JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

Design and performance analysis of high throughput and low latency double precision floating point division on FPGA

Sushmitha V Gopal[1], Shivaputra[2]

M.Tech Student, Department of Electronics and Communication Engineering,
Dr.Ambedkar Institute of Technology,Bengaluru[1]
Assistant Professor, Department of Electronics and Communication Engineering,
Dr.Ambedkar Institute of Technology,Bengaluru[2]
sushmithagopal.817@gmail.com, putrauvce@gmail.com
Publon id : AAE-3813-2019

ABSTRACT
Instruction level parallelism is employed by fully pipelined architecture in latest processors and the existing processor utilizes various strategies. The system proposed in this work is implemented by employing fully pipelined architecture FPU in field-programmable gate array (FPGA) and when compared to other strategies, the proposed architecture is considered to be an optimized architecture in terms of area, latency and throughput. There exist numerous challenges in performing division operation in Digital signal processing. As a result, it is essential to implement division operations in FPGA due to their complexity which is main module in many complex operations. The pipeline in every cycle of clock must be ready to accept the divider results and store in memory. A Virtex-5 FPGA uses Verilog code to build and implement DPFP (double precision floating point), which makes use of single precision floating point calculations. The proposed DPFP uses Radix-4 multiplier at lower level module to perform partial products and additions. The delay possesses 23 clock cycles and the output rate consists of 21 clock cycles. In other implementations such as high-pipelined, a large number of adders/subtractors are used in some of applications. The process delays, as well as output rate obtained are 25 and 9 clock cycles, respectively. It minimizes the amount of energy dissipated and offers several advantages such as lower area usage as well as lower latency. In most floating-point applications, the floating-point division is considered to have low-frequency as well as high-latency of operation. On the other hand, processor designers are being compelled to pay close attention to all aspects of floating-point computations because of the growing demand for excellent graphics and the widespread use of performance standards like SPEC marks.. This work outlines the most widely known algorithms for floating-point division as well as implementation options that are available for the DSP applications. It also demonstrates how conventional applications of floating-point can help the designer in formulating decisions as well as trade-offs using a system-level analysis as a foundation.

Keywords: Double precision floating point division, Low power, FPGA, and Clock gated technique

1. INTRODUCTION

Nowadays, the computational complexity of modern computer applications has been increased. To address the calculation demands of current applications, the development of high-speed floating-point (FP) arithmetic is necessary and also have become more complicated in the past few years. The requirements on computation processors' abilities have been increased because of the focus on high-performance graphics processing systems. On the other hand, extensive application performance standards like SPECmarks drive chip developers to specifically focus on floating-point calculation. Floating-point operations like addition, multiplication, as well as division, are used in applications similar to the one that is stated. Nowadays in recent FPUs, the priority is given mainly in developing faster adders as well as multipliers along with division receiving less priority. The conventional addition, as well as multiplication latency ranges from 2 to 4 cycles and 2 to 8 cycles respectively. In recent FPUs, 7 to 61 cycles are the latency range for double-precision division [4]. This is due to the common

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

misconception that divide is a rare operation in modern floating-point applications. Due to lower frequencies, the decrease in total performance is observed by using a slow divider and therefore more attention has been made to increase addition as well as multiplication performance. Floating-point algorithms, as well as applications, were gradually rebuilt as the performance differences between two operations as well as division were increased. Therefore, the use of the divide was reduced to compensate for this difference. As a result, most of the modern applications as well as standards are developed with the assumption that divide is a time-consuming operation and must be utilized rarely. Whereas the process for developing high-performance cost-effective adders as well as multipliers is widely recognized and the development of dividers continues to be an important design issue which is usually referred to as a "black art" by system developers. A detailed theory exists that describes the division theory and the execution of division on the other hand is less focused with a minimal concentration on analyzing the impact of FP division on the performance of the system. The relationship between system performance as well as FP divide is investigated in detail in the further context. In conventional floating-point applications, this connection is been examined. Whenever assessing a system performance, selecting appropriate applications is a difficult as well as challenging task. The NAS Parallel Standards [5], the Perfect Standards [8], as well as the SPECfp92 [10] standard programs were also examined for this research work. The SPEC standards possess the highest frequency of floating-point operations according to the early analysis of instruction distribution. Hence, they were chosen for this study to represent the floating-point behavior for demanding applications.

Microprocessors enable floating-point computation. FPU (Floating Point Unit) has become a prominent design for the past few years. Adding as well as multiplying numbers has become progressively efficient and effective. However, the square root, as well as division assistance becomes uneven. Variation in different types of algorithms differs in reliability as well as efficiency. However, radix-64 and radix-16 division and square roots are used to implement the floating-point division and square root units.. A 32-bit adder employs the carry skip logic. For execution, the adder is subdivided into many blocks and the signals carry delay, as well as the ultimate targets delay is restricted with the size of every block. An approach was employed to determine the highest size of adders that fulfill the delay of the target. In this research work, the delay of full adders is employed as the unit of measurement. Nowadays many applications rely on floating-point operations that operates with a high frequency and are further limited by the efficiency of FP hardware. As a result, the FPU with high efficiency is considered as an important unit of these systems. Performing operations such as addition as well as multiplication has become highly proficient. However, competence for division as well as other basic operations like square root are unequal. Division operation and square root operation are the most common FP operations in recent processors. The division is a member of the FP family and is regarded as a counterpart.

Floating Point Architecture: The binary representation of a real number is termed as floating-point number. Binary as well as decimal interchange are the two systems in IEEE 754 [11] and in this research work we make use of binary format. The binary format displayed in Fig.1 also includes a sign bit, an exponent, and a mantissa or 23-bit fraction.

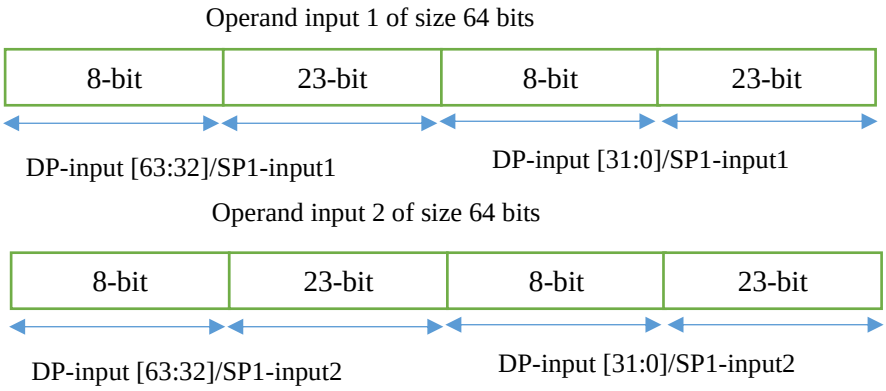


Figure.1: Proposed Double precision floating division operation using Single Precision Binary Format in IEEE 754

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

The number V possess few values in any floating point operation and described below:

1. V= Nan ("It will not work as a number"), when F is non zero and E is 255.
2. The value of V will be -infinity when the value of F and E is 255, as well as S is assumed to be one.
3. The value of V will be infinity when the value of F and E is 255, then S is assumed to be zero.
4. If we assume that 0<E<255, then we can say that the value of V is (-1) , the exponent ranges between -127 to +128.
5. A value of V=(-1) **S *2 **(-126) * (0.F). A "un-normalized" distribution is assumed for the data.
6. Assume if M is equal to 0, then E will be equal to 0, S will be equal to 0, as well as V will be equal to 0
7. Assume if F is equal to 0, then E will be equal to 0, S will be equal to 1, as well as V will be equal to 0.

