Journal of
Electrical & Electronic Systems

Daniel Kho et al., J Electr Electron Syst 2018, 7:4
DOI: 10.4172/2332-0796.1000278

Research Article Open Access

Hardware Implementation of Low-Latency 32-bit Floating-Point Reciprocal

Daniel Kho CK*, Ahmad Fauzi MF and Lim SL

Faculty of Engineering, Multimedia University, Malaysia

Abstract

As the speed requirements of imaging and communications systems increase, the latency requirements of digital circuits also become stringent. Due to such tight latency or timing requirements, large-stage pipelined circuits need to be redesigned to meet the low-latency requirements. Most modern imaging and communications systems rely on digital signal processing (DSP) that compute complex mathematical operations. The emergence of powerful and low-cost field programmable gate array (FPGA) devices with hundreds of arithmetic multipliers has enabled many such DSP hardware applications, traditionally implemented only as software solutions. The reciprocal square root algorithm is a popular technique for computing square roots, used widely in many software applications. This paper shows how this algorithm can be implemented efficiently on hard-ware, and is suitable for low-latency mathematically-intensive applications. Using a low-cost FPGA device, the algorithm takes up less than 1000 look-up-tables (LUTs), which on an Artix XC7A200T device, translates to less than 1% of all the LUT resources in the chip.

Keywords: Reciprocal; Square root; Inverse square root; Non-restoring; FPGA; ASIC; VHDL

Introduction

Many signal processing and imaging applications de-pend on the computation of non-linear equations. The square root is one such non-linear operation that is used by such applications. In image and video processing, the square root is used to calculate the magnitude of the gradient of an image, root mean square error, vector normalisation, among others.

Most square root computations are expensive and takes many iterations to complete. In hardware, these iterations translate to clock cycles. A highly-iterative operation such as those using Newton-Raphson algorithms will require a high-latency hardware circuit. Therefore, such hardware are not suitable to be used in low-latency applications such as high-speed communications and image processing. With an already high system clock frequency, increasing the frequency even further will most likely result in timing issues on hardware.

Pipelining the circuit just partially works around this problem. Due to fast-changing input data, the result from the computation of a highly-iterative high-latency circuit could be available only after the next request is received. Each request to compute usually comes with a new set of data. These fast changes in the input data while the computation is still in progress may corrupt the final result which is yet to be computed. To work around this problem, more storage logic is being added to buffer the incoming data, which in turn increases the area and power consumption, slows down the entire circuit, and increases the complexity unnecessarily Figure 1 illustrates this problem in a timing diagram. This problem is compounded if the result from the computation is available only after two or more such requests is received.

Square Root Algorithms

There are several techniques to compute the square root. The floating-point square root based on Taylor series expansion using lookup tables and multipliers is discussed in detail by Wang [1] and Ercegovac [2] et al. Iştoan [3] de-scribes several techniques for fixed-point implementations of the square root. Li and Chu [4] describes the FPGA im-plementation of the single-precision floating-point square root, while Nanhe et al. [5] describes both fixed- and floating-point square root implementations. These square root implementations are based on concepts such as the non-restoring algorithm [4], tabulate-and-multiply, bipartite, and Taylor series expansion [1-3]. The Newton-Raphson iterations is a common technique to improve the precision of the result, and usually acts as the final step in many of these techniques, including for the method described in this paper.

This paper uses the techniques described by Lomont [6] and Robertson [7]. However, most of these techniques were implemented as software algorithms. Zafar S [8] presented a similar FPGA-based implementation which took 12 clock cycles to complete the processing. Here, we present a low-latency FPGA-based hardware implementation which has only 3-cycle latency. This algorithm involves multiplication, bit-shifts, and subtraction from a "magic" constant. This magic constant was algebraically derived by Robertson, which confirms the same value computed numerically by Lomont in his earlier paper.

Our magic constant also uses the value 0x5f375a86 rather than 0x5f3759df as was used in Zafar's paper. The constant 0x5f3759df was originally used in the Quake III program, and was proven by these authors as being less accurate compared with using new constant. In our paper, we shall refer to this algorithm as the fast reciprocal square root (FRSR).

The Newton-Raphson iterations

The Newton-Raphson method is an iterative technique that takes

***Corresponding author:** Daniel Kho CK, Faculty of Engineering, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia, Tel: 60163330498; E-mail: daniel.kho@synvue.com

Received October 04, 2018; **Accepted** October 25, 2018; **Published** November 01, 2018

Citation: Daniel Kho CK, Ahmad Fauzi MF, Lim SL (2018) Hardware Implementation of Low-Latency 32-bit Floating-Point Reciprocal. J Electr Electron Syst 7: 278. doi: 10.4172/2332-0796.1000278

a current approximation, y_n, and returns a new approximation y_{n+1}, aimed at estimating the final result more accurately. Equation 1 shows how a new estimate is being computed.

$$y_{n+1} = y_n - \frac{f(y_n)}{\dot{f}(y_n)} \tag{1}$$

Where $f(y) = 0$ is the function to be approximated, and $\dot{f}$ is the first-order derivative of f.

