Novel Pipelined Architecture for Efficient Evaluation of the Square Root Using a Modified Non-Restoring Algorithm

I. Sajid, M. M. Ahmed
Department of Electronic Engineering, Mohammad Ali Jinnah University, Islamabad, 44000, Pakistan

Sotirios G. Ziavras
Electrical and Computer Engineering Department, New Jersey Institute of Technology, Newark, NJ 07102, USA

JOURNAL OF SIGNAL PROCESSING SYSTEMS,
SPRINGER PUBL.,
ACCEPTED FOR PUBLICATION.

Abstract

The square root is a basic arithmetic operation in image and signal processing. We present a novel pipelined architecture to implement N-bit fixed-point square root operation on an FPGA using a non-restoring pipelined algorithm that does not require floating-point hardware. Pipelining hazards in its hardware realization are avoided by modifying the classic non-restoring algorithm, thus resulting in a 13% improved latency. Furthermore, the proposed architecture is flexible allowing modification as per individual application needs. It is demonstrated that the proposed architecture is approximately four times faster than its popular counterparts and at the same time it consumes 50% less energy for envelope detection at 268 MHz sampling rate.

Keywords; Pipelining, fixed-point arithmetic, square root, non-restoring algorithm, Field-Programmable Gate Array (FPGA).

1. Introduction

In addition to the basic arithmetic operations (+, -, * and ÷), the square root is also an essential operation frequently employed by signal and image processing applications. Envelope detection is a simple but vital technique used to recover the original signal in the demodulation process of a communications receiver [1-3]. The envelope detector uses the square root operation numerous times for signal demodulation. In software implementations, the envelope detector uses the floating-point square root operation because of its precision, regardless of speed. Nevertheless, the floating-point square root is costly in hardware realization compared to the integer square root operation because it occupies more space on the chip and consumes higher number of cycles for the same non-pipelined operation [4-8].

Fixed-point hardware with appropriate software support is often used to achieve low-cost implementation of algorithms. This alternative approach usually provides accuracy very close to that of floating-point hardware. Furthermore, the fixed-point accuracy heavily depends upon the operand values and selection of *q.n* format for fixed-point operations. Small or extremely large values of operands may produce minor errors for complex operations due to the rounding and the truncation of least significant digits [9-11]. However, real-time portable applications emphasize time and power efficient solutions since minor errors in the answer do not often affect the quality of the results.

Fixed-point addition and subtraction operations are relatively easy to implement on an FPGA, while rounding and truncation errors are negligible. The multiplication, division and square-root operations require complex procedures towards accuracy. They often rely on special types of algorithm for embedded system realization. Out of these three operations, the square root is the most complex one [8,12] because it usually involves convergence or approximation algorithms, and thus becomes computationally intensive [4,13-15].

Numerous techniques have been reported in the literature to implement the square root operation on FPGAs [8,16-19]. FPGA implementations provide tradeoffs between general purpose and ASIC solutions. They offer flexibility near that of a general purpose solution and performance closer to an ASIC since they support low-level hardware design; also, GPUs are not capable of such hardware specialization. They can even provide dynamic reconfiguration capabilities for run-time architecture changes. They are cost effective for customized applications and are commonly used for prototyping before ASIC realization [20]. However, end products containing FPGAs are now commonly employed to create time and power efficient designs for real-time portable applications.

It is worth mentioning that RISC microprocessors like the MIPS R4400, SUN Ultra SPARC and Ultra SPARC T_2 have high latency to throughput ratio for the square-root operation [13,15]. Further, it is to be noted that some newer versions of microprocessors avoid floating-point based square-root instructions in their instruction sets in order to make the processor architecture more efficient [15,21,22]. On the other hand, the square-root has a significant role in image and signal processing algorithms [23, 24], and thus it is not advisable to omit it altogether from an instruction set for such application areas.

The Newton-Raphson, SRT-redundant and non-restoring division techniques form the bases of the three main categories of algorithms used for the evaluation of the square root. The Newton-Raphson method has self-correcting capability and provides relatively better precision but requires an initial guess using a seed generator while it converges quadratically [25]. Furthermore, it is based on an iterative approximation technique which requires frequent multiplications so a pipelined implementation of Newton's method requires large multipliers which may not be a feasible option. On the other hand, the SRT-redundant and non-redundant algorithms are both recursive in nature and their implementation is costly especially for higher order radices [12].

A square root implementation involving the non-restoring algorithm is a simple and efficient technique [26] because it employs addition/subtraction and shift operations only. There are many research articles in the literature describing the pipelined implementation of the non-restoring algorithm. Pipelining is usually adopted to make a system time efficient but requires extra effort to resolve timing and data hazards.

In this article we present a pipelined implementation of the square root using the non-restoring algorithm and fixed-point arithmetic. The proposed pipelined architecture avoids data dependence hazards by employing a modified non-restoring division algorithm. This architecture shows a time and power efficient evaluation of the square root compared to floating-point architectures. Furthermore, the proposed pipeline yields a four-fold speedup in the evaluation of the square root and almost doubles the efficiency of a digital communication receiver.

2. Pipelining of Non-Restoring Algorithm

In digital applications, pipelining is the most common parallel scheme used to improve the throughput. A complex task is divided into subtasks which are implemented independently in their respective execution stages. It is required that the output of each subtask become available to the subsequent unit as and when needed. A write-after-read (*WAR*) hazard may arise if a stage in a pipelined system tries to read data from a register written by an instruction that follows in program order.

The non-restoring division technique can be used to evaluate the square root without restoring the remainder of a division [26]. It generates one bit of the result iteratively using two bits of data. Let $D = D_N D_{N-1}\ldots..D_1$ represent an N-bit radicand of an unsigned number. By using the non-restoring algorithm, the square root of D may be represented by $Q = Q_{N/2} Q_{N/2-1}\ldots.Q_1$ after the $N/2^{th}$-iteration. The remainder $R = R_{(N/2)+2} R_{(N/2)+1\ldots} R_1$ has $(N/2)+2$ bits. The relation among R, D and Q is $R = D-Q^2$, where D is less than $(Q+1)^2$ because $R < 2Q+1$. A functional block representation of the non-restoring based square root evaluation is given in Figure-1. The figure shows that there are only addition/subtraction and shift operations involved in the evaluation of the square root.

