International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869, Volume-3, Issue-4, April 2015

FPGA Implementation of Fixed point Integer Divider Using Iterative Array Structure

Narendra K., ShabirAhmed B.J., Swaroop Kumar K, Asha G.H.

***Abstract*— an array structure for high speed division algorithm has been described in this paper. The objective is to develop the division algorithm first with the basic technique and later enhance the performance by pipelining the execution process. For implementation, we consider restoring dividers (i.e., those that keep the actual residue value at every step). Three different types of division algorithms are developed which serve for different applications. The first algorithm is 'Combinatorial Array Divider' which uses an array of processing units consisting of a full adder and a multiplexer. It is the direct implementation of hand-division method and it gives the basic understanding of the division process. The second algorithm is 'Fully Pipelined Array Divider' which uses an array of processing units along with large number of flip-flops for storing the intermediate results. Pipelining is one way of improving the overall processing performance. This reduces the execution time which is very helpful in certain real-time applications but on the contrary it increases the hardware, resulting in an increase in the cost and area. The third algorithm is 'Iterative Restoring Divider' which uses just a couple of shift registers and a control unit. This reduces the hardware (area and cost) but in turn it takes higher number of clock cycles to execute. This is preferred in some non-real-time applications where execution time is of least essence. A synthesizable model of a divider that can be implemented in FPGA is developed and the implementation has been parameterized (i.e. it can be implemented for any size of the operand).**

***Index Terms*— Combinatorial Array Divider, Fully Pipelined Array Divider, Integer Divider, Iterative Restoring Divider, parameterized, Restoring.**

I. INTRODUCTION

Division is a complex operation whose VLSI implementation is generally slower and more area consuming than the other three basic arithmetic operations (i.e. addition, subtraction and multiplication). However, with more complex digital signal processing (DSP) algorithms being implemented in VLSI, the divider is increasingly becoming an indispensable VLSI block for digital design [6].

Manuscript received April 12, 2015.

Narendra K, Student (M.Tech) Digital Electronics and Communication systems, Malnad College of Engineering, Hassan, Karnataka, India, +91-9738543811.

ShabirAhmed B J, Student (M.Tech) Digital Electronics and Communication systems, Malnad College of Engineering, Hassan, Karnataka, India, +91-9738417860.

Swaroop Kumar K, Student (M.Tech) Digital Electronics and Communication systems, Malnad College of Engineering, Hassan, Karnataka, India, +91-7411379265.

Asha G H, Associate Professor, Dept. of Electronics and communication, Malnad College of Engineering, Hassan, Karnataka, India, +91-9448033837.

Furthermore, the number of clock cycles for integer division varies depending on the operands' values. Every general-purpose microprocessor of recent design provides a hardware support for arithmetic division. Also, in digital signal processors for some applications such as three-dimensional graphics, there are increasing demands for high-speed dividers [7]. However, frequently used division algorithms are based on sequential recurrences producing one quotient digit per iteration, which causes significant increase in computation steps and sometimes imposes severe limitations on system performance.

Integer division is a critical operation in CPU design, since the number of clock cycles to complete an integer is usually very long and unpredictable. The role of division is becoming more and more critical, owing to the requirement of signed computer arithmetic, modulus computation, the calculation of encryption keys, and so on. Pipelining is one way of improving the overall processing performance of a processor. This architectural approach allows the simultaneous execution of several instructions. Pipelining is transparent to the programmer; it exploits parallelism at the instruction level by overlapping the execution process of instructions. It is analogous to an assembly line where workers perform a specific task and pass the partially completed product to the next worker [2].

This paper is organized as follows. Section-II gives the introduction into some standard integer division algorithms. Section-III describes the basic implementation of the division algorithm. Section-IV, V & VI describes the implementation of Combinatorial Array Divider, Fully Pipelined Array Divider and Iterative Restoring Divider respectively. The results and conclusions are presented in Section- VII and VIII.

II. INTEGER DIVISION

The division is a basic arithmetic operation requiring two inputs Dividend (A) and Divisor (B) to produces the two outputs i.e. Quotient (Q) and Remainder (R) such that $Q = int(A/B)$ and $R = A - Q.B$ under the condition $R < B$. The division is a series of subtractions of the divisor from the dividend producing the partial remainder values.

The standard fixed-point algorithm follows a "paper-and-pencil" technique: in every iteration, it produces a fixed number of quotient bits. This involves the addition, multiplication and shift operations. For a proper division, normally the dividend is greater than the divisor (A > B). If we consider the dividend to be n-bits $(A_{n-1}A_{n-2} \dots A_1A_0)$ and the divisor to be m-bits $(B_{m-1}B_{m-2} \dots B_1B_0)$ where $n \geq m$ then the quotient will be of n-bits $(Q_{n-1}Q_{n-2} \dots Q_1Q_0)$ and the remainder will be of m-bits $(R_{m-1}R_{m-2} \dots R_1R_0)$.

Many arrays for division operation have been proposed and they can be broadly classified into two categories: (i) restoring and (ii) non-restoring. In restoring division, the divisor is subtracted from the dividend (or from the previous

remainder); if the remainder is negative, the previous dividend is restored and the quotient bit is taken as zero. Otherwise the quotient bit is one and the process is continued without any change. In non-restoring method, the division process is carried out without restoring the previous dividend irrespective of the sign of the result. The organizations of two types of divisors are quite similar and only the designs of the basic cells are slightly different. But later on it was proved that the speed of the two types of arrays is almost equal and the restoring technique gives a true remainder. In a divider array the subtraction can be achieved either directly or by adding 2's complement of the divisor.

III. Implementation

Given two unsigned numbers A (n-bits) and B (m-bits), we wish to design a circuit that produces two outputs Q (n-bits) and R (m-bits), where Q is the quotient of A/B and R is the remainder. This can be implemented by shifting the digits in A to the left, one digit at a time, into a shift register R. After each shift operation, R is compared with B. If $R \geq B$, a 1 is placed in the appropriate bit position in the quotient and B is subtracted from R. Otherwise, a 0 bit is placed in the quotient. For the implementation, we follow the hand-division method. We grab bits of A one by one and comparing it with the divisor. If the result is greater or equal than B, then we subtract B from it. This algorithm is described using pseudo-code. The notation R||A is used to represent a 2n-bit shift register formed using R as the left-most n bits and A as the right-most n bits.