Performing this iteration on a reciprocal square root function $y = \frac{1}{\sqrt{h}}$ gives

$$f(y) = \frac{1}{y^2} - h \tag{2}$$

which yields the estimate y_{n+1} as

$$y_{n+1} = \frac{y_n}{2}(3 - hy_n^2) \tag{3}$$

Equation 3 will later be computed in floating-point hardware.

The Reciprocal Square Root

In the IEEE 754 floating-point system, a 32-bit single-precision floating-point number f can be physically represented as shown in Figure 2. The representation consists of three parts: the sign bit s, the exponent field q, and the mantissa field M.

We can write f in equation form, as shown in Equation 4.

$$f = (-1)^8 .m.2^{q-127} \tag{4}$$

Where s denotes the sign and indicates whether the mantissa, is even or odd, m denotes the normalised mantissa, and 2^{q-127} denotes the biased exponent. The normalised mantissa component is calculated from the mantissa field represented physically as shown in Figure 2.

$$m = 1 + \frac{M}{2^{23}} \tag{5}$$

The reciprocal square root of f can be written as

$$\frac{1}{\sqrt{f}} = \frac{1}{\sqrt{m}} \frac{1}{\sqrt{2^{q-127}}} \frac{1}{\sqrt{2^{q0}}} = \frac{(\sqrt{2})^{1-q0}}{\sqrt{m}} 2^{63-\frac{q}{2}} \tag{6}$$

Let $zq0$ represent the mantissa field of Equation 6

$$z_{q0} = \frac{(\sqrt{2})^{1-q0}}{\sqrt{m}} \tag{7}$$

When q is even, $q0 = 0$. We have

$$z_0 = \frac{\sqrt{2}}{\sqrt{m}} = \sqrt{\frac{2}{m}} \tag{8}$$

When q is odd, $q0 = 1$, giving us

$$z_1 = \frac{1}{\sqrt{m}} \tag{9}$$

Depending on whether the physical mantissa M is large or small, the estimate $\hat{y}$ can be found [6]. Equation10 estimates the mantissa field of $\hat{y}(x)$. Because this approximation works for all valid exponents, $\hat{y}(x)$ can be considered the general approximation for the reciprocal square root $1/\sqrt{x}$ without needing different approximations for the mantissa and the exponent fields of a float.

$$\hat{y}(x) = \begin{cases} \sqrt{2}\left(\frac{3}{4} + \frac{t}{2} - \frac{x}{4}\right), q0 = 0, x \le 1 + 2t \\ \sqrt{2}\left(\frac{5}{8} + \frac{t}{4} - \frac{x}{8}\right), q0 = 0, x > 1 + 2t \\ 1 + \frac{t}{2} - \frac{x}{4}, \quad q0 = 1 \end{cases} \tag{10}$$

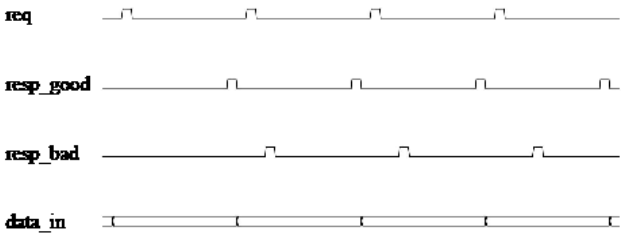

Figure 1: Example timing issue of a high-latency processing block in a high-speed circuit.

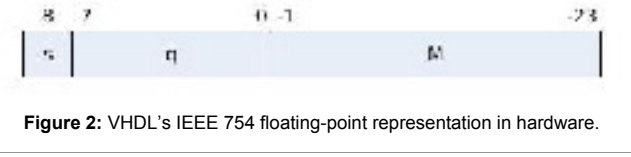

Figure 2: VHDL's IEEE 754 floating-point representation in hardware.

The Newton-Raphson step of Equation 3 can be applied to $\hat{y}(x)$ to give

$$\hat{y}_{n+1}(x) = \hat{y}_n(x)\left(\frac{3}{2} - \frac{x}{2}\hat{y}_n(x)\right) \tag{11}$$

where $x \in [1,2)$

The estimate $\hat{y}$ is then minimised to give the least possible error relative to the actual reciprocal square root function $1/\sqrt{x}$. The maximum relative error for $\hat{y}$ can be expressed as

$$e(x) = \left|\frac{\hat{y}(x)}{1/\sqrt{x}} - 1\right| \tag{12}$$

where $x = f(t)$.

