JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Approximate Floating Point Divider Design

[1]Katta Keerthi, [2]Mahankali Uma Devi, [3] Koppolu Praveen Kuma,[4]Kotha Venkat Praveen
[1]Student of ECE, [2] Student of ECE, [3] Student of ECE,[4] Student of ECE
R.V.R &J.C. College of Engineering

Abstract : Approximate computing is a promising solution to design faster and more energy efficient systems, which provide an adequate quality for a variety of functions. Division, in particula r floa ting point division is one of the most important ope rations in multimedia applications, which has been implemented less in hardwa re due to its significant cost and complexity. Approximatio n is nothing but approximating the output by reducing the constraints in it. There are several different strategies for approxi mate computing like Approximating the circuits, Approximating the software, Approximating the storage etc. This project uses Approximating the Circuit strategy. Approximating the circuits includes approximating the adders subtractors, multipliers, dividers etc. In this project, we will approximate the floating point divider using the concept of Approximate computing. The focus of this work will be on implementing a Verilog-based Approximate divider for floating point divider. Both accuracy and power will be calculated. We expect an improvement in accuracy and reduction in power.

***IndexTerms* - Approximate computing, Accuracy, Power efficient, Floating point representation, Division.**

I. INTRODUCTION

In today's world of technologies increasing usa ge of energy consumption by computer systems is a serious problem that needs to be addressed. Computer systems will be processing massive amounts of data o this requires more power. Making the models energy efficient will save a lot of power. Simply said, energy efficiency is nothing but using less energy for doing the same function, in other words, eliminating energy wa ste. The power consumption of current consumer electronics has become a crucial design concern as the speed, mobility and compactness of these products have increased.

Numerous approaches have been introduced in the past to improve energy efficiency across circuit, architectural, and system abstraction levels. An approach to decrease energy and power usage in the system model involves approximating the hardwa re. This method of approximation is suggested for designing the IEE754 floating point divider. The utilization of floa ting point blocks has grown in various applications, including arithmetic architecture, due to their low production cost and ease of use. In ar ithmetic digital design, the floatingpoint divider is considered a highly complex logic block.

Current sensory data algorithms tend to be statistical in nature, which makes them able to perform approximate computations. In embedded devices, precision would be repla ced by energy efficiency as approxima te computing continues to ga in popularity due to its low energy consumption. Attention has la tely focused on the design of approximation units, e.g. adder and multiplier. But one of the key operations in multimedia applications is division, especially with regard to floating point divisions.

In recent work it has been attempted to speed up the division in order to allow approximation. In this study [1], they propos e e FPAD, a novel approximation floating point divider based on the basic mathematical method of breaking down the mantissa division into a sequence of shifts and addition operations. A wide range of services is also offered by them. The FPAD is designed with a variety of errors and tradeoffs with the PDP. At each level of the proposed divider, a limit is set for inaccuracy. To increase the different areas, power, and delay, the number of adders at each levelm a y be changed.

In this research [2], they are offering a revolutionary adjustment of the approxima tion coefficient that splits floating point data in an efficient and precise wa y. CADE removes the costly mantissa division and repla ces it with a single subtraction between two input operands mantissa. The tuning procedure consists of defining the amount of error, then correcting it by using both i n p u t mantissa's initial N bits. The CADE may be configured to vary degrees of precision depending on the N va lue. According to our findings, CADE is the first floating point divider to include a new GPU knob for configuring the amount of approximation at runtime based on the application/user accuracy requirements.

In this paper, the main a im is to reduce the error and power consumption of the floating point divider. By studying different algorithms for floating point division and the main brief from it is the floating point dividers mantissa consumes more power and energy and is the ma in block that is responsible for the obtained error rate. This leads to finding a different algorithm for floating point division, which has fewer errors, power and energy than the current ones. In order to reduce the error rate, approximate calculations are performed. Since the mantissa is the ma in block of implementation so we primarily focused on mantissa block and performed different operations to analyze the error, power, utilization and dela y of system. The a im of this work is to approximate the floating point divider, analyze and compare the power, accuracy and error rates of the proposed model.

