Arithmetic Calculus Modeling for Approximate Circuits

Alain Aoun[1], Mahmoud Masadeh[1,2], Osman Hasan[3] and Sofiène Tahar[1]
a_alain@ece.concordia.ca, mahmoud.s@yu.edu.jo, osman.hasan@seecs.edu.pk, tahar@ece.concordia.ca
[1]Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada
[2]Computer Engineering Department, Yarmouk University, Irbid, Jordan
[3]Department of Electrical Engineering, National University of Sciences & Technology, Islamabad, Pakistan

***Abstract*—Approximate computing (AC) is a paradigm that introduces errors for reduced design metrics. AC has been recommended for implementation in error-resilient applications. Previously proposed AC implementations can be modified to generate new configurations and thus result in a large design space of AC. In this paper, we propose a mathematical modeling technique that allows a fast analysis of hardware designs with the aim of studying the quality of large design spaces. Mathematically modeling the AC circuits facilitates the quality to be assessed using various error metrics while reducing the assessment time. The proposed approach has been applied to various arithmetic units, including an array multiplier, a multiply accumulate unit and a divider. The experimental results showed a quality assessment equivalent to the one obtained by the Monte-Carlo simulation, while offering a remarkable reduction in CPU time.**

***Keywords*-Approximate Computing, Approximate Arithmetic, Mathematical Modeling, Sobel Filter, Digital Circuits**

I. Introduction

According to a study published in 2012 by the International Data Corporation (IDC), the world generated 2.8 zettabyte in 2012 and projected the world to generate 40 zettabyte by the year 2020 [1]. However, according to a recent study published by the IDC in 2021, the world generated 64.2 zettabytes in 2020 [2] and hence 50% more than what have been previously anticipated. Furthermore, the IDC projects that just the Internet of Thing (IoT) devices will generate 80 zettabyte by 2025 [3] which exceeds the data generated by all users and devices in the year 2020. On the other hand, the semiconductor manufacturers are unable to cope with the current demand. For instance Taiwan Semiconductor Manufacturing Company (TSMC) have seen an annual growth of 2 to 6% in recent years [4]. As hardware manufacturing is unable to meet the demand, actions are becoming necessary. One approach is the resource optimization for a given task.

Approximate computing (AC) or inexact computing emerged as a computing paradigm to optimize hardware usage. AC subsides output quality for saving on resources such as area, delay and power. Thus, AC has been predominantly recommended for implementation in error-resilient applications such as big data, image processing and machine learning [5]. An application is considered error-tolerant when its nature incorporates some of the following factors [6]: (*i*) noisy input signal, e.g., digital signals from various analog sensors; (*ii*) absence of golden result, e.g., search engines; (*iii*) imperfect sense of humans, e.g., dropped frame in a video call; and (*iv*) implementing algorithms with self-healing and error attenuation patterns. AC can be implemented at both hardware and software levels of computing systems. For instance, the work in [7] investigated an implementation of AC in software using a loop perforation technique to skip some portions of the code while having a minimal impact on the output quality. On the other hand, approximate hardware implementations mainly focus on modifying the structure of the circuit in order to deliver an approximate behavior. For instance, the work in [8] investigated implementations of approximate mirror full adders (FAs). The authors of [8] generated 5 versions of approximate FA in a manual iterative fashion which consists of removing a pair of transistors every time. This resulted in a FA that consists of only two buffers in the last iteration.

Given AC's viability for many applications, design space exploration (DSE) becomes essential in order to uncover the best implementation. DSE of AC can be performed either manually, such as the work in [8], or using automated tools such as "Approximate Units Generator (AUGER)" [9]. To the best of our knowledge, most of the DSE methods rely on Monte-Carlo simulation which is time consuming. Subsequently, we propose the analysis of AC hardware using mathematical modeling that allows the evaluation of the hardware using arithmetic calculus, e.g., using derivatives and integrals. In the rest of this paper, we present in Section II relevant error metrics used for the analysis of AC designs. In Section III we delve into the proposed modeling technique of AC design. Thereafter, we present experimental results in Section IV and conclude this paper in Section V.

II. Preliminaries

In this section, we overview conventional error metrics and quality assessment methods used for AC design evaluation. The generated error, due to the introduction of approximations, must not surpass a given threshold in order to make the AC design acceptable for practical usage. The output quality of an AC design is assessed by measuring the error in one of these forms:

- Error Distance (ED) is the arithmetic difference between the exact value (E_v) and the approximate value (A_v) for a given set of inputs. Hence the ED can be written as: $ED = |E_v - A_v|$.
- Relative Error Distance (RED) is the ratio of the ED with respect to the exact value (E_v), i.e., $RED = \frac{ED}{E_v}$.
- Mean Error Distance (MED) is the average of all ED in space.

