A Low Error Add and Shift-based Efficient Implementation of Base-2 Logarithm

Pervaiz Kareem
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
291 Daehak-ro, Yuseong-gu, Daejeon, South Korea
Email:pervaiz@kaist.ac.kr

Syed Rameez Naqvi
Department of Electrical Engineering
COMSATS Institute of Science and Technology
GT Road, Wah Cantt., Pakistan
Email: rameeznaqvi@ciitwah.edu.pk

Chong-Min Kyung
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
291 Daehak-ro, Yuseong-gu, Daejeon, South Korea
Email:kyung@kaist.ac.kr

***Abstract*—A real-time and area-efficient hardware implementation of base-2 logarithm (log_2) finds many applications. In this paper, we analyze previous shift and add-based approximations of logarithm. Based on the analysis we present a new two region add and shift-based method to approximate log_2 in hardware. The proposed approach results in 81.55% less percentage error and 42.85% less average error compared to previously reported best two region based approximation approaches with comparable area cost and latency . The proposed method uses most significant four bits of fractional part of shift and add-based method with different weights to approximate fractional part of log_2. Weights to these bits are assigned by simple addition, logical OR and logical AND operation to make hardware implementation more efficient.**

I. Introduction

Numerous digital signal and image processing applications require complex arithmetic operations such as multiplication, division, square, square-root, inverse, and exponential, to name a few. Although these operations can be performed easily in software but their hardware implementation for real time systems is very difficult.

Use of log_2 is one simple solution to perform these arithmetic operations in hardware [1]-[16]. If logarithms of inputs are calculated first then using properties of log_2 these complex operations simply become addition and subtraction followed by $anti - log_2$. Since the accuracy of the result after taking $anti-log_2$ is solely dependent on the calculation of logarithm, an efficient hardware implementation of log_2 is very important.

Assorted approximations have been proposed over the years to implement logarithm function efficiently in hardware. These implementations can be sorted into three categories. First set is based on digital recursion which has low area and low error, but suffers from long delays (See, for instance, [1]–[3]). Look up table (LUT)-based implementations constitute another set of approximations which in turn result in fast and low error designs. However, the area cost of such implementations is a major draw back. We refer, for example, to [4]–[11] for LUT-based implementations. As size of LUT is directly proportional to the number of input bits, these methods require huge memory as input bandwidth increases. The third set consists of shift and add-based methods. The index of most significant 'on' bit is considered as integral part of log_2 whereas the remaining number excluding most significant 'on' bit is considered as a binary fraction, furnishing the fractional part of log_2. This method initially proposed by Mitchel [12] in 1968, appeared to be very simple with low area and low latency. Unfortunately the error was quite high.

Many researchers used multi-region based approaches in order to optimize the accuracy of Mitchel's approximation (see, for instance, [13]–[16]). Even though some improvement in accuracy is achieved, the maximum percentage error and average error of these approaches still need to be improved. Recently Juang et al. [16] proposed a bit level manipulation scheme for calculation of log_2 which results in less percent error and low area as compared to previous add and shift-based approaches. However, average and maximum absolute errors are quite high due to the correction terms being more dependent on second most significant bit of the fractional part.

In this article we propose a two region based approximation scheme for calculating log_2 in order to reduce maximum error, average error and percentage error. Correction terms of our approach just involve logical OR, addition and logical AND operation for an efficient hardware implementation. By using these correction terms we are able to reduce average error by 42.85% and percentage error by 81.55% as compared to [13] and [16] respectively with negligible increase in latency and area cost.

The main contributions of this work are as follows.

- A new hardware-efficient two region shift and add-based approximation method for calculation of log_2 is proposed.
- The proposed method is efficiently implemented in ASIC using AND, OR logic gates, and single bit half and full adders.

The rest of this article is organized in the following manner. Section II is a brief review of previous shift and add-based approximations. The proposed method is presented in Section III, which is followed by error analysis and comparison in Section IV. Section V shows hardware implementation and comparison of synthesis results with previous implementations. In Section VI we summarize our findings.

