A Low-Power Dynamic Divider for Approximate Applications

Soheil Hashemi
School of Engineering
Brown University
Providence, RI 02912
soheil_hashemi@brown.edu

R. Iris Bahar
School of Engineering
Brown University
Providence, RI 02912
iris_bahar@brown.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI 02912
sherief_reda@brown.edu

ABSTRACT

In this work, a low-power, low-error divider design is proposed that can achieve significant power and area savings, while introducing insignificant inaccuracies to the output. The design of our divider is highly scalable, offering a wide range of power and inaccuracy trade-offs based on the application requirements. Furthermore, the proposed divider has a lower delay compared to the accurate design, enabling its use on the critical path. We theoretically analyze the error of our design as a function of its configuration, and we thoroughly evaluate the error and power characteristics of our divider in a standalone case and demonstrate that the proposed design can achieve up to 70% in power savings, while introducing an mean average absolute error of only 3.08%. We also implement three image-processing applications in hardware using our divider and conclude that use of the proposed divider will not perceptibly impact their quality-of-service while achieving power benefits of up to 75%.

1. INTRODUCTION

Energy efficiency has emerged as one of most critical criteria in computing systems design in recent years. Computing systems, in general, are designed to generate accurate outputs regardless of the application which can lead to higher power and complexity that what is needed. In particular, applications in domains of image and signal processing, computer vision, and machine learning can tolerate amounts of inaccuracies in their computations but still lead to results within acceptable bounds. This error tolerance can originate from imperfect human perception, lack of a universal best result, or redundancy or noise in the input signal [3]. Approximate computing enables the designer to exploit this error tolerance to add a new dimension to design optimization, where computational accuracy is traded for reductions in power consumption and design complexity.

In approximate hardware design, inaccuracies are introduced to the circuit by design. In this approach, hardware elements are simplified to reduce the power consumption and design area, therefore adding inaccuracies. The implementation of these approximations are mostly focused on basic arithmetic building blocks (e.g. adders and multiplier), due to their high utilization in different applications. These approximate designs can then replace or augment their accurate counterparts in general purpose processors or ASIC designs and accelerators.

While a number of designs for approximate arithmetic have been proposed for adders and multipliers, the approximate design space of other arithmetic units, such as dividers, is not as well investigated. This work aims to address this issue, by proposing an approximate divider targeted toward error resilient applications. Our contributions are as follows.

- We present a novel approximate divider that is designed to dynamically select the most relevant bits of each of the operands and reduce complexity by discarding the lower, less significant bits.
- Our highly scalable design approach enables designers to arbitrarily trade-off accuracy for power and area savings during design process based on their application requirements. Our divider utilizes a smaller accurate divider design in its core, which offers flexibility to the designer as it facilitates the integration of designer's preferred divider design.
- We derive mathematical formulation for the introduced average error and average absolute error as a function of the divider's parameters. We verify our analysis with experimental results. Furthermore, we thoroughly evaluate our divider for different configurations in terms of accuracy performance, design area, power consumption, and timing analysis.
- We use our divider to implement three applications in image processing, namely change detection, foreground extraction and JPEG compression, to evaluate the performance of our divider in real applications. We implement our applications in Verilog and report the accuracy, design area, and power consumption of the approximate designs.

The rest of the paper is organized as follows. In Section 2, we review some of the related work in approximate computing and approximate arithmetic. In Section 3, we describe the proposed design of the approximate divider architecture and propose our analytical error analysis. In Section 4 we evaluate the performance of the proposed method in a standalone case and also within the context of applications. Finally, we summarize our conclusions in Section 5.

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2897965

2. PREVIOUS WORK

In this section, we review some of the related prior work done in the area of approximate design. Approximations can be integrated in the underling hardware design, enabling the designer to introduce inaccuracies in a controlled and deterministic manner. Several approximate adder cells were proposed by Gupta *et al.*, where some of the transistors used in traditional mirror adders are removed [2]. Mahdiani *et al.* proposed an approximate multiplier, where the output is approximated in partial product addition stage [8]. A highly configurable approximate multiplier with near zero error distribution is proposed by Hashemi *et al.* [4]. New metrics for evaluation of approximate designs are proposed in [7, 9, 5]. An automated technique for approximate synthesis of behavioral Verilog is proposed by Nepal *et al.* [10].