Whenever an additional bit is added to the mantissa then this is referred to as significant. When E is more than 0 but less than 255, 1 will appear in the MSB significand, resulting in a balanced number. Consider a situation in which a real number is represented by an equation (1)

V=(-1s) * 2 (E-Bias) * (1.M) ----------(1)

General Division Floating Point Algorithm: In Fig.2 flow chart, the description of floating-point multiplication algorithm is been shown. S1, E1, and M1 are corresponding sign bit, exponent, as well as mantissa for operand N1 whereas S2, E2, M2, are corresponding sign bit, exponent, as well as mantissa for operand N2. Let us say x is equal to N1 as well as D is equal to N2, then the end outcome q will be equal to x/d.

There are four steps in the floating-point division:

Significantly divide as well as subtract exponents to arrive at a result.

S=S1 XOR S2
E=E1-E2
M=M1/M2

- Mantissa M i.e., moved to the left or right by a value 1 as well as E exponent is been updated.
- The output should be rounded to fit in the specified bits.
- The determination of special values as well as exceptional flags.

1.1 Single and double are the two fundamental floating-point formats

All 32 bits of the IEEE single format are used to represent 24-bit significand precision. The IEEE double system makes use of 64 bits in total and seems to have a precision of 53 bits for the significand. Single extended and double extended floating-point formats exist. As a result, it is impossible to determine the format's actual size and accuracy using the benchmark. A double extended format of IEEE should possess at least 64 bits and 79 bits significand accuracy as well as an overall length. Remainders, additions, subtractions, divisions, multiplications and square roots are all examples of floating-point operations that demand high levels of precision.. The comparison operations as well as the remainder should be accurate. Until there is no such output or if the output is not in the destinations format, the operations must not be sent to its target. Once it is accurate output is sent to the target. Process must transmit updated output as well as a new, accurate result as indicated below, as per the rounding mode rules that have been previously defined. Decimal strings conversions as well as binary floating-point numbers conversions into primary floating-point formats must satisfy accuracy, monotonicity, as well as identity constraints. In the case of operands having a range within the stated limitations, such conversions should provide as precise results as feasible, or significantly change such accurate results in compliance with the needs of the specified rounding modes. There must be no more than a predetermined tolerance in the outputs of such conversions, and the rounding mode is used for operands that do not fall within the expected range. Five categories of

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

IEEE floating-point errors as well as the rules exists for notifying the user when these sorts of errors arise. Overflow, underflow, and inexact floating-point errors are all examples of the five types of floating-point errors.. Four rounding directions: When there are two nearest representable numbers then "even" values are chosen with respect to negative infinity i.e., down, positive infinity i.e., up and 0 (chop). Rounding accuracy: When a system in double extended format provides results, the user must be allowed to choose whether the output should be rounded to the single format precision or double format precision. Compatibility for user exceptions recovery is also suggested in the IEEE standard below.

- Numerical computing and IEEE standard's criteria involve a variety of technologies, including interval arithmetic, retroactive anomaly detection, effective implementations of fundamental arithmetic operations like exp and cos, multiple-precision math, and many more.
- Users have more flexibility over computation using IEEE 754 floating-point arithmetic than with other type of floating-point calculation. The IEEE standard makes it easier to write mathematically complex, adaptable programs just by placing specific requirements on complying implementations and it also enables complying frameworks to improve as well as enhance the standard.

To determine how it stores as well as evaluates the floating-point integers, the Microsoft Excel was developed in IEEE 754 standard. The Institute of Electrical and Electronics Engineers (IEEE) is a worldwide organization that sets specifications for computer software as well as hardware in several areas. Floating-point numbers in binary computers are stored according to the 754 standards, which are generally used. To store and perform operations on floating-point values in a minimal amount of memory, it is essential. All PC-based microprocessors that execute floating-point arithmetic, such as those made by Intel, Motorola, Sun, and MIPS processors, use 754 benchmarks to measure their performance. Every digit or fractional number can be represented by a binary number once the numbers are stored. The fraction 1/10 can be written as 0.1 in a decimal numeral system and in binary representation the same number yields the following repeating binary decimal:

0001100110011100110011 (and so on)

This can be carried out indefinitely. Since the number is too large it cannot be represented in a finite (restricted) region. As a result, whenever this value is stored, it is rounded down to the nearest -2.8E-17 value. The IEEE 754 specification possess many drawbacks and it is classified into three categories:

- Maximum/minimum constraints
- Accuracy
- Binary numbers repetition

Maximum/Minimum constraints: Because the amount of bits accumulated is finite, the maximum and minimum numbers that a computer can process and store are finite as well. 2.250738585072E-308 and 2.2507385072E+308 are the min and max positive numbers that Excel can store.

1.3 Cases that are followed in IEEE 754

Underflow: Whenever a number is extremely small to represent, then underflow takes place. The result is zero in both IEEE as well as Excel with the exemption that Excel does not has a perception of 0 whereas IEEE has a perception of 0.

Overflow: Whenever a number is extremely large to represent, then overflow takes place. For this instance, excel has its specific representation (#NUM!).

1.4 Cases that are not followed in IEEE 754

Denormalized numbers: The exponent of zero represents the denormalized number. In such a scenario, the mantissa stores the complete number and there is no implied leading 1 in the mantissa. Thus, as the number is lowered, the reduction in accuracy is also more. Numbers down below have an accuracy of just one digit.

Example: Because of the implied leading 1, the normalised number becomes 10011001 when the mantissa represents 0011001. When a denormalized number lacks an inferred leading one, the denormalized number stays the same, i.e. 001101. There are eight significant digits in the normal number (10011001), whereas there are five significant digits in the denormalized number (11001).

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

Denormalized numbers are used to store numbers that fall below the lower limit of the normal range. Since denormalized integers have a variable number of significant digits, Microsoft does not perform this specified portion of the standard. As a result of this, computations may contain significant errors.

Positive/Negative Infinities: Whenever we divide by 0, we obtain infinity. In certain scenarios, excel does not support infinities but rather displays a #DIV/0! Error.

Not-a-Number (NaN): Invalid operations like infinity/infinity, infinity-infinity, or the square root of 1 may still be represented using NaNs, even if a programme suffers an error. On the other hand, excel displays an error message #NUM! or #DIV/0!

Precision: Sign, exponent, as well mantissa are the three forms of storing floating points in binary within a range of 65-bit. As long as the exponent is +1,023 or -1,022 (the min and max values of 2 respectively), the mantissa retains its original value.. The mantissa's finite storage area confines the vicinity of two consecutive floating-point integers. The exponent, as well as the mantissa, are retained as independent components. Thus, the measure of accuracy that may be achieved varies based on the scale of the number (the mantissa) that is being adjusted. Despite excel can retain values between 1.79769313486232E308 and 2.2250738585072E-308, it can execute only to an accuracy of 15 digits. This constraint is an outcome of precisely following IEEE 754 standards and not an Excel constraint also other spreadsheets possess this level of precision. The following is the representation of floating-point numbers, wherein exponent is the binary exponent:

X = Fraction * 2^(exponent - bias)

The leading bit is always made one by adjusting the exponent whereas fractional part of a number has been normalized. This eliminates the need for storage as well as obtaining one more precision bit. Subsequently, a "implied bit" exists and this is analogous to representation where the exponent has only one number that is modified to the left side of the decimal point; but with binary, there are only 1s and 0s and one can adjust the exponent so that the first bit is 1; thus, there is a "implied bit.". To prevent retaining negative exponents, the bias value is utilized. Numbers with single-precision possess a bias of 127, while numbers with double-precision possess a bias of 1,023 (decimal) thus Excel uses double-precision to store numbers. Exponent subtracts, division mantissa i.e., bias subtraction is not required to compute the sign bit, as well as the output should be rounded to accommodate the specified bits. Normalization is also used to illustrate operations of square root.