II. Review of Shift and Add Based Implementations

Let an unsigned input number R be in the interval $0 \leq R \leq 2^n$, whose binary representation can be given as vector

$$\mathbf{R} = [r_n \quad r_{n-1} \quad r_{n-2} \quad \ldots \quad r_0 \quad r_{-1} \quad \ldots \quad r_{-m}],$$

where each entry of $\mathbf{R}$ is a bit. If r_k be the most significant nonzero bit, then we can represent R as

$$R = 2^k(1 + x), \tag{1}$$

where x lies in interval $0 \leq x \leq 1$. Using basic properties of logarithm, from (1) it can be easily observed that

$$log_2(R) = k + log_2(1 + x). \tag{2}$$

In (2), k represent the integer part of $log_2(R)$, whereas $log_2(1+x)$ represent fractional part of $log_2(R)$. For calculating $log_2(1 + x)$ some approximation is needed. Here we briefly review previously reported approximations for $log_2(1 + x)$.

A. Mitchel's Approximation

In [12] Mitchel simply approximate $log_2(1+x)$ by x . Then k and x are added to get the value of $log_2(R)$. This method of approximation is simple but leads to large error as shown in Fig.1.

B. SanGregory et al.'s Approximation

In [13] SanGregory et al. proposed a two region-based approximation which is given below

$$log_2(1 + x) = \begin{cases} x + 2^{-2}x_{4MSBits} & 0 \leq x < 0.5 \\ x + 2^{-2}\overline{x_{4MSBits}} & 0.5 \leq x < 1. \end{cases} \tag{3}$$

In (3) $x_{4MSBits}$ denotes four most significant bits of x. If number of bits in x is N then $x_{4MSBits}$ can also be represented as $\left\lfloor \frac{x}{2^{N-4}} \right\rfloor 2^{N-4}$. This conversion method can achieve

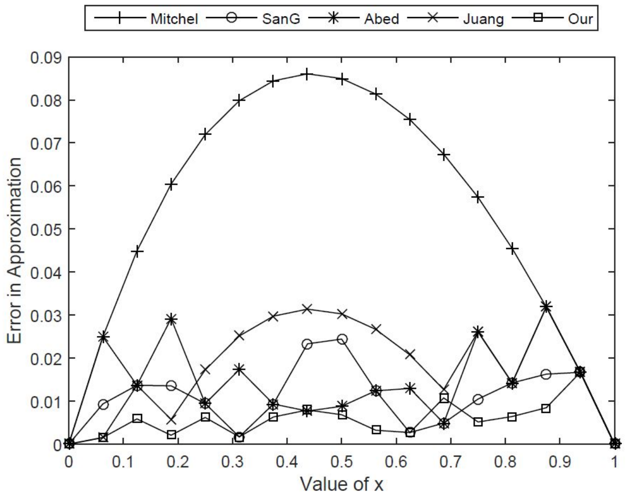

Fig. 1: Error comparison of different approaches with different values of x.

higher accuracy as compared to [12] because $2^{-2}x_{4MSBits}$ or $2^{-2}\overline{x_{4MSBits}}$ is added to the fractional part based on the value of most significant four bits. Instead of multiplying every bit based on its contribution in error of [12] with different number, all four bits are multiplied with 2^{-2} due to which it highly deviates from exact value of log_2 for some input values which can be seen in Fig.1.

C. Abed and Siferd's Approximation

Another two region-based approximation method was proposed in [14]. This approach is similar to [13] with just one difference only first three most significant bits are used instead of most significant four bits. Mathematical expression of this approach is given

$$log_2(1 + x) = \begin{cases} x + 2^{-2}x_{3MSBits} & 0 \leq x < 0.5 \\ x + 2^{-2}\overline{x_{3MSBits}} & 0.5 \leq x < 1. \end{cases} \tag{4}$$

Similar to previous representation $x_{3MSBits}$ represent most significant three bits of x. It can be observed from Fig.1 that this approximation suffers from same problem as [13] due to same multiplier used for all three bits. Although this approach is area efficient because of using three bits, however error is not negligible. To reduce error in approximation Abed et al. suggested a six region conversion method, but area cost of this method was very high due to increased number of regions.

