Design And Optimization Of Floating Point Division And Square Root Using Minimal Device Latency

Hema Priya K, Arulmurugan L, Praveen Kumar S, Ramesh R

2, Assistant Professor, Bannari Amman Institute of Technology, Department of Electronics and Communication Engineering, Sathyamangalam, Tamilnadu, India.
134, PG Scholar, Bannari Amman institute of Technology, Department of Electronics and Communication Engineering, Sathyamangalam, Tamilnadu, India.

hemakandasamy1997@gmail.com, arulmurugan@bitsathy.ac.in

Abstract. New microprocessors have become a compulsory feature of floating-point support. In recent years, multiple generations of floating-point units (FPUs) have been seen by leading architectures. Although the introduction of addition and multiplication has been increasingly effective, the support of division as well as square root will have remained uneven. The types of algorithms employed, as well as the quality and efficiency of the implementations, are drastically different. Here a floating-point division and square root unit is provided that implements a radix-64 floating-point division and a radix-16 floating-point squareroot. Here a floating-point division and square root unit is provided that implements a radix-64 floating-point division and a radix-16 floating-point squareroot. Speculation between succes sive radix-4 incarnations is used to achieve a reduced timing. In digit-recurrence implementations, there are three separate components: initialization, digital repeats, and rounding. The repetition of the digit is the iterative component and for multiple loops it follows the same logic. The initialization and rounding steps are partly shared by the division and square root, although each has a separate logic for digital iterations. The result is a floating-point divider and square root with low latency, requiring double, single and half-precision division of standardized operands11, 6, and 4 result cycles, and 15, 8, and 5 cycles of square root cycles. One or two additional cycles are required in the case of subnormal operand(s) or result

Keywords: Floating point, single, double, square root, device latency carry skip adder and carry save adder.

1. INTRODUCTION

Due to constant advances in VLSI technology and the explosion of corporate, technical and recreational applications using floating-point operations, floating-point computational logic has long been a mandatory part of high-performance computer systems as well as embedded systems and smart phone applications. The efficiency of many modern devices with a high frequency of floating point operations is often restricted by the speed of floating point hardware. Therefore an integral part of these systems is a high-performance FPU. Over recent years, several generations of FPUs have been introduced into leading architectures. However while implementations of addition and multiplication have become more and more successful, support for division and other elementary functions such as square root has remained uneven[1]. In modern processors, some of the most representative floating-point functions are division and square root. While they are less common than the two simple arithmetic operations, addition and multiplication, the global efficiency of the processor may be influenced by bad performance when calculating these operations. The Division was considered a minor member of the floating-point family for a long time, which is irritating. Hardware engineers often see divisions as uncommon, low-priority tasks and accordingly delegate design involvement and chip resources, as two to five device cycles require addition and multiplication, whereas division latencies range from thirteen to sixty. While unconventional general-purpose applications, division and square root are comparatively uncommon operations, they are necessary and increasingly significant, in many modern applications, such as CAD tools and 3D graphics processing, in particular, as seen in Fig.1. Moreover, due to the latency difference between addition by multiplication and division by square root, the above operations are increasingly becoming performance bottlenecks. Thus, poor floating-point division and
square root implementations will result in significant performance degradation. The architecture of a floating-point unit implementing a digit-recurrence divider radix-64 and a square root radix-16 is defined in this article. A radix \( r \) digit recurrence algorithm, \( r \) being a power of 2, is an iterative algorithm where each iteration is obtained by a radix-\( r \) digit, \( \log_2(r) \) bits, of the result quotient or root. By overlapping simple radix-4 iterations, both the division and the square root iteration are obtained to achieve an optimal implementation of energy and timing. Therefore in a single loop, three radix-4 division iterations converge, supplying 6 bits of the quotient per cycle, which is equal to a radix-64 iteration. Similarly, in a single loop, two radix-4 square root iterations coincide, producing 4 root bits per cycle, which is equal to a radix-16 iteration.

