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

FPGA Implementation of modified non-restoring square root core

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

***Abstract*— The aim of this paper is to synthesize and implement an algorithm to compute square root efficiently and cost effectively. Square root is a computation required in many mathematical problems required in computer multimedia communication and in many space related data processing. So, there is a requirement to develop this operation efficiently. Hence, in this paper a new type of algorithm is used to design a core for finding square root. The core implemented in this paper uses a modified non restoring division algorithms to find square root. Three types of structures have been developed namely: basic combinational, iterative and pipeline. Basic combinatorial is simple implementation of non-restoring division algorithm, it is nothing but single stage of pipelined architecture. This architecture can be used when cost is the major factor and speed can be compromised. Iterative architecture is hardware efficient cost effective architecture where single hardware unit is used iteratively for a computation. Pipelined architecture is the fastest architecture to be implemented in this paper. It uses various stages for computation i.e. parallel execution is performed and hence speeding up the execution time and process. Pipelined architecture can be used in real time processing system where speed is the major factor. These three structures are developed to compare various parameters like speed, cost, reliability and distinguish them for various application suited by them. The core is developed for any FPGA processor and is simulated and debugged using XILINX ISE 14.1. The architecture is implemented onto VIRTEX family and debugged on Spartan 3 XC3S400TQ144.**

***Index Terms*—Combinatorial, FPGA, Iterative, Pipelined, Spartan, Virtex**

I. Introduction

Square root is an operation required by system graphics and scientific computation applications such as math coprocessors, DSP algorithms, data processing and control [1]. Hence, it is an important computation that need to be enhanced. In 1996, Lu and Chi [1] have proposed a 'new non-restoring square root algorithm' for VLSI implementation, which is better than the existing VLSI algorithms for computing square root. In many VLSI real time image processing applications, it is high prioritized requirement to provide the computation of square root of a binary coded number with low power dissipation and fast computation (low delay propagation). Square root calculation is one of the most useful and vital operations in computer graphics and scientific calculation applications, such as digital signal processing (DSP) algorithms, math coprocessor, data processing and control, and even multimedia applications [1-6]. It is a classical problem in computational number theory, which is oftenly encountered and which is a hard task to get an exact result [7-8].

The paper is divided as follows: Section II describes the algorithm. Section III presents the implemented architectures. Section IV explains the results and analysis, and in the results a detailed comparison between the spartan core and Virtex core is presented. Finally, conclusions is given.

II. Non Restoring Algorithm

The focus of the previous restoring and non-restoring algorithms is on each bit of the square root with each iteration. In this section, non-restoring square root algorithm has been described as in [1]. Each operation consists of addition or subtraction based on the sign of the result of previous operation. The partial remainder generated in each iteration is used in the next iteration even it is negative [1]. At the final iteration, if the partial remainder is not negative, it becomes the final precise remainder.

Radical: 'D' of '2n bits. Square root: 'Q' of 'n' bits:

D:	D_{2n-1}	D_{2n-2}	D_{2n-3}	D_{2n-4}	…	D_1	D_0

Q:	Q_{n-1}	Q_{n-2}	…	Q_0

$$d_k = D_{2n-1}D_{2n-2}\dots D_k, k \le 2n-1$$

$$q_k = Q_{n-1}Q_{n-2}\cdots Q_k, k \le n-1$$

**Note that q_k has 'n-k' bits*

$$r'_{n-1} \leftarrow D_{2n-1}D_{2n-2} - 01$$

$$Q_{n-1}\begin{cases}1, if\, r'_{n-1} \ge 0\\ 0, if\, r'_{n-1} < 0\end{cases}$$

For k = n-2 downto 0 do

Manuscript received April 15, 2015.

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

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

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.

$$r'_k \begin{cases} r'_{k+1}D_{2k+1}D_{2k} - q_{k+1}01, if\ Q_{k+1} = 1 \\ r'_{k+1}D_{2k+1}D_{2k} + q_{k+1}11, if\ Q_{k+1} = 0 \end{cases}$$

$$Q_k \begin{cases} 1, if\ r'_k \geq 0 \\ 0, if\ r'_k < 0 \end{cases}$$

End

Remainder R = r_0

$$r_0 \begin{cases} r'_0\ if\ r'_0 \geq 0 \\ r'_0 + q_1\,01, if\ r'_0 < 0 \end{cases}$$