Robertson and Lomont [6,7] notes that the optimal t value corresponds to the floating-point constant value of 0x5f375a86. This "magic" constant, and earlier approximations of this constant, is being used frequently in many software algorithms that require fast calculations of the reciprocal square root, such as computer graphics. Our proposed hardware implementation also makes use of this constant.

Hardware Architectures of the Fast Reciprocal Square Root

Low-cost Output-Registered Implementation of the FRSR

Figure 3 shows the proposed hardware architecture of our reciprocal square root algorithm. The "magic" constant is implemented as pull-ups and pull-downs in hardware. The input data n is bit shifted by one bit to the right, before being subtracted from the "magic" constant. The bit-shift operation in hardware just requires rewiring the logic to access the appropriate bits. Accessing a register set starting from the first index instead of the zeroth index essentially gives a shift-right-by-one result. The output of the subtractor provides a very good initial guess of the reciprocal square root of n.

If the Newton-Raphson iterations are enabled, the result from the subtractor is multiplied with itself to produce the square of itself, with the input n, and with a floating-point constant 0.5f. The multiplication is performed with floating-point inputs. Depending upon the speed of the target hardware, the multipliers may or may not be pipelined.

The result from this multiplication is then fed to another subtractor, where this result is subtracted from another floating-point constant 1.5f. The output of this subtractor is then multiplied with the result from the first subtractor to produce the reciprocal square root estimate. This result may be looped back to become the new value of *n* if more iterations are required. Finally, the reciprocal square root result can also be multiplied by the input *n* to produce the square root estimate. This final result can be clocked by a flip-flop before being read by other logic.

Pipelined implementation of the FRSR

The same circuit of Figure 3 can also be pipelined if some of the computational elements are taking a long time to complete. An example implementation is shown in Figure 4. In our case, we were using a low-cost Artix 7 FPGA which requires the use of Xilinx's floating-point multiplier core. The simulation of the multi-stage pipelined implementation is shown in Figure 5.

Results

The latency of our simple low-cost output-registered FRSR is only three clock cycles, mainly contributed by the floating-point subtraction core. Running on a 50-MHz clock, we managed to achieve a timing margin of 1.65ns. Our implementation was designed with the VHDL language, and simulated with Mentor Graphics Modelsim. We have implemented the design on a low-cost Xilinx Artix 7 XC7A200TFBG676 FPGA device, using Xilinx Vivado's default synthesis and place-and-route settings. The software version used was Vivado 2017.4.

In the case where the input *n* is a 32-bit unsigned integer, as is the case for our image processing application, the Newton-Raphson iterations can be omitted while still maintaining an accuracy of 0.3. This is especially useful in applications where accuracy is not as important as speed, and a rough estimate of the reciprocal square root is sufficient. If more precision is required, the final output can still be fed back to the input *n* to go through a couple more iterative cycles. An accuracy of 1.6% (0.016) can be achieved for unsigned integer inputs after one Newton-Raphson iteration, while 0.01% (0.0001) accuracy can be achieved after two iterations. The design can also accept floating-point input, where the accuracy would follow the ones reported by Lomont and Robertson [6,7].

Functional simulations and hardware synthesis

Figure 5 shows the simulations of our FRSR implementations. For basic verification on hardware, we used Xilinx's ChipScope integrated logic analyzer to acquire real-time waveforms from our development board.

Table 1 shows the resource utilisation of our implementation compared against previous implementations. In our paper, we have implemented our floating-point algorithm using the 32-bit IEEE single-precision format 32(8,23). Wang [1] noted in her 2007 paper that her 32-bit floating-point implementation takes up 351 slices (LUTs) (1%), 3 block RAMs (2%), and 9 embedded multipliers (6%), on a Xilinx Virtex II XC2V6000 FPGA. Now, with modern FPGAs packing lots of resources, our implementation takes up only 635 LUTs which translates to a mere 0.47% of a low-cost Artix 7 XC7A200TFBG676 FPGA. Our implementation does not use any block RAMs since there is no need for a look-up table in our square root implementation. We use 12 DSP multipliers, which on a modern low-cost FPGA such as the Artix 7, translates to only 1.62% of the entire DSP resources within the FPGA.

Our selection of DSPs and LUTs when generating the floating-point cores enable us to achieve very low latency in our design. The only delay was contributed by the three-cycle latency from the floating-point subtraction. Table 2 show the latency of the design, compared against other implementations or algorithms.