D. Juang et al.'s Approximation

In 2009 Juang et al. [15] reported a new two region based approximation using first four most significant bits of x like [13]. Although bits are not multiplied by 2^{-2} in this approach but same number is multiplied with all four bits just like [13] and [14]. Error range is reduced by use of new multiplier, but at the same time it results in higher percentage, average, and maximum absolute error than [13]. A new approximation is suggested in [16] by Juang et al. to reduce area and error of [15]. Equation (5) shows the approximation suggested in [16] in which symbol '|' represent logic OR while symbol '&' represent logic AND.

$$log_2(1+x) = \begin{cases} x + \frac{x_2|x_3}{32} + \frac{x_2|x_4}{64} + \frac{x_2|x_4}{128} & 0 \le x < 0.5 \\ x + \frac{\overline{x_2\&x_3}}{32} + \frac{\overline{x_2}}{64} + \frac{\overline{x_2}}{64} & 0.5 \le x < 1. \end{cases} \tag{5}$$

As it is clear from (5) that weights given to every bit was different from other, due to which it results in low error range as compared to previous approaches. On the other hand, it results in larger error for some specific inputs, and its average error is also high due to being more dependent on second most significant bit. To overcome these issues, we propose a new approximation in next section which produces lower maximum, percentage, and average error while maintaining the area and latency.

III. Proposed Approximation

To approximate the value of $log_2(1+x)$ it is obvious from Fig.1, that some amount must be added in Mitchel's approximation [12] to reduce error. As mentioned in previous section that average error, percentage error, and maximum absolute errors are the main deficiencies of previous approaches. It can be observed from Fig.1 that error curve of [12] is symmetric and can be easily divided into two parts. The first half of error curve is an increasing function, while in second half its decreasing. From this observation first we divide error curve into two regions and then utilize the symmetry of the curve. Details of our approximation are as follows.

Let we represent binary of fractional part of x as a vector

$$\mathbf{X} = [x_1 \quad x_2 \quad x_3 \quad \dots \quad x_j],$$

in which every entry represent a bit. It can be easily noted from Fig.1 that contribution of every bit is different from other, in other words, instead of linearly changing, error curve is parabolically changing. To decide multiplier for every bit we performed simulations in Matlab. From simulation results we decided to use first four most significant bits of x with different weights to approximate value of $log_2(1+x)$. Our two region based approximation is given in (6) where symbol '&' denotes logic AND.

$$log_2(1+x) = \begin{cases} x + \frac{3(\overline{x_2}\&\overline{x_3}\&x_4)}{128} + \frac{5(\overline{x_2}\&x_3\&\overline{x_4})}{128} \\ + \frac{(\overline{x_2}\&x_3\&x_4)}{16} + \frac{5x_2}{64} & 0 \le x < 0.5 \\ \\ x + \frac{3(x_2\&x_3\&\overline{x_4})}{128} + \frac{5(x_2\&\overline{x_3}\&x_4)}{128} \\ + \frac{(x_2\&\overline{x_3}\&\overline{x_4})}{16} + \frac{5\overline{x_2}}{64} & 0.5 \le x < 1 \end{cases} \tag{6}$$

Equation (6) shows which correction values should be added in which conditions. Values of correction bits for all possible combinations of most significant four bits of x are summarized in Table I.

TABLE I: Value of Correction Bits for All Possible Combinations of Four Most Significant Bits of x

Value of x	Correction Value						
	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
0000	0	0	0	0	0	0	0
0001	0	0	0	0	0	1	1
0010	0	0	0	0	1	0	1
0011	0	0	0	1	0	0	0
0100	0	0	0	1	0	1	0
0101	0	0	0	1	0	1	0
0110	0	0	0	1	0	1	0
0111	0	0	0	1	0	1	0
1000	0	0	0	1	0	1	0
1001	0	0	0	1	0	1	0
1010	0	0	0	1	0	1	0
1011	0	0	0	1	0	1	0
1100	0	0	0	1	0	0	0
1101	0	0	0	0	1	0	1
1110	0	0	0	0	0	1	1
1111	0	0	0	0	0	0	0

To further elaborate (6) we take a simple example. Let $N = 21$ and we want to calculate $log_2(N)$. As $(21)_{10} = (00010101)_2$, so here $k = 4$ and $x = .0101$. From (2) we can write

$$log_2(21) = 4 + log_2(1 + .0101) \tag{7}$$

Now $(.0101)_2$ is less than .5 so from (6)

$$log_2(1 + .0101) = x + \frac{5}{64} \tag{8}$$

$$log_2(1 + .0101) = .3125 + .0781 \tag{9}$$

$$log_2(1 + .0101) = .3906 \tag{10}$$

Now if we put value of $log_2(1 + .0101)$ from (10) in (7) then $log_2(21) = 4 + 0.3906$ which is 4.3906 closer to the exact value of $log_2(21)$ that is 4.3923. In the same way value of $log_2(21)$ can be calculated using Table I. If we look at Table I then correction value which should be added to x is $(.0001010)_2$ when $x = .0101$, so by adding these value we get same .3906. Detailed error analysis and comparison of our proposed approximation is given in next section.

