Fast Inverse Square Root using FPGA

Abhay Chopde, Sharvari Bodas, Varada Deshmukh, Shamish Bramhekar

Department of Electronics and Telecommunication, Vishwakarma Institute of Technology, Pune, India

Corresponding author: Sharvari Bodas, Email: sharvari.bodas21@vit.edu

The Fast Inverse Square Root (FISR) algorithm, originally introduced in the Quake III source code, accomplishes the vector normalization task required in graphics application through basic multiplication and bit-shifting operations. The core of this algorithm relies on the use of approximation techniques to enhance an initial estimation, which is primarily based on a designated "magic" constant. The implemented Verilog code utilizes the Newton-Raphson iterations, modified booth's multiplier, and the inverse square root, featuring a core "Inverse Square Root" module with 32-bit input and output. This paper makes use of two magic constants "0x5f3e34bc" and "0x5f3759df" aiming to improve the accuracy. It selects a magic constant based on the exponent bit. The approximation occurs through two Newton-Raphson iterations. A Booth Multiplier is used, that is using a Radix-4 encoding scheme to reduce partial product generation, making it faster.

Keywords: FPGA, Fast inverse square root, Radix 4 Booth's multiplier.

in: *Advancements in Communication and Systems*. Ed. by Ashish Kumar Tripathi and Vivek Shrivastava. Computing and Intelligent Systems, SCRS, India., 2023, pp. 231–239. DOI: https://doi.org/10.56155/978-81-955020-7-3-21

1. Introduction

In the field of computer 3D graphics, and scientific computing, accurate representation of real numbers in computers often requires the use of floating-point arithmetic. One common task is the calculation of inverse square root, which is typically implemented through iterative algorithms with initial approximations obtained from lookup tables or magic constants [1]. As a result, it makes it computationally difficult as more divisions as well as direct square root operation is needed which makes more usage of hardware and hence more utilization cost. In most of the cases due to this issue, accuracy might be compromised. This article focuses on fast inverse square root, aiming to reduce relative errors and improve accuracy, while also minimizing the number of required multiplication operations. The computation of the inverse square root function, a common operation in the gaming industry, is crucial for tasks such as vector normalization in computer graphics and ray tracing. Modern games involve millions of these calculations per second, emphasizing the need for an optimized approach. The Fast Inverse Square Root algorithm, initially introduced in Quake III's source code, stands out for its efficiency, as it approximates the inverse square root with only one division operation. Despite its origins remaining unclear and the absence of formal literature discussing it, various attempts have been made to understand its mathematics and derive improved "magic constants" for enhanced accuracy and computational speed.

The Fast Inverse Square Root algorithm is well-known for its speed and clever use of special numbers and bit operations. Our project aims to go even faster and more accurately by using two iterations of the Newton-Raphson method, a new way to compute the inverse square root that should be very efficient and precise. In addition to the two iterations of the Newton-Raphson method we have also designed a set of modular components for addition, subtraction and multiplication to minimize latency and maximize throughput, ensuring that our algorithm operates with the utmost efficiency. Additionally, at the core of our project to optimize arithmetic, we use the Modified Booth's Algorithm. It's a widely recognized method that makes multiplication more efficient by cutting down on the number of bit changes and time needed for each calculation. The core of Fast Inverse Square Root uses two magic constants to first calculate the approximate inverse square root. By including this technique in our project, our main goal is to provide an improved and thorough solution for calculating the inverse square root, with a strong focus on speed and accuracy. The hardware versions of the algorithm use special hard- ware like Field Programmable Gate Arrays (FPGAs) to make the calculations much faster. It was first created in 2007 and is used in inexpensive embedded systems and specialized hardware parts in CPUs and GPUs, mainly for graphics tasks.

2. Literature Review

The 'Fast Inverse Square Root' algorithm, often denoted as 0x5f3759df, is an enigmatic method for rapidly calculating the reciprocal square root ($y = 1/\sqrt{x}$). Although its origins remain mysterious, it outperforms previous approaches by approximately fourfold and was first noticed in the Quake III source code before being utilized in game engines. This paper focuses on implementing the algorithm on an FPGA, emphasizing parallelism to enhance performance, making it one of the few attempts to create a hardware equivalent of this algorithm [2].

In this paper, the authors have introduced two adjustments to the well-known InvSqrt algorithm for efficient inverse square root computation. The first, named InvSqrt1, retains the original "magic" constant but enhances accuracy through modifications to the Newton-Raphson method, slightly increasing computational cost while improving precision by a factor of two or seven after one or two iterations, respectively. The second, InvSqrt2, employs a different "magic" constant with computational costs similar to InvSqrt but achieving accuracy akin to that of InvSqrt1. These modifications may prove beneficial in applications like embedded systems, microcontrollers without a Floating-Point Unit (FPU), and potentially FPGA-based solutions [3].