At each iteration, q_k (the square root of d_{2k}), is computed. Since $d_{2k} = d_{2(k+1)}D_{2k+1}D_{2k}$, that is $D_{2k+1}D_{2k}$ is attached to $d_{2(k+1)}$ to form d_{2k}, it can be inferred that $D_{2k+1}D_{2k}$ must be used to get q_k. That explains the fact the algorithm attaches $D_{2k+1}D_{2k}$ to r'_{k+1} to form r'_k in order to get q_k. The remainder at each iteration, called r_k, has 'n-k+1' bits, one more bit than q_k [1]: $r_k = R\ _nR_{n-1}R_{n-2}\ ...\ R_k$, But the algorithm uses an estimated remainder, called r'_k, that has 'n-k+2' bits, the MSB is the sign bit, which decides the value of Q_k, and it can be demonstrated that only the 'n-k+1' least significant bits of r'_k are used to get the next estimated remainder r'_{k-1}. Also, in order to get the real remainder R = r_0, only the 'n+1' LSBs of r'_0 are needed (the MSB determines Q_0). It lessens the gate count, since a register of only 'n-k+1' bits is needed for r'_k.

III. Architectures

As mentioned in abstract three types of architectures have been implemented which will be described in this section.

A. *Pipelined*

To implement this architecture we need to unfold the algorithm explained in section II. Therefore 'n' stages with 'n' adders/subtractors will appear. By observing the first iteration, a reduction is obtained:

$$R'n\text{-}1 \leftarrow D2n\text{-}1D2n\text{-}2 - 01$$
$$Q_{n-1} \leftarrow 1,\ if\ r'_{n-1} \geq 0$$
$$Q_{n-1} \leftarrow 0,\ if\ r'_{n-1} < 0$$

There is no need to perform the first subtraction and wait one cycle, if the result from the first iteration can be obtained directly from the first 2 MSBs of D. So the first stage can be embedded into the second stage, and there will be 'n-1' pipeline stages.

This architecture is depicted in Figure 1. The computation of the remainder is not considered, although the core computes it if the user wants. Note that the dotted rectangles indicate the registers that would have appeared if the reduction of the first stage hadn't been performed. Such architecture can obtain a new square root each cycle. The initial latency is 'n' cycles.

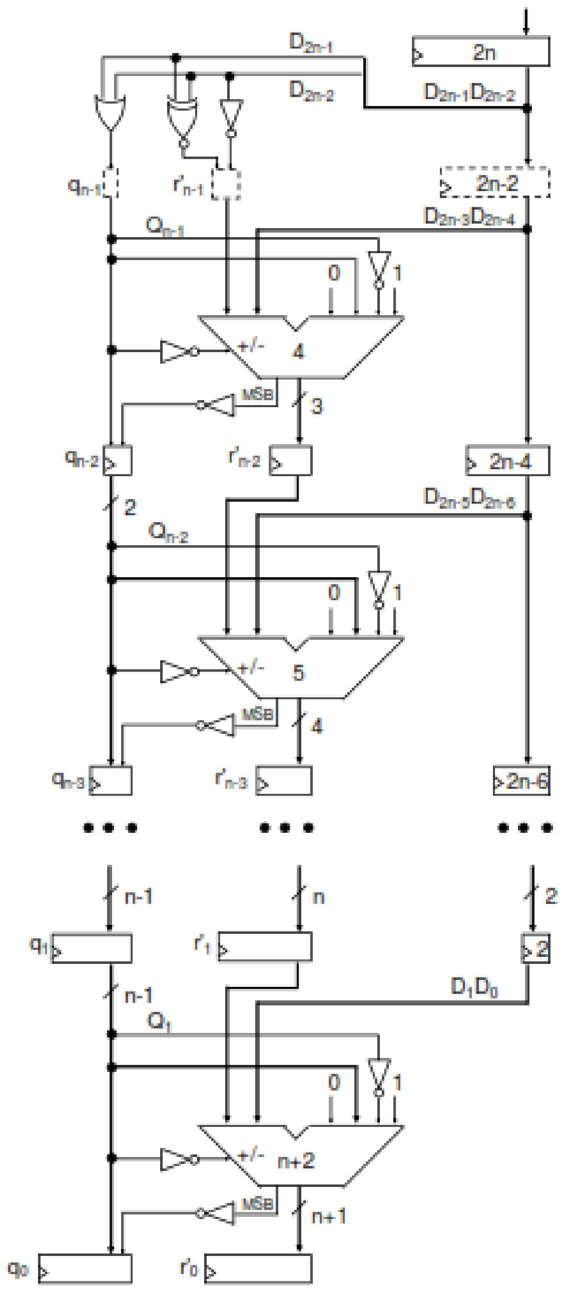


Figure 1: Pipelined architecture

The longest path delay occurs in the last stage, because the adder/substractor increases in size as stages advance. A further improvement can be made if the last stages are pipelined, and the initial ones merged.