$R = 0$;
for $i = 0$ to $n - 1$ do
  Left-shift $R||A$;
  if $R \geq B$ then
    $q_i = 1$;
    $R = R - B$;
  else
    $q_i = 0$;
  end if ;
end for ;

A. Subtraction of Unsigned Numbers Represented With n-Bits: T=R-B.

This point deserves special attention as the divider hardware relies on the result obtained here. We usually determine the sign of the subtraction by sign-extending R and B so that they are in 2's complement representation with $n + 1$ bits. Then, we do $T = R + not(B) + 1$, where $T = t_n t_{n-1} t_{n-2} \dots t_0$, and t_n determine the sign of the subtraction operation. However, when R and B are unsigned, we can compute $not(B)$ without sign-extending B. We then analyse $cout_n$: (i) If $cout_n = 1 \rightarrow R \geq B$ (and $R - B$ is equal to $T = t_n t_{n-1} t_{n-2} \dots t_0$, i.e. it is an unsigned number with n bits). (ii) If $cout_n = 0 \rightarrow R < B$ (here $R - B$ is NOT equal to $T = t_n t_{n-1} t_{n-2} \dots t_0$)

B. Demonstration of the computation of R-B with n bits:

We have $0 \leq R$ and $B \leq 2^n - 1$ where R and B are unsigned binary numbers represented by $R = r_{n-1} r_{n-2} \dots r_0$ and $B = b_{n-1} b_{n-2} \dots b_0$ respectively. To compute $R - B$, we sign-extend R and B to $n + 1$ bits turning them into two numbers in 2's complement representation. The sign-extension actually amounts to zero-extending. Then, $R = 0 r_{n-1} r_{n-2} \dots r_0$ and $B = 0 b_{n-1} b_{n-2} \dots b_0$. In 2's complement, we have that: $0 \leq R$ and $B \leq 2^n - 1$. It follows that: $-(2^n - 1) \leq R - B \leq 2^n - 1$. Thus $R - B$ can be represented in 2's complement with $n + 1$ bits (as expected). Let $K = not(B) + 1$ and is represented by $K = k_n k_{n-1} k_{n-2} \dots k_0$. In unsigned representation, $K = 2^{n+1} - B$.

Equ.1 shows the operation $R - B$ by using: $+K$, where $K = not(B) + 1$ .We let 1 be held by cin. If $B = 0$ then $K = 2^{n+1}$ (here is represented by the second operator as well as $c_{in} = 1$)

$$\begin{array}{l} \qquad\qquad\qquad\qquad 1 \leftarrow c_{in} \\ R: \quad 0r_{n-1}r_{n-2}\dots r_0 - \\ B: \quad 0b_{n-1}b_{n-2}\dots b_0 \end{array} \quad \Rightarrow \quad \begin{array}{l} R: \quad 0r_{n-1}r_{n-2}\dots r_0 + \\ K: \quad 1k_{n-1}k_{n-2}\dots k_0 \end{array}$$

equ.1: Operation $R - b = R + K = R + not(B) + 1$

Table I: To determine the value of k_{n-1} :

Case	K	$k_n k_{n-1} k_{n-2} \dots k_0$	k_n
$B \neq 0$ $(or\ B > 0)$	2^n	**1**00...0	$k_n = 1$
	$2^n + 1$	**1**00...1	
	...	...	
	$2^{n+1} - 1$	**1**11...1	
$B = 0$	2^{n+1}	**10**00...0	$k_n = 0$

Case-1: $R - B < 0$

- Since $R \geq 0$ Implies that $B > 0$ and hence $k_n = 1$
- We have

$$R + 2^{n+1} - B = \sum_{i=0}^{n-1} r_i 2^i + 2^{n+1} - \sum_{i=0}^{n-1} b_i 2^i < 2^{n+1}$$

$$R + K = \sum_{i=0}^{n-1} r_i 2^i + k_n 2^n + \sum_{i=0}^{n-1} k_i 2^i < 2^{n+1}$$

- Hence,

$$\sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i < 2^n$$

- The $n + 1$ bit sum (considering the operation as unsigned) of R and K is lower than 2^{n+1}. Then, there is no overflow in the $n + 1$ bit unsigned sum. Thus $c_{n+1} = 0$.
- The n bit sum (considering the operations as unsigned) of R and $k_{n-1} k_{n-2} \dots k_0$ is lower than 2^n. Thus, there is no overflow of the n bit unsigned sum. Thus $c_n = 0$.

Case-2: $R - B \geq 0$

- We have

$$R + 2^{n+1} - B = \sum_{i=0}^{n-1} r_i 2^i + 2^{n+1} - \sum_{i=0}^{n-1} b_i 2^i \geq 2^{n+1}$$

$$R + K = \sum_{i=0}^{n-1} r_i 2^i + k_n 2^n + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1}$$

- Hence
$$\sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1} - k_n 2^n$$
- The $n+1$ bit sum (considering the operation as unsigned) of R and K is greater or equal to than 2^{n+1}. Then, there is overflow of the bit $n+1$ unsigned sum. Thus $c_{n+1} = 1$.
- For the n-bit sum of R and $k_{n-1}k_{n-2} \dots k_0$, we have two cases

If B > 0, then $k_n = 1$and hence

$$\sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1} - 2^n$$

$$\sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^n$$

If B = 0, then $k_n = 0$ and hence

$$\sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1}$$

- In both cases, the n-bit sum (considering the operands as unsigned) of R and $k_{n-1}k_{n-2} \dots k_0$ is greater or equal to than 2^n. So, there is overflow of the n-bit unsigned sum. Thus $c_n = 1$ when $R \geq B$.

For the 2's complement operation of R-B with $n+1$ bits, there is no overflow of the subtraction as $c_n = c_{n-1}$. For $R - B \geq 0$: The result $T = R - B$ is a positive number, thus $T_n = 0$. Therefore $t_{n-1}t_{n-2} \dots t_0$ contains $R - B$ in unsigned representation.

In conclusion: (i) If $R < B \rightarrow c_n = 0$, then the n bits $t_{n-1}t_{n-2} \dots t_0$ do not contain the result $R - B$. (ii) If $R \geq B \rightarrow c_n = 1$, then the n bits $t_{n-1}t_{n-2} \dots t_0$ do represent $R - B$ in unsigned representation.

C. Restoring Array Divider For Unsigned Numbers.

Let A and B be two positive integers in unsigned form of representation. $A = a_{N-1}a_{N-2} \dots a_0$ with N bits, and $B = b_{M-1}b_{M-2} \dots b_0$ with M bits, with the condition that $N \geq M$. We have $A = (B \times Q) + R$, where Q is the quotient and R is the remainder. In this parallel implementation, the result of every stage is called the residue R_i. The Fig. 1 depicts the parallel algorithm with N stages. For each stage , $i = 0,1, \dots N-1$, we have

- R_i : denotes the output of stage i which represents the residue after each stage.
- Y_i: denotes the input of stage i which holds the minuend at each stage.