2.1 Pipelining Hazard

The conventional non-restoring algorithm was implemented for XC2VP2 Xilinx FPGA and the results were simulated in Integrated Simulated Environment (ISE) 8.2 and ModelSim 5.7 XE. Figure-2 shows the post layout simulation for a 2-stage pipelined implementation. Instability in the *q2* signal representing Q_2 was observed, which is associated with the fact that a register is receiving data from a stage while the next stage is trying to read the same register in the same clock cycle (this represents a WAR hazard).

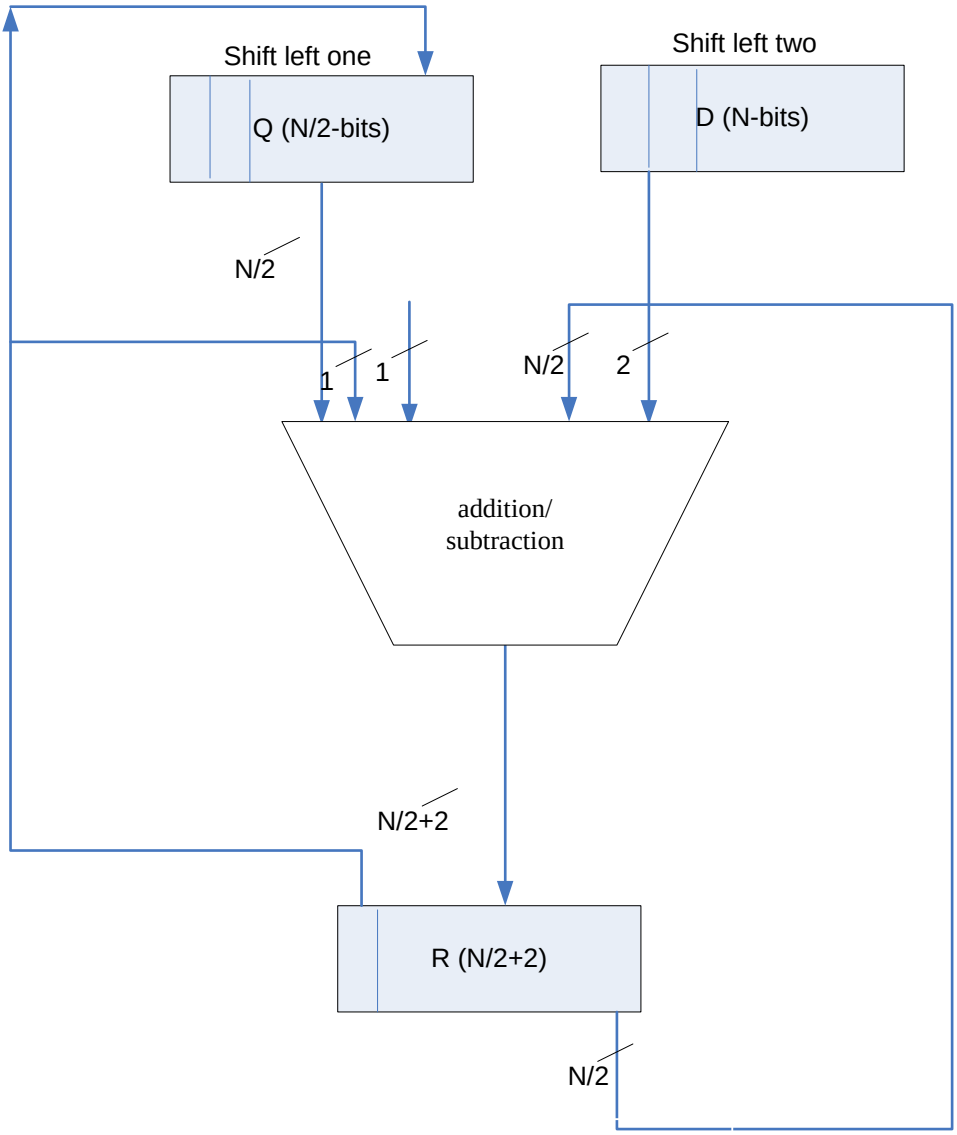


Figure. 1. Block diagram with non-restoring division.

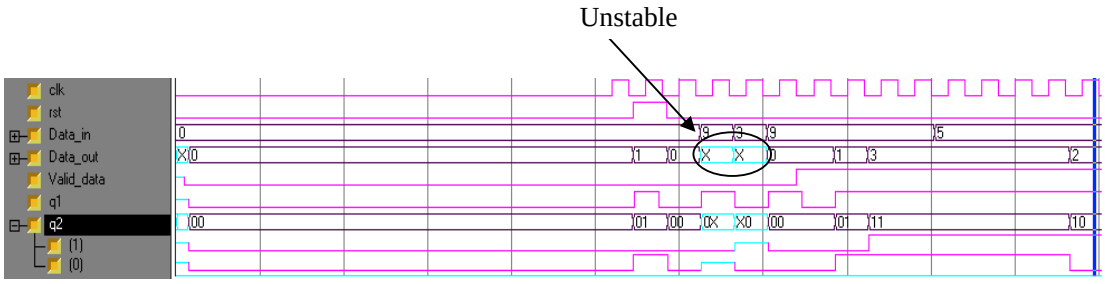


Figure.2. Unstable output due to a WAR hazard for the non-restoring algorithm (pipelined implementation with the XC2VP2 FPGA device).

Let D be an N-bit unsigned integer, Q be a k-bit unsigned integer and R be a (k+2)-bit unsigned integer

Declare $R_1=R_2=.....R_k$ and $Q_1=Q_2=…..Q_k=0$

// (R and Q registers are initially filled with zeros to aviode the junk values).