In this paper, the authors have introduced a modified version of the widely known InvSqrt algorithm, tailored for the swift computation of the inverse square root in single-precision floating-point numbers. The revised code retained the same "magic" constant but underwent alterations in its second segment, which involves Newton- Raphson iterations. With just one iteration, the new code, InvSqrt1, maintained a similar computational cost to the original but delivered twice the level of accuracy. When two iterations were employed, the computational overhead increased marginally, while accuracy improved nearly sevenfold [4].

The method introduced allows for the efficient computation of elementary functions such as reciprocals, square roots, inverse square roots, logarithms, and exponentials through a unified approach. It offers competitive computational performance for reciprocal operations, while excelling in the calculation of square roots and inverse square roots, especially in double precision. Utilizing fixed-point arithmetic and table look-ups, it provides accuracy and versatility across a range of elementary functions [5].

In this paper, the system introduces an efficient architecture for calculating square root and inverse square root values in IEEE 754 single-precision floating-point numbers. It employs a modified Quake's algorithm with the Newton-Raphson method and utilizes a lookup table of carefully chosen magic numbers, implemented in VHDL and synthesized on a Xilinx Virtex 5 FPGA. This innovative design simultaneously computes both square root and inverse square root with high precision in just twelve clock cycles, offering significant value for numerical computing applications [6].

Scaling-less fixed-point Newton-Raphson implementation for an inverse square root operator, enabling size-generic and ready-to-use IP core integration. The design is pipelined for high-frequency processing and eliminates the need for memory blocks. It outperforms memory-based architectures, offering flexibility and efficiency for high-throughput digital signal processing applications [7].

This paper details a hardware implementation of the Newton-Raphson rounding algorithm, which is used for division and square root calculations. The hardware consists of a recode circuit to encode the multiplier into a redundant binary representation, a direct rounding mechanism for handling IEEE 754 rounding modes, and a sticky digit generation circuit for error compensation. These components work in tandem to ensure accurate and efficient floating-point calculations while using the Newton-Raphson method [8].

This paper presents a scaling-less fixed-point Newton-Raphson implementation for an inverse square root operator, eliminating the need for mandatory scaling. It offers a versatile and resource-efficient IP core that adapts the first approximation based on a Leading One Detector and logical operations. The fully-pipelined design achieves a high clock frequency, making it ideal for diverse digital signal processing applications and is available as an open-source project to facilitate accessibility and further enhancements [9].

The paper introduces a fast method for calculation of inverse square roots. This approach uses a modified Newton-Raphson iteration which minimizes delay as well as power consumption. It significantly reduces memory and area requirements compared to previous methods, making it ideal for applications involving IEEE single precision floating-point numbers and benefiting fields like 3D graphics [10].

The paper delves into optimizing the InvSqrt algorithm, used for calculating inverse square roots in applications such as 3D graphics and hardware implementations. It investigates the selection of a "magic constant" (R) and offers mathematical insights into the code, highlighting the impact of different magic constants on algorithm accuracy. The study introduces optimal magic constants and their adaptability, which can enhance the precision and performance of the algorithm in various contexts, particularly in floating-point hardware components [11].

This paper presents an enhanced design for Modified Booth Encoder (MBE) multipliers, crucial in Digital Signal Processing (DSP) applications. The improved MBE architecture employs a sophisticated modified Booth encoder and selector to stream- line and minimize partial products, eliminating the need for an extra row in the multiplier array. Additionally, a novel hybrid two's complementation logic leverages input signal arrival time disparities, resulting in substantial reductions in area and power consumption compared to conventional and alternative designs [12].

3. Methodology

Verilog code has been implemented for the calculation of the inverse square root using the iterative Newton-Raphson method. The "InverseSquareRoot" module is at the core of this computation, taking a 32-bit integer input, and producing a 32-bit output, which represents the inverse square root of the input. To achieve this, the code that we have implemented, relies on registers and constants to store vital values and constants. Right bit shift operation is performed to half the number and it will be used in calculations with magic constant. The implemented code extracts the exponent bit.