For the next stage, we append the next bit of A to R_i. This becomes Y_{i+1} (the minuend) $Y_{i+1} = R_i || a_{N-1-I}$ for $i = 0,1,..,N-1$. At each stage i, the subtraction $Y_i - B$ is performed. (i) If $Y_i \geq B$then $R_i = Y_i - B$, (ii) If $Y_i <$ Bthen $R_i = Y_I$

Table II: Restoring algorithm for division

Stage	Y_i	Computation of R_i	R_i bits
0	$Y_0 = a_{N-1}$	$R_0 = Y_0 - B, if\ Y_0 \geq B$	1
		$R_0 = Y_0, if\ Y_0 < B$	
1	$Y_1 = R_0 \| a_{N-2}$	$R_1 = Y_1 - B, if\ Y_1 \geq B$ $R_1 = Y_1, if\ Y_1 < B$	2
2	$Y_2 = R_1 \| a_{N-3}$	$R_2 = Y_2 - B, if\ Y_2 \geq B$ $R_2 = Y_2, if\ Y_2 < B$	3
…	…	…	…
M-1	$Y_{M-1} = R_{M-2} \| a_{M-N}$	$R_{M-1} = Y_{M-1} - B, if\ Y_{M-1} \geq B$ $R_{M-1} = Y_{M-1}, if\ Y_{M-1} < B$	M

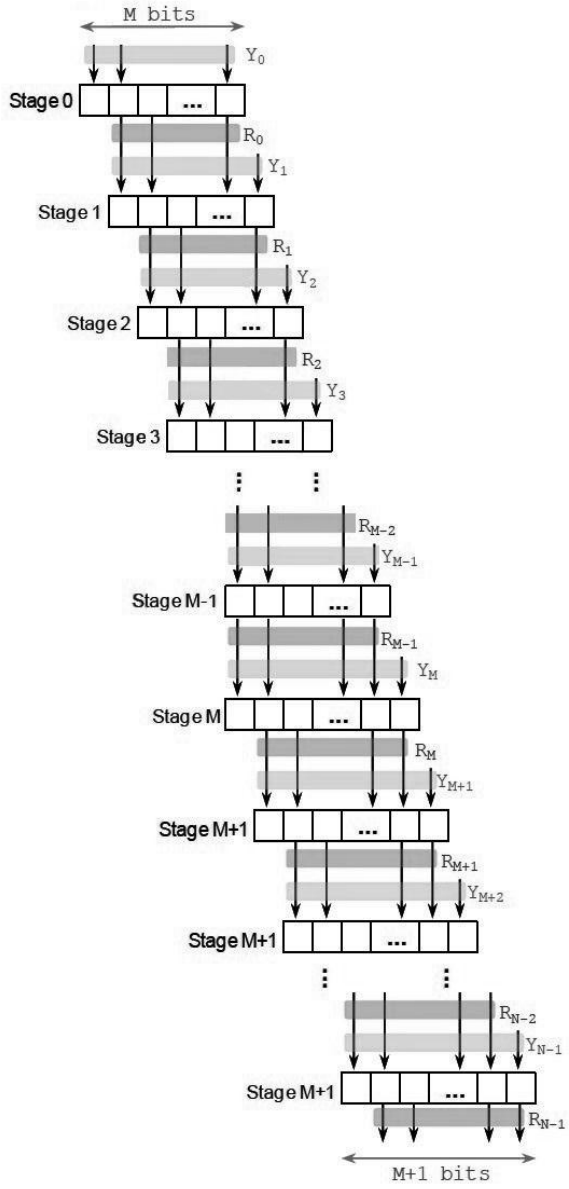


Fig. 1: Parallel implementation algorithm

Since B has M bits, the operation $Y_i - B$ requires M bits for both operands. To maintain consistency, we let Y_i be represented with M bits. R_i Represents the output of each stage. For the first M stages, R_i requires $i+1$ bits. However, for consistency and clarity's sake, since R_i might be the result of a subtraction, we let R_i use M bits.

For the stages in between 0 to $M-2$, R_i is always transferred onto the next stage. Note that we transfer R_i with $M-1$ least significant bits. There is no loss of accuracy here since R_i at most requires $M-1$ bits for stage $M-2$. We need R_i with $M-1$ bits since Y_{i+1} uses M bits.

For the stages in between $M-1$ to $N-1$, Starting from stage $M-1$, R_i requires M bits. We also know that the residue requires at most M bits (maximum value is 2^M-2). So, starting from stage $M-1$ we need to transfer M bits. As Y_{i+1} now requires $M+1$ bits, we need $M+1$ units starting from stage M.

To implement the operation $Y_i - B$ we use a subtractor. If $Y_i \geq B$ then $cout_i = 1$, and when $Y_i < B \rightarrow cout_i = 0$. This $cout_i$ becomes a bit of the quotient: $Q_i = cout_{N-1-i}$. This quotient Q requires N bits at the most. Also, the final residue is the result of the last stage. The maximum theoretical value of the residue is $2^M - 2$, thus the residue R requires M bits where $R = R_{N-1}$. Also, note that we should avoid a division by 0. If B=0, then, in our circuit: $Q = 2^N - 1$ and $R = a_{M-1}a_{M-2} \ldots a_0$.

IV. Combinational Array Divider

The Fig. 2 shows the hardware of this array divider for N=8 and M=4. Here the first M=4 stages only require 4 units, while the next stages requires 5 units. This is fully combinatorial implementation. Each level computes R_i. It first computes $Y_i - B$. When $Y_i \geq B \rightarrow cout_i = 1$, and when $Y_i < B \rightarrow cout_i = 0$. This $cout_i$ is used to determine whether the next R_i is $Y_i - B$ or Y_i. Each Processing Unit (PU) is used to process $Y_i - B$, one bit at a time, and to let a particular bit of either $Y_i - B$ or Y_i be transferred on to the next stage.

