DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications

Team 6 - Anoushka Vyas[*1]
20171057

Abstract

In this report, I have described the **D**ynamic **R**ange **U**nbiased **M**ultiplier for Approximate Applications (DRUM) [1]. The purpose of DRUM is to reduce computational expense of the task of multiplication in terms of area, power and error. The dynamic nature of the design enables the multiplier to "zoom in" on the most significant digits of the numbers, all while making the multiplier highly scalable and configurable. In this report, I have showed that the error with respect to the accurate multiplier is less and the model is resilient to changes in the approximation of the operands in terms of error. Various applications of the DRUM multiplier are explored in the area of image processing.

1 Introduction

Power efficiency has emerged as one of the most critical goals of hardware design and the emerging applications in digital signal processing, computer vision, and machine learning have created new power consumption challenges due to their high computational demands. These applications some error tolerance because of some unavoidable inaccuracies. This feature can be exploited for reducing power consumption. One such example is voltage over-scaling (VOS) where the supply voltage is reduced below the safe operating threshold in favor of saving power but this approach creates timing errors on the critical paths as the circuit slows down with lower voltages.

The novel approach DRUM focuses on creating a multiplier which has unbiased error distribution, where the errors cancel out on repeated computations instead of accumulating. DRUM also uses a smaller multiplier than accurate multiplier thus saving on area and computation power. The design is flexible to incorporate any kind of multiplier for the purpose.

The report is organised in the following way, section 2 describes the preliminary design blocks required like the Leading One Detector, Priority Encoder, Multiplexer, Barrel Shifter and Wallace Tree Multiplier to understand the DRUM design, section 3 states the model design and the error analysis, section 4 presents the various results obtained through different experiment studies of the model, section 5 explains in detail the applications of DRUM in areas of image filtering, JPEG compression and perceptron classifier and section 6 concludes the report.

2 Preliminaries

Before delving into the DRUM design, cover some basics of the building blocks of DRUM which will be used in later sections to develop the DRUM design.

[1] International Institute of Information Technology, Hyderabad, India

2.1 Leading One Detector

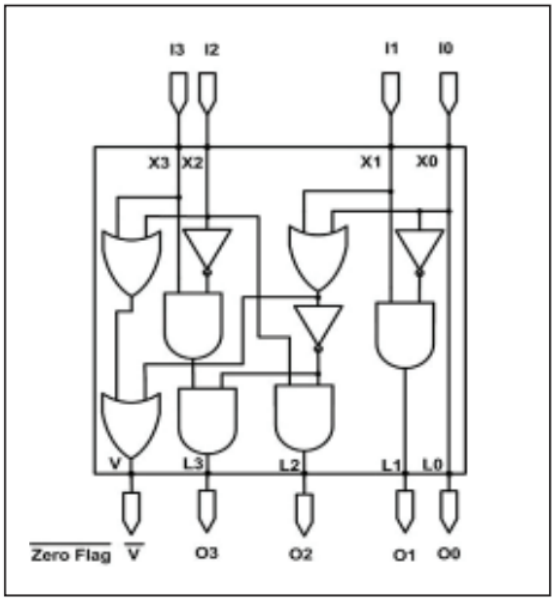

Figure 1: A 4-bit LOD (Image courtesy [2])

Leading One Detectors or LODs determine the location of the most significant one or a leading bit in a given binary. The position of leading one is used for the normalization process and shifting process in the floating-point multiplication, floating-point addition and also in binary logarithmic converters.

The LOD produces logic-1 in the output where the input has the leading 1 and all other outputs will be logic 0. The 4-bit LOD is made using the basic 2-input logic gates viz., AND, OR, NOT and WIRE and this gate level circuit shows performance benefits. Using the evolved 4-bit LOD the higher order LODs of various sizes are can be constructed. In Figure 1, the best known 4-bit LOD is shown.

2.2 Priority Encoder

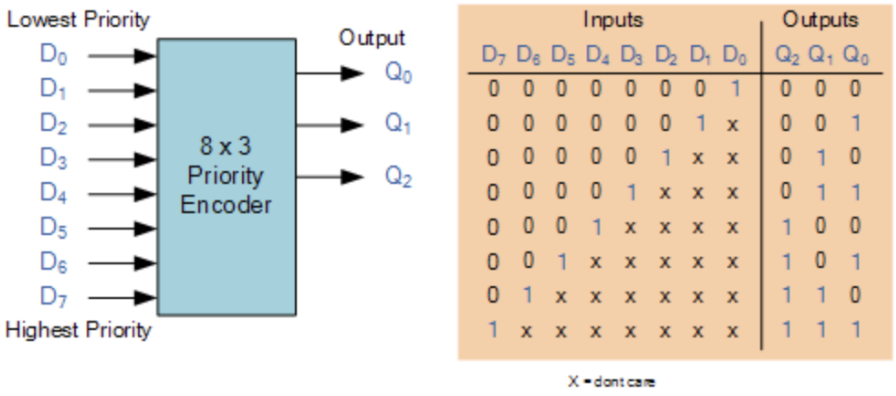

Figure 2: An 8 x 3 priority encoder diagram and truth table (Image courtesy [3])