B. *Combinatorial Architecture*

This architecture is implemented because some non-real time applications need it, and also in order to establish a comparison with the core that does have a fully-combinatorial architecture. The architecture is very simple: It is the fully-pipelined architecture without the pipelining registers. It only has one register at the input and one at the output.

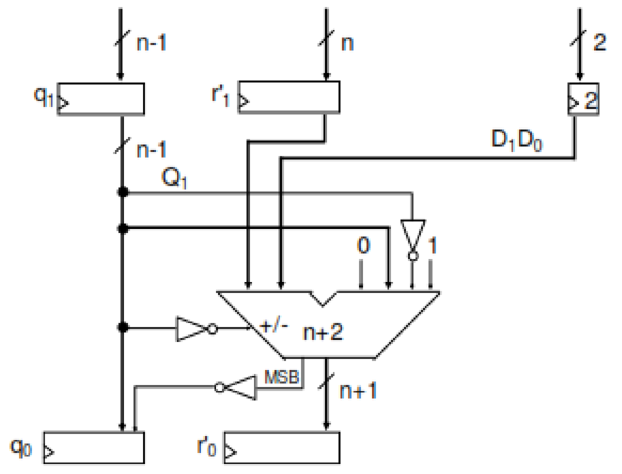


Figure 2: Combinatorial Architecture

C. *Iterative Architecture*

The size of the elements (registers, adder/subtractor) will be the size of the last stage of the pipelined architecture:
Register R 'n+1' bits Register Q 'n' bits Adder / subtractor 'n+2' bits.
Since all iterations are embedded in one stage, the reduction of Section III A cannot be used.

But a simplification for this case exists:

In the adder/subtractor: the 2 LSBs performs either 'xy-01' or 'xy+11', 'xy' is the pair of D bits used at each step. The operation yields: 'cba'. The truth table is shown:

cba = xy + 11		cba = xy - 01	
xy	cba	xy	cba
00	011	00	111
01	100	01	000
00	101	00	001
01	110	01	010

'C': carry-in for the next stage of the adder/subtractor 'ba': result of the operation.

'Ba' depends only on 'xy', but 'c' depends on the type of operation. Luckily, a conventional adder/substractor with carry-in (e.g. the lpm_add_sub megafuntion) treats the carry-in as positive logic when adding, and as negative logic when subtracting [3] (this is done to reduce gates usage). So, for subtraction, we have to invert 'c' to assure the proper working of the adder/subtractor. The new truth table is:

cba = xy + 11		cba = xy - 01	
xy	cba	xy	cba
00	011	00	011
01	100	01	100
00	101	00	101
01	110	01	110

Now, 'c' and 'ba' depends only on 'xy':

$$c = x + y \quad b = (x \oplus y) \quad a = y$$

This reduces the width of the adder/subtractor by 2 bits. The result 'ba' is obtained in parallel and the carry-in comes from just an OR gate. So the new adder/subtractor uses 'n' bits and has carry-in. Also, note that the MSB of the second operator of the adder/subtractor is '0' as in the pipelined case. Figure 3 depicts this architecture.