979-8-3503-7939-6/24/$31.00 ©2024 IEEE

- Normalized Mean Error Distance (NMED) is measured to have a better analysis for the worst case scenario error. NMED is computed as $NMED = \frac{MED}{MAX}$ with MAX representing the maximum value in the domain, e.g., 255 for an 8-bit design.
- Mean Squared Error (MSE) provides a general overview of the quality by a design for a given set of n outputs. The MSE can be written as $MSE = \frac{1}{n}\sum_{i=1}^{n} ED_i^2$.
- Peak Signal to Noise Ration (PSNR) is used to measure the fidelity of the design. The PSNR represents the maximum possible value divided by the MSE and is expressed as a logarithmic quantity. The PSNR can be written as $PSNR = 10\log_{10}\left(\frac{MAX^2}{MSE}\right)$.
- Bit-Error Rate (BER) is the percentage of faulty bits in the output. It is noteworthy that the BER does not differentiate in error in the LSBs or the MSBs and thus it neglects the arithmetic difference, i.e., error magnitude.

Given these various error metrics, the AC designer has to carefully select the appropriate metric which is typically defined by the targeted application. For instance, the quality of a multimedia application is generally measured using PSNR. This is due to the fact that each metric highlights a given property of the results. Alternatively, it is noteworthy that the position of an erroneous bit is important for an arithmetic approximate unit. Thus, the BER metric is not the most suitable one for arithmetic units due to its inherent indifference of error occurring in the MSB, i.e., large error-magnitude, or in the LSB, i.e., small error-magnitude. Furthermore, the error or quality analysis of an AC hardware is generally conducted using simulation by testing various sets of inputs while logging the output. On the other hand, the simulation can be performed for all possibilities in space, in a sub-domain of interest or randomly [9]. A drawback of simulation-based analysis regardless of the chosen domain is the requirement of computation power, which can be unrealistic for large designs. On the other hand, a probabilistic error estimation may not represent the actual error rate.

III. Proposed Modeling

In this paper, we present the proposed modeling approach for combinational logic circuits (AC circuits included) using arithmetic calculus as depicted in Fig. 1. An arithmetic hardware that accepts two binary inputs and generates one binary output as shown in Fig. 1(a), can be modeled in the form of $Z = g(X, Y)$, where X and Y are the applied decimal inputs and Z is the decimal output of the hardware unit as depicted in Fig. 1(b). The aim of the proposed modeling technique is to assess the quality of an approximate hardware using Calculus theories instead of the commonly used techniques as they are difficult even under bounded and well behaved scenarios [10]. For instance, by using the Calculus theories of derivatives and integrals, various error behaviors of the AC circuit can be determined. Towards this goal, we propose the implementation of two operations. The first operation generates the Boolean equation of the outputs as function of the Boolean inputs. The second operation is a conversion of the generated Boolean equations in terms of the binary representations of the inputs into a single decimal equation in terms of the decimal representation of the inputs. To this aim, we propose a three step conversion that consists of:
C1) Decimal to Binary converter.
C2) Equivalence of logic operation in arithmetic calculus.
C3) Binary to Decimal converter.
The third step (**C3**) of this conversion is commonly known, i.e., $(D)_{10} = \sum_{i=0}^{n-1} f_i \times 2^i$ where $(D)_{10}$ is the decimal value of a n-bit binary number consisting of f_i binary values. Its computation complexity is also known to be $\boldsymbol{O(1)}$ since it is the summation of n terms for an n-bit representation. In the sequel, we describe the first two operations (**C1** and **C2**) and discuss their computation complexities.