2. FLOATING POINT STANDARDS

The coasting side of the point idea methods there would no altered amount from claiming digits previously, then following those decimal point; that is, the decimal point will coast. Skimming purpose representations need aid slower What's more lesquerella exact over settled purpose representations. Yet they camwood handle a wider situated for numbers. Since floating point number math requires an incredible bargain of computing power, large portions of microprocessors come with An chip known as a floating perspective unit (FPU), which is particular over skimming purpose math execution. Math coprocessors and numerical coprocessors are also called FPUs. A complex encoding scheme with three simple components has a floating-point representation: mantissa, exponent, and symbol. Gliding purpose numbers are spoke to Likewise absolute exactness (32-bit) What's more twofold precision (64-bit) floating-point numbers utilizing double numeration and forces for 2. Concerning illustration demonstrated for fig. ,both absolute Furthermore twofold precision numbers. 1 is specified by standard IEEE 754. For a single precision format, 8-bits are reserved for an exponent with a bias value of +127 for a single precision format, and 23 bits are reserved for mantissa. It means a negative number when the sign bit is 1, and when it is 0, it claims a a positive number.

![Figure 1. Types of floating point](image)

For double precision formats where exponents are skewed to +1023, the comparable explanation is expanded. In several digital signal processing (DSP) applications, division and square root are major operators, including matrix inversion, vector normalization, and Cholesky decomposition. Many various floating-point formats, like common IEEE formats, are supported by floating-point split and square root operators. The two modules display a strong trade-off between zone, latency, and performance. They are often completely pipelined to assist the designer in executing designs that are fast, complicated and pipelined. In order to get a stronger figure of merit, it is the most essential objective of a designer to increase the efficiency of the ALU, thus reducing its design complexity. The foregoing operations are gradually becoming output bottlenecks due to the latency gap between addition by multiplication and division by square root. As the output distance between addition by subtraction by multiplication and division by square root widens, many signal and image processing algorithms have been rewritten to prevent the use of division and square root to incorporate floating-point operations. In comparison, applying square root on hardware is challenging. Thus, bad floating-point division and square root implementations result in significant performance loss.

2.1 Digit-Recurrence Algorithms For Division And Square Root

The Digit-recurrence is a class of iterative algorithms that measure a digit \( pi+1 \) radix-\( r \) result with a power of \( r^2 \) and a residual \( \text{rem}[i] \) for each iteration[5],[7]. The remainder is used to gain the next digit of radix-\( r \). Notice that each radix-digit expresses the effects of \( \log_2(r) \) bits.

Before the iteration I the partial result is defined as

\[
R(i) = \frac{p_i \times r^2}{r^2} \quad \text{(1)}
\]

For a short iteration, the remainder is kept in the signed-digit redundant representation carry-save. In our execution, with a positive and a negative term, we selected a radix-2 signed-digit representation for the remainder.
2.2 General Architecture

In this unit, division and square root are the two floating-point operations being carried out. In figure, the general organisation is seen. The three areas of the implementation of digit-recurrence, preprocessing, digital iterations and processing, are specifically marked. The records for the partial quotient by root, remainder, and certain other signals are exchanged by these three components. Operands are unpacked in the pre-processing stage and the first division or square root iteration is carried out. In addition, the divisor and dividend are scaled to place the divisor in a small interval about +1 in the case of division, so that the quotient-digit selection is simplified. In this step, every subnormal operand is normalised, depending on the number of subnormal operands, the normalisation logic is used for 1 or 2 cycles. The original values of the partial quotient by root and remainder are passed on to the digit iterations.

The iterative portion of the digit-recurrence algorithm is the digit iteration stage. Since the registers are shared, the division and square root have a distinct digit iteration logic. This logic is divided into two parts: (1) the calculation of the remainder and (2) the calculation of the quotient by root digit and the partial modification of the quotient by root. As stated in the introduction, each loop obtains radix-64 division and radix-16 square root iterations by overlapping simpler radix-4 iterations: three radix-4 iterations in division, and two radix-4 iterations in square root. Another esteem to the incomplete quotient/root may be ascertained to each cycle and the leftover portion. For both, fractional result and rest, Redundant radix - 2 signed-digits representation is utilized for both halfway come about What's more remainder; hence, two words, certain statement Furthermore negative word, need aid used to speak to whatever remains. Also incomplete result; the altered buildup Furthermore halfway quotient Toward root are passed go for the following circle. Then afterward the most recent iteration, the unrounded quotient Toward root and the last leftover portion would moved of the post-processing step.