2. PROPOSED DOUBLE PRECISION DIVISON FLOATING OPERATION USING SINGLE PRECISION FORMAT

Fig.2 depicts the suggested architecture. It is made up of three stages that are connected by a pipeline. One dividend and one divisor are required for each 64-bit operand size, as well as the mode-control signal dp sp (double precision or dual single precision). Both input operands can be either DP operands (a whole 64-bit pair) or two parallel SP operands, as shown in Figure 2. (as two sets of 32-bit pair). An exceptional case manager and sub-normal processing are all part of the data extraction phase. Pre-fetching the divisor mantissa inverse first approximation value is also included in this mantissa division unit. Using the major operands as input, the data extraction computation computes the signs, exponents, and mantissas components in double and single precision, respectively, based on their standard forms. Data from sub-normal (SN) sources are handled using traditional methods (exceptional). Subnormal, infinity, and NaN (Not a Number) have all been tested using this functionality spanning SP-2 and DP-2 . because the DP exponent's 8 MSB bits overlap with the SP-2 exponent's 8 MSB bits (as seen in Fig 2). DP and SPs have both used it for division by zero (dbz) and zero (_z) tests. As indicated in Fig. 2, two 2:1 MUX (as seen in Fig. 1) are utilised to merge the mantissa (M1 and M2) of either DP or both SPs into one set (M1 and M2). As a consequence, the design of a bespoke datapath processing for following stage computing may be optimised, resulting in more efficient resource sharing. The LOD and dynamic left shifter, the two units that come after it in this stage, both do sub-normal processing. They normalise any mantissa that is out of the ordinary. [2] Information on dual-mode LOD and dynamic left shifter designs is given here.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

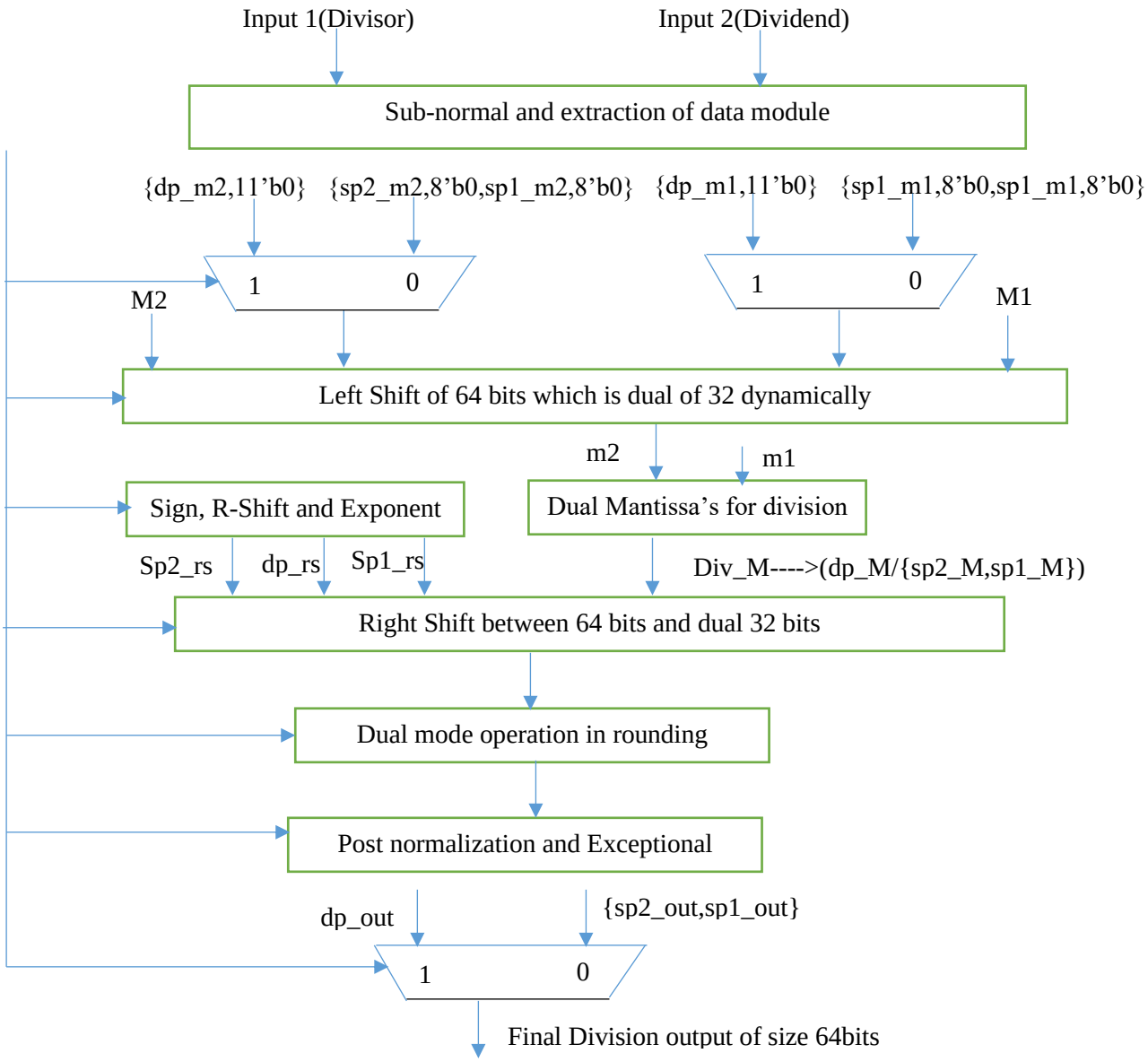
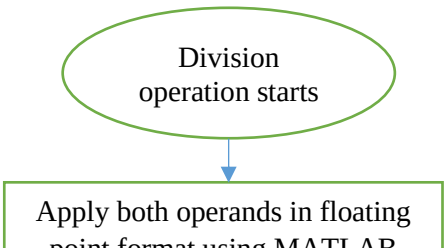


Fig.2 Proposed Architecture of double precision floating division operation using single precision

As illustrated in Fig. 2, the above procedure converts mantissas into normalised forms m1 and m2. An additional 8-bit MSB component (a1) is used to pre-compute the first estimation of the inverse of the normalised divisor mantissas (m2). For initial approximation in both DP and SP-2, the DP SP2 LUT (256x53) is utilised, as can be seen in the first stage of Fig.3, while the SP1 LUT (256x24) is only used in SP-1. At the second stage of architecture, the sign, exponent, and mantissa processing of arithmetic is done. Standard techniques are used to compute the exponent and the right shift amount. Individual calculations are done for the DP and both SPs. The FP division architecture's most critical feature is its dual-mode mantissa division processing.. A dual-mode booth multiplier is used to iteratively build this calculation .The operands used for multiplication in each stage are measured using dual operations using Finite State Machine (FSM)



JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

Figure. 3: Flow Chart for proposed double precision Floating-Point Division

2.1 Application of division that applied to Square Root Floating Point:

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

According to the research work, positive properties of square root is included in the algorithm. As the number of bits increases the drawback of the algorithm can be identified and this algorithm is also dealt with a value of 23-bit mantissa. The suggested design's algorithm is as follows:

Sign Bit: The initial number is termed as sign bit.

Computation of Exponent: An exponent bit is referred to as a number that has a biased exponent in the output. 127 is added only when the number is odd and shifts the entire sum to the right. (division by two operations).

$$E_r = \frac{E_a+127}{2} \quad \text{---------- (2)}$$

126 is added only when the number is even and shifts the entire sum to right. (division by two operations).

$$E_r = \frac{E_a+126}{2} \quad \text{----------(3)}$$

Furthermore, the shift flag is utilized to signify the mantissa by shifting it to the left as indicated by 1 bit, and this is performed before the square root algorithm.