While a few designs have been proposed for both adders and multipliers, dividers and other arithmetic units have not received nearly as much attention. This is due in part to lower utilization of blocks such as dividers compared to adders and multipliers. Lower utilization of dividers, however, is in part a result of high cost of dividers in terms of delay and power. Therefore, divider design with significant improvements in these areas can facilitate utilization of these arithmetic blocks. Kelly *et al.* propose a multi-cycle unsigned integer divider design for timing critical applications [6]. The proposed approximate design trades accuracy for timing improvements, achieving as much as 22.5% speedup compared to a radix-4 SRT divider with 99.4% probability of correctness in the output. In this work, the focus is on speedup improvements, and the power and design area characteristics of the design are not evaluated thoroughly. A non-restoring divider design is proposed by Chen *et al.* where a 8/4 divider array is approximated by using their proposed approximated subtractor cells [1]. The utilization of pass transistor logic in the design limits their applicability. A floating point non-iterative divider using curve fitting is proposed by Wu *et al.* where a systematic approach provides a range of trade-offs between area-delay product and accuracy [13].

3. PROPOSED APPROXIMATE DIVIDER

In any binary number, the higher-order bits represent the most significant bits of the number and the significance of the bits decrease as we move toward lower-order bits. In our design, we take advantage from the aforementioned fact and propose to limit the number of bits in the operands actually divided, by carefully selecting a dynamic range of bits for each of the two operands. To be more specific, we reduce a large, and hence high power, divider to a significantly smaller core accurate divider and some steering logic, where the steering logic is defined as hardware necessary for dynamically detecting the most significant bits of each operand and routing the selected bits to the accurate divider.

In our design, inaccuracies are introduced as part of the steering logic component, where each of the operands are approximated by truncating some of the lower bits. The truncated operands are then divided accurately using a divider. If the design area and power consumption reduction due to the smaller divider outweighs the overheads due to the steering logic, then the proposed implementation is well-justified and results in power and design area savings. In our design, the steering logic utilizes leading one detector (LOD) units to effectively "zoom in" on the most significant bits of each of the operands. Then, for each of the operands a predefined number of following consecutive bits are selected and routed to the accurate core divider, while the rest of the input bits are truncated. We define the parameter k (or range of the divider) as the number of bits put through the accurate divider. This parameter can be used by the designer to select among different trade-offs between accuracy and design metrics, giving the design significant flexibility.

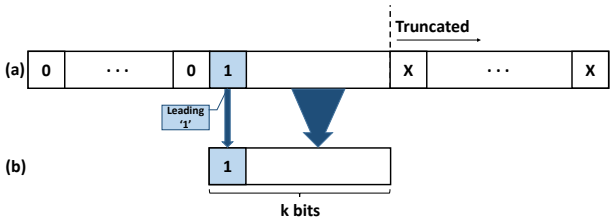

Figure 1: An example of the approximation process. (a) Original input. (b) Approximated input.

Figure 1 illustrates the process implemented for approximation of each of the operands. For an n bit operand, first the most significant one is detected using a LOD block. The next $k-1$ bits are then selected and routed directly to the arithmetic logic using multiplexers. The remaining bits are then truncated. As this truncation is done on both dividend and divisor the final approximate result of the division could either underestimate or overestimate the final accurate result; thus, errors could potentially cancel each other out in series of operations.