The algorithm of division What's more square root is those radix-4 digit-recurrence calculation for three divisions or two square roots) iterations for every circle and the representational from claiming digit set f 2; 1; 0; +1; +2g; i. E. R = 4, a = 2, respectively, the radix and the digit situated. At whatever 2 odds of the quotient alternately root will be finished with the radix-4 calculation. Every circle need obtained 6 odds of the quotient, which is equal with a radix-64 divider, since three radix-4 iterations are performed clinched alongside division for every clock cycle. Similarly, two radix-4 iterations for every clock cycle are performed in the square root, Furthermore every cycle gets 4 odds of the root, which will be equivalent to a radix-16 square root.

3 EXISTING WORK

Figure 2 General organisation unit diagram
In this unit, the two floating-point operations being carried out are division and square root. Figure 1 demonstrates the general organisations. The three aspects of the implementation of digit recurrences, preprocessing, digital iterations, and post processing are specifically marked. Three parts are traded Toward those documents for those fractional quotient/root, remainder, and a few different signs. The operands are unpacked in the pre-processing stage and the To begin with division alternately square root cycle is conveyed crazy. In addition, the divisor and dividend are scaled to bring the divisor around +1 in a small interval in order to optimise the quotient-digit collection in the case of division. In this step, every subnormal operand is normalised, The number of subnormal operands is based on, the normalisation logic is used for 1 or 2 cycles. The original values of the partial quotient by root and remainder are moved to the digit iterations. The digit iteration stage is the iterative part of the digit-recurrence algorithm. The division and square root have a distinct digit iteration logic, as the registers are shared. This logic is divided into two parts: (1) the calculation of the remainder and (2) the calculation of the quotient by root digit and the partial modification of the quotient by root. As stated in the introduction, by overlapping simpler radix-4 iterations in each sloop, radix-64 division and radix-16 square root iterations are obtained three radix-4 iterations in division, and two radix-
For every iteration, another value is computed to the fractional quotient/root and the r. Excess radix-2 signed-digit representational will be utilized for both fractional Conclusion Also remaining; hence, two words, sure expression Also negative word, are used to representable the remaining Also fractional outcome; those changed leftover portion Also incomplete quotient/root would moved over for the following circle. Then afterward the most recent iteration, the unrounded quotient/root and the last leftover portion are moved of the post-processing stage. There is an alternate digit cycle rationale for those division What's more square root, Similarly as said preceding. Notwithstanding there are exactly past hypotheses that fuse the rationale for division What's more square root digit tedious. In spite of the fact that the region might be smaller, our execution might influence the stage timing; done particular, it might make was troublesome to have three iterations of the radix-4 division in the same chance. Those remaining progress may be attained with 3-to-2 carry-save adders in the division iteration, Also for 4-to-2 carry-save adders in the square root cycle. On the whole division What's more square root rest modification, 4-to-2 carry-save adders necessity should make utilized within a joined unit over whatever division Furthermore square root r modification; thus, three radix-4 iterations might not work to one cycle and the execution might be corrupted. Perceive that those region might be smaller, In this way that any of the carry-save adders Might make evacuated.

3.1 Division Architecture

Displayed Pre-scaling could be awell-knowntechnique to simplify the selection function for quotient digits. Here we provide a brief description of this method. an in depth description are often found in. The divisor is scaled to a worth near 1 in order that the quotient-digit selection is independent of the divisor the choice function is kind of simple: there remainder estimation is compared with some low-precision wired selection constants. If the divisor isn't pre-scaled, the choice function in equation (7) would become qi+1 = SEL(~rem [i]:d) Although the divisor is that the same for each iteration and known earlier, it's to be taken into consideration for the quotient-digitselection. Consequently, a look-up table storing the choice constants for each quotient interval would be needed for the quotient-digit selection. This look-uptable isn't required when the divisor is pre-scaled. It has been determined that for a radix-4 digit recurrence it's enough to possess the scaled divisor within the range [1-1/ 64 ; 1 + 1/ 8). The divisor is multiplied by a scaling factor $M = 1 + b *23$, with $0 \leq b \leq 8$, and $b \neq 7$, which depends only on the three most-significant bits of the divisor. The pre-scaling will be implemented because the addition of the divisor plus 2 or 1 multiples of the divisor. The dividend should be pre-scaled by the identical amount to induce the right result. The diagram of this cycle is shown in Figure 2.