IV. Error Analysis

Proposed approximation together with previously mentioned approaches were simulated using Matlab to confirm the accuracy of new approach. Comparison of all approximations with exact value of log_2 for different ranges is shown in Fig.2. It can be seen from Fig.2 that overall results of our approximation are more closer to exact value of log_2 with maximum error of just 0.025 which is lesser than all the other previous approximations.

We further compared our proposed approximation with existing two region schemes [12]-[16] with respect to maximum absolute error, average error and percentage error for detailed analysis. Fig.3 shows comparison of absolute error in methods [12]-[16] and our proposed method. Let a and b denotes approximated value of $log_2(N)$ and exact value of $log_2(N)$ respectively, then

$$Absolute \quad Error = |a - b|. \tag{11}$$

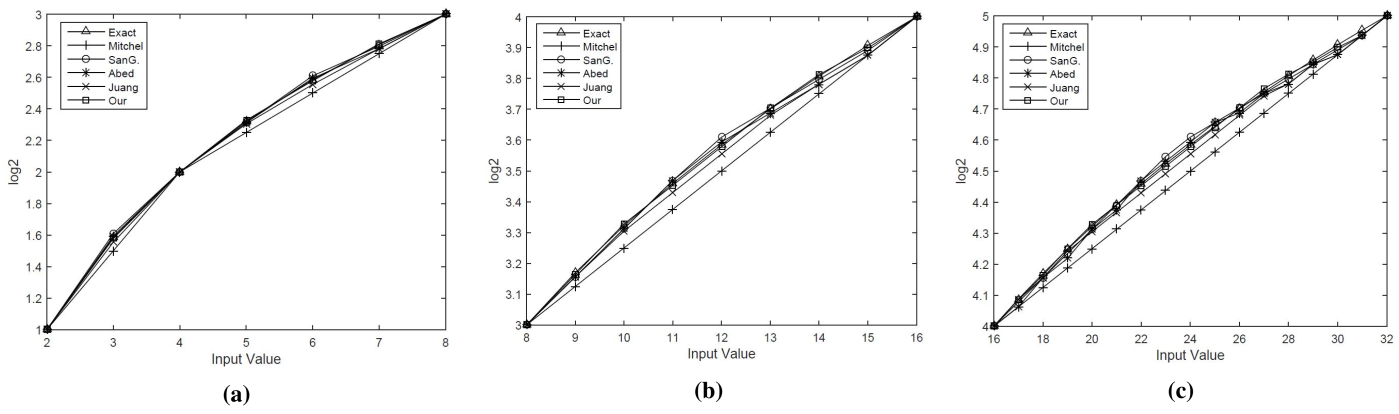

Fig. 2: Comparison of approximation schemes with exact values of log_2 for input ranges from (a) 2-8, (b) 8-16, and (c) 16-32