Figure 2 shows the building blocks of the proposed hardware and their interconnections. In order to maintain consistency with previous work, we design our divider to have a 2/1 ratio between the dividend and the divisor. While the ratio is kept constant throughout this work, the proposed design can be readily used to implement dividers with any input widths without restrictions. Furthermore, we define parameter n to be equal to the bitwidth of the dividend to simplify future references. Figure 2 shows the bitwidths and their relations to each other. Here, the LOD blocks, encoders, multiplexers, and barrel shifter are all part of the steering logic while the accurate core divider is the only component in the arithmetic logic. If the leading '1' is within the least significant k bits, no truncation is necessary and the accurate divider can handle all of the non-zero bits in the operands. Therefore, in this case, the least k bits are directly forwarded to the divider and as a result no inaccuracy is introduced to the division output. This special case is integrated in all our evaluations, but is not shown in the schematics in order to avoid over-complicating the

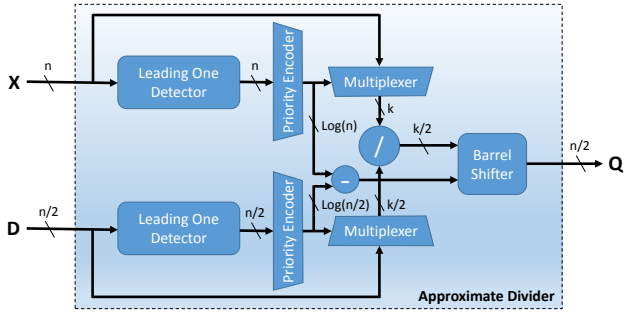

Figure 2: Simplified schematic for the proposed approximate divider. The result is shifted by the value specified in the output of the subtractor.

(a) 0001 **0111** **010**0 1101 ÷ 0011 **00**10
(b) 1011 1010 ÷ 1100
(c) = 1111
(d) 0111 1000
(e) 0111 0111

Figure 3: An example of a division using the proposed method. (a) Input operands ($5965_{10}/50_{10}$); (b) Approximated operands ($186_{10}/12_{10}$); (c) Arithmetic unit result (15_{10}); (d) Final approximate result (120_{10}); (e) Accurate result (119_{10}).

schematics. As a feature of the proposed methodology, the utilization of an accurate divider as the core components enables the designers to use their preferred divider architecture. We will demonstrate in Section 4 that this design will lead to significant improvements in power consumption, design area, and delay.

Figure 3 gives a numeric example demonstrating the operation of the proposed approximate divider with input size 16/8 and $k = 8$. In this figure, bold numbers represent the selected bits that will be routed to the arithmetic logic as they are. In this example, the approximation has a relative error of -0.84%.

It is worth mentioning that the proposed design can easily be used to divide signed numbers, where a preprocessing logic can convert signed operands to unsigned before forwarding them to the approximate divider. The output of the divider can then be negated if the sign bits of the operands do not match.

As our proposed method dynamically detects the most significant bits for operation, the benefits of our design increase as the divider input width grows. Note that often the complexity of accurate divider designs grows as a quadratic function of input width $O(n^2)$. However, in the case of our design, while the complexity of the steering logic increases as the input size grows, the complexity of the arithmetic logic does not need to grow to maintain accuracy. It is critical to note that in our design $k << n$. With this in mind, one can easily show that in our proposed divider the steering logic grows as $O(n \log n)$, while the arithmetic logic grows as $O(k^2)$. Therefore, overall as the input width grows, the increase in complexity for our divider is much less significant than the increase in an accurate divider, resulting in more substantial benefits. This makes the proposed divider highly scalable to higher input widths. Thorough analysis of the proposed divider in Section 4 will confirm our argument.

Mathematical Analysis: We also derive mathematical formulation for the expected error of our proposed divider based on the design parameters k and n. Figure 4 highlights the generic definition of the variables as used in this section for each of the operands. Here, t, L, and U are defined as the index of the leading one, the value of the lower bits, and the value of higher bits respectively.