Discussion

Many digital signal processing (DSP) and digital imaging hardware avoid implementing the square root function due to its perceived complexity [9-14]. As the demand for faster processing increases, latency is also becoming an issue that increasingly needs to be resolved. A low-latency reciprocal square root implementation on hardware could help many computationally-intensive DSP hardware main-tain its precision while also complete computations in the fewest clock cycles possible. We have presented a feasible and inexpensive square root and reciprocal square root circuit suitable for implementation in low-latency FPGA or ASIC hardware.

The FRSR algorithm has been traditionally used in software requiring high-speed computations of the square root, such as graphics-intensive video games. Li and Chu [4] have shown two hardware implementations of their floating-point non-restoring square root algorithms, whose performance results have been highly cited. The first low-cost iterative version, requires 25 clock cycles (latency) for the result to be computed. The second pipelined version, requires 15 clock cycles for the result to be computed. For the pipelined version, it is mentioned the issue rate is 1 clock cycle, which means new data can be issued as

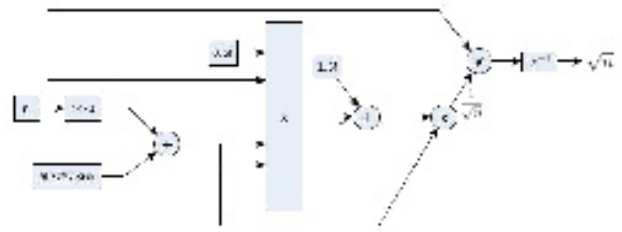

Figure 3: Simple circuit of the FRSR algorithm.

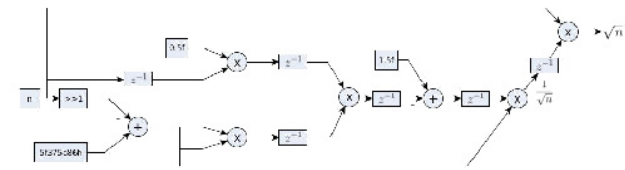

Figure 4: Pipelined example of the FRSR algorithm.

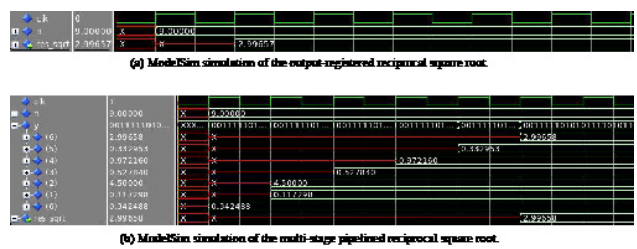

Figure 5: Simulation results for output-registered and pipelined versions of reciprocal square root.

Implementation	LUTs	FFs	DSP
Y. Li & W. Chu (Iterative) [4]	82	138	0
Y. Li & W. Chu (Pipelined) [4]	408	675	0
X. Wang [9]	351	-	9
S. Zafar [5]	1939	748	2
Output-registered FRSR	635	82	12
(1 N-R iteration)	(0.47%)	(0.03%)	(1.62%)

*Percentage values are measured with respect to the Xilinx Artix 7 XC7A200TFBG676 FPGA on the AC701 board.

Table 1: Resource utilization of fast reciprocal square root.

Implementation	Latency
Y. Li & W. Chu (Iterative) [4]	25
Y. Li & W. Chu (Pipelined) [4]	15
M. Iştoan [7]	15 to 23
X. Wang [9]	13
S. Zafar [5]	12
Output-registered FRSR	3

*Only data using 32-bit fixed and single-precision 32(8,23) floating-point formats are shown.

Table 2: Latency comparison.

well as new results are available every clock cycle, although there is still a 15-cycle delay between an input data and its corresponding result. Zafar and Adapa [8] presented an implementation of their reciprocal square root algorithm based on the fast inverse square root algorithm as well, however, their implementation requires 12 clock cycles to complete [15-17].

As we apply this algorithm on FPGA hardware, we had to ensure that the floating-point FPGA cores were inferred correctly by the tool. This was done by characterising the functionality of the floating-point cores in the laboratory. We observed incorrect physical behaviour of the floating-point basic calculations when we synthesised our design by allowing the synthesis tool to automatically infer the floating-point cores from the standard VHDL-2008 floating-point library. After directly instantiating the Xilinx floating-point cores, correct behaviour was observed in the lab. The results presented are using the directly-instantiated cores which exhibit the intended behaviour [18-20].