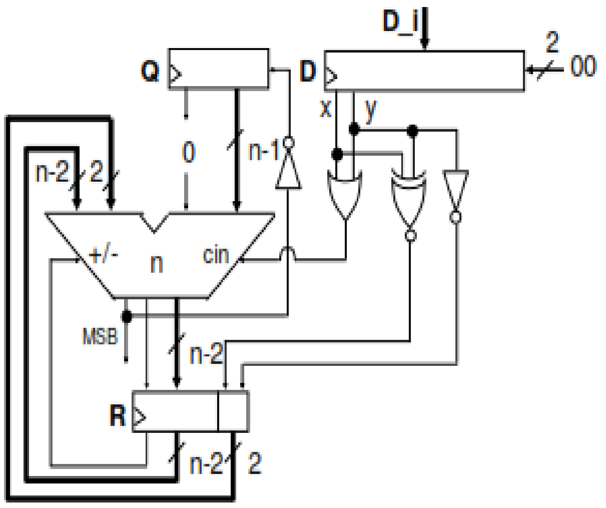


Figure 3: Iterative Architecture

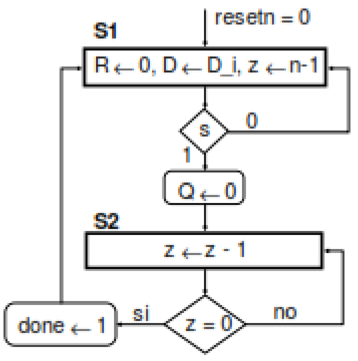


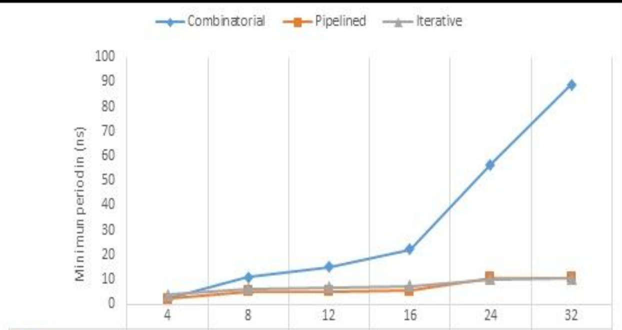
Figure 4: FSM for iterative architecture

Finite state machine of iterative architecture is depicted in figure 4. This FSM controls the iterative architecture. The process start when s = 1.After 'n' clock cycles, the result is obtained in register Q, done = 1, and a new process can be started.

IV. Results

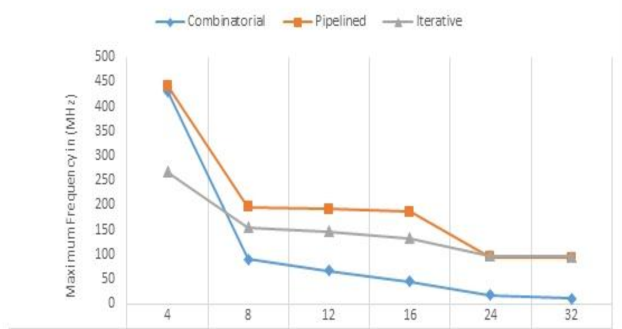
The architecture were synthesized using XILINX ISE v14.1 successfully. After synthesizing the core were implemented on FPGA device XC3S400-TQ144 (Xilinx Spartan-3 family) with speed grade -5. The core presented does not compute the remainder, since it is rarely used. Figure 5 depicts the core with all its options. Table 1 establishes a comparison between this core and the ALTERA core.

Results are shown only for a specific device (Spartan 3) because of large results data with just one device and these results are enough to demonstrate the benefits of the core implemented.



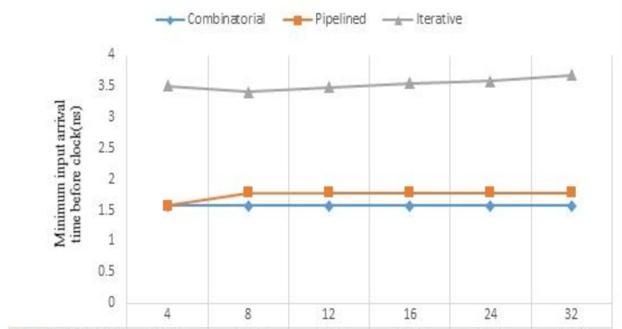
	4	8	12	16	24	32
Combinatorial	2.321	11.067	15.023	22.259	56.5	88.862
Pipelined	2.263	5.094	5.205	5.344	10.596	10.68
Iterative	3.74	6.426	6.82	7.534	10.226	10.448

'D' --> size of input bits



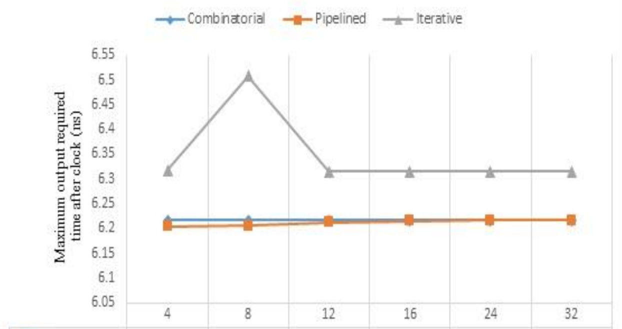
	4	8	12	16	24	32
Combinatorial	430.765	90.357	66.567	44.925	17.699	11.253
Pipelined	441.803	196.298	192.112	187.133	94.375	93.637
Iterative	267.355	155.625	146.623	132.734	97.79	95.712