II. METHODOLOGY

2.1 IEEE-754 FLOATING POINT REPRESENTATION

In order to separate decima l and floating point va lues, the floating point representation of a number represents where the po int should be indicated. In order to allocate the va lues mentioned, a specific number of bits is allocated to the representation. It's where the float point has to float rather than being a fixed point. This is one of the ma in differences in representation by fixed and float points. For floating point, we use the IEEE-745 format because it conta ins various set formats and operation modes that are widely used in a number of hardware applications.

IEEE-754 floating point representation can be represented in two different precisions SINGLE PRECISION [32 bit] and DOUBLE PRECISION [64 bit]. The three main components of IEEE-754 floating point representation are SIGN BIT, EXPONENT and MANTISSA. Sign bit is a single bit value that represents whether the given number is positive or negative. If th e bit va lue is '0' then the number is said to be positive number. Exponent bits are the bits that represent both positive and n ega tive values that describe where the floa ting point (or decima l point) needs to be pla ced. Represented as power of 2. Mantissa bits are the bits that describe the floating point numbers of binary values, always positive. This part holds the main bits of the number.

The precision ma inly differs by its size i.e. the bit size. In single precision the total float point representation is a 32 -bit representation. As said earlier they contain sign, mantissa and exponents bits. This contains 1 sign bit, '23' mantissa bits and '8' exponent bits. This precision is also called binary 32. In the 24 -bit mantissa, the MSB (most significant bit) is always '1' so this bit need not to be stored. So though theoretically the mantissa is a 24 -bit va lue but technically mantissa is a 23-bit value. So, this MSB '1' is called as "Hidden bit". As the exponent is an 8-bit value the range of value it can hold is 2 ^ (-126) to2^ (127).

In double precision the total floa t point representation is a 64 -bit representation. The total bit size of double precision is twice that of single precision, that is why this is called double precision. As said earlier they contain sign, mantissa and exponent bits.The below dia gram depicts the distribution of the three in a 64 -bit single precision representation. This conta ins '1' sign bit, '52' mantissa bits and '11' exponent bits as shown. This precision is also called as binary64.In the 53 -bit mantissa, the MSB (most significant bit) is always '1' so this bit need not to be stored. So though theoretically the mantissa is a 53 -bit va lue but technically mantissa is a 52-bit value. So, this MSB '1' is called as "Hidden bit". As the exponent is an 11 -bit value the range of va lue it can hold is 2 ^ (-1022) to2^ (1023).

For converting a given decimal number into a floa ting point representation we need to follow certa in steps. Firstly, we need to convert the decima l number into binary number then normalize the binary value resulting in an exponent va lue (in integer) and mantissa value (in binary). In the given formula , S represents the sign bit, M represents the mantissa and E represents the exponent value. We use a formula as shown

$$\text{Number} = (-1)^{(s)} * (1.M) * 2^{(E-127)}$$

2.2 FLOATING POINT DIVISION

Let us consider two numbers X & Y ,floating point representation of these numbers is as follows: X =(-1)Sx*(1.Mx)*2(Ex-127) and Y =(-1)Sy*(1.My)*2(Ey-127) If we divide the two equations X/Y the resultant is Z =(-1)Sz*(Mx)/(My)*2(Ey-Ex).

The following are few steps to perform generalized floating point division:

Step-1: Output of sign bit is XOR of the input sign bits i.e. Sz =Sx xor Sy.

Step-2: Exponent is Ex =(Ey-Ex) +bias. [for single precision the bias value is +127].

Step-3Mantissa Division Mz =(Mx)/(My). This is the general binary division of the two input mantissas.

Normalization: This is the case where the final output is normalized. We need to normalize by moving the mantissa towards the left and reducing the final exponent value. This step is not mandatory, if required we need to perform this operation.

Percentage Error:

$$\text{Percentage Error} = (\text{Actual value} - \text{Obtained value})/\text{Obtained value} * 100\%$$

III. PROPOSED ALGORITHMS

IEEE 754 floating point number ($X_{31}, X_{30}, \ldots X_0$) consists of 3 parts i.e. sign(S_{31}), exponent($E_{30}, E_{29}, \ldots E_{23}$) and Mantissa ($M_{22}, M_{21}, \ldots M_0$). The operation that we perform is floating point division.