#Define Con = Concatenation (x,y)

// (In HDL the bits of two variables x and y are concatenated using curly brackets by comma i.e. {x,y})

loop i=1:k

if (i=1)

R_i=Con(k zeros, D[N-2(i-1)-1:N-2(i-1)-2]) - Con(k+1 zeros,1), sel=0

else

sel = R_{i-1}[k-1]

if(sel=0)

R_i=Con(R(i-1)[k-1:0],D[N-2(i-1)-1:N-2(i-1)-2])– Con(Q(i-1),R(i-1) [k-1],1);

Q_i=Con(Q(i-1)[i-1:1], R_i' [k+1]);

else

R_i=Con(R(i-1)[k-1:0],D[N-2(i-1)- 1:N-2(i-1)-2])+Con(Q(i-1) , R(i-1) [k-1],1);

Q_i=Con(Q(i-1)[i-1:1], R_i' [k+1]);

endif

Figure. 3. Modified non-restoring algorithm for pipelined fixed-point based N-bit square root realization.

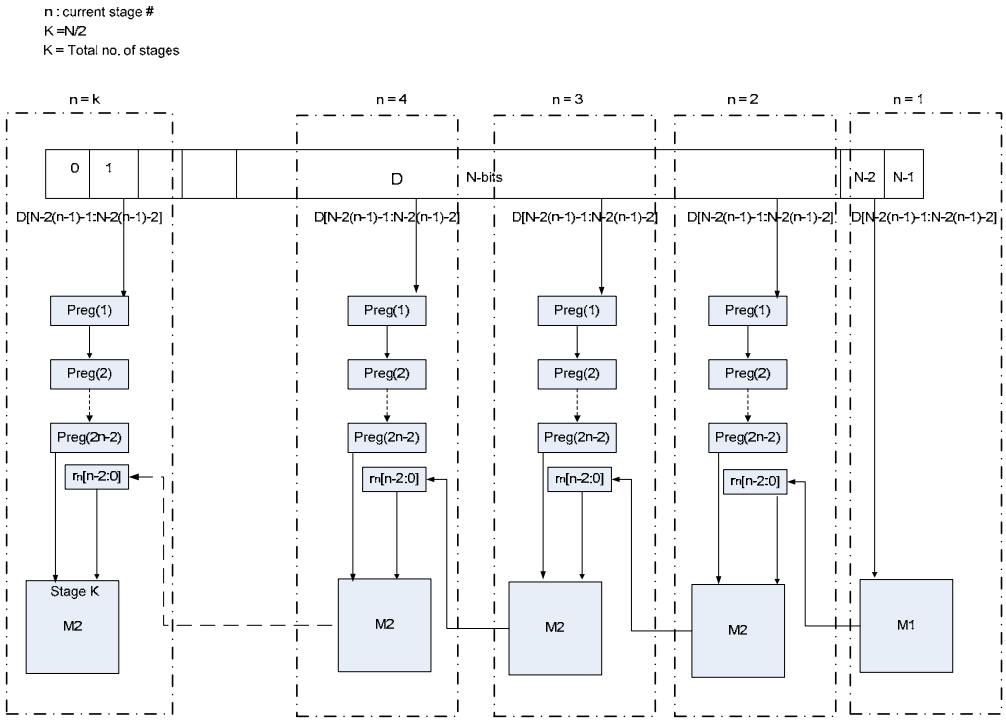


Figure. 4. N-bit pipelined architecture for fixed point based square root implementation.

To overcome this *WAR* hazard, we modified the design as shown in Figure-3. *k* is a variable related to the bit-width of the *R* and *Q* registers, as required by the algorithm. With this modification, the '*if condition'* of the conventional version [4, 12] has been replaced with a simple NOT gate. The shift operation, which is an integral part of the non-restoring algorithm, works well in the non-pipelined implementation where only one *Q* register is required as shown in Figure-1.

In the proposed pipelined implementation, an arrangement has been made for the proper placement of bits in the *Q* register from different wires in the same clock cycle without a shift operation. This provides stable and correct data for subsequent stages in the pipeline without losing clock cycle.

Figure-4 shows a conceptual flow of the pipelined implementation for the modified non-restoring algorithm. However, more *Q* registers for a pipelined implementation are required and their exact number depends upon the number of stages involved in the pipelined system. Two bits are fed to each stage from a *D* input register whose length is equal to *N* radicand bits. In stage-1, the two most significant bits (MSBs) are directly connected to M_1 and there are no pipeline registers. In stage-2, there are three pipeline registers, five pipeline registers in stage-3, and so on.

The circuitry associated with the M-modules of Figure-4 is shown in Figure-5. In this figure, S_n is a computing unit (ALU) and is labeled as S_1 for stage-1, S_2 for stage-2, and so on.