V. Fully Pipelined Array Divider

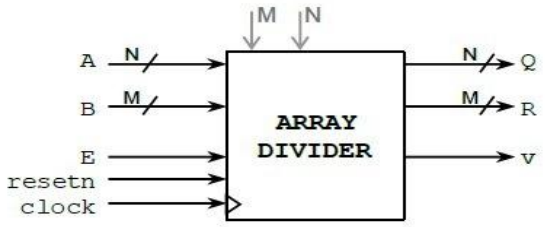


Fig. 2: Combinational Array divider block schematic

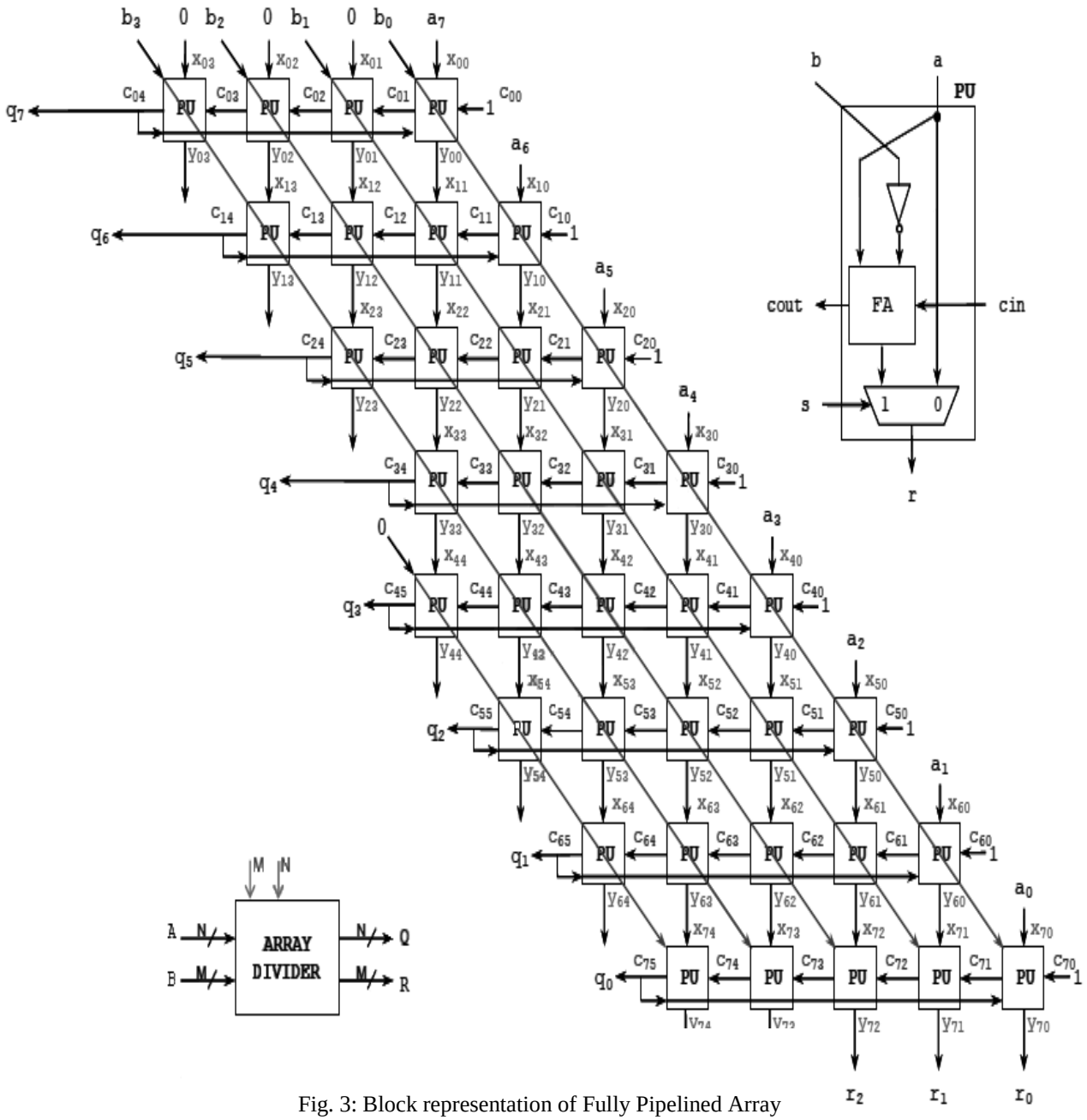


Fig. 3: Block representation of Fully Pipelined Array divider

As shown in Fig. 3 the hardware core of the fully pipelined array divider with its inputs, outputs, and parameters. The Fig. 4 shows the internal architecture of this pipelined array divider for N=8, M=4. Note that the first M=4 stages only require 4 units, while the next stages require 5 units. Note that the enable input 'E' is distributed across the enable inputs of all flip flops. The exception is the shift register on the left, which is used to generate the valid output.