1) During this cycle, additionally to the operands pre-scaling, the first iteration is dispensed, and therefore the operands are comd to force the result to be in as a part of the pre-scaling, redundant carry save representations of pre-scaled divisor and dividend are obtained.
2) So as should actuate those leftover portions after the main iteration, those excess pre-scaled divisor and profit would assimilate should. An non-redundant representation, scaled divisor Also scaled profit inside the figure, rem. The non-redundant divisor is utilized inside these digit iterations besides (see equation (8)).
3) The operands are compared and also the dividend is left-shifted by 1bit if $x < d$ to avoid wasting time, the comparison is disbursed in parallel with the pre-scaling.
4) In parallel for the excess with non-redundant interpretation of the operands, those basic digit, q1 of the quotient, will be Moreover acquired. This could be a simplified digits quotient calculation, a result that the quotient is certain and to, the integer radix 4 digit might only take qualities $qs1 = +1$; or $qs1 = +2$. For $x < d$ What's more $x d$, the basic digit comparison is repeated, getting two quotient digit candidates, for profits more stupendous over divisors Furthermore for profits more diminutive over divisors. Those outcomes of the examination select the right digit.
5) The next remainder, Rem is derived from the non-redundant scaled dividend (the remainder's positive word) and the non-redundant scaled divisor (the remainder's negative word). The contrast of operands is often used in thescaled dividend selection in such a way that if the divider is greater than the dividend, the scaled dividend is 1-bit left-shifted. Conversely, if the quotient digit is +2, the scaled divisor is 1-bitleft-shifted and is not shifted if the quotient digit is +1.

Figure 3: Pre-processing stage for division: pre-scaling and integer quotient digit calculation
3.2 Square Root Architecture

The estimation of the square root consists of multiple iterations, leading to the accuracy (number of bits) of the final product of the number of iterations. The higher the number of iterations, the lower the lag. By missing the first iteration, the number of iterations is decreased by 1 and the latency is therefore reduced by 1 cycle with such floating-point accuracies and organisation. In the initialization process, the first iteration is skipped and inserted. From the input value $x_0$ in equation (14), the partial root and the rest for the second iteration are readily obtained.

The actual implementation of the floating-point square root performs two radix 4 iterations per cycle. So, taking into account this fact, the reasoning has been refined. The block diagram of a digit-iteration loop is seen in figure 4; this is the computation of two iterations of radix 4. Note that the figure implementation is divided into two parts, (1) residual estimation, and (2) digit collection and root updating. As for division, according to equation (11), the remainder is speculatively determined. Therefore, 5 remainders are determined for each iteration, one remainder for each root digit number, and when the digit is obtained, the right remainder is chosen. Notice that as part of the next remainder estimate, the remainder has to be left-shifted by two bits.

3.3 Latency

The latency of the measurement of the division and square root of the proposed unit as seen in Table 3. The table displays the latency for the interest, double, single and half-precision, DP, SP and HP, respectively, for the three floating-point accuracies, and for some mixture of regular and subnormal operands and effects. Remember that the product of the square root cannot be subnormal and that it only has one operand. The latency is the sum of the number of pre-processing cycles, plus the number of cycles for digital iterations, plus the number of post-processing cycles.

The number of pre-processing cycles depends on the number and the operation of sub normal processes. Indivision can be one, three or four cycles of preprocessing. If all operands are regular unpacking and pre-scaling in the same step. Three separate cycles for unpacking, normalisation and pre-scaling are required in the case of one subnormal operand. Finally, if all operands are sub-normal, there is one unpacking cycle, two normalisation cycles, and one pre-scaling cycle. Similarly, if the operand is normal, there will be one pre-processing cycle for unpacking for square root, or two pre-processing cycles for unpacking and normalisation, if the operand is subnormal. The number of digit cycles depends on the number of bits required. The number of fractional bits of the quotient to be obtained by the algorithm in division is 53 double accuracy (52 fractional bits plus the guard bit), 24 single accuracy (23 fractional bits plus the guard bit) and 11 half accuracy (10 fractional bits plus the guard bit). There is also an integer digit that may be 1 or 2, but this integer digit is gained in parallel with the pre-scaling and does not count in the intervals of digit iterations. Finally, the number of post-processing cycles is one if result is normal or two if the result is subnormal (only indivision).