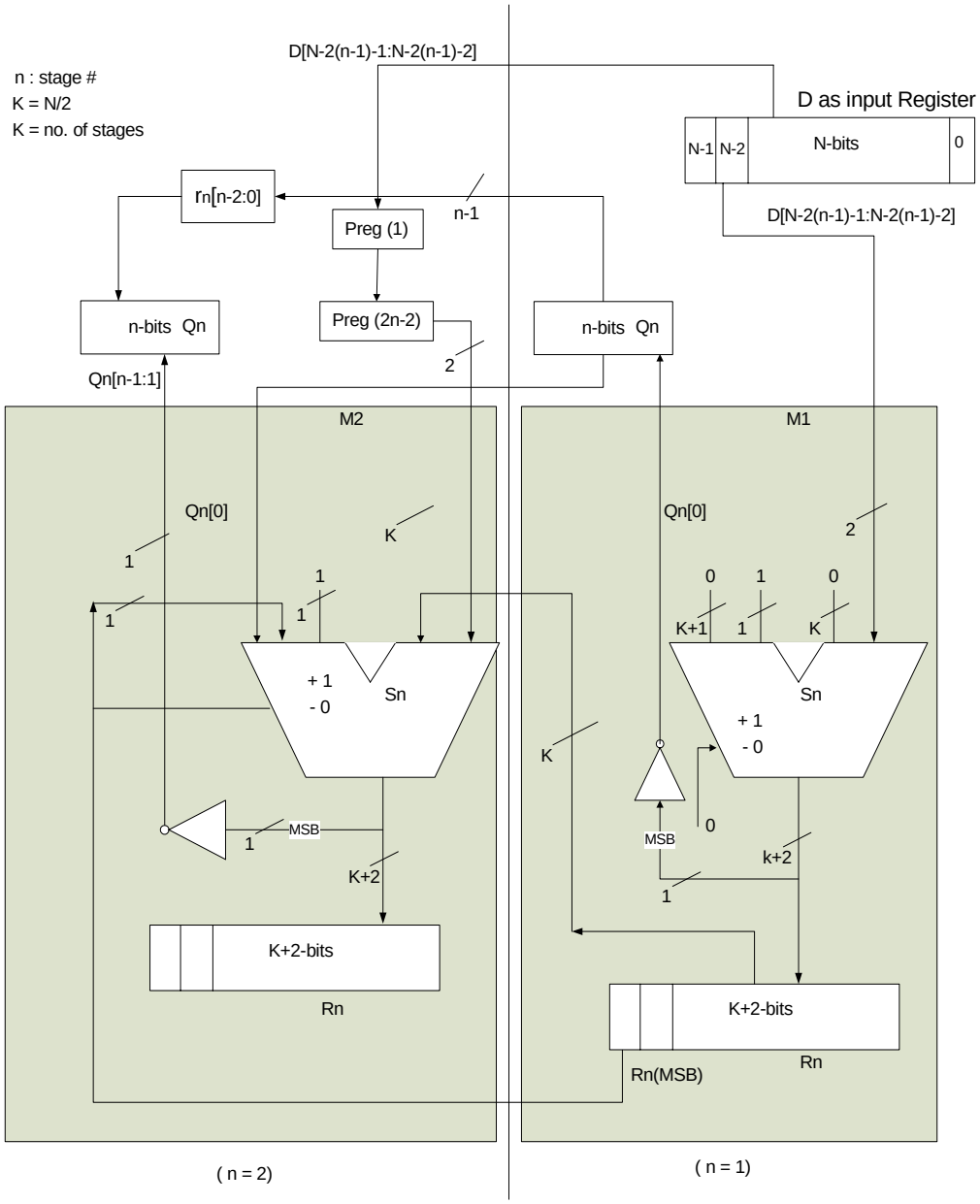


Figure. 5. The details of the M1and M2 modules and their inter-connections.

S_1 gets the two LSB bits from the *D* register as its right operand, while the remaining bits of the left and right operands are initialized with one's and zero's as shown in Figure-5; on the other hand, the LSB of the left operand of S_2 is fixed as one whereas the remaining bits are fed from the previous stage of the pipeline. S_1 is treated as a subtractor in stage-1 because its select bit is always low. S_2 could be a subtractor or adder depending upon the select bit whose value is defined by the MSB of the R_1 register.

To demonstrate the validity of the modified non-restoring realization, the concept in Figure-4 for *N=32* was implemented on XC2VP2 and XC3S100 Xilinx devices. The software tools are ISE 8.2 and ModelSim 5.7 XE. For N = 32, the number of stages defining the pipeline architecture is 16. In Figure-4, *Preg*1&2 act as padding registers while the r_2 receives data from stage-1. For *n* stages in the pipelined architecture, *2n-2 Preg* registers are required plus a register r_n which receives data from the previous stage, where *n* is the stage number. The r_n register is primarily used to store partial results from the Q_{n-1} register of the previous stage and provides the same data to the Q_{n+1} register of the subsequent stage in the pipeline. Therefore, the *WAR* hazard, which is related to read and write timing the Q register in Figure-2, has been solved.

3. Operation

To observe the operations in the proposed pipeline system, one stage at a time, consider Figure-5 with *N=4*. In the first clock cycle (T_1) the two MSB bits of the *D* register are transferred into the LSB positions of the right operand of S_1. The remaining *k* MSBs in the right operand of S_1 are fixed to zeros. The *k* as a variable is equal to half of the radicand bits and *k+1* bits are concatenated with 1 for the left operand of S_1. The result of the above operation is not immediately available from S_1 to R_1 because of the combinational delay in S_1. After that delay, *k+2* bits from S_1 are available in R_1 but not placed in R_1. Furthermore, the MSB of S_1, after passing through an inverter, is available at Q_1 during T_1, but is not placed in Q_1 registers because the sequential units operate in a clock edge-triggered manner. Furthermore, in T_1 the two LSBs from the input *D* register are transferred into the *Preg*(1) which is a part of stage-2.

In T_2, *Preg.*(2) receives data from *Preg.*(1), Q_1 from the inverter and R_1 from S_1 which is available to S_2 and ready for computation because S_2 is a combinational unit. The LSB of the left operand for S_2 is always high and its next bit is defined by the MSB of R_1 (it determines addition or subtraction) after T_2 (because data from *Preg.*(2) is available to S_2).

In T_3, S_2 performs addition or subtraction and the pipeline register r_2 receives data from Q_1. Register r_2 contains just one bit and is part of stage-2. In this cycle, data is not placed in R_2 due to the delay in S_2, and similarly data from r_2 is not available to Q_2. In T_4, the final result will be computed and will become available in Q_2 after receiving data from S_2 and r_2.

Figure-6 shows a dry run of the proposed system. It explains the stage-wise changes in the *R* and *Q* registers per cycle, once the pipeline is filled. The answer is available at Q_4 for 8-bit data. Similarly extending the same logic, the answer for a 32-bit radicand will be available in Q_{16}.