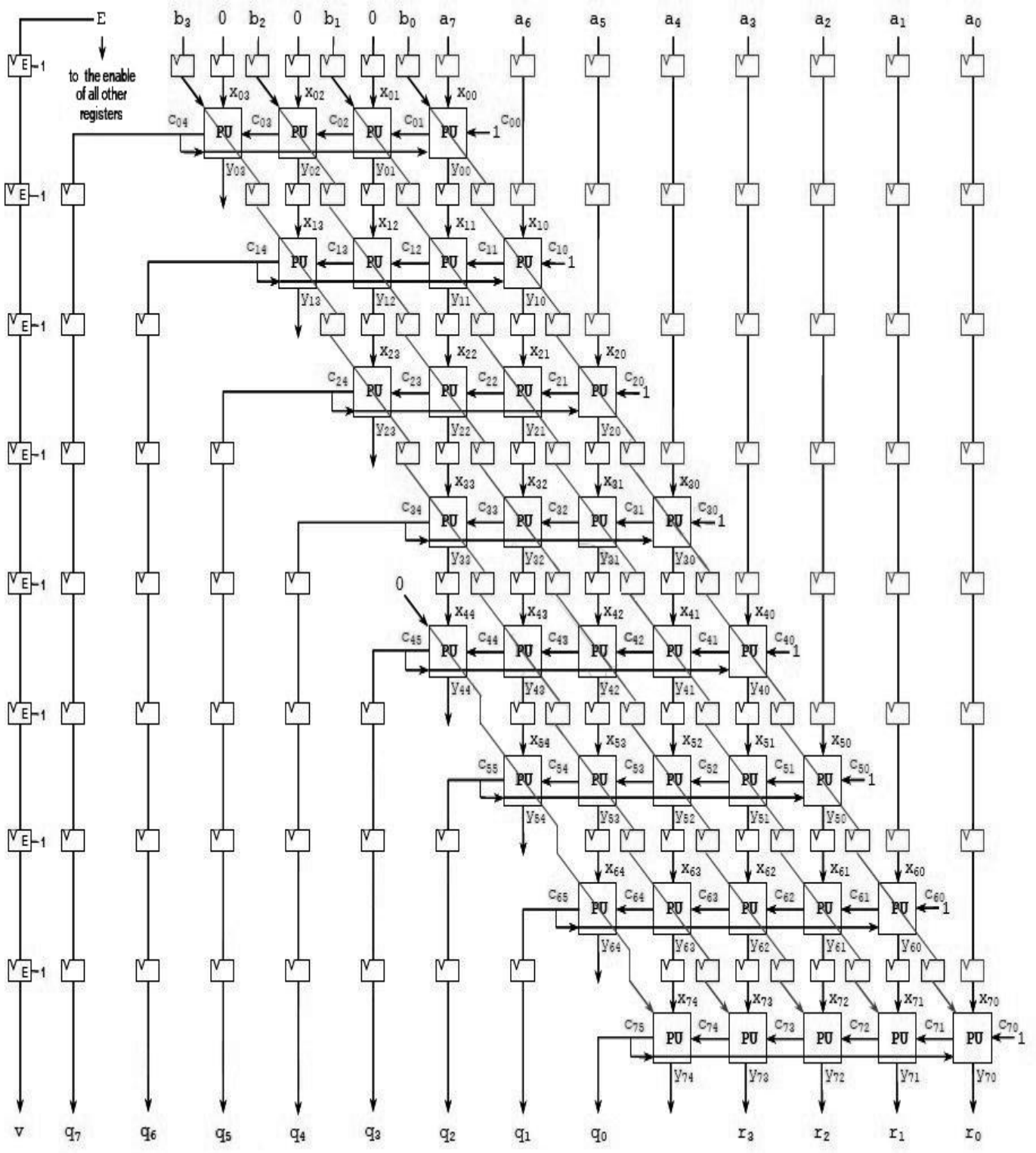


Fig. 4: Schematic of Fully Pipelined Array divider

VI. Iterative Restoring Divider

The Fig. 5&6 shows the iterative hardware architecture and the state machine. Here, R_i is always held at register R. The subtractor computes $Y_i - B$. This requires $M+1$ bits in the worst case. If $Y_i \geq B$ then $R_i = Y_i - B$. Y_i here is the minuend. $Y_i - B$ is loaded onto register R. Note that only M bits are needed. If $Y_i < B$, then $R_i = Y_i$. Here only Y_i is loaded onto register R. This is done by just shifting a_{N-1} into register R. Here, R requires M bits since it holds the residue at every stage. Also, since we always shift $cout_i$ onto register A, the quotient Q is held at A in the last iteration.

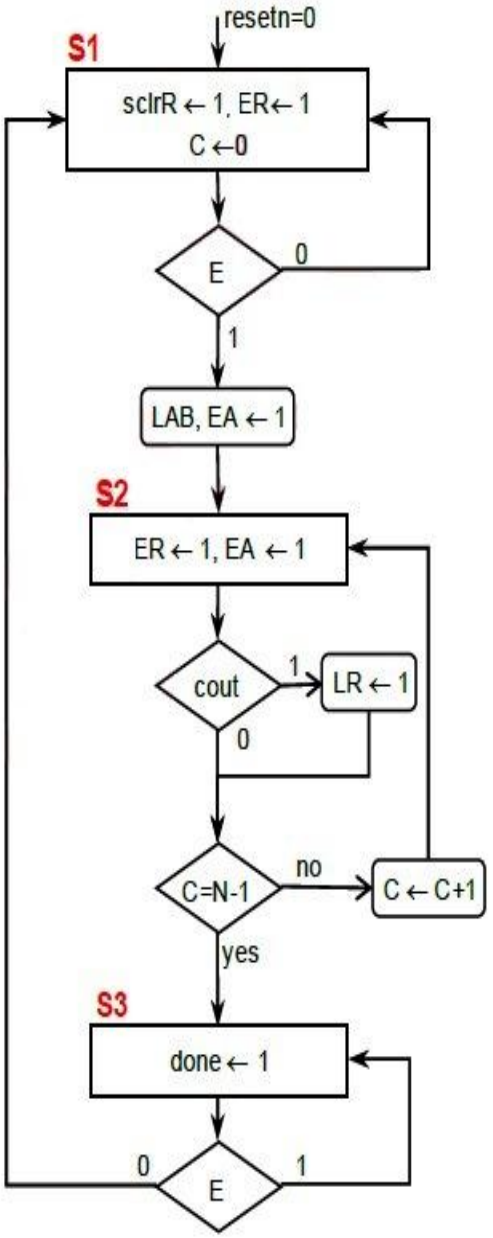


Fig. 5: State Machine of Iterative Restoring Divider

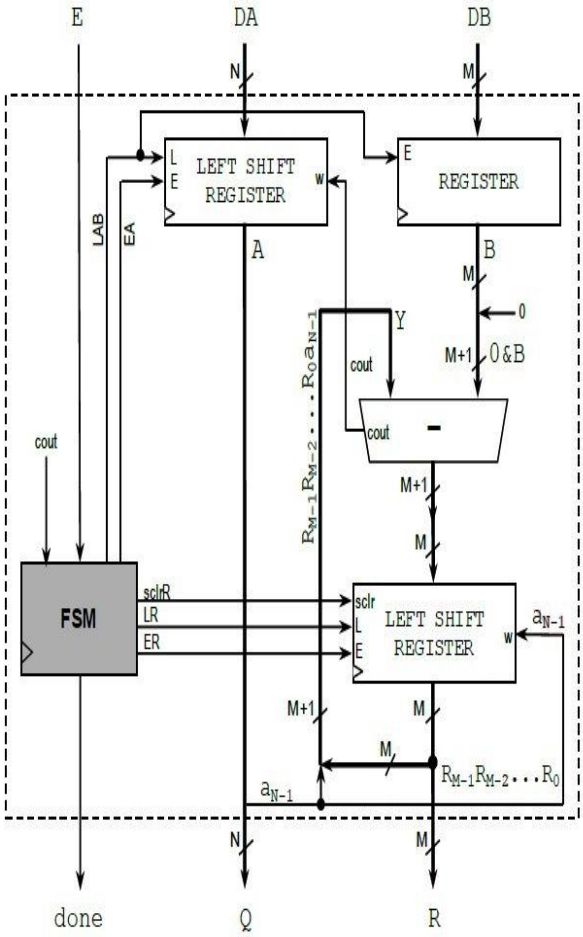


Fig. 6: Iterative Divider Architecture

VII. Result

The described divisor models were implemented in the FPGA device XC3S400-TQ144 (Xilinx Spartan-3 family) with speed grade -5 and in in the FPGA device XC4VFX12 -SF363 (Xilinx Virtex-4 family) with speed grade -12. The development system ISE v 14.1 with default settings was used. The implementation results – Maximum combinational path delay (for Combinatorial Array Divider) and Minimum period, Maximum Frequency, Minimum input arrival time before clock and Maximum output required time after clock (for Fully Pipelined Array Divider and Iterative Restoring Divider) obtained by Synthesize-XST are given in following tables (Table 3 and Table 4).