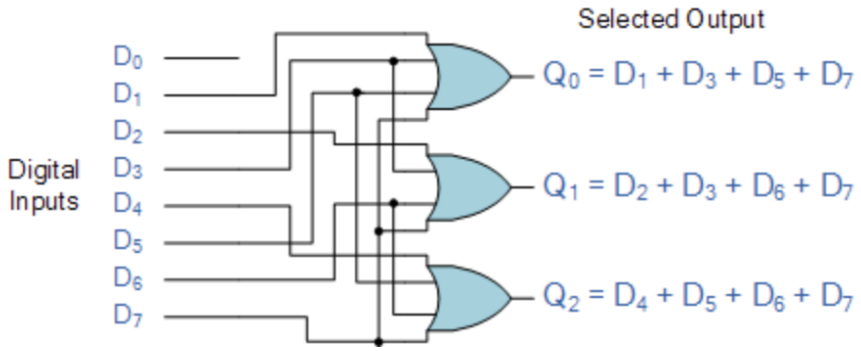

Figure 3: An 8 x 3 priority encoder gate diagram (Image courtesy [3])

Priority encoders were introduced to solve the problem of wrong output code generation when more than one bit position have logic "1". The priority encoders output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored as shown in Figure 2.

Priority encoders output the highest order input first for example, if input lines "D2", "D3" and "D5" are applied simultaneously the output code would be for input "D5" ("101") as this has the highest order out of the 3 inputs. Once input "D5" had been removed the next highest output code would be for input "D3" ("011"), and so on. The gate level design of the priority encoder is given in Figure 3.

2.3 Multiplexer

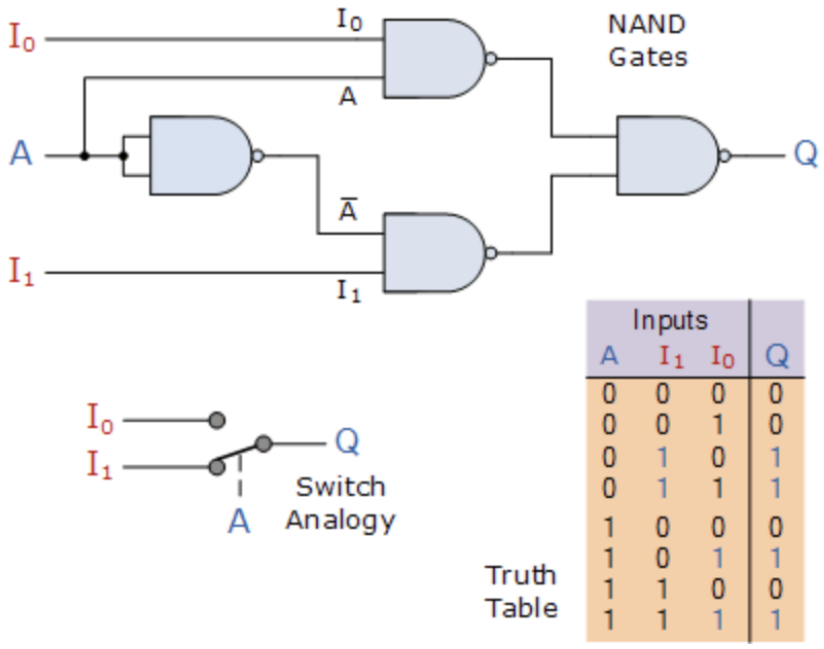

Figure 4: A 2 x 1 multiplexer gate diagram and truth table (Image courtesy [4])

Multiplexer is a device which controls which signal will go on a common transmission line

through a switch. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output.

Consider the 2 x 1 multiplexer shown in Figure 4, the input A acts to control which input (I_0 or I_1) gets passed to the output at Q. From the truth table in Figure 4, we can see that when the data select input, A is LOW at logic 0, input I_1 passes its data through the NAND gate multiplexer circuit to the output, while input I_0 is blocked. When the data select A is HIGH at logic 1, the reverse happens and now input I_0 passes data to the output Q while input I_1 is blocked.

2.4 Barrel Shifter

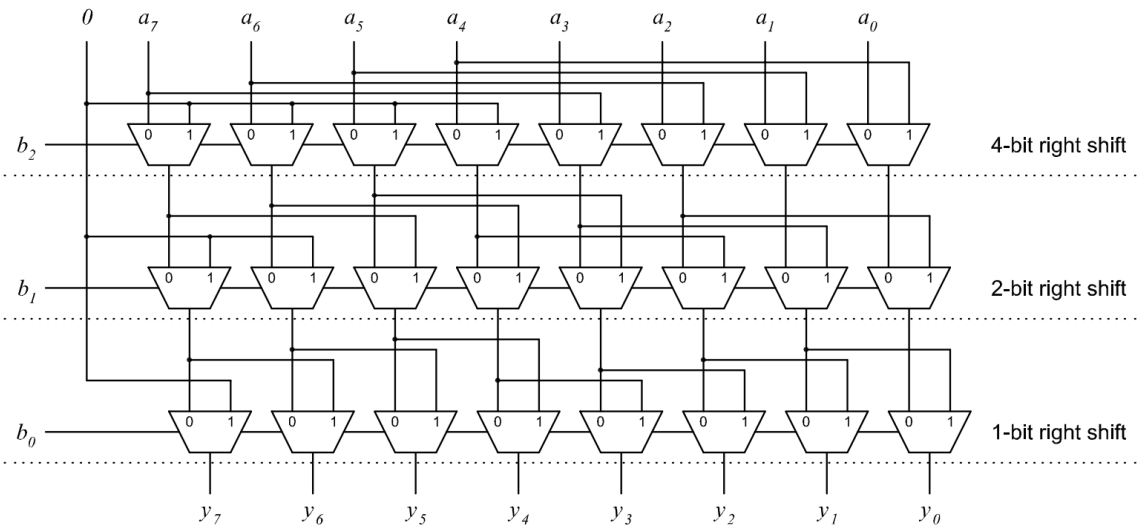

Figure 5: A 8-bit logical right shifter (Image courtesy [5])