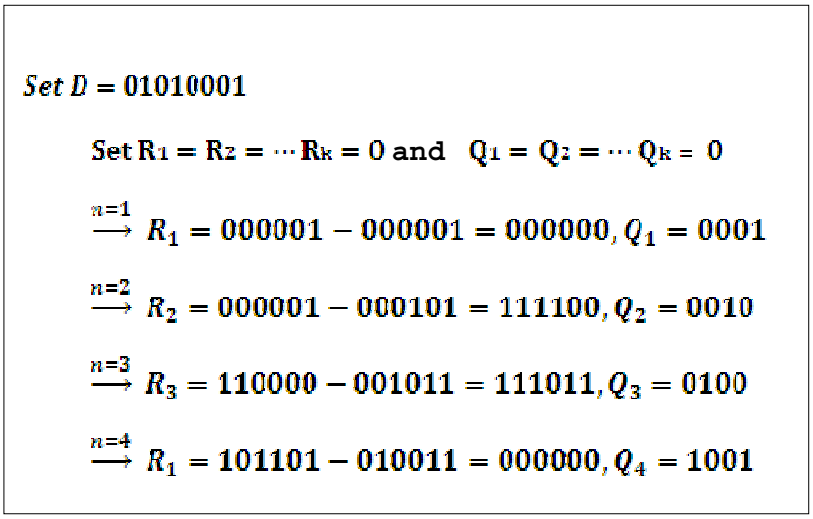


Figure. 6. Dry run for an example using a 4-stage pipelined system (8-bit radicand).

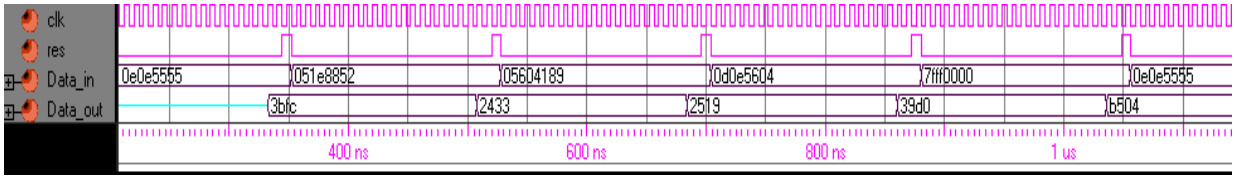
4. Results and Discussion

Figure-7 (a&b) shows post layout results of non-pipelined and pipelined implementations of non-restoring algorithm on a Virtex2Pro (XC2VP2) device. Figure-7(c) represents once again pipelined implementation of the same algorithm on a Spartan3E (XC3S100) device. The post layout simulations shown in Figure-7 were generated after the place-and-route process of the Xilinx tools. In this figure, data_*in* and *data_out* show the input and output data, respectively. To demonstrate the validity of the proposed system, hundreds of randomly selected radicand values for the evaluation of the square root were used in simulations. The same output from various Xilinx devices shows that the proposed architecture is device independent. The total latency of the system was evaluated for various devices. This latency of the modified non-restoring algorithm is about 118 nsec whereas it is 135 nsec on the Virtex2Pro for the classical version, as shown in Figure-7 (a). Therefore, the proposed system demonstrates a 13% improved latency with respect to the classical system.

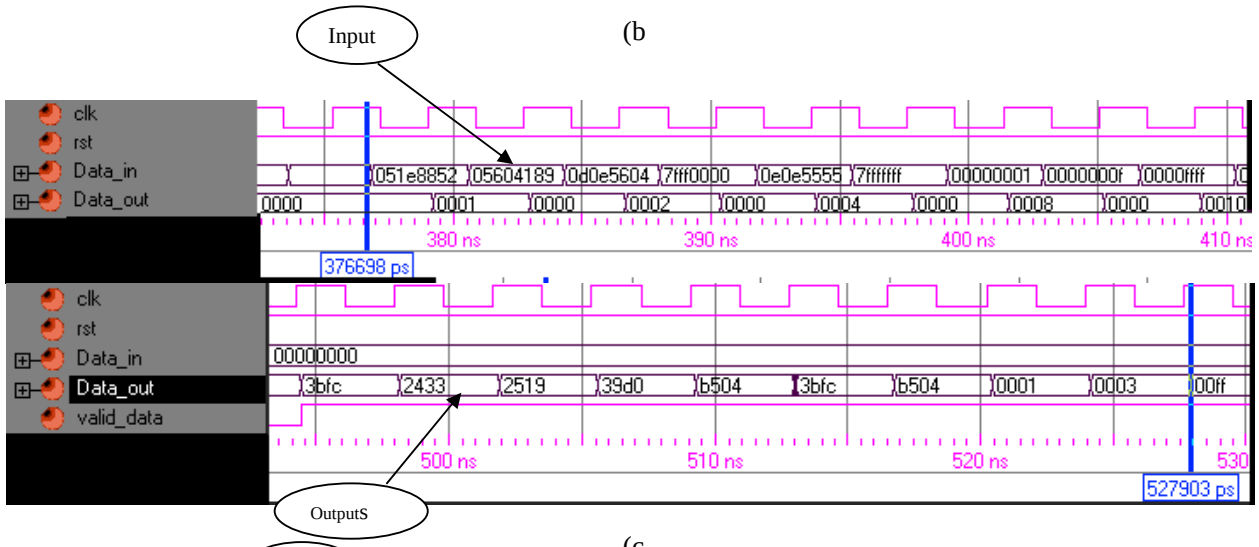
Table-1 shows time to evaluate a square root with non-pipelined and pipelined systems using Virtex2Pro from Xilinx and Flex from Altera. It is useful to consider power consumption along with the execution time because they are reciprocally dependent upon each other.

Various performance metrics have been reported for the evaluation of FPGA-based square root systems, but they are not comparable [18,19]. Throughput (the inverse of the clock period for the slowest pipelined stage) is used as a performance metric. The throughput of a pipelined system is dependent upon number stages. Thus, a system comprising on larger number of stages will require higher resources and as a result it will consume more power.