3.1 ALGORITHM-1: DIVIDE & SUBTRACT

Figure-3.1 shows a block diagram for the proposed algorithm -1. This illustrates the whole system architecture in which operations are carried out for each block.

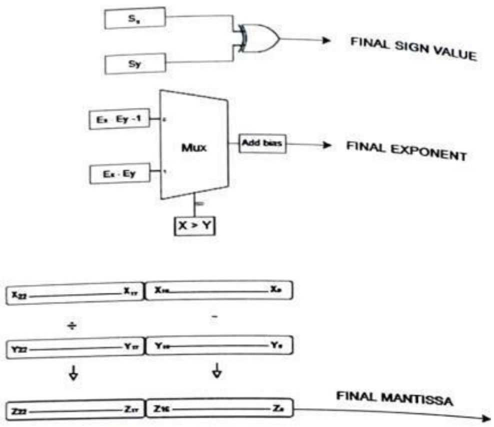

Figure-1: Block diagram of Algorithm-1

Sign Bit: The sign bit operator of the divider output is "Exor". We perform Exor operation of both the inputs i.e. $S_Z=S_X$ XOR S_Y gives the sign bit value of the divider output.

Exponent Bit: The exponent bit is deducted by comparing the input mantissas. During this comparison we will be having two cases -

CASE 1:$M_X < M_Y$: In this case we subtract the two input exponent va lues and then a ga in subtract 1 from the obta ined va lue. And the key point here is we should always add the bia s to the exponent to get the stored va lue. So, the final exponent va lue of the divider output is $E_Z=E_X - E_Y$ -1.

Case 2: $M_X>M_Y$: In this case we just need to subtract the two input exponent values. And the bias va lue which is + 127 should be added to the exponent to get the stored value. So, the final exponent value of the divider output is $E_Z = E_X - E_Y$

Mantissa Bit: As mentioned earlier in the notation of mantissa bits, the mantissa bits are (M22,M21,….M0). In the proposed algorithm, we consider the first eight input bits are chosen for division and the rest of the 15 bits are chosen for subtract ion. This is done to reduce the hardwa re and power consumption. The 8 - bit division is done by considering the hidden bit also. As the division operation consumes a lot of power and indeed requires a lot of hardwa re to implement the 32 - bit division. So, the final operation is $(M_{Z22}M_{Z21}....M_{Z0}) = (M_{X22}M_{X21}....M_{X17})/ (M_{Y22}M_{Y21}....M_{Y17}) +((M_{X16}M_{X15}...M_{X0})- (M_{Y16}M_{Y15} M_{Y0}))$. The final divider outputs are obtained from the proposed "Algorithm 1".

3.2 ALGORITHM-2:DIVIDE & ALTERNATE '10'

Figure-3.2 shows a block diagram for the proposed algorithm -1. This illustrates the whole system architecture in which operations are carried out for each block.

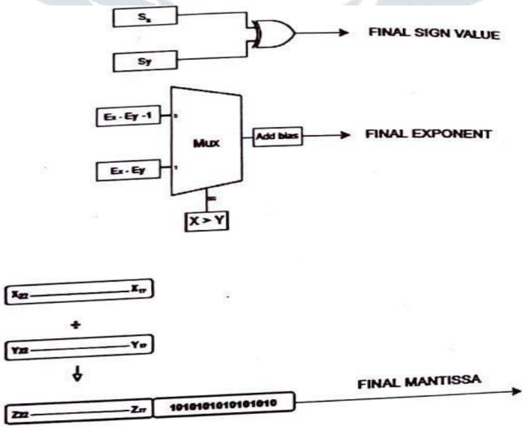

Figure-2: Block diagram of Algorithm-2

Sign Bit: The sign bit operator of the divider output is "Exor". We perform Exor operation of both the inputs i.e. $S_Z=S_X$ XOR S_Y gives the sign bit value of the divider output.