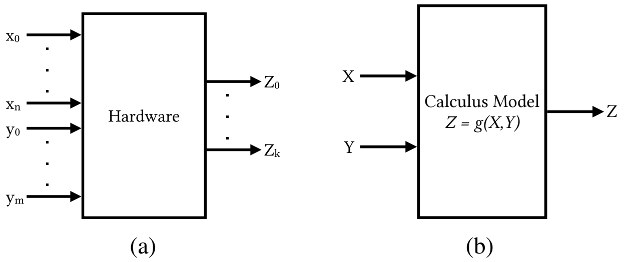


Fig. 1: Schematic of the: (a) Hardware to be Assessed; and (b) its Calculus Model

1) Decimal to Binary Conversion: The decimal to binary conversion can be achieved by either dividing the decimal number by two or finding the largest possible power of 2 that is less than the decimal number. The first approach, which we adopted in this work, computes the LSB first while the second computes the MSB first. In the first approach, the remainder of the n-th division by 2 will identify the n-th bit. Furthermore, a division of 2 with the remainder of 1 identifies an odd dividend while a remainder of 0 identifies an even dividend. On the other hand, a similar property can be deduced for the division of a number by looking at the fraction part of the result, i.e., $frac(x) = x - \lfloor x \rfloor$ where x is the result of the division. The property classifies an even dividend when the resulting division by 2 has a $frac(x) \in [0; 0.5[$ while the division of an odd dividend has a $frac(x) \in [0.5; 1[$. Hence, the property of the resulting $frac(x)$ can be used to deduce the binary representation.

Table I shows the division of 22 by 2^n, where we notice that the binary representation of 22 matches the Boolean values of $frac\left(\frac{22}{2^n}\right) \in [0.5; 1[$ when we identify "$true = 1$" and "$false = 0$". Furthermore, if $\frac{x}{2^n} \leq 0.5$ then all the subsequent divisions by 2^n result in a $frac(x) \leq 0.5$. Thus, all subsequent divisions will result in a logic '0' and a faulty deduction of a logic '1' in the MSB is not possible. For instance, all divisions of 22 by 2^n where $n \geq 6$ will result in a $frac(x) \in [0; 0,5[$ and hence all the deduced MSBs from the subsequent divisions are logic '0'.

TABLE I: Decimal to Binary Conversion for $x = 22$

n	1	2	3	4	5	6	7
$x/2^n = 22/2^n$	11	5.5	2.75	1.375	0.6875	0.34375	0.171875
$frac(x/2^n) \in [0.5; 1[$	×	✓	✓	×	✓	×	×
Binary Representation	0	1	1	0	1	0	0

Thereafter, we propose a mathematical function that classifies any number x to one of two constants, i.e., 0 or 1, based on its fractional part. Since the classification is periodic for every interval of 1, we propose the usage of a sin function as it offers periodic output. Moreover, in general, dividing any mathematical function by its magnitude, i.e., absolute value, results in a value of -1 or 1 if and only if its magnitude is not 0. Thus, applying this division to the sin function generates a square signal that has two values -1 and 1. The proposed mathematical equation with a period of 1 is:

$$h(x) = -\frac{\sin(2\pi x)}{|\sin(2\pi x)|} \tag{1}$$

Fig. 2 represents $h(x)$ graphically. The proposed function has a periodic interval of 1 with the half period being 0.5. Subsequently the proposed $h(x)$ shown in Eq. (1) classifies all numbers x to -1 or 1 if the $frac(x) \in [0;0.5[$ or $frac(x) \in [0.5;1[$, respectively if and only if $frac(x) \neq 0$ and $frac(x) \neq 0.5$. Furthermore, since $\sin(k\pi) = 0$ for $k \in \mathbb{N}$, the proposed $h(x)$ in Eq. (1) is undefined if $frac(x) = 0$ or $frac(x) = 0.5$. For this reason, we add a small positive number, i.e., ε, to the argument of sin to shift its graph to the left by a distance of $\sim \varepsilon$. In general, the upper bound of $frac(x)$ for an n-bit representation when $frac(x) \in [0.5;1[$ is equal to $\frac{2^n-1}{2^n}$, i.e., $x = (2^n - 1)$ where all its binary digits are logic '1' and all division results in $frac(x) \in [0.5;1[$. Alternatively, the upper bound of $frac(x) \in [0;0.5[$ for an n-bit representation is $\frac{2^{n-1}-1}{2^n}$, i.e., all divisions result in $frac(x) \in [0.5;1[$ except the last one where $frac(x) \in [0;0.5[$. Hence, to avoid undefined results we write $\varepsilon < \left(1 - \frac{2^n-1}{2^n}\right)$ which can be simplified to $\varepsilon < 2^{-n}$. The added ε makes Eq. (1) defined when $frac(x) = 0$ and $frac(x) = 0.5$ while computing the anticipated value when $frac(x) \in [0;0.5[$ and $frac(x) \in [0.5;1[$. On the other hand, the binary numbers are 0, 1 instead of -1, 1; and x is a result of division by 2^n, the function $h(x)$ proposed in Eq. (1) has to be adopted to accommodate these behaviors. Hence, we propose the addition of $+1$ and dividing the function by 2 to achieve the 0 and 1 output. We replace x with X divided by 2^n to represent the n-th bit. Therefore, the equation $b(X,n)$ that accommodates $h(x)$ to the requirements stated earlier can be written as:

$$b(X,n) = 0.5 \times \left(1 - \frac{\sin\left(\frac{\pi X}{2^n} + \varepsilon\right)}{\left|\sin\left(\frac{\pi X}{2^n} + \varepsilon\right)\right|}\right) \tag{2}$$

using $b(X,n)$, i.e., the n-th $b$inary representation of X, proposed in Eq. (2), finding the n-th bit representation of a number X is transformed to a relation that computes the n-th digit independently without the need of computing the preceding or succeeding bits. Since the usage of Eq. (2) to compute the i-th and j-th binary representation of a decimal number X (where $i \neq j$) requires the same computational complexity, i.e., by simply replacing n in Eq. (2) with i and j, we identify the proposed conversion as an $\boldsymbol{O(1)}$ equivalence.

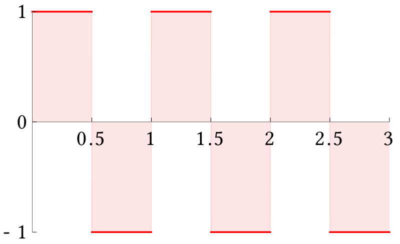


Fig. 2: Graphical Representation of $h(x)$

2) Logic Operators Equivalence: using the conversion proposed earlier, the Boolean outputs of an arithmetic unit that produces k binary outputs, i.e., $f_1,\ f_2, ...,\ f_k$, and accepts two n-bit and m-bit inputs, i.e., $x_1, x_2, ..., x_n$ and $y_1, y_2, ..., y_m$, can now be written in terms of the decimal inputs, $f_1(X,Y), ..., f_k(X,Y)$, instead of the binary inputs, $f_1(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m), ...,\ f_k(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m)$, where X and Y are the decimal representations of the binary inputs x and y, respectively. The concept can be generalized to any arithmetic unit that produces k binary outputs, i.e., $f_1,\ f_2, ...,\ f_k$, using m decimal inputs, i.e., $IN_1, IN_2, ..., IN_m$, in the form of $f_1(IN_1, IN_2, ..., IN_m), ..., f_k(IN_1, IN_2, ..., IN_m)$. The next step is to convert the Boolean operations into arithmetic calculus operations. This conversion aims to transform $f_1,\ f_2, ...,\ f_k$ into arithmetic functions and hence the third step, i.e., $(D)_{10} = \sum_{i=0}^{k-1} f_i \times 2^i$, becomes a function of the decimal inputs, e.g., $IN_1, IN_2, ..., IN_m$. Towards this goal, we propose the usage of the equivalences shown in Table II.

TABLE II: Logic-Arithmetic Equivalence

Gate	Boolean Operator	Arithmetic Equivalence
INV	$!A$	$(1 - A)$
AND	$A \wedge B$	$A \times B$
OR	$A \vee B$	$A + B - (A \times B)$
XOR	$A \oplus B$	$A + B - (2 \times A \times B)$

The equivalences for the OR and XOR operations do not scale well when applied recursively, i.e., doubling the size of the equation after every iteration. On the other hand, the INV and AND equivalences scale linearly. Hence, in the scope of our application, we propose rewriting the OR operation in terms of INV and AND solely. For instance, the Boolean operation $A \vee B \vee C$ can be rewritten as $\overline{\overline{A \vee B \vee C}} = \overline{\overline{A} \wedge \overline{B} \wedge \overline{C}}$. Thereafter, the arithmetic conversion of the three input OR becomes $\{1 - [(1-A) \times (1-B) \times (1-C)]\}$ instead of $\{A + [\ B + C - (B \times C)]\ - A \times [\ B + C - (B \times C)]\ \}$. Alternatively, the XOR of n-terms is defined by the number of occurrences of logic a '1', i.e., odd and even occurrences of logic '1' result in a logic '1' and '0', respectively. Therefore, we propose the usage of $\sin^2\left(\frac{\pi x}{2}\right)$. The proposed equation results in a value of '0' and '1' when x is even and odd, respectively. Subsequently, the XOR of n-terms, i.e., $A_1,\ A_2, ...,\ A_n$, can be written as $\sin^2\left(\frac{\pi \times (\sum_{i=1}^{n} A_i)}{2}\right)$. For instance, we convert the Boolean operation $A \oplus B \oplus C$ using sin to an arithmetic operation in the form of $\sin^2\left(\frac{\pi \times (A+B+C)}{2}\right)$. The conversion using the equivalences of INV and AND from Table II along with the proposed alternative representations of OR and XOR results in a constant growth of computation complexity. Consequently, we identify the proposed conversion as $\boldsymbol{O(1)}$.