'D' --> size of input bits



	4	8	12	16	24	32
Combinatorial	1.572	1.572	1.572	1.572	1.572	1.572
Pipelined	1.572	1.778	1.778	1.778	1.778	1.778
Iterative	3.504	3.413	3.486	3.554	3.584	3.685

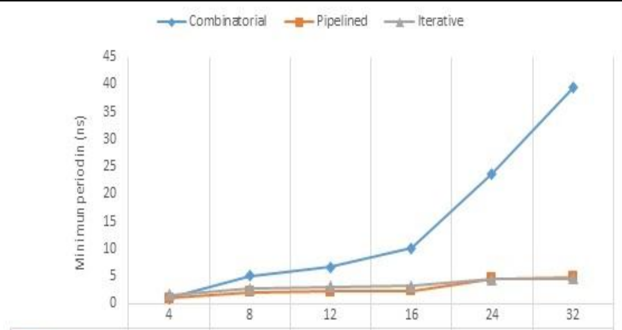
'D' -- size of input bits



	4	8	12	16	24	32
Combinatorial	6.216	6.216	6.216	6.216	6.216	6.216
Pipelined	6.204	6.206	6.212	6.215	6.216	6.216
Iterative	6.318	6.507	6.314	6.314	6.314	6.314

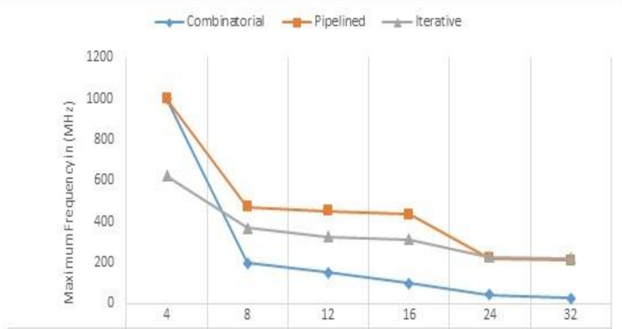
'D' -- size of input bits

Figure 5: Parameter comparison graph for Spartan



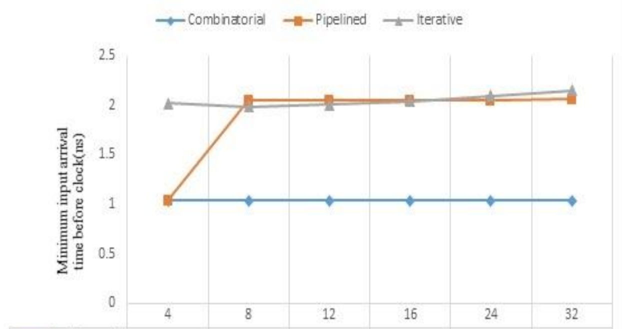
	4	8	12	16	24	32
Combinatorial	1.003	5.054	6.613	10.148	23.728	39.414
Pipelined	1.003	2.121	2.212	2.305	4.527	4.75
Iterative	1.605	2.708	3.061	3.212	4.415	4.551

'D' --> size of input bits



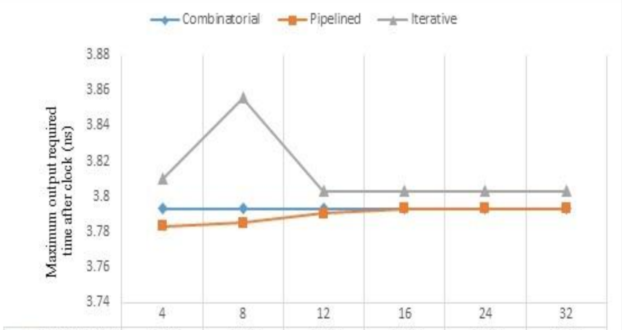
	4	8	12	16	24	32
Combinatorial	997.208	197.875	151.215	98.544	42.145	25.372
Pipelined	997.208	471.509	452.028	433.858	220.909	210.513
Iterative	622.917	369.263	326.728	311.318	226.503	219.734