Using this notation, for the proposed architecture, the approximate value of the operand is equal to

$$\widehat{a} = \left\lfloor \frac{a}{2^{t-k+1}} \right\rfloor \quad (1)$$

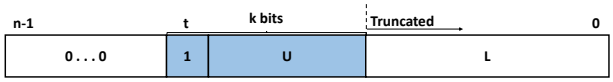

Figure 4: The general case of the approximation and the analytical variables.

where $\widehat{a}$ represents the truncated operand encoded with k bits. Note that $\widehat{a} = a$ if the leading one is within the lower k bits. Here, the value of $\widehat{a}$ represents an intermediate step (using k-bit rather than n-bit) in the calculation of the division and therefore $|a - \widehat{a}|$ represent the truncation error. The relative truncation error is given by

$$\frac{|a - \widehat{a}|}{a} = \frac{L}{2^t + U \times 2^{t-k+1} + L}.$$

Based on Equation 1 and in the case of division, assuming uniform distribution on the input operands, and using the same calculation for the second operand, the expected error of the divider is given by

$$E(Error(\frac{a}{b})) = \frac{1}{2^n}\frac{1}{2^{n'}} \times \sum_{a=1}^{2^n-1}\sum_{b=1}^{2^{n'}-1}\left(\left\lfloor\frac{a}{b}\right\rfloor - \left\lfloor\frac{\widehat{a}}{\widehat{b}}\right\rfloor \times \frac{2^{u(a-2^k)\times(t-k)}}{2^{u(b-2^{k'})\times(t'-k')}}\right) \quad (2)$$

where $u(\cdot)$ is a step function equal to one when its argument is non-negative and equal to zero for negative inputs (e.g., when an operand is passed to the core divider accurately). Further, the parameters related to the dividend (a) are shown as n, k, t while the parameters of the divisor (b) are shown as n', k', t'. Here, the flooring function represents integer division. Furthermore, the formula is driven for general case of the division where in the case of this work, the values of n', k' can be replaced with $n/2, k/2$ to maintain the 2/1 ratio. In subsection 4.1 we validate our analytical analysis against experimental results.

4. EXPERIMENTAL RESULTS

In this section, we empirically evaluate the proposed approximate divider design in terms of computational accuracy as well as power and area performance. For the core divider, we implement a conventional array divider. All designs are coded in Verilog and synthesized using Synopsys DC compiler with an industrial 65-nm standard cell library at the typical process corner. We use MentorGraphics ModelSim to evaluate the numerical output of the designs and assess its accuracy. In Subsection 4.1, we evaluate the accuracy, power consumption, and design area characteristics of our approximate standalone divider and compare these characteristics against those of an accurate divider. Then, in Subsection 4.2 we evaluate the quality of solution when our divider is used in real applications. Here, we also report the power and area benefits achieved when implementing each of the applications in Verilog and as a custom design.

4.1 Standalone Divider

In this subsection, two ten-million randomly generated vectors are used to evaluate the divider and its error characteristics. Similar to the approach proposed in [7], we evaluate the performance of our divider in terms of both the relative error and error distance (ED) distribution. Here, relative error is defined as RE = (Acc − App)/Acc, while error distance is simply defined as ED = |App − Acc| where App and Acc are the results generated by the approximate and accurate circuits respectively. We analyze the error distributions and show that the proposed approximate divider has near zero error bias which results in improvements in applications where results of divisions are summed together.

As discussed in Section 3, our divider is highly configurable. As the value of the parameter k can be anywhere

Table 1: Accuracy results for the standalone divider for different values of k ($n = 16$).

	range				
	$k = 4$	$k = 6$	$k = 8$	$k = 10$	$k = 12$
Max. ED	64	44	26	13	6
Average Abs Error %	13.57	6.37	3.08	1.42	0.59
Error Bias %	-1.78	-1.49	-0.93	-0.48	-0.23
Standard Deviation %	17.16	8.55	4.60	2.56	1.50

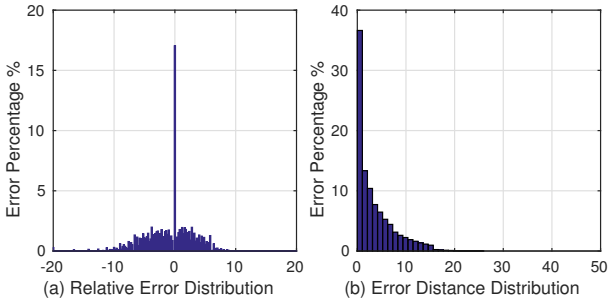

Figure 5: The error distributions of our divider.