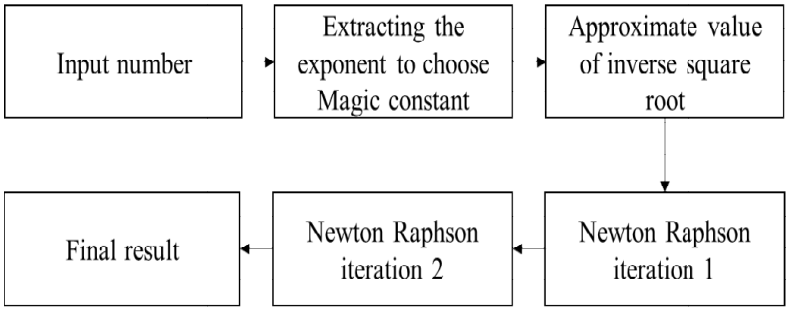

Figure 1. System architecture of Fast Inverse Square Root

Above Figure 1. Represents the system architecture for the modified algorithm.

Depending on this exponent bit, the code selects an appropriate magic constant, 'R', which significantly impacts the precision of the approximation. The approximation process itself occurs in two iterations, following the Newton-Raphson method, gradually refining the result. Finally, the calculated approximation is assigned to the output signal.

4. Verilog Code

The "TopModule" serves as the top-level module that instantiates the "InverseSquareRoot" module, facilitating input and output at the system level. Thus, the Verilog code that we have implemented provides an effective methodology for approximating the inverse square root, employing an iterative Newton-Raphson approach with the consideration of registers, constants to ensure accuracy and precision. Different modules have been created for each specific operation e.g., rightbitshift for shifting the input number to right thus dividing the number in half or subtractor module for subtractions. For multiplication operations, modifiedbooth has been create which essentially performs multiplications. A Booth Multiplier is a hardware-based digital circuit used for binary multiplication in computer architecture and digital signal processing. It employs a Radix-4 encoding scheme to reduce the number of partial products generated during multiplication. By efficiently encoding and processing four bits of binary data at a time, Booth Multi- pliers significantly optimize the multiplication process, making

them faster and more resource-efficient than conventional multipliers [13]. They are especially well-suited for applications requiring high-speed and low-power binary multiplication, such as digital signal processing, arithmetic units, and microprocessor design [14].

In Top Module all these other modules are called as per their functionality. In this way, to perform newton Raphson iteration 1, subtractor and modifiedbooth module will be called as per the operation required.

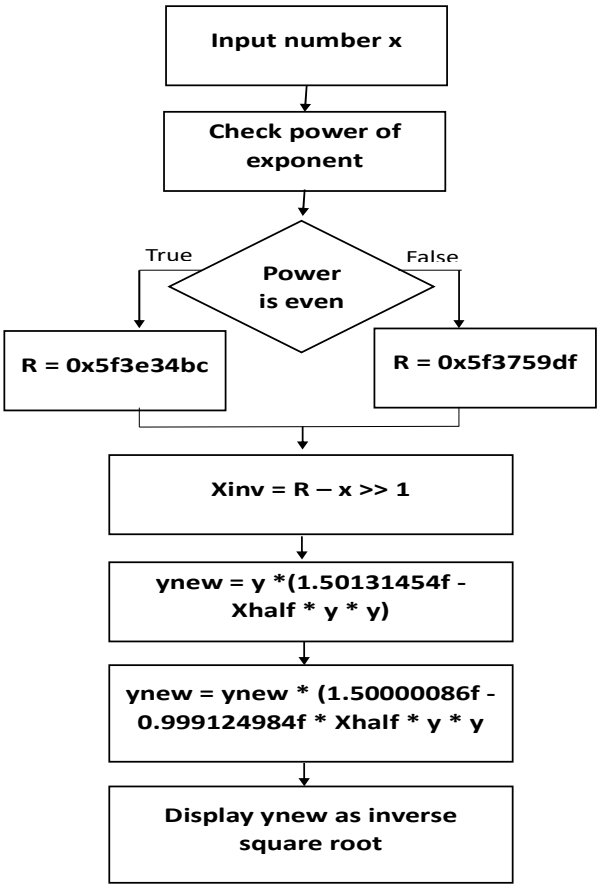

Figure 2. System flow of the algorithm

Figure 2. represents flowchart of the system in detail. It uses right bit shift module for dividing the number In half, subtractor module for the process of subtraction in Xint, and multiplier module for newton Raphson iterations.