'D' --> size of input bits



	4	8	12	16	24	32
Combinatorial	1.037	1.037	1.037	1.037	1.037	1.037
Pipelined	1.037	2.05	2.05	2.05	2.05	2.06
Iterative	2.023	1.987	2.004	2.04	2.095	2.149

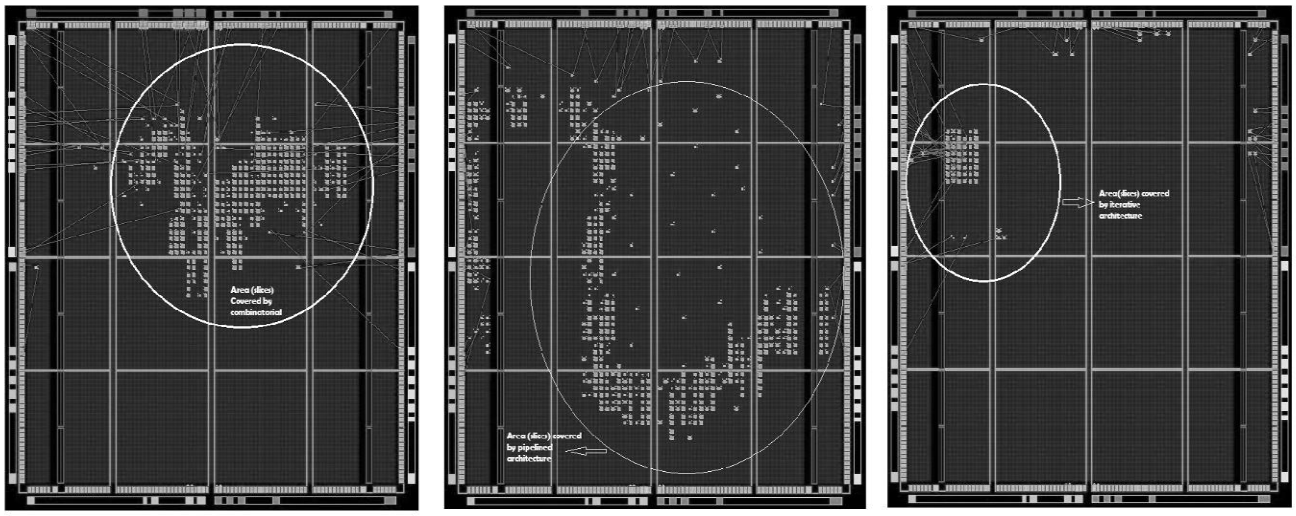
'D' -- size of input bits



	4	8	12	16	24	32
Combinatorial	3.793	3.793	3.793	3.793	3.793	3.793
Pipelined	3.783	3.785	3.7903	3.793	3.793	3.793
Iterative	3.81	3.856	3.803	3.803	3.803	3.803