using the three consecutive conversions proposed in the previous sections, the combinational circuit can be represented in arithmetic calculus, i.e., $Z = g(X,Y)$. The modeling will allow the analysis of quality to be performed in a mathematical manner rather than the usage of excessive simulation. For instance, if the targeted arithmetic operation is multiplication, then the exact operation can be written as $f(X,Y) = X \times Y$. Thus, the error distance can be analyzed using $ED(X,Y) = |f(X,Y) - g(X,Y)|$. Other arithmetic metrics can also be modeled such as the RED where it can be expressed as $RED = \frac{ED(X,Y)}{f(X,Y)}$. Moreover, by computing the integral of $ED(X,Y)$ the MED and MSE metrics can be estimated. For instance, we can estimate $MED = \left[\frac{1}{X_{limit} \times Y_{limit}} \int_{X=0}^{X_{limit}} \int_{Y=0}^{Y_{limit}} ED(X,Y)\ dY\ dX\right]$ and the estimate of the MSE can be computed as $MSE = \left\{\frac{1}{X_{limit} \times Y_{limit}} \int_{X=0}^{X_{limit}} \int_{Y=0}^{Y_{limit}} [ED(X,Y)]^2\ dY\ dX\right\}$. Subsequently we can compute $NMED$ and $PSNR$. Moreover, the worst-case error, i.e., $max(ED(X,Y))$, can be found by studying the sign of the derivative of $ED(X,Y)$ and hence determining the peak point(s) of the function $ED(X,Y)$. On the other hand, if for some specific applications the BER metric is the point of interest, the proposed modeling technique can be adapted to analyze AC configurations in such fashion.

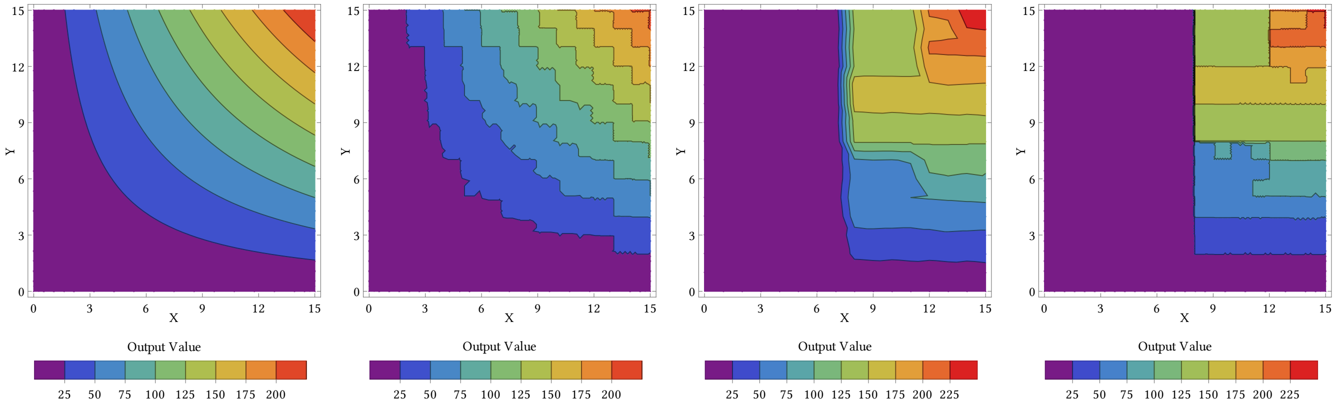
In summary, the proposed calculus modeling and three-step conversion operations (**C1-C3**) have a constant growth in the computation complexity, i.e., ***O*(1)**, where the number of mathematical operations have a constant growth regardless of the size. We will show its application on large design spaces in the forthcoming experimental results section.

IV. Experimental Results

The proposed mathematical modeling of AC circuits is tested on *i*) an array multiplier, and *ii*) a *SOBEL* filter [11] using Wolfram Mathematica [12]. Moreover, the conversions of Boolean equations to a single calculus equation requires to search for patterns in the text and modify appropriately. For this purpose, we used REGEX [13] as it is one of the simplest and efficient way to search and replace patterns in a text. The Mathematica scripts and REGEX replacement are executed on a computer with Intel(R) Core(TM) i5-4278U CPU @ 2.60GHz and 8GB of RAM running macOS 10.15.