2.2 Evaluation of Mantissa

Assume that TEMP, as well as ANS, are the two registers with 27-bit as well as 26-bit respectively. 16(100002) is considered when the square root is taken as an example. Consider the values of TEMP and ANS as 0000...0000 and 0100...0000 respectively and the iteration procedure is performed. In the primary iteration, the TEMP is loaded to Mantissa and then constrasted with the ANS value, further the outcome is based on the shift operation. Floating-point employs division as well as square root operations. Pre-processing, processing, and post-processing are the three processes of digit recurrence. Further, the signals are further subdivided by the register into various parts. quotient/root. There are no packed operands in the pre-processing block, therefore it executes division/square root in 1st iterations. Dividends, as well as a divisor, are scaled over +1 in the division to execute the division in the narrow interval and thereby enabling quotient-digit selection easier. During this phase, any subnormal can be normalized. It relies on the number of subnormal operands for cycles 1 or 2 and thereby it employs normal logic. The starting value is forwarded on to partial quotient/root and reminder digital iterations. The recurrence algorithm is a digit-iteration with an iterative component. Division as well as square root, are distinguished in the shared register by digit iteration such as (1) Computational reminder (2) partial quotient/root update as well as digit quotient/root computation. In the beginning, the overlapping technique of radix-16 square root as well as radix-64 division is used to produce the easier iteration radix-4 for every cycle such as 2 radix-4 and 3 radix-4 iteration in square root as well as in division respectively. Every iteration calculates the reminder as well as the new value's partial quotient/root. The redundant radix-2 is represented as a signed-digit for both partial reminders as well as the outcome. Positive and negative words might thus serve as a reminder and even a partial output. The quotient/root is substantially passed in the next cycle as well as the remainder will be updated. Finally, the unrounded quotient/root with last reminders are provided to the post-processing phase. There are distinct square root and division operations in Digit iteration. Digit iteration logic, on the other hand, uses a similar idea that makes use of square root and division.. This compactness will have an effect on our implementation's timing phase, because it is difficult to make three radix-4 division iterations follow a similar pattern of timed processing.. However, in the logic of digit iteration, there exists a similar concept that combined the square root as well as division. In the division iterations and the square root iterations, the update takes 2 to 3 CSA and 2 to 4 CSA respectively. Division and square root are updated using common components with 4 to 2 carry-save adders. Simultaneously it will not accommodate 3 radix 4 iterations in a single cycle and thus reduces the efficiency. The carry-save adder has been eliminated by providing only a small area. For non-redundant unrounded and reminder sign output, integration is necessary in post-processing step after round of the quotient/root. This implies that the negative to non-negative term must be subtracted to obtain the non-redundant unrounded as well as reminder sign output which is then normalized to the ultimate root or quotient. A subnormal number is the quotient at the end of the floating-point division. To get an IEEE standard output, the mantissa must be moved to

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

the right. The standard floating-point rounding step will take less time in the post-processing stage.. For division and square root/cycle algorithms, a 2 and 4 signify the digit set f2; 1; 0; +1; +2g, which is the digit set for the division and square root/cycle, respectively, as well as the number of iterations used. Every time the radix 4 algorithm iterates, the quotient or root is 2 bits. With each division 3 clock cycle, iteration radix-4 occurs and the quotient of bits with 6 in each cycle yields the divide by 3 clock cycle radix-64 outcome. When using a square root clock, two radix-4 iterations are typically used to determine the root of four bits, which in this case is equivalent to the square root of 16..

3. PROPOSED DOUBLE-PRECISION FLOATING POINT DIVISION (DPFPD)

A system of add or subtract as well as shifting operations are taken into account. Up to division restoration, the corrections of quotient bit, different signs of dividend as well as the reminder has been ignored in the following algorithm phases. Therefore, this technique eliminates the need for restoration. The DPFPD algorithm has numerous benefits as well as the division of non-positive number should be compatible in 2's complement.

Proposed Floating point division:

Process 1: Initialize temporary register value to 4''b001

Process 2: Assume a register B and is set to 24 bit 0's

Process 3: Let one operand as Divisor Register is M1 and its size 24 bit

Process 4: Let one operand as Dividend register is M2 and its size 24 bit

Process 5: Concatenate both B and M2 and result store in temporary register

Process 6: Steps 1 to 5 are repeated "m" times (where m is the number of bits in the divisor):

Process 7: If B is a 1 in the sign bit, then do M+B, where M is the total of M1 and M2 in the end procedure.

Process 8: The division should be verified in as indicated in Fig.3 with check zero exception.

Process 9: The output register's B register is combined with the reminder and M2 of the 23-bit register to produce a final division output as per IEEE 754 standard..

For negative numbers, it employs various techniques. The non-positive operand is converted into 2's complement. Perform 1's complement and then add one to the result to get the 2's complement.. Whenever both the numbers are negative, a normal NRD is carried out. Follow the following algorithm if one of the operand is non-positive and the others are non-negative:

Process 1: Initialize temporary register value to 4''b001

Process 2: Assume a register B and is set to 24 bit 0's

Process 3: Let one operand as Divisor Register is M1 and its size 24 bit

Process 4: Let one operand as Dividend register is M2 and its size 24 bit

Process 5: Concatenate both B and M2 and result store in temporary register

Process 6: If given number is positive then perform Non-recursive division (NRD) and if it is negative then perform 2's complement.

Process 7: From the division output, if remainder is not equal to 0: then increment the quotient by one. Then calculate Reminder using Reminder= divisor* quotient-dividend.

Process 8: Sign bit of quotient is 1.

Process 9: Check the Division by 0 exceptions.

Process 10: A 23-bit register value M2 is assigned to the final output, which is placed in the register output and B, and the output register is divided according to IEEE 754 format..

3.1 Square root Efficient is determined by the Proposed division Algorithm

It is not a "square root each bit" but a "partial reminder" on every iteration for not restoring the data.." Each iteration requires only 1 conventional add or subtract which further eliminates the need for hardware such as multiplexors or multipliers. It also provides a correct solution for the bit that is present at the last. As a result, there is no need for any add or precise operation to obtain the reminders. This is performed in a fast rate clock that has repetitions on basic operations.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

Phase 1: Keep the register reminder as (Reminder>(i+1)) &3) if the reminder register is equal to zero or more than zero. –((Quotient>(i+1)) &3) is the value for the reminder. +((Quotient2) |1) is given as the reminder value once again.

Phase 2: Otherwise, use (Reminder>(i+1)) &3) as the reminder. Specify the reminder values of Reminder+((Quotient2) |3)

Phase 3: When the reminder value is higher than or equal to zero, the quotient is stated as ((Quotient<<1) |1).

Phase 4: Or else Quotient value is ((Quotient<<1) |0)

Phase 5: When the value of the reminder is less than zero, the remainder is supplied as Reminder+((Quotient <<1) |1)).

The M2 register and register remainder provide square root value as well as remainder respectively and obtaining the appropriate bit in every iteration is included in this algorithm. Every iteration is predicated on previous iteration depending on add/subtract sign of the result. The remainder is generated for each and every iteration even if the value is non-positive. Finally, the last remainder will be taken by adding the partial.

4. RESULTS AND DISCUSSION

For instance, a common technique based on this method may be used to demonstrate division and the square root, which is more optimistic. It is now easier to get away with using less slices and LUTs. The existing as well as proposed systems use 1125 slices and 1507 slices respectively. As a result, the density packages are improved by almost 23.18% and 5.122 ns is the maximum delay time. When calculating for application of division i.e square root, the time consumed is 7.192 ns and it is the time consumed to perform square root computation as shown in Table.1. As a result, the task can be completed in very less time and thereby increases the speed of operation. Further, the code is then programmed into the FPGA kit. Both proposed as well as existing systems make use of the Xilinx tool with Virtex-5 Device helps in the synthesis of design. The "Device Summary" is synthesized as well as displayed by the program. The utilization of "Target Device" is synthesized and reported by design as well as "Timing of the crucial path" is calculated and further displays the time.