(a



(b



(c

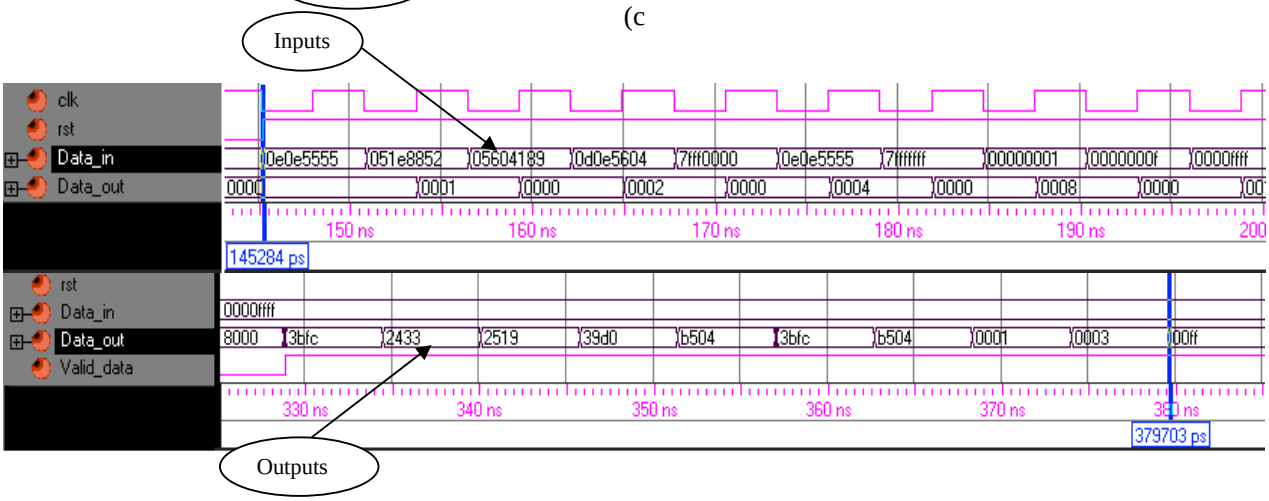


Figure.7. Post layout simulation of the proposed algorithm (a) using a non-pipelined architecture operating at 8 ns on the Virtex2Pro (b) using a pipelined architecture operating at approximately 3.5 ns on the Virtex2Pro (c) using a pipelined architecture at 5.63 ns on the Spartan3E.

Table 1. Throughput of non-pipelined (NP) and the proposed pipelined (PL) architecture.

	Time per square root evaluation (ns)	
Number of operations	**NP**	**PL**
1.00E+00	1.60E+02	1.25E+02
1.00E+01	1.60E+02	1.56E+01
1.00E+02	1.60E+02	4.71E+00
1.00E+03	1.60E+02	3.62E+00
1.00E+04	1.60E+02	3.51E+00
1.00E+05	1.60E+02	3.50E+00
1.00E+06	1.60E+02	3.50E+00
1.00E+07	1.60E+02	3.50E+00

Million floating-point operations per second per slice (MFLOPS/slice) is another relevant metric used to evaluate FPGA based systems. It is pertinent to mention that the power required by a slice of an FPGA is defined by its operating frequency [27]. Thus, the time and power consumption of a system are inter-related for the same FPGA. A benchmark is required which can provide time and power measures simultaneously.

Table-2 illustrate a comparison based on power, execution time and number of stages engaged by an architecture reported in the literature [4,12,14,18,28]. For the proposed system, the power consumption was calculated using the Xpower utility [29], whereas the time required for the same is measured from the post layout simulation. The last column of Table-2 was evaluated by using our proposed benchmark. The data in the fifth column represents the expression $C*S*F$, where S is the number of slices, F is the maximum operating frequency, and C= 0.001475 or 0.001629 is a constant value related to power and deduced from the device data sheet for the Virtex2 and Spartan3E, respectively. The examination of Table-2 reveals that our proposed system produces the minimum value for $C*S*F$ among all square root solutions. Furthermore, in comparison to [18] the great advantage of our proposed pipelined system is that it yields a four-fold speedup; the 28-stage implementation of the former shows a 25% improved C*S*F product whereas its 59-stage solution almost doubles the consumption in comparison to our solution.

Furthermore, the proposed system has been exhaustively tested for accuracy. The maximum error and dB loss in the output, using a wide range of values, was simulated and shown in Table-3. The maximum and average dB losses are 9.20E-04 and 4.77E-4, respectively, compared to floating-point operations whereas the maximum observed error is 6.23E-2. The data in Table-3 show errors on the order of E-05 and E-04 for values less than unity and greater than unity, respectively. Of course, the input value range depends upon the bit-width of the architecture. The maximum input value supported by the proposed architecture is dependent upon the bit-width of the radicand register that takes only unsigned integers. Table-4 presents tradeoffs involving the resource consumption (in number of slices), execution time, mean square error (MSE) and average relative error (ARE) for a 0.02-0.99 data range. This range was chosen due to its special significance in digital communication receivers [1-3]. Table-4 shows that the MSE value is reduced with a larger slice occupancy and, thus the lowest MSE is observed for the 16-stage pipelined system.

Table 2. Performance comparison based on the power consumption and time for fixed and floating-point based systems and various devices (TESQR: Time to evaluate the square root).