As an example, the cycles are shown below for the single precision division,

1) Normal inputs, normal result:
PRE – DGT – DGT – DGT – DGT – RND
2) Normal inputs, subnormal result:
PRE – DGT – DGT – DGT – DGT – RND – RSH
3) One subnormal input, normal result:
UNP–NM–PSC–DGT–DGT–DGT–DGT–RND
4) One subnormal input, subnormal result:
UNP–NM–PSC–DGT–DGT–DGT–DGT–RND–RSH
5) Two subnormal inputs, normal result:
UNP–NM–NM–PSC–DGT–DGT–DGT–DGT–RND

3.4 Area

The region is highly based on the algorithm that is used for division and square root, digit-recurrent or multiplicative. In general, if the multipliers or FMA units are shared with other floating-point instructions, multiplicative algorithms have lower area requirements. In the case of digit-recurrence algorithms, the radix also has a great impact, the greater the region of the radix. The area of our division by square root unit is also much larger than the area of the other units in the table, but our attention was on keeping a division/square root unit with low latency. The rounding stage area is not included in the debate since for all the units it should be about the same.

3.5 Timing

The Conceptual Effort Model [26] is used in this section for critical path delay estimation. The pause with respect to FO4 and its pico second counterpart of the simple gates (upper part) and the main modules in Figures 2 and 3 (middle and lower parts, respectively) is outlined in Table 5. We have considered a FO4 delay of 6-ps. The load was taken into account for each signal, so that a fan out of n contributes a delay equal to log 4 n FO4. In specific, the fan-out affects the Pick division module in Figure(3) and the comparators in Figure(4). The performance of the modules, the quotient or root digit, has a high fan out, about 64 gates.

3.6 Carry Save Adder (CSAA)

The addition of 3 numbers to the addition of 2 numbers is reduced by the carry save adder. Regardless of the number of bits, the propagation delay is 3 gates. The carry-save unit consists of a full address, each measuring a single amount and holding a bit depending entirely on the respective bits So the three input numbers. By shifting the carry sequence left by one place and adding a 0 to the front of the partial sum sequence (most important bit) and adding the resulting n + 1-bit value to this sequence with RCA, the entire sum can then be calculated. This process can be continued indefinitely, adding an input of full adders without any intermediate carry propagation for each point. With the logarithmic average delay in the number of inputs to be applied and the number of bits per input being invariant, these phases can be arranged in a binary tree structure. The major use of the carry save algorithm is that the multiplier architecture is well known for the robust implementation for high-speed CMOS of a much wider variety of algorithms +6. In the partial product line of array multipliers, CSA was included in the partial product line of array multipliers to accelerate the carry propagation in the array. In Figure (4), the Hold Save Adder concept diagram is shown. The simulation result is shown in Figure (3b).

3.7 Drawbacks

- Hardware utilization is high.
- Number of adders used is high.
- It propagate the carry to the next upcoming bits automatically.
PROPOSED WORK

4.1 32-bit carry-skip adder design

Along with carry-skip logic (SKIP), carry-generate logic (CG), and group generate propagate logic (PG), it utilises a mixture of RCAs. The full adder is split into a variety of blocks of variable width. Using AOI and OAI circuits, both the carry generation and skip logic. The width of each block is restricted by the delay T goal. There is a further division of each block into subblocks. In a recursive fashion, a sub block can contain additional levels of sub blocks. The number of variable width RCAs are formed into the lowest-level sub block. The structure of an adder is defined as follows:

Here the 32-bit adder is split into four separate blocks. Figure 2 displays a block diagram of the first three blocks (A0, A1, and A2). The first block A0 (LSB) alone is a complete adder.