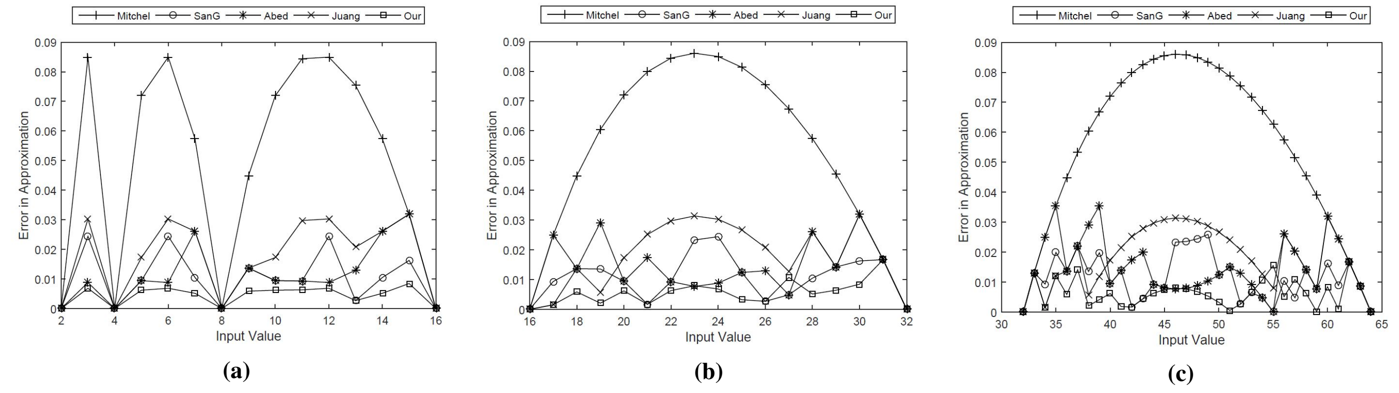

Fig. 3: Comparison of absolute error of our approach with previous approaches for input range (a) 2-16, (b) 16-32 and (c) 32-64.

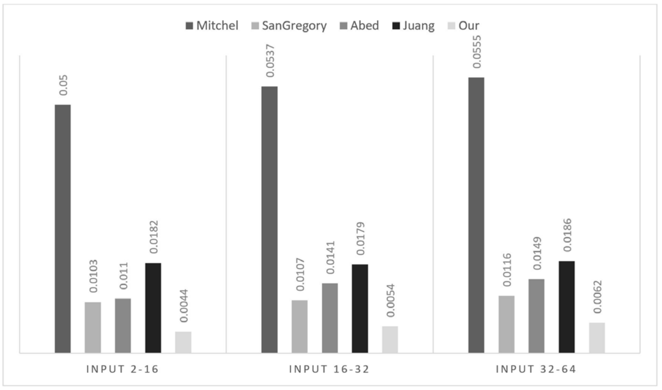

Fig. 4: Average error in our approach and previous approaches for different input ranges.

It can be observed from Fig.3 that our approach has the minimum error compared to all other previous approaches.

We also evaluated previously reported approaches and our proposed method with respect to average error in value of log_2. Average error is defined as sum of difference between exact value of log_2 and approximated value for certain input range divided by number of inputs in this range. let input interval m contains n numbers starting from i to j, and let a and b denotes approximated value of log_2 and exact value of log_2 respectively then we can write expression for average error as (12)

$$Average \quad error = \frac{\sum_{m=i}^{j} |a_m - b_m|}{n}. \qquad (12)$$

We checked average error for multiple ranges. It can be observed from Fig.4 that average error of our approach is significantly less than all other approaches. Moreover from Table II it can be calculated that average error of proposed approach is 42.85% less then [13] which has the least average error than previously reported method.

In addition to absolute error and average error we compared approximation schemes with respect to percentage error. The ratio of error in approximation to the exact value of log_2 is definition of percentage error which is shown in (13)

$$Percentage \quad error = \frac{Absolute \quad error}{Exact \quad value \quad of \quad log_2}. \qquad (13)$$

Similar to maximum error and average error, the proposed method resulted meaningfully less percentage error in all ranges. This reduction in percentage error is as higher as 81.55% compared to the best resulting percentage error method. Fig.5 shows comparison of error percentages for different input ranges. Comparison of our proposed method and previously reported methods is summarized in Table II.