A barrel shifter is a combinational circuit that shifts a data word by a specified number of bits. An n-bit logarithmic barrel shifter uses $log_2(n)$ stages. Each bit of the shift amount, B, controls a different stage of the shifter. The data into the stage controlled by b_k is shifted by 2^k bits if $bk = 1$; otherwise it is not shifted. Figure 5 shows the block diagram of an 8-bit logical right shifter, which uses three stages with 4-bit, 2-bit, and 1-bit shifts.

2.5 Wallace-Tree Multiplier

Wallace-Tree multiplier is an efficient parallel multiplier. In the conventional Wallace-Tree multiplier, the first step is to form partial product array (of N^2 bits). In the second step, groups of three adjacent rows each, is collected. Each group of three rows is reduced by using full adders and half adders. Full adders are used in each column where there are three bits whereas half adders are used in each column where there are two bits. Any single bit in a column is passed to the next stage in the same column without processing. This reduction procedure is repeated in each successive stage until only two rows remain. In the final step, the remaining two rows are added using a carry propagating adder. Figure 6 shows the working of a 4-bit Wallace-Tree multiplier.

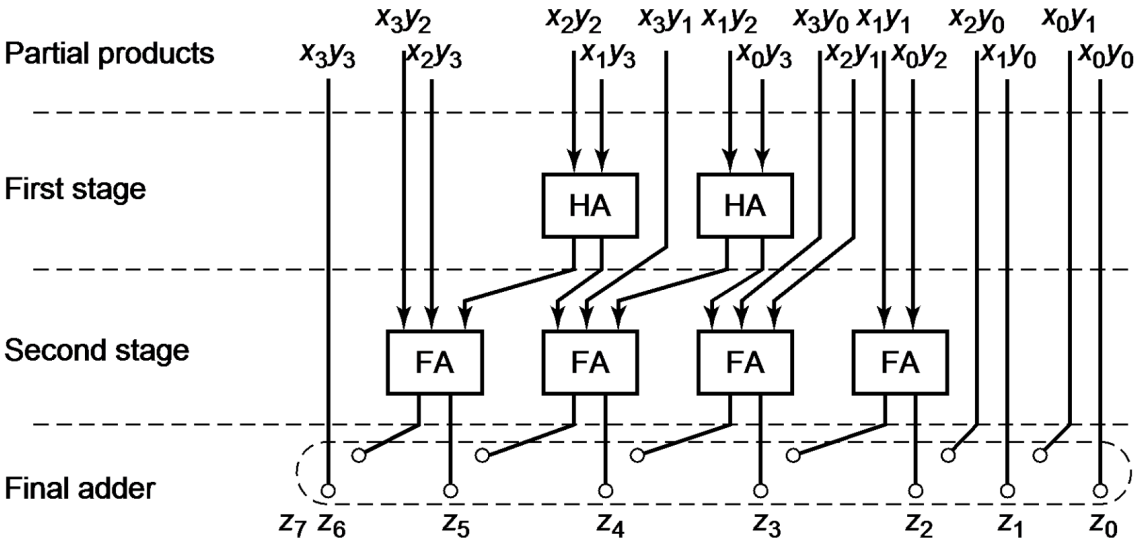

Figure 6: A 4-bit Wallace-Tree multiplier (Image courtesy [6])

3 DRUM

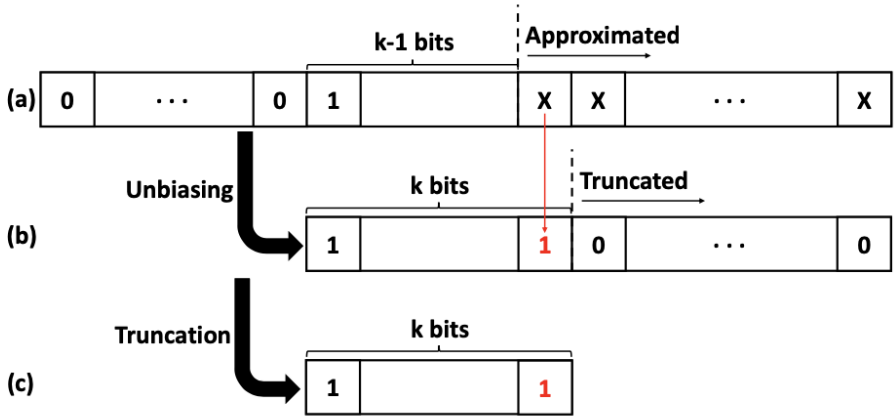

Figure 7: An example of the approximation process (a) Original number (b) Number after unbiasing (c) Final approximated input (Image courtesy [1])

The following section explains the model design in detail and various theoretical analysis done on the design for error calculation.