Table I: Comparison between Existing and Proposed floating point division

Parameters	Existing [1,2,3]	Method for Proposed
LUT'S	1245	1507
Slice Registers	66416	1125
Power consumed in Watts	90.47	2.52104497
Frequency in Maximum MHz	124.356	195.243
Delay in ns	38.22	5.122

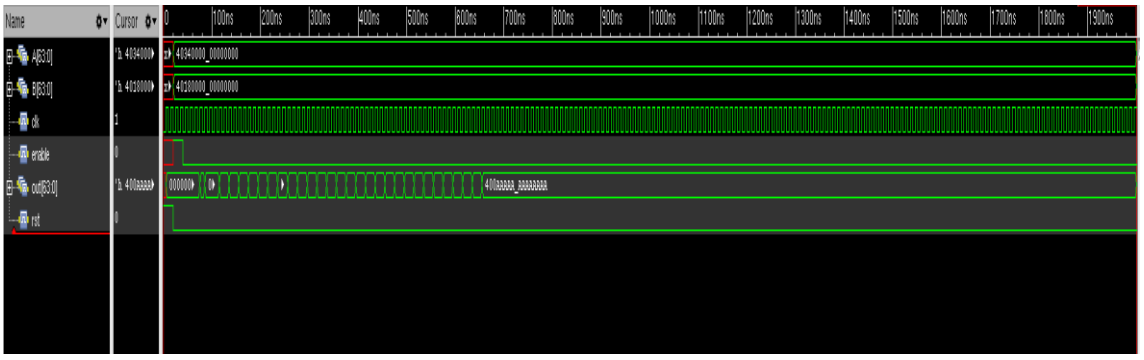


Fig.4. Simulated results for input operands 20 and 6 and its division output

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

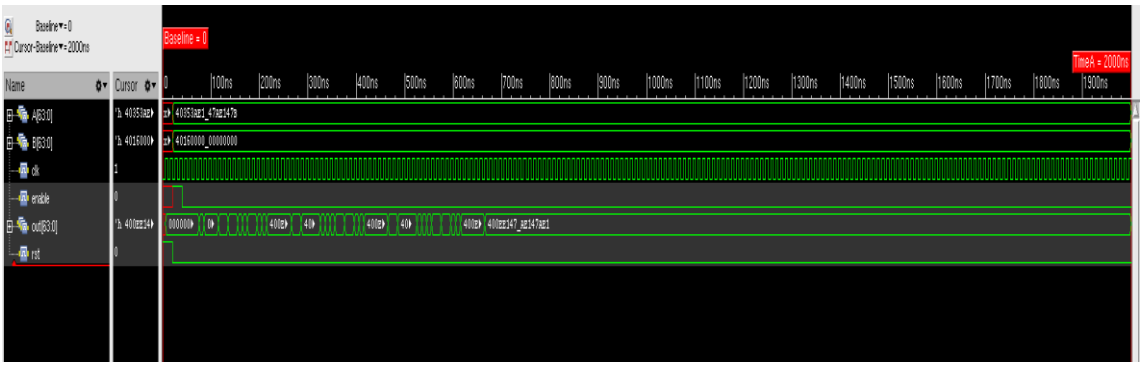


Fig.5. Simulated results for input operands 21.23 and 5.5 and its division output

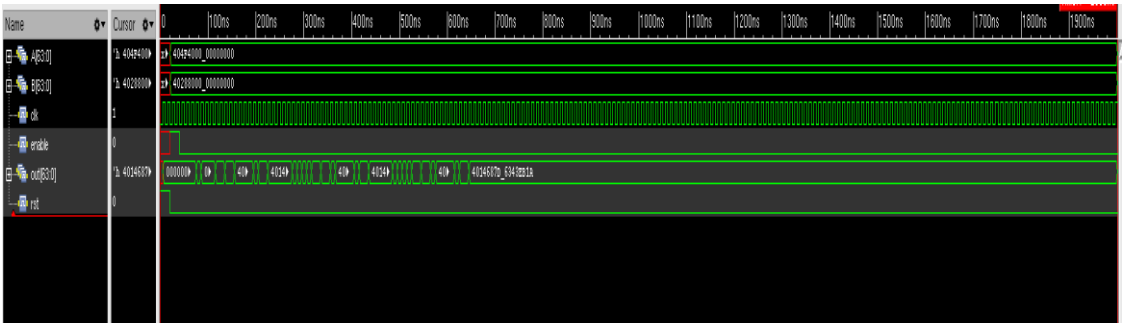


Fig.6. Simulated results for input operands 62.5 and 12.25 and its division output

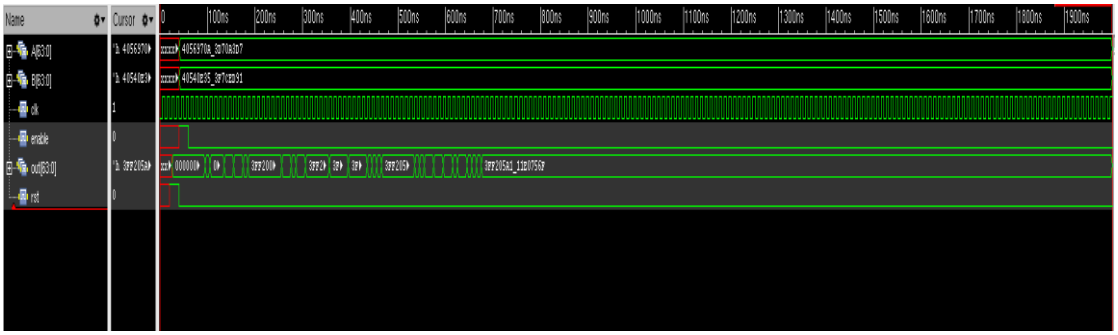


Fig.7. Simulated results for input operands 90.36 and 80.222 and its division output

The proposed design is successfully simulated and synthesized using Cadence software tool and validated for different floating numbers and some simulated output results are shown in Fig.4 to Fig.8. For example, the Fig.8. the inputs are A=111.6482 and its hexadecimal value is 405be97c1bda511a and B=2.2361 and its hexadecimal value is 4001e3886594af4f and final division output is A/B= 49.9299 and its hexadecimal value is 4048f7063d4b9830. The Fig.9. is Clock tree synthesis after elaboration of complete design and Fig.10 is CTS with elaboration of floating division and Fig.11 is RTL diagram.

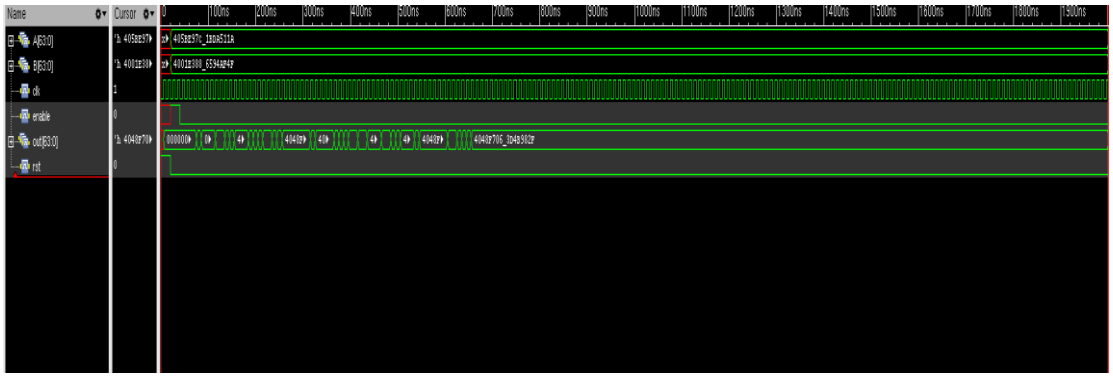


Fig.8. Simulated results for input operands 111.6482 and 2.2361 and its division output

