Maximum output required time after clock-4.283ns	Maximum input arrival time after clock-3.700ns Maximum output required time after clock-4.310ns	Maximum input arrival time after clock-3.886ns Maximum output required time after clock-4.450ns
Area Analysis	No. of Slice-18	No. of Slice-18	No. of Slice-1	No. of Slice-3
			No. of Slice Flip-Flop-2	No. of Slice Flip-Flop-5
	No. of 4 input LUT- 30	No. of 4 input LUT- 33	No. of 4 input LUT-2	No. of 4 input LUT-4
Power Analysis	Total power- 0.083W	Total power- 0.083W	Total power- 0.081W	Total power- 0.081W

VIII. Conclusion

In this paper analysis of various division algorithms are done. Their timing, area & power comparison are obtained.

From the analysis I have seen that though the number of LUT utilization in non-restoring algorithm is more but the timing performance of the non-restoring algorithm is much better. So from these to algorithm analysis it can be concluded that the non restoring algorithm is by far better than restoring algorithm.

Later in the next part a polynomial divider is implemented using LFSR and the simulation is done using Verilog. The results obtained are then compared and we see that the delay is much less in this method. We also see that more the maximum power of denominator more is the delay. The power consumed by the polynomial divider circuit is also bit less. Overall it can be said that by implementing the polynomial division the delay and area can be minimized a lot.

As the complexity in designing a polynomial divider circuit is high, need for more efficient design is very high in market. Thus seeing the growing needs my project can be highly beneficial.

IX. Future work to be done

Up to this point research on divider circuits like Booth's algorithm, Restoring algorithm & Non-restoring algorithm are done and a polynomial divider using LFSR's serially is implemented.

So, the next work will be to implement the LFSRs in parallel to reduce the time required for execution of the division method.

Acknowledgement

The road of knowledge is long and full of obstacles, but going over those obstacles is what ultimately enlightens the individual and strengthens ones spirit. I have been very fortunate to walk through the road of knowledge, not alone, but accompanied by the brightest and warm hearted individual I have ever met.

I take this opportunity to thank my project guide Sri Anindya Sen (Phd) for guiding me in this paper.

He has been a great source of motivation for me, which inspired me a lot to take up the project and deliver it in a nicest possible way.

Finally, I would also like to thank all the faculty members and staff of Electronics and Communication Engineering Department for their time to time support and co-operation, which has helped me a lot.

PURBAYAN DEB

References

[1]. Performance analysis of various multiplication and division algorithms for large numbers Harpreet Kaur, 2010

[2]. Patterson, David and Hennessy, John. Computer Organization and Design - The Hardware / Software Interface. San Francisco: Morgan Kaufmann Publishers, 1998.

[3]. Lecture 5 *Multiplication* and Division. ECE 0142 Computer Organization

[4]. Division Algorithms and Hardware Implementations, Sherif Galal Dung Pham EE 213A: Advanced DSP Circuit Design

[5]. Computer Principles And Design In Verilog HDL Yamin Li Hosei University, Japan

[6]. https://asnaikblog.wordpress.com/video-tutorial-for-non-restoring-method-of-division-operation/

[7]. Linear Feedback Shift Registers Theory and. Applications. *Kewal K. Saluja*. Department of Electrical and Computer Engineering. University of Wisconsin-madison

[8]. Implementation and evaluation of a polynomial-based division algorithm Examensarbete utfört vid Elektroniksystem, Linköpings Tekniska Högskola Av Stefan Pettersson

[9]. Polynomial Multipliers and Dividers, Shift Register Generators and Scramblers Phil Lucht Rimrock Digital Technology, Salt Lake City, Utah 84103

[10]. Mr. Hiren G. Patel, Dr. D.M.Patel, Mr. Milan A. Chaudhari, Mr. Mahavirsinh A. Zala "An Automated CRC Engine" International Journal of Engineering Research And Management (IJERM) ISSN : 2349- 2058, Volume-1, Issue-3, June 2014, PP. 11-15

[11]. https://www.freemathhelp.com/forum/archive/index.php/t-56823.html

[12]. Oberman, Stuart F. and Flynn, Michael J. "Division Algorithms and Implementations." IEEE Transcation on Computers (1997): 833-854.