Comparison of AREA:

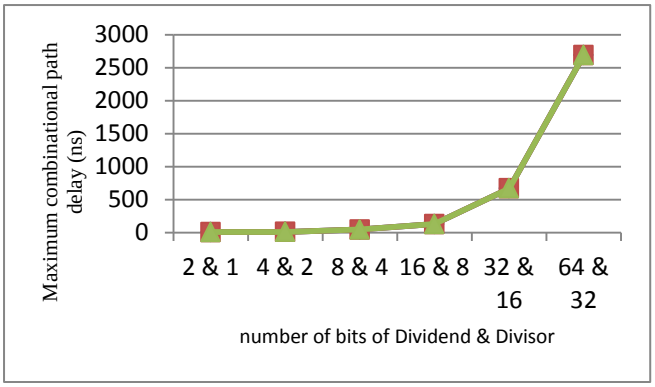
As seen in the Fig. 8 & 9, the amount of area required to implement on these devices have been compared. These comparisons are done for the three designs based on the implementation in the FPGA device XC3S400-TQ144 using Implementation Design-Analyze Timing/Floor plan Design (Plan Ahead).

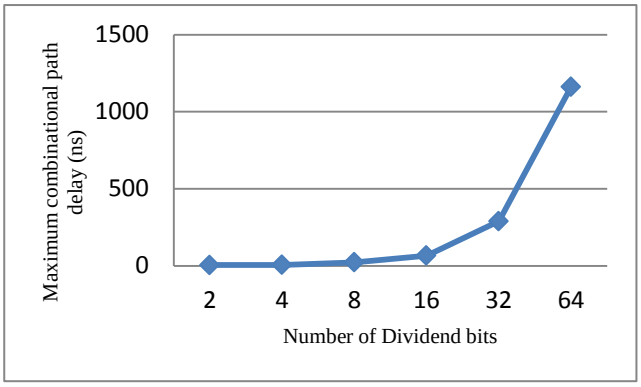
Table 3: Comparison table of timing analysis for Spartan 3

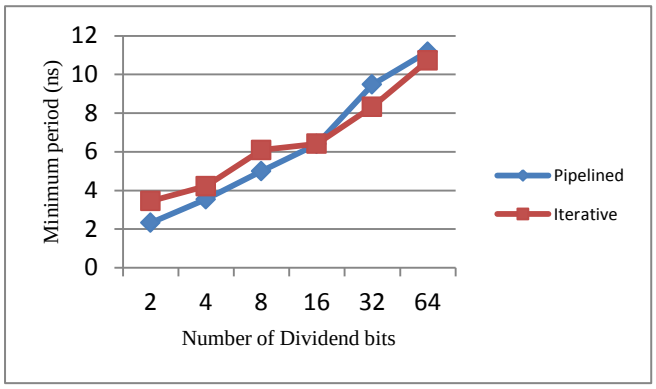
Device Family	Divide nd bits	Divisor bits	Division Algorithm	Maximum combinational path delay (ns)	Minimum period (ns)	Maximum Frequency (MHz)	Minimum input arrival time before clock (ns)	Maximum output required time after clock (ns)
Spartan-3	2	1	Combinational	7.850	…..	…..	…..	…..
			Pipelined	…..	2.321	430.765	2.160	7.735
			Iterative	…..	3.451	289.809	3.611	6.441
	4	2	Combinational	9.150	…..	…..	…..	…..
			Pipelined	…..	3.538	282.622	2.804	10.302
			Iterative	…..	4.222	236.860	3.679	6.456
	8	4	Combinational	45.999	…..	…..	…..	…..
			Pipelined	…..	4.998	200.094	3.238	12.045
			Iterative	…..	6.095	164.077	4.578	6.456
	16	8	Combinational	127.412	…..	…..	…..	…..
			Pipelined	…..	6.399	156.266	5.044	17.465
			Iterative	…..	6.418	155.818	4.247	6.544
	32	16	Combinational	669.188	…..	…..	…..	…..
			Pipelined	…..	9.475	105.541	9.421	28.046
			Iterative	…..	8.330	120.053	5.058	6.895
	64	32	Combinational	2691.854	…..	…..	…..	…..
			Pipelined	…..	11.175	89.484	9.207	48.893
			Iterative	…..	10.720	93.284	5.339	7.159

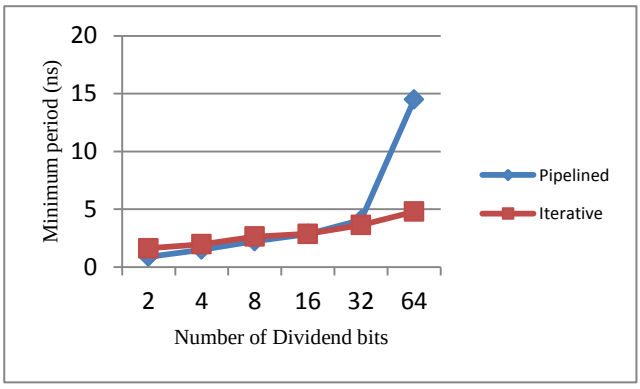
Table 4: Comparison table of timing analysis for Virtex 4

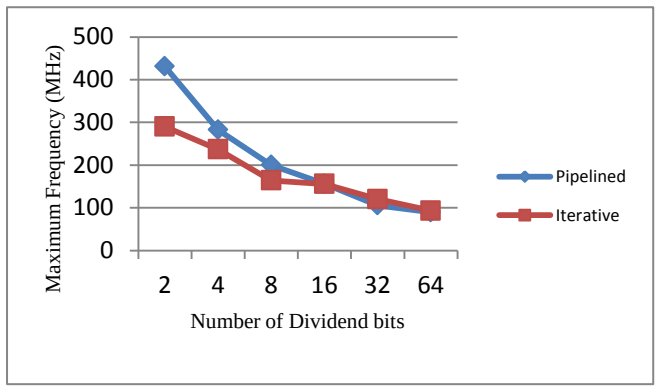
Device Family	Dividend bits	Divisor bits	Division Algorithm	Maximum combinational path delay (ns)	Minimum period (ns)	Maximum Frequency (MHz)	Minimum input arrival time before clock (ns)	Maximum output required time after clock (ns)
Virtex-4	2	1	Combinational	4.871	…..	…..	…..	…..
			Pipelined	…..	0.885	1130.199	1.492	4.467
			Iterative	…..	1.619	617.608	2.117	3.856
	4	2	Combinational	5.586	…..	…..	…..	…..
			Pipelined	…..	1.495	668.762	1.843	5.532
			Iterative	…..	1.966	508.660	2.142	3.856
	8	4	Combinational	22.135	…..	…..	…..	…..
			Pipelined	…..	2.219	450.592	2.100	6.653
			Iterative	…..	2.627	380.713	2.179	3.856
	16	8	Combinational	66.053	…..	…..	…..	…..
			Pipelined	…..	2.845	351.512	3.161	8.992
			Iterative	…..	2.862	349.424	2.348	3.964
	32	16	Combinational	289.622	…..	…..	…..	…..
			Pipelined	…..	4.058	246.418	4.936	13.416
			Iterative	…..	3.619	276.304	2.863	4.074
	64	32	Combinational	1160.928	…..	…..	…..	…..
			Pipelined	…..	14.487	69.029	5.379	22.442
			Iterative	…..	4.788	208.862	3.000	4.230