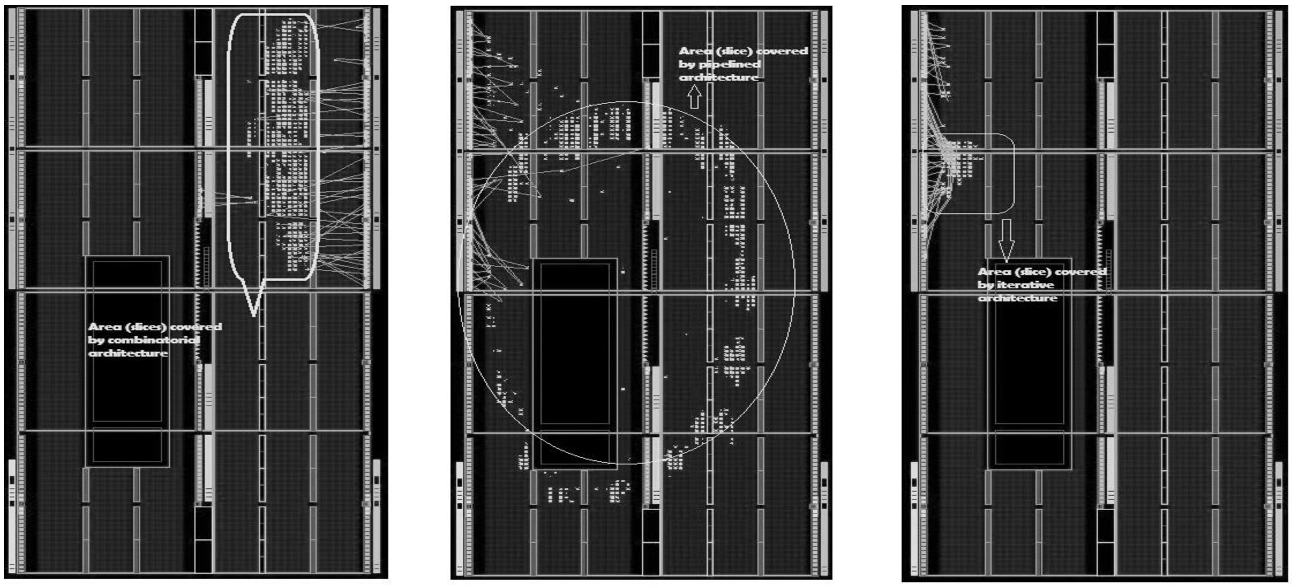
'D' -- size of input bits

Figure 5: Parameter comparison graph for Virtex


Area (slices) Covered by combinatorial
Area (slices) covered by pipelined architecture
Area(slices) covered by iterative architecture

a. Combinatorial b. pipelined c. Iterative

Figure 7: Area comparison of all three architecture for SPARTAN 3


Area (slices) covered by combinatorial architecture
Area (slice) covered by pipelined architecture
Area (slice) covered by iterative architecture

a. Combinatorial b. Pipelined c. Iterative

Figure 7: Area comparison of all three architecture for VIRTEX 4

Result analysis:

From the graphs shown in the figure 5 and 6 we can infer that latency is more in combinatorial. But in pipeline the latency till 16 bits are low than iterative. After 16 bits iterative and pipeline latency time crossover: i.e. for large input bits latency increases in pipeline. Hence efficiency in pipeline architecture is the highest. Maximum frequency operable is highest in pipelined architecture.

Area Comparison:

As shown in figure 7 and 8 area covered by all three architecture can be seen. Plan Ahead tool of XILINX was used to generate area and compare them. We can infer from figure 7 and 8 that pipeline architecture uses more area and iterative uses the least area.

V. Conclusion

The core implemented achieves high speed at minimum cost since it only uses only an adder/subtractor unit to perform the operations. The architecture is very flexible, so that the user can choose the best architecture for his application. The efficiency of this core can be observed from graph and table.

The results are better in terms of speed and resource effort than the earlier implementation. An improvement i.e. simplification for the iterative architecture can be applied to each stage of the pipelined architecture.

REFERENCES

[1] Y. Li and W. Chu, “A New Non-Restoring Square Root Algorithm and Its VLSI Implementations”, Proc. Of 1996 IEEE International Conference on Computer Designs: VLSI in Computers and Processors, Austin, Texas, USA, October 1996, pp538-544..

[2] J. Hennessy and D. Patterson, Computer Architecture, A Quantitative Approach, Second Edition, Morgan Kaufmann Publishers, Inc., 1996.

[3] G. Knittel, “ A VLSI-Design for Fast Vector Normalization” Comput. & Graphics, Vol. 19, No. 2, 1995. pp261 - 271.

[4] J. Bannur and A. Varma, “The VLSI Implementation of A Square Root Algorithm”, Proc. IEEE Symposium on Computer Arithmetic , IEEE Computer Society Press, Washington D.C., 1985. pp159 - 165.

[5] J. O’Leary, M. Leeser, J. Hickey, M. Aagaard, “NonRestoring Integer Square Root: A Case Study in Design by Principled Optimization”, Proc. 2nd International Conference on Theorem Provers in Circuit Design (TPCD94) , 1994. pp52 - 71.

[6] K. C. Johnson, “Efficient Square Root Implementation on the 68000”, ACM Transaction on Mathematical Software , Vol. 13, No. 2, 1987. pp138 - 151.

[7] H. Kabuo, T. Taniguchi, A. Miyoshi, H. Yamashita, M. Urano, H. Edamatsu, S. Kuninobu, “Accurate Rounding Scheme for the Newton-Raphson Method Using Redundant Binary Representation”, IEEE Transaction on Computers , Vol. 43, No. 1, 1994. pp43 – 51

[8] Brown & Vranesic. Fundamentals of Digital Logic with VHDL Design, McGraw Hill, 2000

[9] U. Meyer – Baese, Digital Signal Processing with Field Programmable Gate Arrays: Springer-Verlag Berlin Heidelberg, May 2001

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

.

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

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.