3.1 Design

The approximate multiplier exploits the fact that not all bits of the operand are equally important. Thus, by carefully selecting a range of bits, the hardware cost of the multiplier is greatly reduced. Some steering logic is required to detect and route the selected range of bits of each of the operands to the smaller core multiplier.

Each operand has n bits and the design uses two leading one detector (LOD) circuit blocks to dynamically locate the most significant "1" in each of the two operands as illustrated in Figure

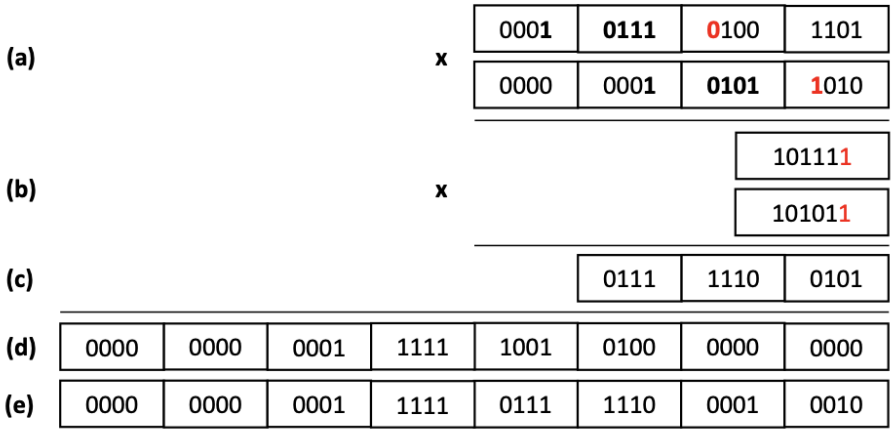

Figure 8: A working example of DRUM ($n = 16$, $k = 6$) (a) The input numbers (b) Approximated inputs (c) Result before shifting (d) Approximate result (e) Accurate result (Image courtesy [1])

7(a). The next $k-2$ bits are selected based on the most significant "1" where k is the bandwidth of the core accurate multiplier. To reduce the error, the value of the remaining lower bits is taken as the expected value. If the leading one is detected at index t, where $0 \leq t \leq n-1$, then the value of the remaining $t-k+2$ lower bits is between 0 and $2^{t-k+2}-1$. Assuming a uniform distribution for the values of the operands, then the expected value is 2^{t-k+1}. Unbiasing is performed by setting $t-k+1$ bit to 1 and all the bits from $t-k$ to 0 to zero (Figure 7(b)). The last $t-k+1$ bit are truncated and only k bits are left (Figure 7(c)). The output of the k bit multiplier is shifted using a barrel shifter according to the location of the leading one. Figure 8 shows a working example for DRUM.

Figure 9 shows the DRUM design in a compact manner. The steering logic is responsible for dynamically selecting the right range of the input operands and feeding them to the multiplier. The output of LOD block is a number with only one 1 showing the location of the leading one for both the operands. These numbers are fed to the priority encoder which encodes these numbers in $log_2 n$ bits which are used to identify the location of leading one in both operands. The multiplexer responsible for selecting $k-2$ bits from each of the two operands, starting from most significant 1. The select line of the multiplexer is actually the output of the priority encoder. The output of the multiplexers is then multiplied using k bit Wallace-Tree multiplier (any multiplier can be used). The last step includes shifting the output to the left to get the final output. In the special case where the leading one is within the least significant k bits, then the steering logic forwards the least k bits directly to the multiplier.

To deal with signed operands a preprocessing logic needs to be added which converts the signed operands to unsigned inputs using twos complement before forwarding them to an unsigned approximate block. The sign signal for the result is then calculated separately and the output will be negated if necessary. The steering logic has a complexity of $O(nlogn)$ and the multiplier part has the logic of $O(k^2)$.

3.2 Error Analysis

Given an operand A in Figure 10, t is the index of the leading one of the operand, U denotes the value of the consecutive $k-2$ bits that are subsequent to the leading one, and L is value of