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

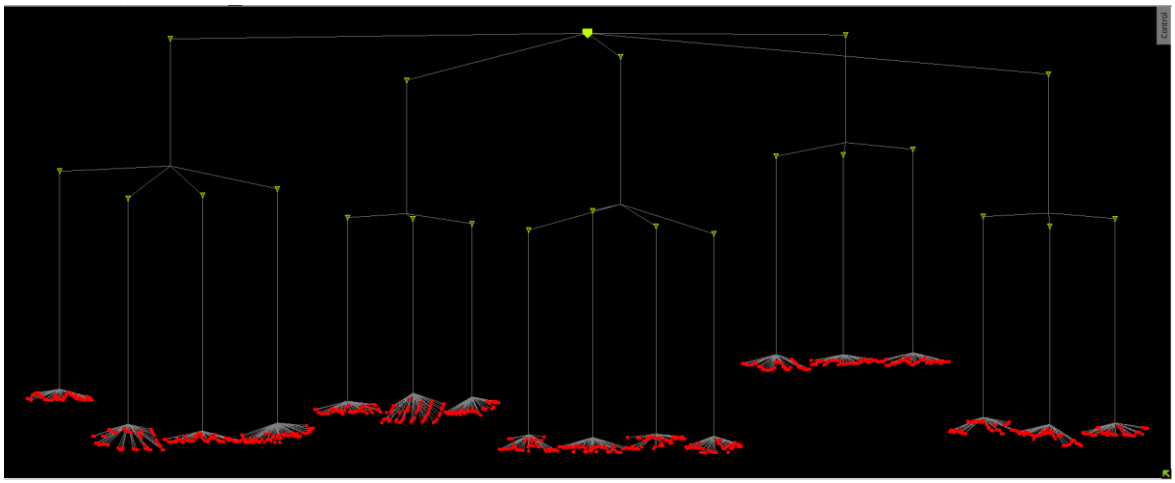


Fig.9. Clock Tree Synthesis (CTS) of proposed floating point division operation

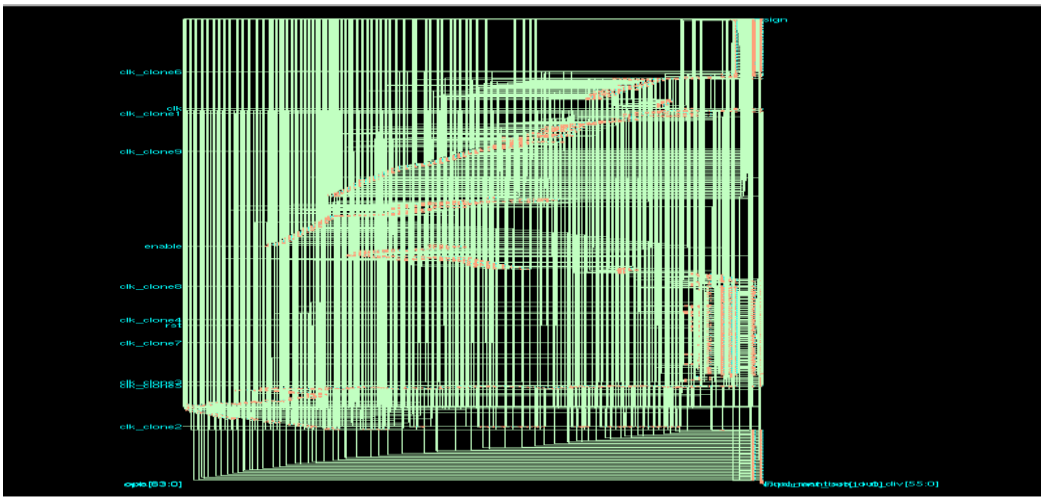


Fig.10. Elaborated Clock Tree Synthesis (CTS) of proposed floating point division operation with clock separation for each internal modules

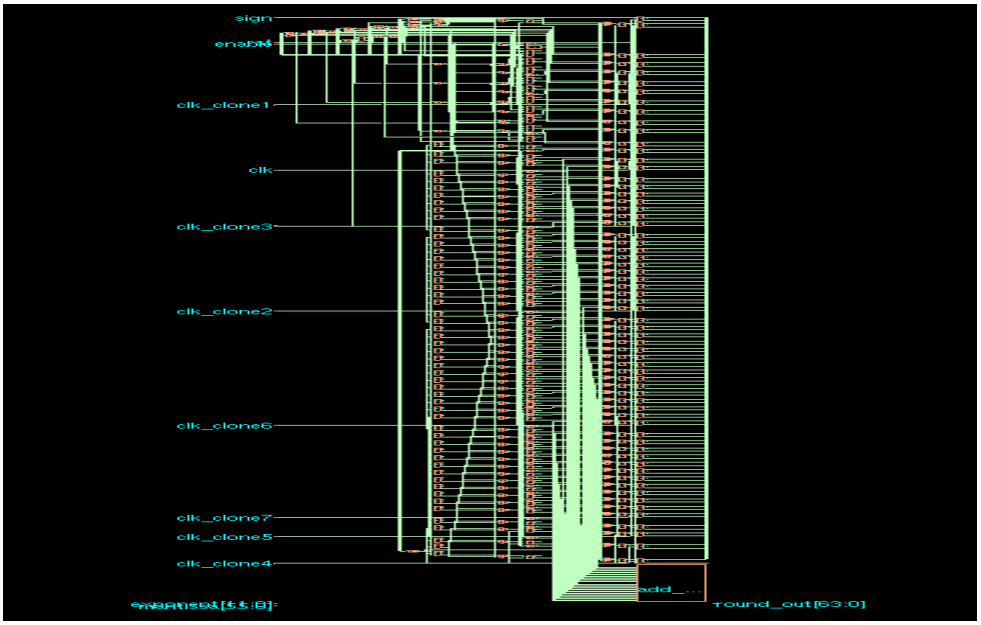


Fig.10. Elaborated RTL diagram of proposed floating point division operation with input and output ports of each internal modules

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

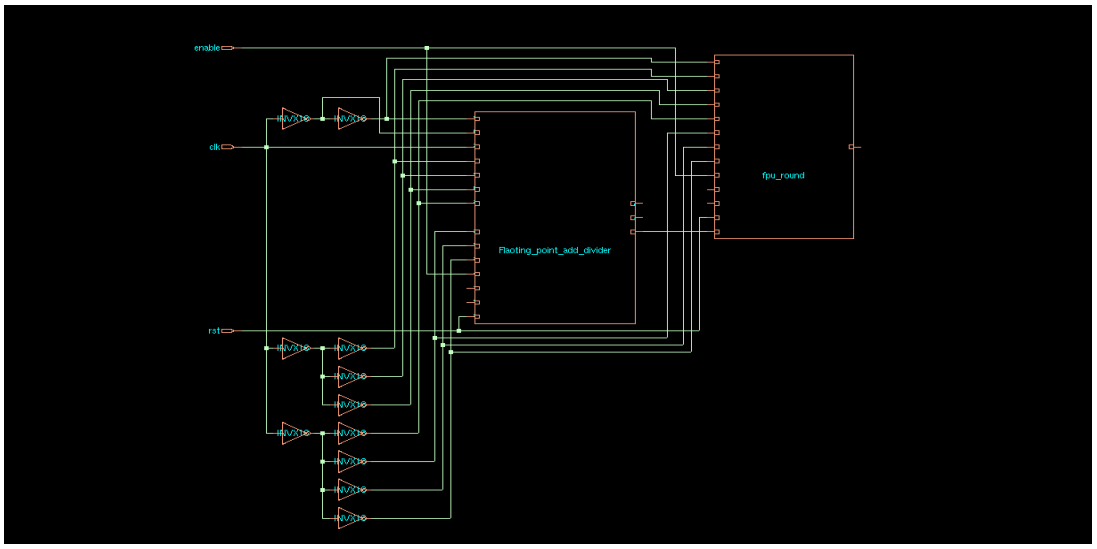


Fig.11. Top level Schematic elaborated RTL diagram of proposed floating point division operation with CTS

Setup mode	all	reg2reg	default
WNS (ns):	1.462	1.462	7.777
TNS (ns):	0.000	0.000	0.000
Violating Paths:	0	0	0
All Paths:	2238	1445	1256

Table.2 Timing summary of proposed floating point division and its includes worst positive and negative delay in setup timing.