4.2 Carry Skip Adder (CSKA)

A carry-skip adder consists of a plain ripple carry-adder with a skip chain called a special speed up carry chain. When adding a large number of bits, carry skip adder is a fast adder compared to ripple carry adder; carry skip adder has O(n) delay along with a simple and standard structure, offers a good balance in terms of delay. This chain determines the distribution of ripple carry blocks that make up the skip adder. By assisting the propagation of a carrying bit around a portion of the entire adder, a carry-skip adder is built to speed up a large adder. The ripple bearing adder is actually quicker for small N values. Nevertheless these days, the industrial specifications that most desktop computers use 32-bit word lengths like multimedia processors make the carry skip structure more interesting. The crossover point between the ripple-carry adder and the carry-skip circuitry consists of two logic gates. The AND gates accepts the carry-in bit and uses the individual propagate values to equate it with the group propagate signal. The output from the AND gate is ORed with RCA cout to generate the next group of adders. The design schematic of Carry Skip Adder is shown in Figure below.

Figure 9 Schematic of Carry Skip Adder
4.3 Square Root

The effective algorithm that contains the positive attributes of the mentioned square root computation. With the huge increase in the number of bits, the square root algorithm discovers its limits as this paper deals with the representation of the floating point where mantissa is 23-bit broad. As follows, the basic algorithm for the proposed design is:

1) Sign Bit

The result's signal bit is the same as the initial number's sign bit.

2) Exponent Computation

The performance exponent relies on the skewed number exponent. If the skewed exponent is odd, it adds 127 and the final sum is moved appropriately (divide by action 2).

$$E_r = \frac{E_a + 127}{2}$$

If the skewed exponent is even, 126 is applied and the final sum is moved appropriately (divided by 2 operations). Furthermore the change flag is set to show that the mantissa should be pushed 1 bit to the left before measuring its square root.

$$E_r = \frac{E_a + 126}{2}$$

3) Mantissa (Square Root) Evaluation

The code block that performs square-root computation is based on an iterative approach where two registers are handled, respectively temp and 27-bit wide ANS and 26-bit wide Mr. Consider an example of the square root of 16 (100002) evaluation.

TEMP is initialised as 0000...0000 and ANS is initialised as 0100...0000. The method of iteration is carried out. TEMP is loaded with mantissa on the first iteration and then compared with ANS, based on the comparison performance, shift operation is performed. If the contents of TEMP are greater than or equal to ANS, they are subtracted from those of ANS, moved to the left, and placed in TEMP. The contents of ANS, starting from the current pointer location, are moved one bit to the left. In ANS, aS1 is then inserted into the current bit location. Notice that in any iteration, a pointer points to the current bit that will be replaced by ANS. If TEMP is below ANS, its contents are transferred to the left and deposited in TEMP. The contents of ANS, starting from the current pointer location, are moved one bit to the right. So a 0 is inserted in the ANS bit of the current bit location. Since the last iteration, the contents of the ANS are the bits known as the final product, with the exception of the two least important bits. Fig. Fig. 4 displays the square root computation's architectural block diagram.

![General block Diagram of FP sqrt algorithm](image)

**Figure. 10** General block Diagram of FP sqrt algorithm

It starts with the study of machine arithmetic. The IEEE standard 754 on binary floating point arithmetic was evaluated next. A number of high-performance floating point arithmetic algorithms have been identified. For floating point numbers, arithmetic functions consist of addition, subtraction, multiplication and division. Functions are conducted using algorithms similar to those used on integers of sign magnitude (because of representation similarity) — for example, just add numbers of the same sign. If the numbers have the opposite symbol, subtraction needs to be performed. In computing systems, floating point numbers are commonly used for numerical calculations. Arithmetic units are more complex than fixed-point estimates for floating-point numbers. Implementation of floating-point arithmetic digit recurrence algorithms such as binary division and square root.

Three inputs are taken from the floating-point adder: the two floating-point operands and a control field. The area of influence is 5 bits long. The two least essential bits (LSBs)describe the mode of rounding. To allow the hardware underflow trap handler, Bit 2 is used. To allow the hardware underflow trap handler, Bit 3 is
used. In order to pick the procedure, bit 4 is used. There are two outputs to the adder: the product of the floating point, and a flag sector. The flag field is also 5 bits long and has zero divider flags (for consistency with other floating point elements), invalid, erroneous, overflow, and exception flags for underflow. Due to the complexity of the algorithms, square root operation is difficult to enforce on FPGAs. It implemented a Digit-recurrence algorithm and two algorithm-based floating point square root implementations with very simple single precision. The Digit-recurrence square root algorithm also uses the representation of the two complements for the square root result.