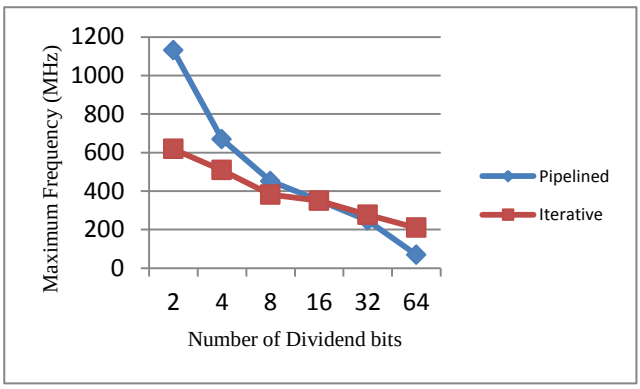


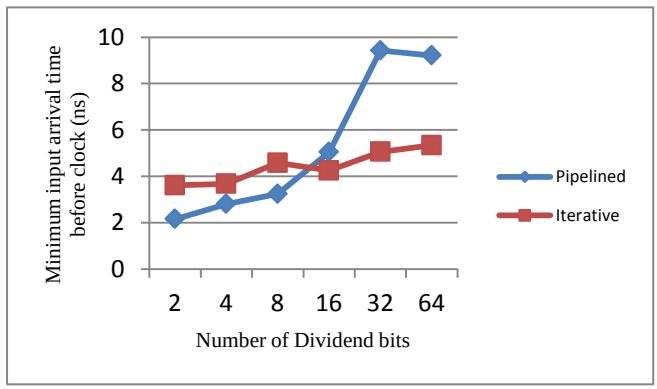


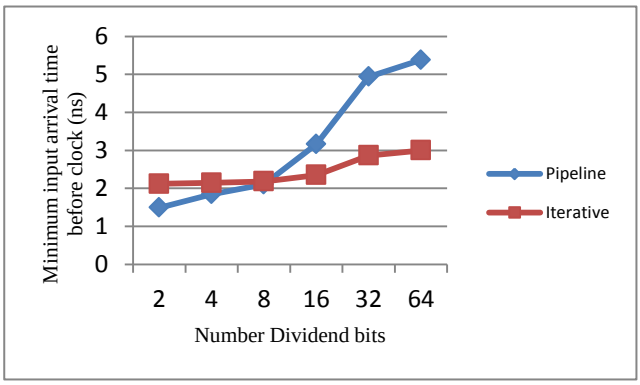


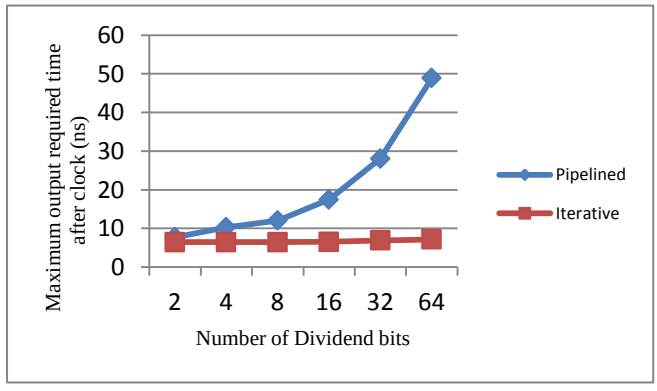












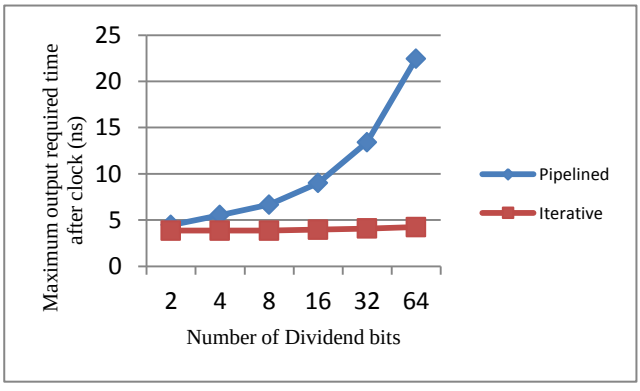


Fig. 7: Various parameter graph for Spartan 3

Fig. 8: Various parameter graph for Virtex 4

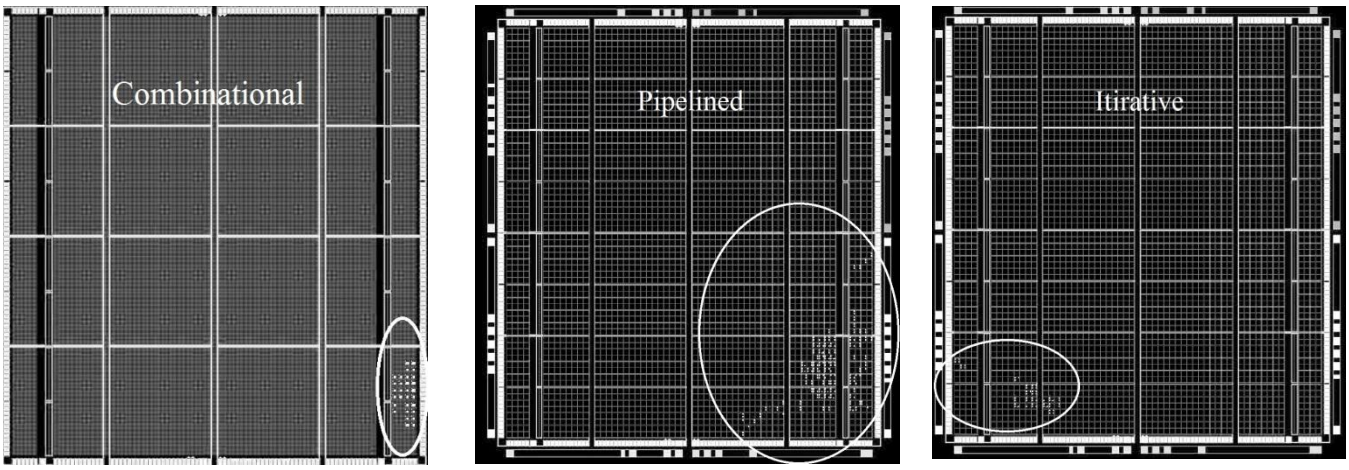


Fig. 9: Area comparison for Spartan 3

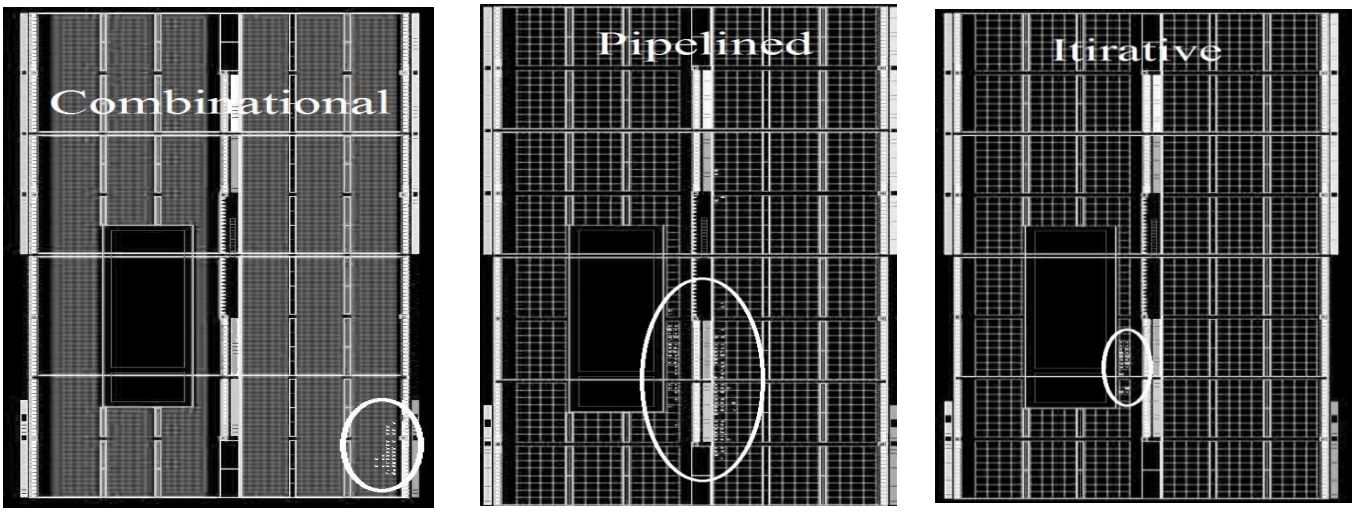


Fig. 9: Area comparison for Virtex 4

VIII. CONCLUSION

The paper introduces three types of synthesizable model of the divider that can be implemented in any FPGA devices. In our approach the designs have been targeted to Spartan-3 & Virtex-4 and the results have been compared respectively. The Maximum combinational path delay (ns) in the 'Combinatorial Array Divider' increases exponentially as the number of dividend (or divisor) bits increases. Hence, the overall execution time increases exponentially with the increase in the input operand value. The minimum period (ns) time for execution in 'Fully Pipelined Array Divider' is less than the execution time in 'Iterative Restoring Divider' only when the number of dividend/divisor bits is less than 16/8. But, when these number of bit values are increased, then the 'Iterative Restoring Divider' works effectively as compared to 'Fully Pipelined Array Divider'. This inference is justified by the results obtained by both the Maximum frequency (MHz) and the Minimum input arrival time before clock: this is because the Maximum output required time after clock (ns) remains almost constant for 'Iterative Restoring Divider' whereas it increases exponentially for 'Fully Pipelined Array Divider'. The 'Iterative Restoring Divider' requires least amount of area to be implemented, while 'Combinatorial Array Divider' requires moderate and the 'Fully Pipelined Array Divider' requires the maximum amount of area to be implemented.

REFERENCES

[1]. Fedra, Z.; Kolouch, J., "VHDL procedure for combinational divider," Telecommunications and Signal Processing (TSP), 2011 34th International Conference on , vol., no., pp.469,471, 18-20 Aug. 2011
[2]. Agrawal, Dharma P., "Optimum array-like structures for high-speed arithmetic," Computer Arithmetic (ARITH), 1975 IEEE 3rd Symposium on, vol., no., pp.208,219, 19-20 Nov. 1975