Table 2 shows a technologically independent comparison using Gate-Count for area, FO4-delay for timings, cycle counts for latency and throughput, and a unified metric Area Period (FO4)Throughput (in clock cycle) for comparison (which should be smaller for a better design). Isseven et al. [1] introduced an iterative DPdSP division architecture without subnormal support based on the Radix-4 SRT division algorithm. Isseven et al design requires much more area and has a lower Area Period Throughput than the recommended architecture. [2], like the previous study [2,], requests for a big area with a low Area Period Throughput. This design is likewise infeasible due to the single-cycle implementation of [2]. As a result, the present architecture proposal is superior in terms of design metrics.

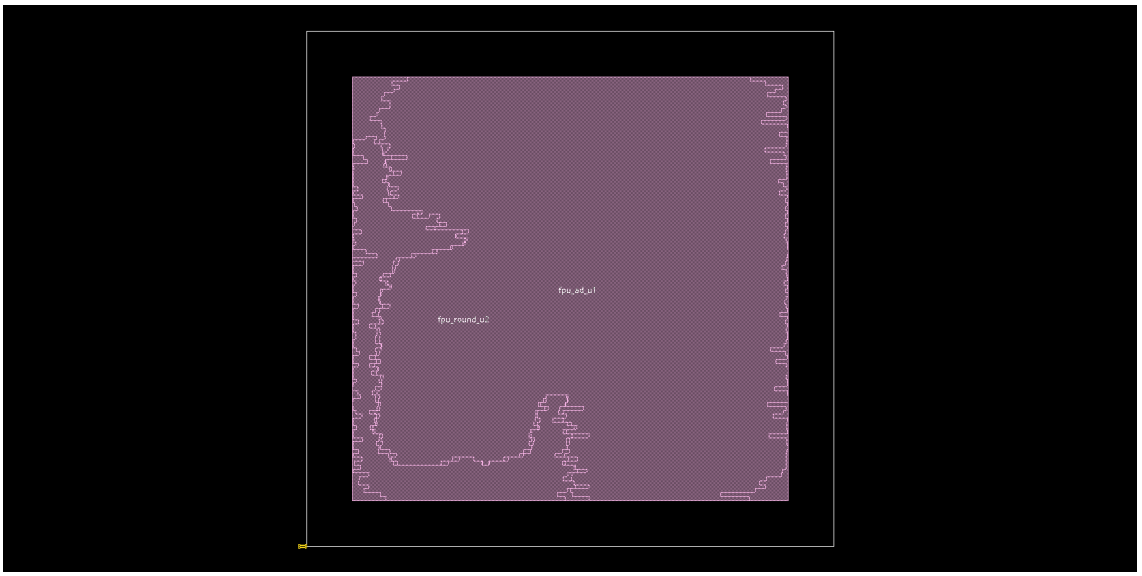


Fig.12. Chip level design of proposed floating point division operation

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2302-2317
https://publishoa.com
ISSN: 1309-3452

The carry skip adder is used than a carry-save adder in the proposed system since the carry will surely propagate. There will be a lot of people to use this adder, which means a lot of hardware. Whenever the outcome is high, the output will be added at the last and whenever the output is low it will not be propagated. Furthermore, the proposed strategy will maximize the application of hardware and thereby reduce the number of adders utilized. No issue occurs because of the application of floating-point. Systems such as computers or electronic devices are utilized to execute as well as implement. The division is more significant in arithmetic operations and to boost ALU performance, the proposed system used several iterative architectures. The technique of logical shifting is used to obtain an expected output. Silicon dye is used to reduce the speed, area and thereby provides efficiency of 43.38%. The proposed system enhances the performance by minimizing cost as well as size. Very little time is taken to compute quick and accurate calculations. 10.95% is the power dissipated, and it ignores the thermal absorbers as well as fans in the cooling process. As a result, when compared to previous systems the proposed square root/division will be more efficient. This research work mainly deals with the problems that arise during the implementation of the FP splitter. To resolve various issues about the floating-point divider implementation, the frequency as well as interlock distance of divide instructions in SPECfp92 standards were computed along with other significant measurements. The initial query was whether a hardware FP split unit in a system is required or not. The analysis proves that the CPI cost is more than 0.50 for the slowest hardware divider with $l > 60$ cycles. This signifies that some type of hardware division is necessary to attain acceptable system performance. The ability of the compiler to enhance the performance of the system due to the division was then examined. The compiler's capacity to minimize, divide latency by 30% was demonstrated in this research work. As a result of optimization from the fundamental compiler at the O2 level, the performance gain was increased. Furthermore, optimization resulted in only a minor improvement. After that, the impacts of several problems on divide latency were examined. It was noticeable that urgency u was increased by enhancing the number of instructions received for every cycle. CPI increased by 38%, 94%, and 120% whenever the number of instructions transmitted for every cycle was increased to 2, 4, and 8 respectively. By broadcasting two or more instructions every cycle, big issue machines may take use of instruction-level parallelism in a variety of applications. Whereas this lowers the processor's base CPI as it displays the functional unit delays to a greater extent. The concept of utilizing the current FP multiplier by utilizing the multiplication-based division technique was then studied. The CPI cost ranges between 0.025 and 0.040 for a divide latency l of approximately 12 cycles as per the observations. The architectural risk is relatively rare because of the consequence of low-frequency divide operations, as well as multiply instructions that take place among divide output execution as well as implementation. The CPI cost is minimal when multiplier is distributed as well as updated to additionally execute the division. The developer must examine the impacts of multiplier's latency which might affect the cycle time. The requirement of conversion, as well as on-the-y rounding was the last aspect addressed in this work. Lack of on-the-y rounding as well as the conversion does not represent a significant proportion of additional CPI for divide latencies that are more than 10 cycles and it is not necessary as shown in Fig.12 and Fig.13. Even though division is a comparatively rare operation in floating-point-intensive applications, it fails to implement and thereby reduce the performance of the system. This research work tries to describe the basic factors of developing an FP divider in hardware and thereby addressing various architectural problems associated with FP division.