Exponent Bit: The exponent bit is deducted by comparing the input mantissa. During this comparison we will be having two cases -

CASE 1:$M_X < M_Y$: In this case we subtract the two input exponent va lues and then a ga in subtract 1 from the obta ined va lue. And the key point here is we should always add the bia s to the exponent to get the stored va lue. So, the final exponent va lue of the divider output is $E_Z=E_X - E_Y$ -1.

Case 2: $M_X > M_Y$: In this case we just need to subtract the two input exponent values. And the bias va lue which is + 127 should be added to the exponent to get the stored value. So, the final exponent value of the divider output is $E_Z = E_X - E_Y$.

Mantissa Bit: As mentioned earlier in the notation of mantissa bits, the mantissa bits are (M_{22}, M_{21},….M_0). In the proposed algorithm, we consider the first eight input bits to be chosen for division and the rest of the 15 bits are not considered fo r any of the operation. The 8 - bit division is done by considering the hidden bit also. This is valid as the output of the divider is mostly based on the first few MSB va lues of the mantissa. And for the final result the rest of the bits i.e. the bits left after storing t he division output are made a lterna te '10' (1010101...). This is done to reduce the hardwa re and power consumption. As the division opera tion consumes a lot of power and indeed requires a lot of hardware to implement the 32 - bit division. So, the final operation is $M_{Z22}M_{Z21}....M_{Z0}$) = ($M_{X22}M_{X21}....M_{X17}$)/ ($M_{Y22}MY_{21}$ M_{Y17}) +"1010101010101010". The final divider outputs are obtained from the proposed "Algorithm 2".

33 ALGORITHM-3: DIVIDE & ZERO

Figure-3.3 shows a block diagram for the proposed algorithm -1. This illustrates the whole system architecture in which operations are carried out for each block.

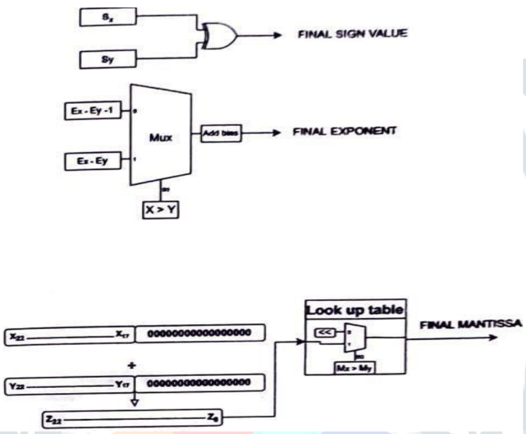

Figure-3: Block diagram of Algorithm-3

Sign Bit: The sign bit operator of the divider output is "Exor". We perform Exor operation of both the inputs i.e. $S_Z = S_X$ XOR S_Y gives the sign bit value of the divider output.

Exponent Bit: The exponent bit is deducted by comparing the input mantissa. During this comparison we will be having two cases -

CASE 1:$M_X < M_Y$: In this case we subtract the two input exponent va lues and then a ga in subtract 1 from the obta ined va lue. And the key point here is we should always add the bia s to the exponent to get the stored va lue. So, the final exponent va lue of the divider output is $E_Z = E_X - E_Y - 1$.

Case 2: $M_X > M_Y$: In this case we just need to subtract the two input exponent values. And the bias va lue which is + 127 should be added to the exponent to get the stored value. So, the final exponent value of the divider output is $E_Z = E_X - E_Y$.

Mantissa Bit: As mentioned earlier in the notation of mantissa bits, the mantissa bits are (M_{22}, M_{21},….M_0). In the proposed algorithm, we consider the first eight input bits to be chosen for division and the rest of the 15 bits are not considered fo r any of the operation. The 8 - bit division is done by considering the hidden bit also. This is valid as the output of the divider is mostly based on the first few MSB va lues of the mantissa. And for the final result the rest of the bits i.e. the bits left after storing the division output are made zeros (0000000...). This is done to reduce the hardwa re and power consumption. As the division operation consumes a lot of power and indeed requires a lot of hardwa re to implement the 32 - bit division. We then introduced a look up table into the algorithm to provide operational range without requiring more time and la rge gate count which results in reduc ing the cost of operation. This look up table includes a comparison of mantissa for further operation.