Table 2: Accuracy for the standalone divider for different input sizes ($k = 8$).

	divider size		
	$n = 16$	$n = 24$	$n = 32$
Max ED	26	420	6772
Average ABS Error %	3.03	3.09	3.09
Error Bias %	-0.97	-0.84	-0.84
Standard Deviation %	4.6	3.84	3.77

between 2 (to maintain the 2/1 ratio) and n, the range of possible power and accuracy trade-offs (which is a function of k) is large. We first examine the impact on the divider performance by varying the value of k from 4 to 12, while assuming a constant value of $n = 16$. Table 1 summarizes the accuracy results with respect to the accurate design. As expected, the accuracy increases for all of the metrics as the value of k increases. In addition, we notice that as k increases, the average absolute error decreases roughly by a factor of two. This is expected given that increments in the range translate into one more bit calculated in the result. This trend makes the error performance of the divider easily calculable and predictable.

Furthermore, we plot the error distance and the relative error distributions for a divider with $n = 16$ and $k = 8$. Figures 5(a) and 5(b) show the empirical error distributions. The biggest portion of the results are close to 0 in both distributions such that more than 99.5% of the relative error is within 15% error bound. The results in Figure 5(a) also show that the relative error is relatively balanced around 0 resulting in lower error rates in applications where results of different divide operations are added together to generate the final result.

Figure 6 gives the total power and design area savings relative to an accurate divider for different values of k ($n = 16$). We can observe that with average absolute error values ranging from 0.64% to 13.7% it is possible to obtain power savings of 29% to 90%, respectively. Our approach provides great flexibility to meet application needs.

Next, we consider the impact of changes in the divider input size on the performance. We summarize the accuracy results in Table 2 with $k = 8$ and for input widths of 16/8, 24/12, and 32/16. While the maximum error distance is increasing significantly with input size, it is critical to note

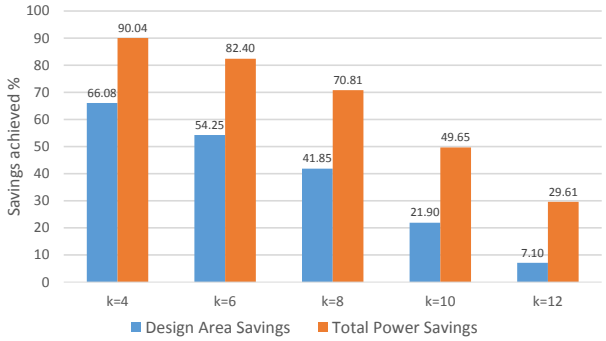

Figure 6: Area & power savings as a function of k.

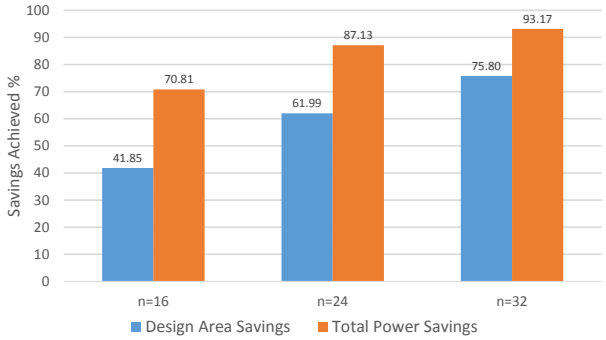

Figure 7: Area & power savings as a function of n.

that the value of the accurate results is also increasing, keeping the *error/Acc* ratio within the same bounds, therefore resulting in small changes in average relative error. These results support our claim that we do not need to increase the range of the divider to higher bit widths to maintain accuracy. Therefore, our design can achieve improved benefits as the divider input width increases.