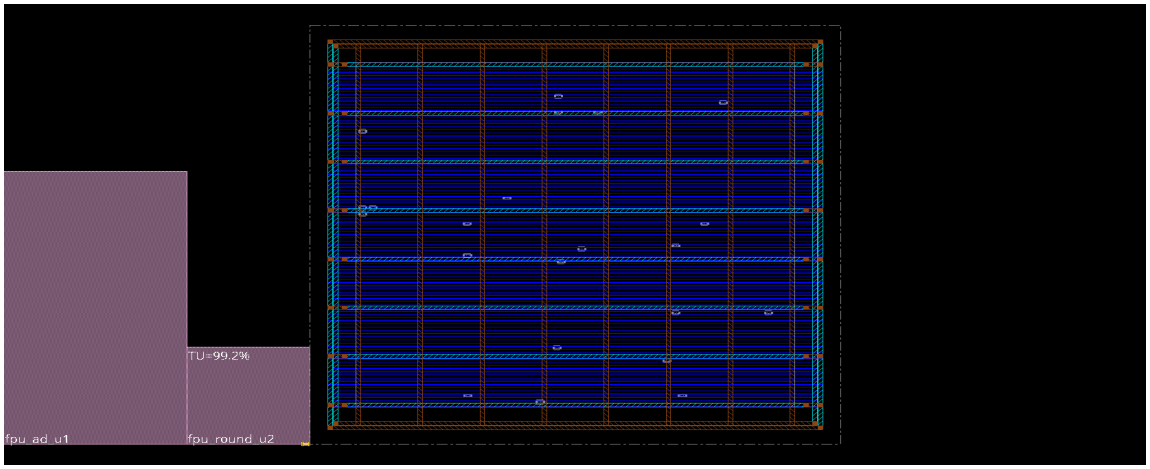


Fig.13. Floor Plan and its routing of proposed floating point division operation

CONCLUSION

For DP-FP division, this work provides a dual-mode iterative architecture that may be used in two different ways. DP and two-parallel SP floating point divisions may be managed by it. Dual-mode Radix-4 Modified, brand new in the market For dual-mode processing, a booth multiplier with decreased overhead has been proposed. In numerous critical design aspects, the dual-mode architecture suggested here outperforms the prior technology. Although floating-point division and subtraction are the most common arithmetic operations in most scientific applications, in some state-of-art on the design of an asynchronous floating-point division. The present AFPA designs use dual-rail coding, which necessitates a large implementation area, but they are faster and use less power than their baseline FPA counterparts. This survey research compared several performance aspects with their respective baseline FPA for all four extant AFPA designs. Because all of the already developed divisions has got characteristics in terms of performance, an absolute comparison of all of them is impossible; however, these are the only AFPA implementations available in the literature, and they show that the asynchronous design technique has the potential to improve AFPA performance. Based on obtained results, it is observed that 23% improvement in area, 16% improvement in LUT's utilization and 9% improvement in throughtput.

REFERENCES

1. Srujana B Malkapur et al, "Design of Generic Floating Point Pipeline Based Arithmetic Operation for DSP Processor", Proceedings of the Second International Conference on Inventive Research in Computing Applications (ICIRCA-2020), 978-1-7281-5374-2/20/$31.00 2020 IEEE.
2. Manish Kumar Jaiswal et al, "SP48E Efficient Floating Point Multiplier Architectures on FPGA", 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems, DOI:10.1109/VLSID.2017.106, 2380-6923/16 $31.00 2016 IEEE.
3. S¨uleyman Savas et al, "Efficient Single-Precision Floating-Point Division Using Harmonized Parabolic Synthesis", 2017 IEEE Computer Society Annual Symposium on VLSI, DOI:10.1109/ISVLSI.2017.28, 2159-3477/17 $31.00 2017 IEEE.
4. Soumya Havaldar et al, "Design Of Vedic IEEE 754 Floating Point Multiplier", IEEE International Conference On Recent Trends In Electronics Information Communication Technology, May 20-21, 2016, India, 978-1-5090-0774-5/16/$31.00 2016 IEEE.
5. Spoorthi M. N. et al, "Decimal Multiplier with Improved Speed Using Semi-Parallel Iterative Approach", 978-1-7281-9369-4/20/$31.00 2020 IEEE.
6. Farhad Merchant et al, "Efficient Realization of Table Look-up based Double Precision Floating-Point Arithmetic", 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems, DOI: 10.1109/VLSID.2016.113, 978-1-4673-8700-2/16 $31.00 2016 IEEE.
7. Deeksha Kiran D K et al, "VLSI Implementation of Braun Multiplier using Full Adder", International Conference on Current Trends in Computer, Electrical, Electronics and Communication (ICCTCEEC-2017), 978-1-5386-3243-7/17/$31.00 2017 IEEE.
8. Sangeetha P et al, "Comparison of Braun Multiplier and Wallace Multiplier Techniques in VLSI", Fourth International Conference on Devices, Circuits and Systems (ICDCS'18), 978-1-5386-3476-9/18/$31.00 2018 IEEE.
9. Sneha Hiratkar et al, "VLSI Design of Analog Multiplier in Weak Inversion Region", 978-1-5090-0396-9/16/$31.00 2016 IEEE.
10. Shuchi Nagaria et al, "Efficient FIR Filter Design using Booth Multiplier for VLSI Applications" 978-1-5386-4491-1/18/$31.00 2018 IEEE.
11. Minal R. Ghonge et al, "VLSI Design of Fixed Width 2's Compliment Multiplier", 2017 International Conference on Innovations in information Embedded and Communication Systems (ICIIECS), 978-1-5090-3294-5/17/$31.00 2017 IEEE.
12. Siva.S.Sinthura et al, "Implementation and Analysis of different 32-bit multipliers on aspects of Power, Speed and Area", Proceedings of the 2nd International Conference on Trends in Electronics and Informatics (ICOEI 2018), 978-1-5386-3570-4/18/$31.00 2018 IEEE.

13. Shirin Pourashraf et al, "±0.25-V Class-AB CMOS Capacitance Multiplier and Precision Rectifiers", IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, DOI: 10.1109/TVLSI.2018.2881249, 1063-8210 2018 IEEE.
14. Jalaja S et al, "Design of low power based VLSI architecture for constant multiplier and high-speed implementation using the retiming technique".
15. Zhibin Zeng et al, "Wideband Common-Mode Suppression Filter With Compact-Defected Ground Structure Pattern", IEEE TRANSACTIONS ON ELECTROMAGNETIC COMPATIBILITY, VOL. 57, NO. 5, OCTOBER 2015, DOI:10.1109/TEMC.2015.2440424, 0018-9375 2015 IEEE.
16. Anand Mehta et al, "Implementation of Single Precision Floating Point Multiplier using Karatsuba Algorithm", 978-1-4673-6126-2/13/$31.00 2013 IEEE.
17. Manish Kumar Jaiswal et al, "Dual-Mode Double Precision / Two-Parallel Single Precision Floating Point Multiplier Architecture", 978-1-4673-9140-5/15/$31.00 2015 IEEE.
18. Ushasree G et al, "VLSI Implementation of a High-Speed Single Precision Floating Point Unit Using Verilog", Proceedings of 2013 IEEE Conference on Information and Communication Technologies (ICT 2013), 978-1-4673-5758-6/13/$31.00 2013 IEEE.
19. Manish Kumar Jaiswal et al, "Configurable Architecture for Double / Two-Parallel Single Precision Floating Point Division", 2014 IEEE Computer Society Annual Symposium on VLSI, DOI:10.1109/ISVLSI.2014.45, 978-1-4799-3765-3/14 $31.00 2014 IEEE.
20. Hao Zhang et al, "New Flexible Multiple-Precision Multiply-Accumulate Unit for Deep Neural Network Training and Inference", DOI:10.1109/TC.2019.2936192, IEEE, 0018-9340 2019 IEEE Transactions on Computers.
21. Yuxuan Wang et al, "GH CORDIC-Based Architecture for Computing Nth Root of Single-Precision Floating-Point Number", IEEE TRANSACTIONS ON VERY LARGE-SCALE INTEGRATION (VLSI) SYSTEMS, DOI: 10.1109/TVLSI.2019.2959847, 1063-8210 2019 IEEE.
22. Hwa-Joon Oh et al, "A Fully-Pipelined Single-Precision Floating-Point Unit in the Synergistic Processor Element of a CELL Processor", 2005 Symposium on VLSI Circuits Digest of Technical Papers, 4-900784-01-X.
23. Addanki Puma Ramesh et, "An FPGA Based High Speed IEEE-754 Double Precision Floating Point Multiplier Using Verilog", 978-1-4673-5301-4/13/$31.00 2013 IEEE.
24. Alexander Godunov et al, "Algorithms for Calculating Correctly Rounded Exponential Function in Double-Precision Arithmetic", IEEE TRANSACTIONS ON COMPUTERS, VOL. 14, NO. 8, AUGUST 2015, 0018-9340 2019 IEEE.