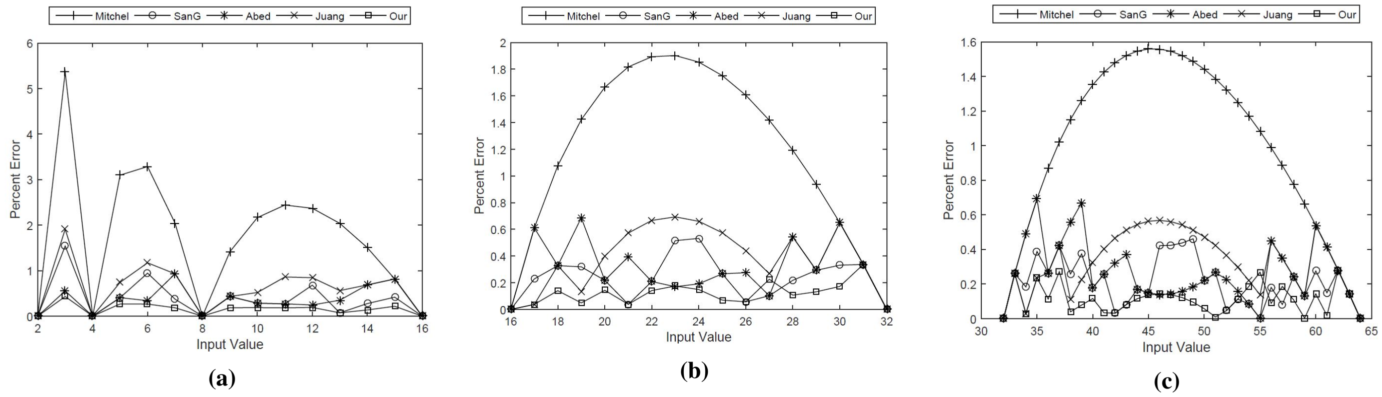

Fig. 5: Percentage error comparison of our approach with previous approaches for input range (a) 2-16, (b) 16-32 and (c) 32-64.

TABLE II: Accuracy of Previous Schemes and Our Proposed Scheme

Parameter	[12]	[13]	[14]	[15]	[16]	Our
Maximum error	.086	.029	.045	.037	.032	.025
Average error	.054	.014	.017	.019	.018	.008
Maximum % error	5.79	2.503	3.838	2.967	2.337	0.431

TABLE III: Post Synthesis Area Results of Previous Methods and Our Proposed Method Under Constant Delay

Method	Area(μm^2)	Delay(ns)
[13]	854	1.59
[14]-6 region	2481	1.59
[15]	3339	1.59
[16]	895	1.59
Our	1003	1.59

V. Hardware Implementation

We designed our proposed approximation scheme using Verilog HDL. For comparison purpose with previous works we used 26 input bits for calculation of log_2. Seven fractional bits out of 26 are used for approximation of $log_2(1+x)$. Fig.6 shows architecture of our proposed approach where HA and FA represent half adder and full adder respectively.

In actual implementation of our approximation many gates are reused, but to make figure easy to understand we avoided to include those details in Fig.6. It can be noted from Fig.6 that there is no multiplexer in our architecture, and HA and FA are simply single bit half adders and full adders, due to which this architecture has less latency and low area cost.

We synthesized our design using TSMC $0.18\,\mu$m technology and compared post synthesis results of our proposed method with previous implementations with constant delay of $1.59ns$. Table III indicates that we achieved 59.57% and 69.96% less area than [14] and [15] respectively, and comparable area cost with [13] and [16] with same latency. We further compared our approach for minimum delay and area product. ADP results of that comparison are presented in Table IV. It can be observed from those results that our approach is better than [14] and [15] while comparable to [13] and [16].

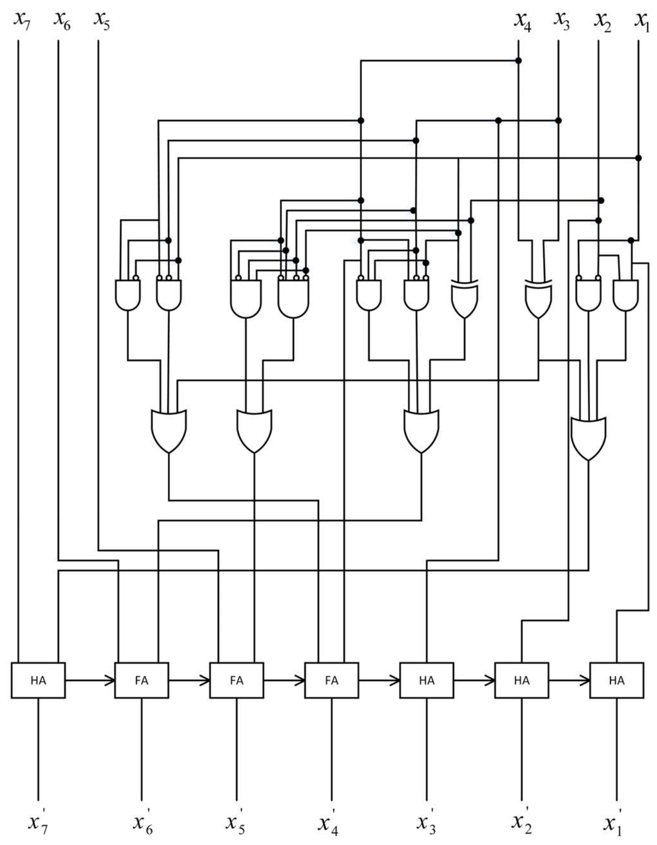

Fig. 6: The hardware architecture of proposed method.