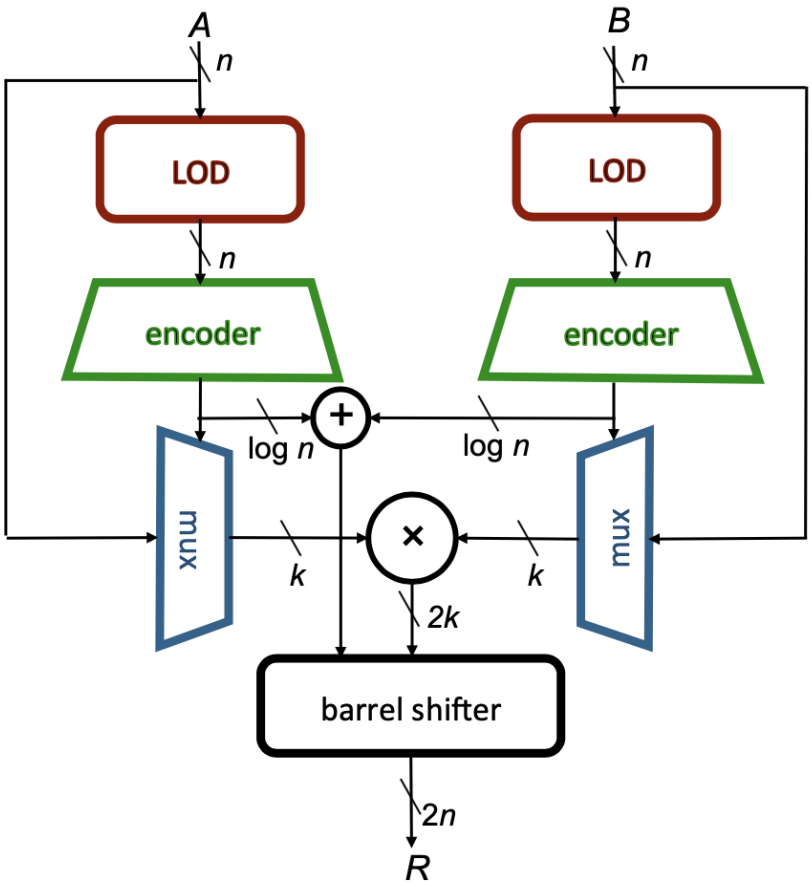

Figure 9: The simplified schematics for DRUM (Image courtesy [1])

the remaining bits excluding the bit at position $t-k+1$. For operand A, the introduced error relative to the number is given by:

$$Err(A) = \begin{cases} X(U,L,t) = \frac{L}{2^t+(2U+1).2^{t-k+1}+L} & if \quad A[t-k+1] = 1 \\ Y(U,L,t) = \frac{L-2^{t-k+1}}{2^t+2U.2^{t-k+1}+L} & if \quad A[t-k+1] = 0 \end{cases} \tag{1}$$

To find the maximum error, both the maximum positive error (MPE) and maximum negative error (MNE) are considered. The MPE happens when L, and therefore t, is maximized and U is minimized. Thus, the MPE is equal to

$$\mathsf{MPE} = \frac{2^{n-k}-1}{2^{n-1}+2^{n-k}-1}, \tag{2}$$

and the MNE happens for minimum values of L, U and t, and therefore it is equal to

$$\mathsf{MNE} = 2^{-k+1}. \tag{3}$$

Using Equation (2) and Equation (3), the maximum truncation error (MTE) is given by Equation (4):

$$\mathsf{MTE} = \max\{\mathsf{MPE}, \mathsf{MNE}\} \tag{4}$$

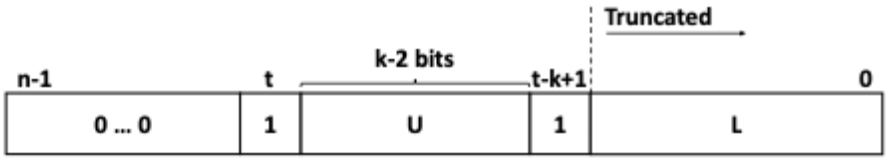

Figure 10: Operand A as used in error analysis in section 3.2 (Image courtesy [1])

To calculate the expected error, a uniform distribution is assumed on all the operands and it is calculated as:

$$E(Err(A)) = \frac{1}{2^n} \sum_{t=k}^{n-2} \sum_{L=0}^{2^{t-k+1}-1} \sum_{U=0}^{2^{k-2}-1} (X(U,L,t) + Y(U,L,t)) \tag{5}$$

If the input operands, A and B, are independent, the multiplication error can now be characterized in terms of truncation errors. Therefore, for maximum multiplication error (MME):

$$\mathsf{MME} = \begin{cases} 2\mathsf{MPE} - \mathsf{MPE}^2 & if \quad \mathsf{MPE} > \mathsf{MNE} \\ 2\mathsf{MNE} + \mathsf{MNE}^2 & if \quad \mathsf{MPE} < \mathsf{MNE} \end{cases} \tag{6}$$

Finally the expected multiplication error is given by

$$E[Err(AB)] = E[Err(A)] + E[Err(B)] + E[Err(A)].E[Err(B)]. \tag{7}$$

4 Experimental Analysis

In this section, I have presented the results for the DRUM multiplier for two different set of operands for different values of k. Next I have described how the various error as described in section 3.2 change for changes in n and k. Lastly, I have shown a gaussian error distribution fitting using mean and standard deviation of error.

4.1 Multiplier Results

The DRUM multiplier is run for two set of operands, one is a smaller 16-bit operand set of 11 and 55 and another one is a bigger 16-bit operand set of 3000 and 125. From Table 1, it is evident that smaller numbers are easy to approximate and thus have no or very less error when k is changed. It is giving the exact value in the particular case I have taken. For the larger number, the obvious trend of decrease in error is observed as k is increased. However, the values are not exact until a sufficient value of k is used, because large numbers require more bits to be approximated correctly.