Although the VHDL language provides very useful fixed- and floating-point packages suitable for DSP-heavy applications, special attention needs to be made to ensure that these packages are supported by tool vendors. A successful synthesis that produces no errors cannot be assumed to exhibit the correct behaviour, and careful lab characterisation is necessary to ensure whether or not the tools are synthesising these packages correctly [21].

Conclusion

The fast reciprocal square root algorithm, popularly used in many software applications, has been shown to be a good alternative for low-latency hardware. While there is a slight increase in logic resources and area, this increase is negligible as the density of modern FPGAs keeps the utilisation percentage for LUTs below 1% for a low-cost Artix device. Although Zafar and Adapa is using a similar algorithm as that shown in our paper, the solution presented was shown to be more expensive and complex with the usage of 1939 LUTs. We have shown that it is feasible and cost-effective to implement the same FRSR algorithm on hardware with much less logic resources (635 LUTs), and also with much lower latency (3 clock cycles).

Acknowledgment

The authors are thankful to the Ministry of Higher Education Malaysia for the award of Fundamental Re-search Grant Scheme FRGS/1/2015/TK04/MMU/02/10 to support this project. Also, we are thankful to Richard Joseph, Jeannie Lau, and Ang Boon Chong for the many technical discussions that helped us complete this project in one way or another.

References

1. Wang X (2007) Variable Precision Floating-Point Divide and Square Root for Efficient FPGA Implementation of Image and Signal Processing Algorithms. Northeastern University, Massachusetts.
2. Ercegovac MD, Lang T, Muller JM, Tisserand A (2000) Reciprocation, square root, inverse square root, and some elementary functions using small multipliers. IEEE Trans Comput 49: 628-637.
3. Iştoan M, Pasca B (2015) Fixed-Point Implementations of the Reciprocal, Square Root and Reciprocal Square Root Functions. HAL Id: hal-01229538.
4. Li Y, Chu W (1997) Implementation of single precision float-ing point square root on FPGAs. IEEE Symp on Field-Programmable Custom Computing Machines, Napa, California, USA, pp: 226-232.
5. Nanhe A, Gawali G, Ahire S, Sivasankaran K (2013) Implementation of Fixed and Floating Point Square Root Using Non-restoring Algorithm on FPGA. IJCEE 5: 533-537.
6. Lomont C (2003) Fast Inverse Square Root. West Lafayette, Indiana.
7. Robertson M (2012) A Brief History of InvSqrt. University of New Brunswick.
8. Zafar S, Adapa R (2014) Hardware architecture design and map-ping of 'Fast Inverse Square Root' algorithm. ICAEE, pp: 1-4.
9. Kanjar De, Masilamani V (2013) A New No-Reference Image Quality Measure for Blurred Images in Spatial Domain. Int J Image Graph 1: 39-42.
10. Kanjar De, Masilamani V (2013) Image Sharpness Measure for Blurred Images in Frequency Domain. Int Conf Design Manufacturing.
11. Kanjar De, Masilamani V (2017) Image Quality Assessment for Blurred Images Using Nonsubsampled Contourlet Transform Features. J Computers 12: 156-164.
12. Halder A, Chatterjee N, Kar A, Pal S, Pramaniket S (2011) Edge Detection: A Statistical Approach. Int Conf Electron Comput Tech (ICECT).
13. Umar A, Li H, Aguirre A, Zhu Q (2012) FPGA-based reconfigurable processor for ultrafast interlaced ultrasound and photoacoustic imaging. IEEE Trans Ultrason Ferroelectr Freq Control 59: 1344-1353.
14. Nelson AE (2000) Implementation of image processing algorithms on FPGA hardware. Vanderbilt University.
15. Ercegovac MD, Muller JM, Tisserand (2005) A Simple Seed Architectures for Reciprocal and Square Root Reciprocal. IEEE.
16. Sajid I, Ahmed MM, Ziavras SG (2010) Pipelined imple-mentation of fixed point square root in FPGA using modified non-restoring algorithm. ICCAE, pp: 226-230.
17. Ananthalakshmi AV, Sudha GF (2017) Design of a re-versible floating-point square root using modified non-restoring algorithm. Microprocess Microsyst 50: 39-53.
18. Lachowicz S (2008) Fast Evaluation of the Square Root and Other Nonlinear Functions in FPGA. IEEE Int Symp Electronic Design Test Appl.
19. Gonzalez RC, Woods RE (2002) Digital Image Processing. 2nd Ed. Prentice-Hall.
20. Proakis JG (1995) Digital communications. McGraw-Hill, New York.
21. Ashenden PJ (2010) The Designer's Guide to VHDL, Volume 3, 3rd Ed. Morgan Kaufmann.