Algorithm 1: Previously proposed algorithm

Input: x

Output: y

Initialisation:

1: xhalf= 0.5 ∗ x

2: xint= (x as an integer)

3: xint= 0x5f3759df – (xint >> 1)

4: y = (xint as a float)

5: y = y ∗ (1.5 – xhalf ∗ y ∗ y)

Algorithm 2: Improved Algorithm

Input: x

Output: y

Initialisation:

1: xhalf= 0.500438180f ∗ x

2: xint= (x as an integer)

3: e = exp(x) % 2 == 0

4: R = e ? 0x5f3e34bc : 0x5f3759df

5: xint= R – (xint >> 1)

6: y = (xint as a float)

7: y = y ∗ (1.5013454f – xhalf ∗ y ∗ y)

8: y*(1.50000086f – 0.999124984f * xhalf * y *y

The values used for Newton-Raphson iteration are used from the paper [4].

Comparison of the two Algorithms: Algorithm 2 is an upgraded version of Algorithm 1 for estimating the inverse square root of a floating-point number 'x'. Both algorithms use dynamic 'R', which is the magic constant , selection based on the exponent of 'x', but Algorithm 2 further refines the approximation through two iterations, uses more precise constants, and includes an additional multiplication term. This leads to improved accuracy, when compared to Algorithm 1, which involves just one iteration and simpler constants. However increased precision leads to a slightly higher computational complexity compared to Algorithm 1, which has a single iteration.

5. Results

The schematic diagram obtained from the implemented code is given below.
Figure 3 and 4 represent the schematic diagram for the fast inverse square root implemented. It uses Right bit shift and subtractor to approximately calculate the inverse square root of the number. Newton Raphson iterations are implemented with the help of subtractors and multipliers to get more accurate results.

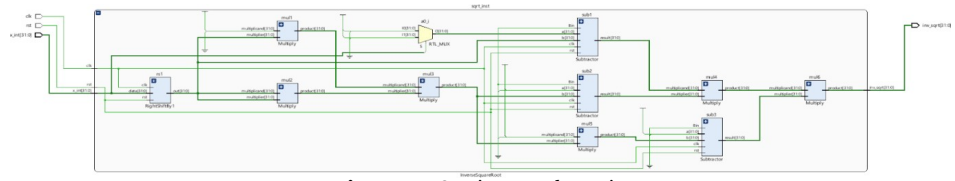

Figure 3. - Continuous schematic

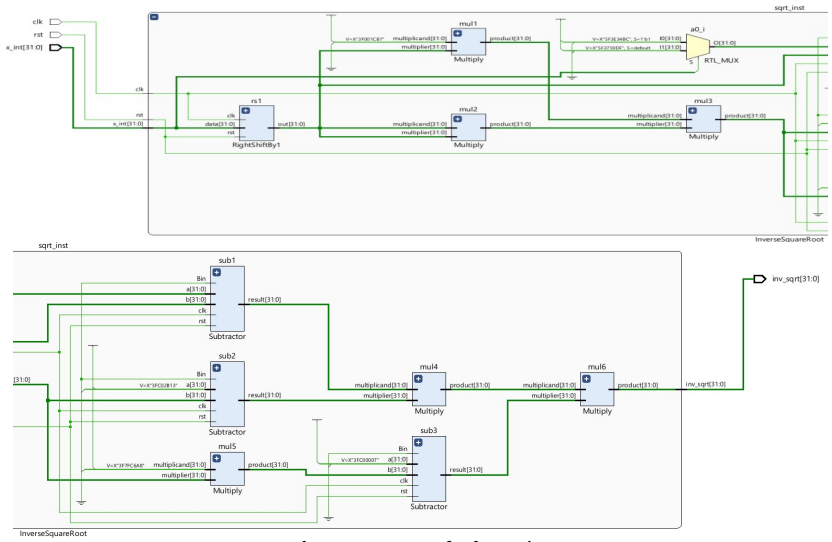

Figure 4. Zoomed schematic

As we can see in the figure 3 of continuous schematic, output of one stage is passed on to the other as its input. The output of the first right bit shift is passed on to the sub1 which essentially performs the operation of xinv = R – xint <<1. This step essentially calculates approximate value of the inverse

square root for the given number. Similarly, outputs of sub1 and sub2 are x_int = R - (x_int >> 1) and 1.50131454f - Xhalf * y * y respectively which are then passed on to mul4 which calculates the first Newton Raphson iteration value of the fast inverse square root.

Table 1. Slice logic utilization

Site Type	Used	Available	Util%
Slice LUTs*	408	17600	2.32
- LUT as Logic	408	17600	2.32
- LUT as Memory	0	6000	0.00
Slice Registers	72	35200	0.20
- Register as Flip Flop	72	35200	0.20

Table 1. Indicates utilization of slice LUTs and slice registers. Utilization of LUTs used as logic is 2.32%. Utilization for these look up tables is only for the purpose of logic and are used as memory unit. Utilization of registers which are used as flip flop for storage purpose is 0.20%.