[3]. Aoki, T.; Nakazawa, K.; Higuchi, T., "High-radix parallel VLSI dividers without using quotient digit selection tables," Multiple-Valued Logic, 2000. (ISMVL 2000) Proceedings. 30th IEEE International Symposium on , vol., no., pp.345,352, 2000
[4]. Cappa, M.; Hamacher, V.C., "An Augmented Iterative Array for High-Speed Binary Division," Computers, IEEE Transactions on, vol.C-22, no.2, pp.172,175, Feb. 1973
[5]. Takagi, N.; Kadowaki, S.; Takagi, K., "A hardware algorithm for integer division," Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on , vol., no., pp.140,146, 27-29 June 2005
[6]. Kei-Yong Khoo; Willson, A.N., Jr., "Efficient VLSI implementation of N/N integer division," Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on , vol., no., pp.672,675 Vol. 1, 23-26 May 2005

[7]. Wang, C.-C.; Huang, C.-J.; Lin, G.-C., "Cell-based implementation of radix-4/2 64b dividend 32b divisor signed integer divider using the COMPASS cell library," Computers and Digital Techniques, IEE Proceedings - , vol.147, no.2, pp.109,115, Mar 2000
[8]. Oberman, S.F.; Flynn, M., "Division algorithms and implementations," Computers, IEEE Transactions on , vol.46, no.8, pp.833,854, Aug 1997,
[9]. N. Sorokin, "Implementation of high-speed fixed-point dividers on FPGA," Journal of Computer Science & Technology, Vol. 6, No. 1, April 2006, p. 8 – 11.
[10]. J. E. Robertson, "A new class of digital division methods," IRE Trans. Electronic Computers, vol.EC-7, pp.218-222, Sept. 1958.
[11]. Hallin, T.G.; Flynn, M., "Pipelining of arithmetic functions," Computer Arithmetic (ARITH), 1972 IEEE 2nd Symposium on , vol., no., pp.1,28, 15-16 May 1972

Narendra K, Student (M.Tech) Digital Electronics and Communication systems, Malnad College of Engineering, Hassan, Karnataka, India, +91-9738543811,

ShabirAhmed B J, Student (M.Tech) Digital Electronics and Communication systems, Malnad College of Engineering, Hassan, Karnataka. India. +91-9738417860,

Swaroop Kumar K, Student (M.Tech) Digital Electronics and Communication systems, Malnad College of Engineering, Hassan, Karnataka, India, +91-7411379265,

Asha G H, Associate Professor, Dept. of Electronics and communication, Malnad College of Engineering, Hassan, Karnataka, India, +91-9448033837.,