Architecture	Stages	Implementation	TESQR(ns)	C*S*F (mw)	Device	Slices/LE	mW*ns
32-bit non-restoring fixed-point [12]	-	Non-pipelined	1170	65.48	Altera 10K20RC240-4	161 (LE)	76611.60
32-bit modified non-restoring fixed-point	-	Non-pipelined	160	11.78	XC2VP2-7	63	1885.58
Single Precision [28]	-	Pipelined	168	1309.19	XC2V1000	1313	219944.30
Single-precision [4]	15-Stages	Pipelined	1.8	6389.70	XC2V4000	4332	11501.46
Double-precision [18]	28-Stages	Pipelined	14.47	152.18	XC2V6000	1433	2202.11
Double-precision [14]	47-Stages	Pipelined	7	364.90	XC2VP2-7	1730	2554.30
Double-precision [18]	59-Stages	Pipelined	7.88	529.59	XC2V6000	2700	4173.19
Fixed-point Modular array, 32-bits [30] [31]	16-Stages	Pipelined	101	30.37	XCV100E-8	312	3067.37
Proposed fixed-point 32-bit	16-Stages	Pipelined	3.50	270.86	Virtex2 XC2VP2-7	709	1002.17
Proposed fixed-point 32-bit	16-Stages	Pipelined	5.63	203	Spartan3E XC3S1600E-5	704	1142.89

Table 3. Maximum error and dB loss of the proposed architecture for different radicand value ranges as compared to pure floating-point implementations.

Range	Max Error	Max dB Loss
0.02 to 0.99	1.53E-05	2.65E-05
1 to 100	3.81E-03	9.20E-04
100 to 1000	3.87E-03	1.24E-04
1000 to 1E+6	6.23E-02	8.39E-04

Carry save adder (CSA) technology was used in the parallel array square root (PASQRT) implementation that can be pipelined without resulting in WAR hazards [4]. Fast execution was achieved at the cost of high resource consumption [32-34]. This is evident in Table-2 since the *C*S*F* value for [4] is very high. Also, a non restoring implementation on a Xilinx FPGA was used to control AC and DC motors [26]. It involved an iterative implementation that is inherently slow and cannot be employed by fast communication systems.

Table 4. Tradeoffs on theVirtex2Pro involving resources, execution time, mean square error (MSE) and average relative error (ARE) for a radicand range from 0.02 to 0.99.

Architecture	Stages	MHz	Slices (out of 1408)	Execution time (ns)	MSE	ARE
32-bit	16	285	709	3.5	9.0E-05	2.91E-05
16-bit	8	310	164	3.2	2.2 E-02	8.08E-03
8-bit	4	368	44	2.7	3.75E-01	1.17E-01
4-bit	2	489	11	2	5.26E-01	1.7E0

In a recent attempt, an ASIC has been developed by employing 90 nm CMOS technology for the evaluation of square root [8]. It has two pipelined architectures simultaneously. As a result, it consumes ~18 % of the total area on the ASIC just for the square root circuit and produces answer in 4 nsec while our approach takes 3.5 nsec that too on an FPGA; an inherently slower device.

Envelope detection is a simple technique to recovery the original signal in a demodulation process of a communications receiver [1-3]. The original signal is extracted by using $y(t)=\sqrt{I^2+Q^2}$, where I and Q are the phase components at time t as shown in Figure-8. In general, the modulation frequency of a communications system is on the order of 100MHz. It implies that a billion operations per signal per second are required. Therefore, an improved throughput for the successive evaluation of the square root can substantially improve the demodulation process and hence the envelope detection. Table-5 shows time and power comparison for a digital communications receiver. Its entries are computed for a 60 sec signal using best floating-point system and our proposed fixed-point architecture.

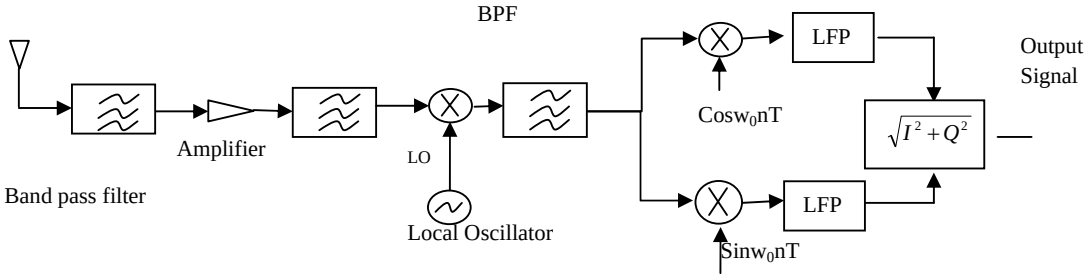


Figure 8. Schematic diagram of a digital signal communications receiver using envelope detection.

The input domain range of radicand values for envelope detection depends upon the application type. The quadrature phase shift keying (QPSK) modulation scheme is often used in cellular communications. The value of the (I^2+Q^2) signal is usually less than unity. For this range of radicands, the MSE, ARE and dB loss values for the proposed system are 9.01E-05, 2.91E-05 and 2.65E-05, respectively, as shown in Table-3 and Table-4.

Table 5. Time comparison to evaluate the square root using purely floating-point (FL) realization and the proposed fixed-point based systems for a digital communications receiver and a minute signal length.

Operating Frequency	Sampling Rate	Time FL SQR (ms)	Time FX SQR (ms)	Power FL SQR (mW)	Power FX SQR (mW)
1 KHz	2 KHz	1.74	0.45	4.23	2.09
1 MHz	2 MHz	1736.4	444	4227.35	2091.55
10 MHz	20 MHz	17364	4440	42273.50	20915.50
36 MHz	72 MHz	62510.40	1598.40	152184.60	75295.80
134 MHz	268 MHz	232677.60	59496	566464.90	280267.70

5. Conclusions

Our proposed pipelined architecture for square root evaluation yields high throughput that can match to the needs of real-time applications. The proposed architecture is reconfigurable to radicand precision, as per user requirements. The accuracy of the system was tested for various ranges of radicand values, resulting in an average loss of 4.77E-4 dB compared to floating-point calculations. Time-power efficiency of the proposed architecture was two times better than the best known floating-point realization of the square root. With this efficiency and accuracy, the proposed system is a viable candidate for digital communications receivers.

References

[1] Sklar, B. (2001). *Digital communications fundamentals and applications* (2nd ed.), Person Education.