16- bit numbers	Accurate	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$
11 & 5	55	55	55	55	55	55	55
3000 & 125	375000	337920	365056	379008	372000	374000	375000

Table 1: Result of DRUM for various operands for different values of k

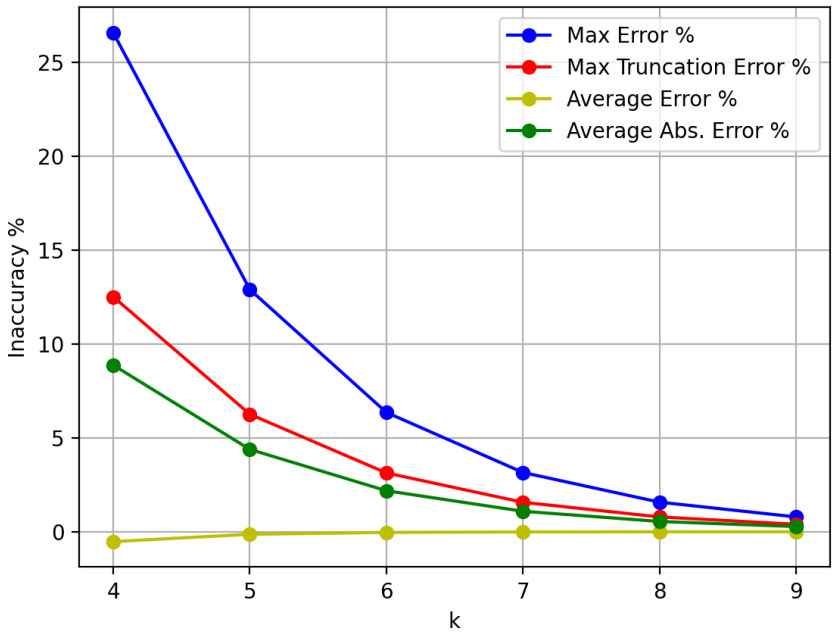

Figure 11: Errors as given by the equations (4), (6), and (7) for different values of k ($n = 16$)

4.2 Error Analysis Results - Changing k

First consider $n = 16$ for a 16-bit multiplier and examine the impact of changing the range as controlled by k. Table 2 summarizes the results by providing the maximum error, average absolute error, average error and maximum truncation error of the error distribution calculated with respect to the accurate multiplier. The results show the expected trend that the error decreases as a function of k; furthermore, the reduction is exponential in k, where the errors are halved for every additional 1 bit in k as expected from the analysis in Section 3.2. Figure 11 plots the maximum error, the expected error, the maximum truncation error and the expected absolute error as a function of k as derived by the equations (4), (6) and (7).

Errors	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$
Max Error %	26.56	12.89	6.35	3.15	1.57	0.78
Max Truncation Error %	12.50	6.25	3.12	1.56	0.78	0.39
Average Error %	-0.53	-0.14	-0.04	-0.02	-0.01	-0.01
Average Abs. Error %	8.86	4.38	2.18	1.08	0.54	0.27

Table 2: Accuracy results for different k ($n = 16$)

4.3 Error Analysis Results - Changing n

Next, the impact of changing the multiplier size n is studied. Table 3 summarizes the accuracy results for the case of $n = 8, 10, 12, 14$ and 16. For all designs, $k = 6$ is fixed. As expected, the results show that the error characteristics of DRUM do not change as a function of the multiplier size, because the design utilizes a dynamic range that always zooms on the first k bits starting at the position of the leading one. This feature enables the design to have an enormous scalability advantage over other approximate multipliers with static ranges. Figure 12 plots the maximum

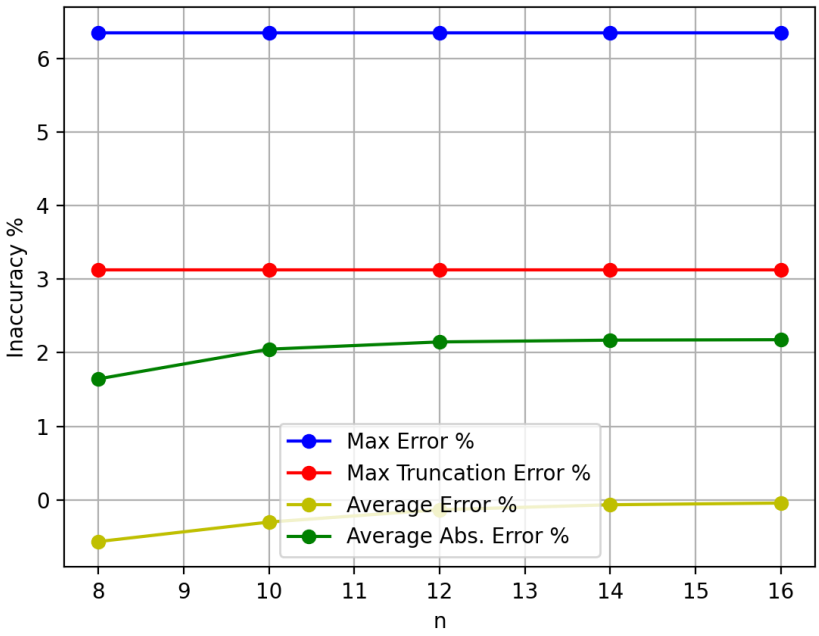

Figure 12: Errors as given by the equations (4), (6), (7) for different values of n ($k = 6$)

error, the expected error, the maximum truncation error and the expected absolute error as a function of n as derived by the equations (4), (6) and (7). The plot is almost parallel to the x-axis.

Errors	$n = 8$	$n = 10$	$n = 12$	$n = 14$	$n = 16$
Max Error %	6.35	6.35	6.35	6.35	6.35
Max Truncation Error %	3.12	3.12	3.12	3.12	3.12
Average Error %	-0.57	-0.30	-0.13	-0.07	-0.04
Average Abs. Error %	1.64	2.05	2.15	2.17	2.18

Table 3: Accuracy results for input size n ($k = 6$)

4.4 Error Analysis Results - Error Distribution

Take the average mean and the standard deviation of the error, to fit a Gaussian distribution on the error. Figure 13 shows the fitted Gaussian error distribution of DRUM for different values of k. The plot shows that as k is increased the error is more and more concentrated towards zero as compared to smaller k values. Smaller k values has larger standard deviation as compared to k values which are larger.