Figure 7 gives the design area and total power savings of our proposed divider relative to an accurate divider of the same size. Here, as explained in Section 3, we maintain a 2× ratio between the input operand for all cases. We observe that the benefits in power savings increase from 42% to 62% to 76% as the divider size grows from 16/8 to 24/12 and then to 32/16. Further improvements to accuracy can be achieved by simply increasing the range of the divider.

We also compare our proposed approximate divider against the approximate divider, AXDnr, proposed by Chen *et al.* [1]. We implement a model of their design in Matlab and compare it with our divider using the same inputs. Since AXDnr is implemented using pass transistor logic, for a fair comparison, we compare AXDnr against an accurate divider using pass transistor logic while comparing our designs against an accurate design using conventional CMOS technology. In each design we report the design area utilization normalized to the accurate divider using the same logic. Since the power consumption and design area have a strong correlation, this comparison provides useful information on how the dividers compare. Figure 8 illustrates this comparison using a Pareto Frontier graph. As demonstrated in the figure, the designs proposed using our approach clearly outperform the previous work in both the accuracy and the design metrics while offering a much broader range of trade-offs.

In our approximated design, even though almost all of the steering and arithmetic logic are on the critical path, the quadratically smaller arithmetic logic leads to significantly

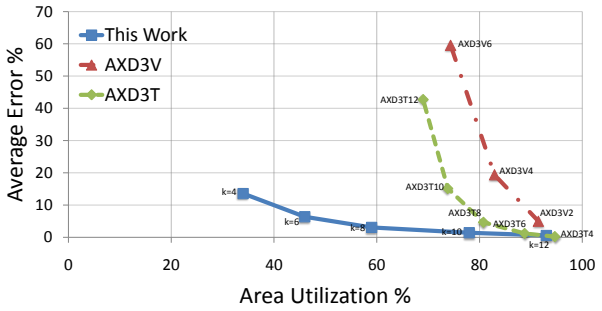

Figure 8: Trade-offs between accuracy and design area for our divider vs. AXDnr [1].

Table 3: Critical path delay as a function of input size ($k = 6$).

Divider Size	Acc. Delay (ns)	App. Delay (ns)	Speedup
16/8	8.39	4.75	1.77×
24/12	17.33	5.26	3.29×
32/16	29.60	5.74	5.16×

shorter critical path delay. Table 3 showcases the delay results and the corresponding speedups, for different divider input widths. A speedup between 1.77× and 5× over a traditional accurate divider is possible using our approach.

4.2 Results for Real Applications

In this section, we evaluate the proposed divider design when used within the context of three different applications from the image processing domain: image change detection, JPEG compression and foreground extraction. The criteria for applications is immunity to small and bounded errors as well as high utilization of the dividers in the combinational logic. We implemented our applications in Verilog and evaluated both accuracy of the result as well as design metrics. We use peak signal to noise ratio (PSNR) to evaluate the performance of our divider in our applications, since PSNR is commonly used for accessing the quality of images after compression or processing. All PSNR values are calculated using the output of the accurate divider as the reference image.

4.2.1 Change Detection

In this application, two images are compared against each other and the coordinates of the changes are highlighted. These coordinates can later be used for remote sensing, medical diagnosis or surveillance applications. Different implementations of this algorithm have been proposed in the literature using either dividers or subtracters [11]. In this work, we are evaluating the proposed divider against accurate divider, and therefore the differences among high-level algorithms are not relevant to this study. In basic implementation of this application, the value of each pixel after pre-processing is divided by the value of the corresponding pixel in the second image. Larger values in the division result correspond to bigger differences between the two input images. A predefined threshold is then used to highlight the coordinates of the changes. We implement this application on regions of interest for each example image to narrow the search space. As our input images, we use two randomly chosen images as proposed in [12], which is a dataset specifically designed for change detection algorithms.