[2] Rappaport, T. S. (2002). *Wireless communications principles and pPractice*, (2nd ed.), Prentice Hall Communication.

[3] Frerking, M. (2003). *Digital signal processing in communications systems*, (9th ed.), Kluwer publisher.

[4] Li, Y. & Chu, W. (1997). Implementation of single precision floating point square root on FPGAs, *Proceeding of the 5th IEEE symposium on FPGA-based custom computing Machines*, pp. 226-232.

[5] Liao, J.R. (2000). Real-time image reconstruction for spiral mri using fixed-point calculation, *IEEE Transactions on Medical Imaging*, (19) 7, pp. 690-698.

[6] Oberstar, E. L. (2007). Fixed-point representation & fractional math, http://www.superkits.net/whitepapers.htm.

[7] Sajid, I., Ahmed, M.M., Taj, I., & Humayun M. (2008). Design of high performance fpga based face recognition system, *Proceeding of Progress In Electromagnetic Research Symposium (PIERS) in Cambridge USA*, pp.504-510.

[8] Kwon, T. & Draper, J. (2009). Floating point division and square root using a Taylor-series expansion algorithm, *Microelectronics Journal*, (40) 11, pp. 1601-1605.

[9] Wilkinson J. H. (1962). Error analysis of eigenvalue techniques based on orthogonal transformations, *Journal of the Society for Industrial and Applied Mathematics*, (10) 1, pp. 162-195.

[10] Ortega, J. M. (1963). An error analysis of Householder's method for the symmetric Eigenvalue problem, *Numerische Mathematik*, (5) 1, pp. 1-225.

[11] Burden, R. L. & Faires, J. D. (2005). Numerical analysis, (7th ed.), Thomson.

[12] Piromsopa, K., Arporntewan, C., & Chongstitvatana, P. (2001). An FPGA implementation of a fixed-point square root operation, *Proceedings of the International Symposium on Communications and Information Technology, ChiangMai, Thailand*, pp. 587-589.

[13] Soderquist, P., & Leeser, M. (1996). Area and performance tradeoffs in floating-point divide and square-root implementations, *ACM Computing Surveys*, (28) 3, pp. 518-564.

[14] Ronald, S. & Viktor, P.K. (2004). Computing Lennard-Jones potentials and forces with recogfigurable hardware, *International Conference on Engineering of Reconfigurable Systems and Algorithms*, pp. 284-290.

[15] Soderquist, P. & Leeser, M. (1997). Division and square root choosing the right implementation, *IEEE Micro*, pp. 52-66.

[16] Louca L., Cook, T. A., & Johnson, W. H. (1996). Implementation of IEEE single precision floating point addition and multiplication on FPGAs, *Proceeding of IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press*, pp. 107-116.

[17] Shirazi, N., Walters, A., & Athanas, P. (1995). Quantitative analysis of floating point arithmetic on FPGA based custom computing machines, *Proceeding of IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press*, pp. 155-162.

[18] Thakkar, A. J., & Ejnioui, A. (2006). Design and implementation of double precision floating point division and square root on FPGAs, *IEEE proceeding Aerospace Conference*.

[19] Thakkar, A. J., & Ejnioui, A. (2006). Pipelining of double precision floating point division and square root operations, *Proceedings of the 44th annual Southeast regional conference*, pp. 488-493.

[20] Deschamps, J., Jean, G., & Sutter, G. (2006). *Synthesis of arithmetic circuits FPGA, ASIC, and embedded systems*, John Wiley & Sons.

[21] Power PC processor reference guide, (2003). Embedded development kit.

[22] Oberman, S. F., & Flynn, M. J. (1997). Design issues in division and other floating-point operations, *IEEE Trans. Computers*, (46) 2, pp. 154–161.

[23] Bravo, I., Jimenez, P., Mazo, M., Lazaro, J. L., & Gardel, A. (2006). Implementation in FPGAS of Jacobi Method to Solve the Eigenvalue and Eigenvector Problem, *Proceeding of International Conference on Field Programmable Logic and Applications*, pp. 1-4.

[24] Ray, A. (1998). A survey of CORDIC algorithms for FPGA based computers, *Proceeding of the ACM/SIGDA 6th International Symposium on Field-Programmable Gate Arrays*, pp.191-200.

[25] Karp, A. H., & Markstein, P. (1997). High-precision division and square root, *ACM Transactions on Mathematical Software*, (23) 4, pp. 561-589.

[26] Mailloux, G. J., Simard, S., & Beguenance, R. (2007). Implementaion of division and square root using XSG for FPGA-based vector control drives, *International Journal of Electrical and Power Engineering*, (5) 1, pp. 524-529.

[27] Shang L., Kaviani, A. S., & Bathala, K. (2002). Dynamic power consumption in Virtex-II FPGA family, *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays, USA*, pp. 157-164.

[28] Wang, X., & Nelson, B. E. (2003). Tradeoffs of designing floating point division and square root on Virtex FPGAs, *IEEE Proceeding of Symposium on Field Programmable Custom Computing Machines*, pp. 195-203.

[29] Altera Application note 74, (2001). Evaluating Power for Altera Devices.

[30] Samavi, S., Sadrabadi, A., & Fanian, A. (2008). Modular array structure for non-restoring square root circuit, *Journal of System Architecture*, (54), pp. 957-966.

[31] DS003-1, Virtex 2.5 field programmable gate arrays, 2001.

[32] Li, Y., and Chu, W. (1997). Parallel-array implementations of a non-restoring square root algorithm, *International Conference on Computer Design, Texas, USA*, pp.690–695.

[33] Li, Y., & Chu, W. (1996). A new non-restoring square root algorithm and its VLSI implementations, *International Conference on Computer Design,USA*, pp. 538-544.

[34] Chu, W., & Li,Y. (1999). Cost/performance tradeoff of n-select square root implementations, in: *Fifth Australasian Computer Architecture Conference*, pp. 9–16.

List of Figure Captions

List of Table Captions