4.5 Power and LUT Analysis

In Table 4, it is evident that power and area is definitely optimized at the cost of small error which is desirable in practicality.

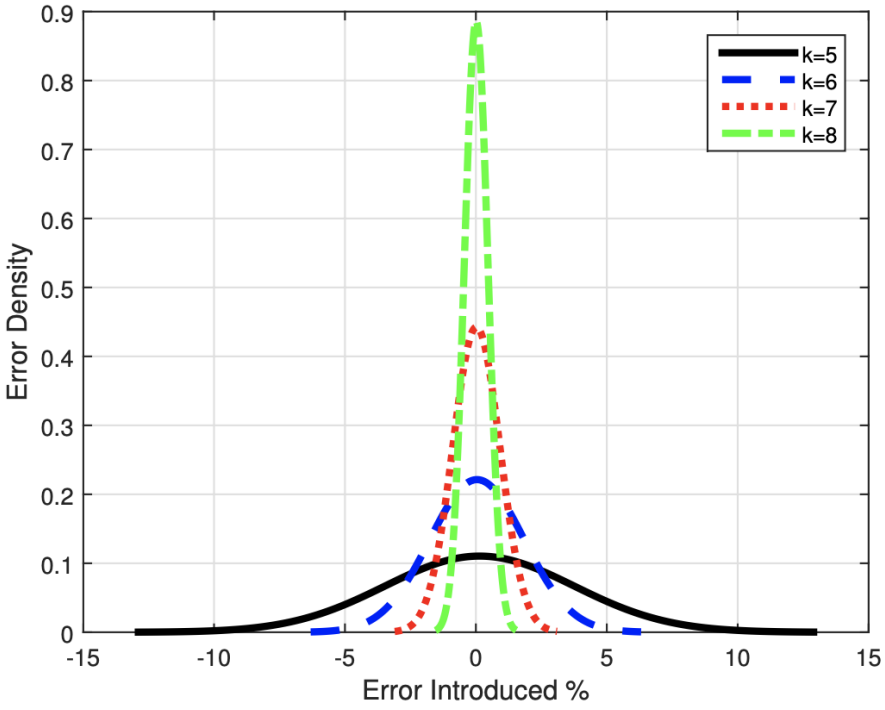

Figure 13: The fitted Gaussian error distributions for DRUM for different k ($n = 16$)

Multiplier	Power(μW)	LUTs
Simple Multiplier (Signed)	1421	455
Simple Multiplier (Unsigned)	1326	328
DRUM (Signed)	1307	249
DRUM (Unsigned)	1240	191

Table 4: Power and LUTs required after synthesising in **Xylinx Vivado** for $n = 6$ and $k = 6$

5 Applications

DRUM design can be used with three different applications from the domains of computer vision, image processing and data classification. The applications are image filtering, JPEG compression and a perceptron classifier. These applications are chosen due to their inherent tolerance to computational inaccuracies.

DRUM can be deployed in two ways for these applications. It could be either part of a general-purpose processor, where the software application invokes DRUM for executing multiplications, or DRUM could be used within custom circuits that execute the applications as accelerators in an ASIC or FPGA.

5.1 Image Filtering

Image smoothing or filtering is performed by convolving the image with a 7x7 Gaussian kernel. The input image is a 200x200 greyscale image with 16-bit pixels. Convolution operation involves shifting and multiplying the image with the kernel for which DRUM can be used to reduce computation complexity.

Figure 16(a) is the original image, Figure 16(b) is the image resulting from accurate filtering. Figure 16(c)-(f) is output of filtering using DRUM for different k values from $(3, 4, 5, 6)$. It can

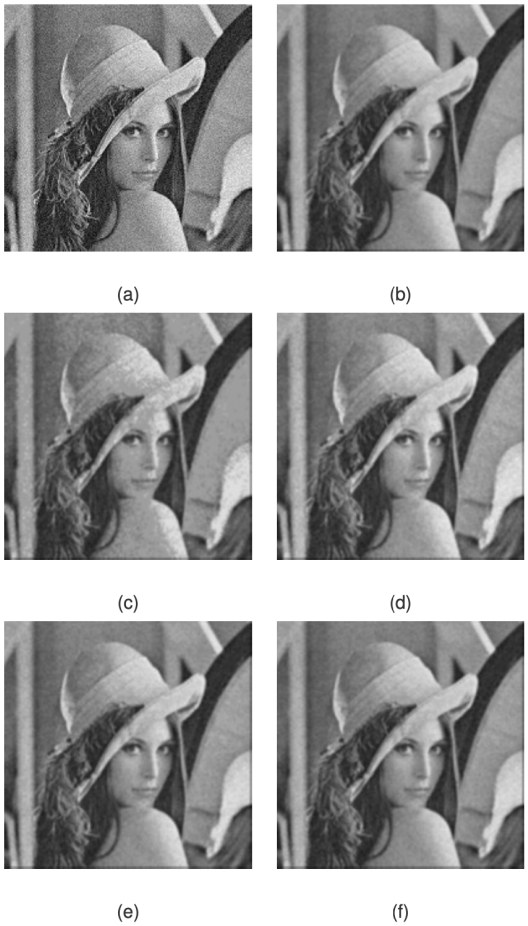

Figure 14: Gaussian filtering results for different values of k (a) Input image (b) Filtered with accurate multiplier (c) $k = 3$ (d) $k = 4$ (e) $k = 5$ (f) $k = 6$ (Image courtesy [1])

be seen that as k increases there is less noise in the image.