Figure 9 illustrates the output images as implemented

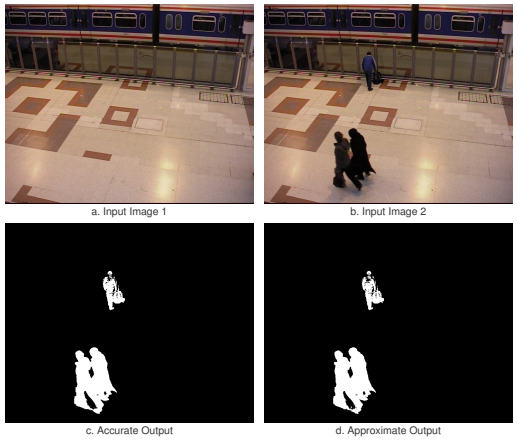

Figure 9: Change detection results for two sets of input images. (a) Input image 1; (b) Input image 2; (c) Detected using accurate divider; (d) Detected using approximate divider. $PSNR = 28.87$ dB.

using exact and approximate dividers. Here, Figure 9(a) and 9(b) show the two input images, Figure 9(c) shows the detection result using an accurate divider, and Figure 9(d) shows the result using the proposed approximate divider with $k = 8$. It can be seen that the degradation in output quality is hardly noticeable to the human eye.

Table 4 summarizes the hardware design characteristics of the application for implementations using both accurate and approximate dividers in their design. Since division modules comprise a substantial portion of the hardware implementation, using our approximate divider can reduce power consumption by more than 75% without any significant degradation in output accuracy. Furthermore, delay is reduced by 2.84×, providing timing slack that can be exploited to further reduce power consumption with voltage scaling.

Table 4: Area, power and delay characteristics using the proposed divider in change detection.

Design	Area (um^2)	Power (mW)	Area Savings	Power Savings	Delay Reduction
Acc.	17360.64	1.6875	–	–	–
App.	10327.68	0.4074	40.51%	75.86%	2.84×

4.2.2 JPEG Compression

We also evaluate our divider in a JPEG compression algorithm using a 512x512 RGB image as the input. Our divider replaces the ones used in the quantization step. We use 16/8-bit fixed-point dividers where the processed input image, after DCT, is mapped to 16-bits, while the quantization divisors use 8-bits. For our approximate design, we use the proposed divider with $k = 8$ and report the accuracy and the resulting JPEG images for comparison. Figure 10 shows the compressed image using both an accurate divider and an approximate divider. As shown in the image, the degradation in accuracy due to use of the approximate divider is negligible. The PSNR of the image outputted using the approximate divider is 24.82 dB in reference to the output of the accurate design.

In Table 5, we summarize the hardware design characteristics of the JPEG compression algorithm utilizing both the accurate and the approximate dividers. Even as a more complicated design, in particular, our proposed implementation achieves power savings of 29.23%.

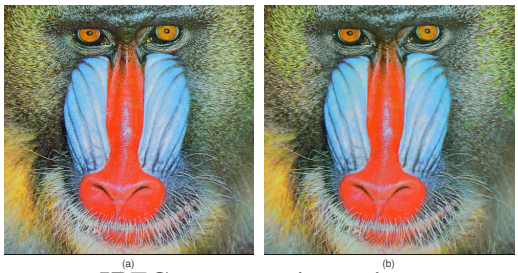

Figure 10: JPEG compression using accurate and approximate dividers. (a) Compressed image using accurate divider; (b) Compressed image using Approximate divider. $PSNR = 24.82$ **dB.**

Table 5: Area, power and delay characteristics using the proposed divider in JPEG compression.

Design	Area (um^2)	Power (mW)	Area Savings	Power Savings	Delay Reduction
Acc.	1291474	9.10	–	–	–
App.	1102510	6.44	14.63 %	29.23 %	1.52 ×

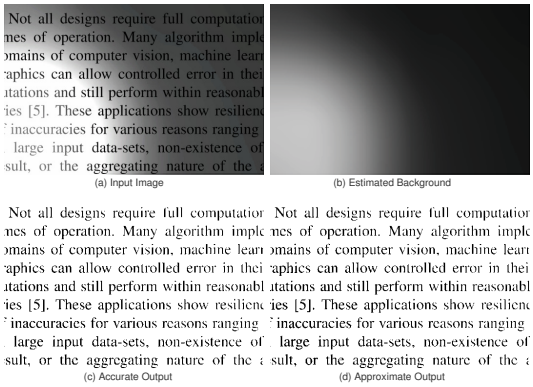

Figure 11: Foreground extraction using accurate and approximate dividers. (a) Input image; (b) Estimated background image; (c) Enhanced image using accurate divider; (d) Enhanced image using approximate divider. $PSNR = 23.96$ **dB.**