6. Conclusion

In conclusion, our project is an innovative and practical approach to computing the inverse square root efficiently. The fast inverse square root algorithm has diverse applications, spanning from signal processing to 3D rendering and gaming engines. By implementing this algorithm with two magic constants, its performance, precision, and efficiency have been significantly enhanced, all while requiring fewer hardware resources and consuming less power. The incorporation of the two magic constants also extends the applicability of the inverse square root operation to a wide range of fields and applications.

7. Acknowledgement

The concept of using two magic constants and deciding the value based off of index of the exponent is implemented with the help of Jay Pargaonkar, Parth Kulkarni and Soumil Paranjpay Department of Electronics & Telecommunication, Vishwakarma Institute of Technology Pune, India.

References

[1] Horyachyy, Oleh, L. Moroz, and Viktor Otenko. "Simple effective fast inverse square root algorithm with two magic constants." Int. J. Comput 18 (2019): 461-470.

[2] Zafar, Saad, and Raviteja Adapa. "Hardware architecture design and mapping of 'Fast Inverse Square Root' algorithm." In 2014 International Conference on Advances in Electrical Engineering (ICAEE), pp. 1-4. IEEE, 2014.

[3] Walczyk, Cezary J., Leonid V. Moroz, and Jan L. Cieśliński. "Improving the accuracy of the fast inverse square root algorithm." arXiv preprint arXiv:1802.06302 (2018).

[4] Walczyk, Cezary J., Leonid V. Moroz, and Jan L. Cieśliński. "A modification of the fast inverse square root algorithm." Computation 7, no. 3 (2019): 41.

[5] Ercegovac, Milos D., Tomás Lang, J-M. Muller, and Arnaud Tisserand. "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers." IEEE Transactions on computers 49, no. 7 (2000): 628-637.

[6] Hasnat, Abul, Tanima Bhattacharyya, Atanu Dey, Santanu Halder, and Debotosh Bhattacharjee. "A fast FPGA based architecture for computation of square root and Inverse Square Root." In 2017 Devices for Integrated Circuit (DevIC), pp. 383-387. IEEE, 2017.

[7] Libessart, Erwan, Matthieu Arzel, Cyril Lahuec, and Francesco Andriulli. "A scaling-less Newton-Raphson pipelined implementation for a fixed-point inverse square root operator." In 2017 15th IEEE International New Circuits and Systems Conference (NEWCAS), pp. 157-160. IEEE, 2017.

[8] Kabuo, Hideyuki, Takashi Taniguchi, Akira Miyoshi, Hitoshi Yamashita, Miki Urano, Hisakazu Edamatsu, and Shigeo Kuninobu. "Accurate rounding scheme for the Newton-Raphson method using redundant binary representation." IEEE Transactions on Computers 43, no. 1 (1994): 43-51.

[9] Libessart, Erwan, Matthieu Arzel, Cyril Lahuec, and Francesco Andriulli. "A scaling-less Newton-Raphson pipelined implementation for a fixed-point inverse square root operator." In 2017 15th IEEE International New Circuits and Systems Conference (NEWCAS), pp. 157-160. IEEE, 2017.

[10] Schulte, Michael J., and Kent E. Wires. "High-speed inverse square roots." In Proceedings 14th IEEE Symposium on Computer Arithmetic (Cat. No. 99CB36336), pp. 124-131. IEEE, 1999.

[11] Moroz, Leonid V., Cezary J. Walczyk, Andriy Hrynchyshyn, Vijay Holimath, and Jan L. Cieśliński. "Fast calculation of inverse square root with the use of magic constant–analytical approach." Applied Mathematics and Computation 316 (2018): 245-255.

[12] Wang, Li-Rong, Shyh-Jye Jou, and Chung-Len Lee. "A well-structured modified Booth multiplier design." In 2008 IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 85-88. IEEE, 2008.

[13] Hussin, Razaidi, Ali Yeon Md Shakaff, Norina Idris, Zaliman Sauli, Rizalafande Che Ismail, and Afzan Kamarudin. "An efficient modified booth multiplier architecture." In 2008 International Conference on Electronic Design, pp. 1-4. IEEE, 2008.

[14] Mishra, Ravi Shankar, Puran Gour, and Braj Bihari Soni. "Design and Implements of Booth and Robertson's multipliers algorithm on FPGA." International Journal of Engineering Research and Applications (IJERA) 1, no. 3 (2011): 905-910.