5.2 JPEG Compression

DRUM can be used in JPEG compression algorithm by replacing the accurate multiplication in the algorithm with DRUM approximate multiplier. Figure 15(a) shows the output using an accurate multiplier and 15(b) shows the output using DRUM. It is evident that there is not much difference in the outputs of both the methods. Also in real applications the average error (error bias) is far more relevant for JPEG compression algorithm than the average absolute error and as DRUM has a near-zero average error, it will outperform biased multipliers (the biased multiplier is assumed to truncate the lower bits without changing the value of the highest bit to "1". In other words, the lower bits are approximated by 0.).

5.3 Perceptron Classifier

Perceptron classifiers are widely used in machine learning applications. 1000 2-D points are used belonging to two classes $\{-1, 1\}$. In the perceptron, input points are multiplied with weights to decide the cluster in which they are to be classified. Instead of using the accurate multiplier, a DRUM approximate multiplier is used. The error is defined as the percentage of mismatch between assigned class and the ground truth.

(a) (b)

Figure 15: JPEG compression algorithm (a) Compressed using accurate multiplier (b) Compressed using DRUM (Image courtesy [1])

Figure 16(a) shows the dataset used for the experiment and Figure 16(b) shows the output using both accurate and DRUM approximate multiplier. The DRUM multiplier classifies four points, from the 1000 points in training set, in class 1 by mistake, and it detects three more points compared to the accurate multiplier. This shows a fairly good performance even though the accurate multiplier is also not a perfect classifier, DRUM performs well compared to that fact.

6 Conclusion

In this report, I have described the **D**ynamic **R**ange **U**nbiased **M**ultiplier for Approximate Applications (DRUM) [1]. I have verified through experimental analysis, the claims that it is robust to changes in n (size of the input) with constant k (bandwidth of the approximated number) and also tested the nature of various errors with changing k. It is observed that with changing k, the error shows an exponential behaviour which is obvious from the fact that a bit change corresponds to a change in power of 2. Moreover, the Gaussian error distribution plot shows that with increasing k there is less error in multiplication. Overall, this method is a good exploitation of that fact that some applications are tolerant to small errors, as a result, at the cost of small error, power and area can be optimized.

References

[1] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 418–425. DOI: 10.1109/ICCAD.2015.7372600.

[2] K. Kunaraj and R. Seshasayanan, "Leading one detectors and leading one position detectors - an evolutionary design methodology," *Canadian Journal of Electrical and Computer Engineering*, vol. 36, no. 3, pp. 103–110, 2013. DOI: 10.1109/CJECE.2013.6704691.

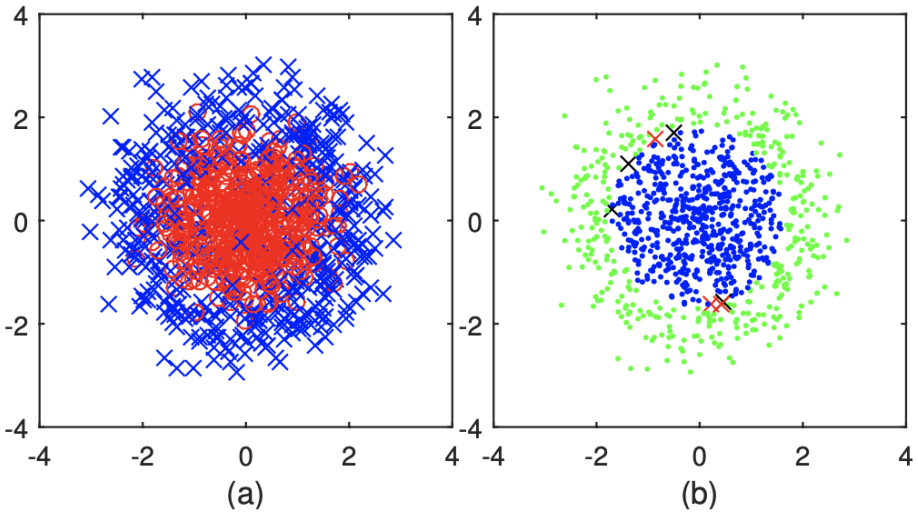

Figure 16: Perceptron classifier (a) Input dataset with classes -1 (blue), 1 (red) (b) The outputs of accurate and approximate multipliers (dots:matching classification, crosses:mismatch, red:Additional detection, black:False alarm)(Image courtesy [1])

[3] . [Online]. Available: https://www.electronics-tutorials.ws/combination/comb_4.html.

[4] . [Online]. Available: https://www.electronics-tutorials.ws/combination/comb_2.html.

[5] . [Online]. Available: https://www.princeton.edu/~rblee/ELE572Papers/Fall04Readings/Shifter_Schulte.pdf.

[6] . [Online]. Available: http://www.cse.psu.edu/~kxc104/class/cmpen411/14f/lec/C411L20Multiplier.pdf.