We have two cases for mantissa operation-

CASE-1 Mx < My: In this case we use the output from the previous block as it is without any further operation and consider this as the finalmantissa of the divider output.

CASE-2 Mx > My: In this case we use a shift operator to the output obtained from the previous block i.e. we shift the number 1 bit towards the left, and then this output is considered as the finalmantissa of the divider output.

So, the final operation is

$$(M_{Z22}M_{Z21}....M_{Z0}) = (M_{X22}M_{X21}....M_{X17}) / (M_{Y22}M_{Y21}..M_{Y17}) + \text{"0000000000000000"}$$

The final divider outputs are obtained from the proposed "Algorithm 3".

IV. RESULTS &ANALYSIS

4.1ERROR ANALYSIS

Error ana lysis is ana lyzing how much percentage error rate has been obta ined for each algorithm. As said earlier we have three different proposed algorithms and two existingalgorithms for comparison.

	Algorithm-1	Algorithm-2	Algorithm-3
Percentage Error Rate	6.0%	5.9%	0.36%

Table-1: Error rates of three proposed algorithms

According to the above table, Algorithm 3 gives a very low percenta ge error rate among all algorithms and actually makes an appreciable improvement in accuracy. Although the error rates of all three algorithms should be taken into account, since eac h algorithm has an error rate below 10 %, in order to produce a high performance model, it is necessary to have minima l errors. So, in our case the algorithm-3 which is given an error rate of 0.36% for N=8 bits (bit size) will definitely be the best model. The error rate is also reduced by increasing the number of bits, but this increases hardwa re and power consumption due to a division block. For both error rate and power consumption, it would therefore be reliable to restrict the bit size to '8'.

4.2 COMPARISON OF POWER, UTILIZATION & DELAY b/w THE THREE PROPOSED ALGORITHMS

The corresponding reports of the three algorithms after simulation are given below.

	Algorithm-1	Algorithm-2	Algorithm-3
Power Report	25.403W	13.808W	11.274W
Device Utilization Report	92 LUT's	76 LUT's	85 LUT's
Delay Report	14.061ns	12.457ns	9.264ns