4.2.3 *Foreground Extraction*

We also use our divider to implement a simple foreground extraction (i.e., background removal) algorithm. Foreground extraction is commonly used as an initial stage for further processing. In this algorithm, the input images usually suffer from background variations (e.g., uneven illumination), affecting the foreground quality. The effect of this background variance can be reduced by dividing the image by its estimated background image. Figure 11 visualizes the input, estimated backgrounds, and output images as generated using approximate and exact dividers. The proposed divider, using $k = 8$, maintains a PSNR of 23.96 dB. In this application, again the degradation of the quality of service in negligible.

The hardware design characteristics of the application are given in Table 6. Here, use of the proposed approximate divider can achieve up to 64% in total power savings compared to the accurate design. In terms of critical path delay, the approximate design can achieve delay reductions of 2.44×.

5. CONCLUSIONS

In this work we proposed a design for approximate dividers targeted toward error resilient applications. Our divider is highly configurable, enabling designers to use different variants based on their accuracy needs. We demonstrate that the benefits of this design approach increase as the input width increases, making it highly scalable. We evaluate the design thoroughly and empirically in terms of accuracy, design area, total power savings, and circuit delay, and show that we can achieve significant savings while adding a small error to the divider results. In particular, our divider achieves up to 70% in power savings while introducing an insignificant 3.08% error in standalone analysis. Finally, we evaluate our design against three hardware implemented applications from the image processing domain and show that we can achieve up to 75% in power savings while degradations in quality-of-service are negligible.

Table 6: Area, power and delay characteristics using the proposed divider in foreground extraction.

Design	Area (um^2)	Power (mW)	Area Savings	Power Savings	Delay Reduction
Acc.	9192.96	0.6345	–	–	–
App.	7511.04	0.2283	18.30%	64.02%	2.44×

Acknowledgments: This research is supported by NSF grant 1420864.

6. REFERENCES

[1] L. Chen, J. Han, W. Liu, and F. Lombardi. Design of approximate unsigned integer non-restoring divider for inexact computing. In *Great Lakes Symposium on VLSI*, pages 51–56, 2015.

[2] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[3] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *European Test Symposium*, pages 1–6, 2013.

[4] S. Hashemi, R. I. Bahar, and S. Reda. Drum: A dynamic range unbiased multiplier for approximate applications. In *International Conference on Computer-Aided Design*, pages 418–425, 2015.

[5] J. Huang, J. Lach, and G. Robins. A methodology for energy-quality tradeoff using imprecise hardware. In *Design Automation Conference*, pages 504–509, 2012.

[6] D. Kelly, B. Phillips, and S. Al-Sarawi. Approximate unsigned binary integer dividers for arithmetic data value speculation. In *Conference on Design and Architectures For Signal And Image Processing*, pages 105–112, 2009.

[7] J. Liang, J. Han, and F. Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 62(9):1760–1771, 2013.

[8] H. Mahdiani, A. Ahmadi, S. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I*, 57(4):850–862, 2010.

[9] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. In *International Conference on Computer-Aided Design*, pages 728–735, 2012.

[10] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 1–6, 2014.

[11] R. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: a systematic survey. *IEEE Transactions on Image Processing*, 14(3):294–307, 2005.

[12] Y. Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar. Cdnet 2014: An expanded change detection benchmark dataset. In *Computer Vision and Pattern Recognition Workshops*, pages 393–400, 2014.

[13] L. Wu and C. C. Jong. A curve fitting approach for non-iterative divider design with accuracy and performance trade-off. In *New Circuits and Systems Conference*, pages 1–4, 2015.