VI. Conclusion

Accurate and real-time hardware implementation of log_2 function is important due to its applications in various fields. Several techniques are proposed in past to calculate accurate value of log_2 in hardware, but some approaches suffer from long latency while some other result in high area cost. Some researchers tried to trade-off between error, area, and latency, but improving one factor resulted in worsening other. We proposed

TABLE IV: Post Synthesis Area Results of Previous Methods and Our Proposed Method Under Minimum Delay

Method	Area(μm^2)	Delay(ns)	ADP
[13]	2042	0.85	1735.70
[14]- 6 region	2481	1.59	3944.79
[15]	3612	1.43	5165.16
[16]	1989	0.79	1571.31
Our	2175	0.81	1761.75

a new method of approximation to reduce error maintaining area cost and delay. Results show that we can achieve an average error of as less as 0.008 , maximum error of just 0.025, and 81.55% leas percentage error compared to previous best approaches, with use of our proposed approximation. In addition to less error, the proposed approach also results in comparable area-delay product to previous approaches. This approximation scheme can be used for accurate calculation of log_2 for real time image and signal processing applications.

References

[1] D. K. Kostopoulos, "An algorithm for the computation of binary logarithms," *IEEE Transactions on Computers*, vol. 40, no. 11, pp. 1267–1270, 1991.

[2] J.-A. Pineiro, M. D. Ercegovac, and J. D. Bruguera, "Algorithm and architecture for logarithm, exponential, and powering computation," *IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1085–1096, 2004.

[3] J.-A. Piñeiro, M. D. Ercegovac, and J. D. Bruguera, "High-radix logarithm with selection by rounding: Algorithm and implementation," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 40, no. 1, pp. 109–123, 2005.

[4] R. Maenner, "A fast integer binary logarithm of large arguments." *IEEE Micro*, vol. 7, no. 6, pp. 41–45, 1987.

[5] P. T. P. Tang, "Table-lookup algorithms for elementary functions and their error analysis." in *IEEE Symposium on Computer Arithmetic*, 1991, pp. 232–236.

[6] M. J. Schulte and E. E. Swartzlander, "Hardware designs for exactly rounded elementary functions," *IEEE Transactions on Computers*, vol. 43, no. 8, pp. 964–973, 1994.

[7] M. J. Schulte and J. E. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Transactions on Computers*, vol. 48, no. 8, pp. 842–847, 1999.

[8] M. Arnold, T. Bailey, and J. Cowles, "Error analysis of the kmetz/maenner algorithm," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 33, no. 1-2, pp. 37–53, 2003.

[9] K. Johansson, O. Gustafsson, and L. Wanhammar, "Implementation of elementary functions for logarithmic number systems," *IET Computers & Digital Techniques*, vol. 2, no. 4, pp. 295–304, 2008.

[10] S. Paul, N. Jayakumar, and S. P. Khatri, "A fast hardware approach for approximate, efficient logarithm and antilogarithm computations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 2, pp. 269–277, 2009.

[11] R. Gutierrez and J. Valls, "Low cost hardware implementation of logarithm approximation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2326–2330, 2011.

[12] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[13] S. L. SanGregory, C. Brothers, D. Gallagher, and R. Siferd, "A fast, low-power logarithm approximation with cmos vlsi implementation," in *Circuits and Systems, 1999. 42nd Midwest Symposium on*, vol. 1. IEEE, 1999, pp. 388–391.

[14] K. H. Abed and R. E. Siferd, "Cmos vlsi implementation of a low-power logarithmic converter," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1421–1433, 2003.

[15] T.-B. Juang, S.-H. Chen, and H.-J. Cheng, "A lower error and rom-free logarithmic converter for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 12, pp. 931–935, 2009.

[16] T.-B. Juang, P. K. Meher, and K.-S. Jan, "High-performance logarithmic converters using novel two-region bit-level manipulation schemes," in *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*. IEEE, 2011, pp. 1–4.