Table-2: Comparison of three algorithms

4.3 POWER CONSUMPTION ANALYSIS

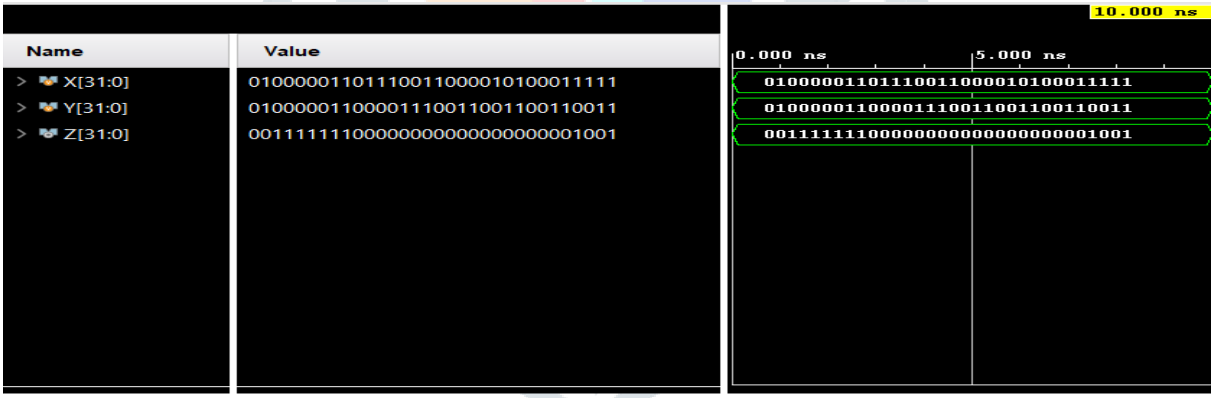

Figure-4: Output obtained from proposed divider design

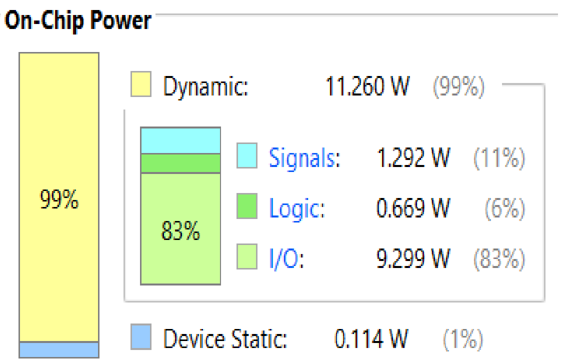

Figure-5: P ower consumption by proposed divider design

The final output of the approximate divider is shown in fig-4.1. This output is represented in binary format. The output is approximated giving a high accuracy and less error rate. For the power ana lysis, fig-4.2 is the power consumption summary of the proposed approximate divider which is 11.260W and this power is split into two categories: dynamic and static. The static and dynamic power distributions are 99% and 1% respectively. Because the dynamic power is more than the static power, we can say that the device has high performance. Because of the leakage current, this device has an extremely low static power consumption. The existing system power consumption is 33.356W which is high. And our design uses power of 11.260W that indicates that we have reduced the power consumption to more than 50%, which is highly reliable.

V. CONCLUSION

In this paper, we proposed a noval method for design of approximate division algorithm. Our method has been shown to perform better in terms of error rate, accuracy and energy consumption compared to many existing methods such as CADE, FPAD etc. When compared to the existing floating point division. We have observed a 50% decrease in power consumption while maintaining the error percentage at 1%. This can further be done by reducing the size of mantissa input selection for division. This can be consider as the finalapproximate algorithm for floatingpoint divider.

VI. REFERENCES

[1]C.K.Jha,K.Prasad,V.K.Srivastava and J.Mekie,"FPAD:A Multistage. Approximation Methodology for Designing Floating Point Approximate Dividers. 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1 -5, doi: 10.1109/ISCAS45731.2020.9180768.

[2] M. Imani, R. Garcia, A. Huang and T. Rosing, "CADE: Configurable Approximate Divider for Energy Efficiency. 2019 Design, Automation Test in Europe Conference Exhibition (DATE), 2019, pp. 586 -589, doi: 10.23919/DATE 2019.8715112.

[3] Linbin Chen, Jie Han, Weiqiang Liu, and Fabrizio Lombardi 2015. Design of Approximate Unsigned Integer Non -Restoring Divider for Inexact Computing. In Proceedings of the 25th edition on Great Lakes Symposium on VLSI (GLSVLSI '15). Association for Computing Machinery, New York, NY, USA, 51-56. https://doi.org/10.1145/2742060.2742063.

[4] B. Yao et al., "Multifractal analysis of image profiles for the characterization and detection of defects in additive manufacturing," Journal of Manufacturing Science and Engineering, vol. 140, no. 3, p. 031014, 2018.

[5] M. Imani et al., "Hierarchical hyperdimensional computing for energy efficient cla ssification," in DAC, pp. 1 –6, IEEE, 2018.

[6] J. Han et al., "Approximate computing: An emerging paradigm for energy-efficient design," in IEEE ETS, pp. 1–6, IEEE, 2013.

[7] H. Amrouch et al., "Towards aging-induced approximations," in DAC, pp. 1–6, IEEE, 2017.

[8] M. Imani et al., "Approximate computing using multiple-access single-charge associa tive memory," TETC, vol. 6, no. 3, pp. 305–316, 2018. [9] C. Liu et al., "A low-power, high-performance approximate multiplier with configurable partial error recovery," in DATE, pp. 1–4, IEEE, 2014.

[9] M. Imani et al., "Cfpu: Configurable floating point multiplier for energy-efficient computing," in DAC, p. 76, ACM, 2017.

[10] X. Jiao et al., "Energy-efficient neural networks using approximate computation reuse," in DATE, pp. 1223 –1228, IEEE, 2018