![Figure 11 Detailed block Diagram of FP sqrt algorithm](image)

Due to the complexity of the algorithms, square root operation is difficult to enforce on FPGAs. It implemented a Digit-recurrence algorithm and two algorithm-based floating point square root implementations with very simple single precision. The Digit-recurrence square root algorithm also uses the representation of the two complements for the square root result.

![Figure 12 Detailed block Diagram of FP Division algorithm](image)

5 FEATURES

- It supports streaming inputs / outputs (pipeline).
- Rounding is to the nearest even number.
- Status flags indicating infinity, not a number and zero values.
- Built in Debug Logic feature to simplify the debugging process.
- Specifically designed for high-speed and high-performance floating point applications.
- Fully synthesizable synchronous design with positive edge clocking.

6 ADVANTAGES OF PROPOSED WORK

- It consumes less power compared to exiting work.
- Area of the circuit reduced.
- Low latency for input to output time.

7 CONCLUSION

The main motive of this paper, to use carry skip logic instead of using carry save adder. Because in CSA, carry will propagate automatically to the next bits. Because of this CSA, the number of adders in the circuit will be larger and so it will utilize the hardware more. But in Carry Skip Logic, carry will not propagate automatically to the next bits. It will propagate when the output is one (1). And it will be added finally at the output. So in the Carry Skip Logics, the number of adders in the circuit will be less and so it will not utilize the hardware more.
References

[1] J.D Bruguera. Radix-64 Floating-Point Divider. Proceedings of 25th IEEE international Symposium on Computer Arithmetic, pp. 879-84, 2018.

[2] A. Fog. Instruction Tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs. https://www.agner.org/optimize/instruction_tables.pdf, 2019

[3] C. Lichtenau, S. Carlough, and S.M. Mueller. Quad Precision Floating Point on the IBM z13TM. Proceedings of 23th IEEE international Symposium on Computer Arithmetic, 2016.

[4] Dheepanachakravarthy, A., Selvan, M.P. and Moorthi, S. Alleviation of Current Quality Issues Caused by Electric Arc Furnace Load in Power Distribution System Using 3-Phase Four-Leg DSTATCOM Journal of the Institution of Engineers (India) - Series B - Springer 100 01 9 - 22 JUN 2018

[5] A. Nannarelli. Performance/Power Space Exploration for Binary64 Division Units. IEEE Transactions on Computers, Vol. 65, No. 5, pp. 1671-1677, 2016.

[6] Sivaraman, P “A New Method of Maximum Power Point Tracking for Maximizing the Power Generation from a SPV Plant” Journal of scientific and Industrial Research Vol.74, No.3 pp.411 - 415 AUG 2015.

[7] D. Piso and J.D. Bruguera. Variable Latency Goldschmidt Algorithm Based on a New Rounding Method and a Remainder Estimate. IEEE Transactions on Computers, Vol. 60, No. 11, pp. 1535-1546, 2011.

[8] E. Ramprasath and P. Manojkumar, "Modelling and analysis of induction motor using labview", International Journal of Power Electronics and Drive Systems, vol. 5, no. 3, pp. 344, 2015.

[9] E. Ramprasath, P. Manojkumar and P. Veena, "Induction motor analysis using labview", Proceeding on International Conference on Electrical Engineering and Technology, vol. 2, no. 5, pp. 498, 2015.

[10] E. Ramprasath, P. Manojkumar, P. Veena "Analysis of Direct Current Motor in LabVIEW", World Academy of Science, Engineering and Technology, 2015.

[11] J. Rupley, J. King, E. Quinnell, F. Galloway, K. Patton, P. M. Seidel, J. Dinh, H. Bui, A. Bhownik. The Floating-Point Unit of the Jaguar x86 Core. Proceedings of 21th IEEE international Symposium on Computer Arithmetic, 2013.

[12] S. Srinivasan, K. Bhudiya, R. Ramanarayanan, P. S. Babu, T. Jacob, S.K.Matthew,R.KrishnamurthyandV.Erraguntla.Split-pathFused Floating Point Multiply Accumulate (FPMAC). Proceedings of 21th IEEE international Symposium on Computer Arithmetic, 2013.