A. Evaluating Correctness of the Calculus Modeling

As a first study, we tested the proposed decimal to binary conversion, i.e, Eq. (2), in order to validate its correctness. For this purpose, we tested all $X < 2^{16}$ where $X \in \mathbb{N}$ with results in line with the conventional decimal to binary conversion. As a next step, we validated the proposed modeling of combinational circuits, where the results of this study are summarized in Fig. 3 which represents the output values for a 4×4 multiplier when generated using different approaches. Fig. 3(a) represents the contour plot of $X \times Y$ for all values less than 16. Fig. 3(b) depicts the contour plot for the exact implementation of the 4×4 multiplier when modeled using the proposed calculus modeling. Figs. 3(c) and 3(d) represent the contour plot for an approximate multiplier when the data is computed using simulation and the proposed calculus modeling, respectively. From all plots, presented in Fig. 3, we can notice that the pattern of output is preserved. The difference between the plots comes from the fact that the proposed binary conversion is a step function in the continuous domain, i.e., $X \in \mathbb{R}$. Fig. 4 depicts the plot of a number X, and its decimal value when generated from its binary representation proposed in Eq. (2). We can notice that the values generated by the conversion are exact only when $X \in \mathbb{N}$ while the error is present when $X \in (\mathbb{R}^+ - \mathbb{N})$. Moreover, one major difference between Figs. 3(c) and 3(d) is the fact that the simulation data is in a discrete domain, while the latter is in a continuous domain. To this aim, we analyzed the equation used in Fig. 3(d) in a discrete domain, i.e., $X \in \mathbb{N}$, and the output values computed matched with the data used to generate Fig. 3(c). Similarly, we analyzed the equation used to generate Fig. 3(b) in a discrete domain and the resulting values are equal to the exact values. Furthermore, since the proposed modeling technique saves computation power when analyzing large circuits, a noisy analysis of output can be deemed acceptable.



(a) (b) (c) (d)

Fig. 3: Contour Plot for: (a) $X \times Y$; (b) Exact Hardware Implementation using the Proposed Mathematical Modeling; (c) Approximate Implementation using Simulation Data; and (d) Approximate Implementation using the Proposed Mathematical Modeling

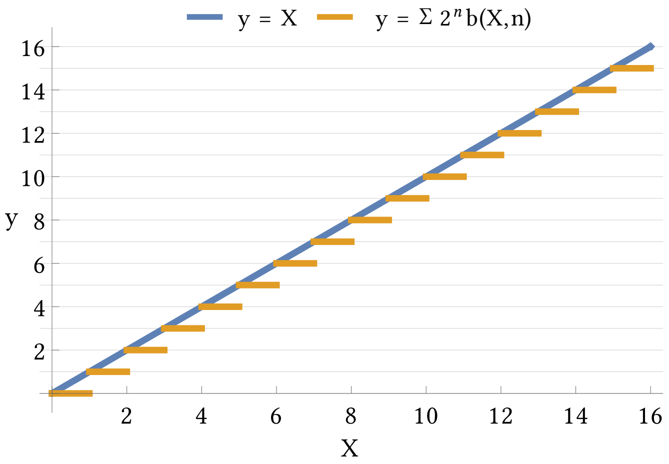


Fig. 4: Decimal Values and the Regenerated Decimal Values from their Binary Representation using Eq. (2)

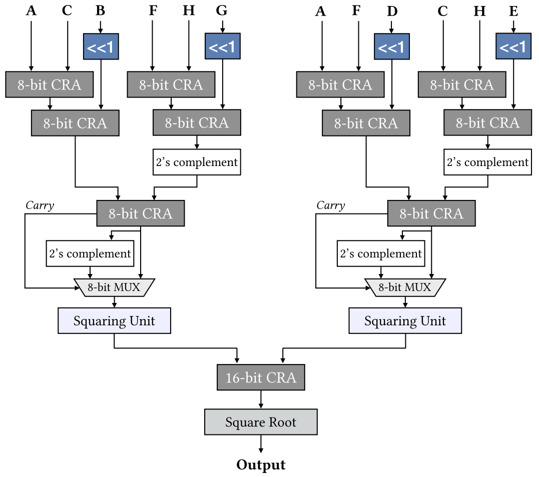
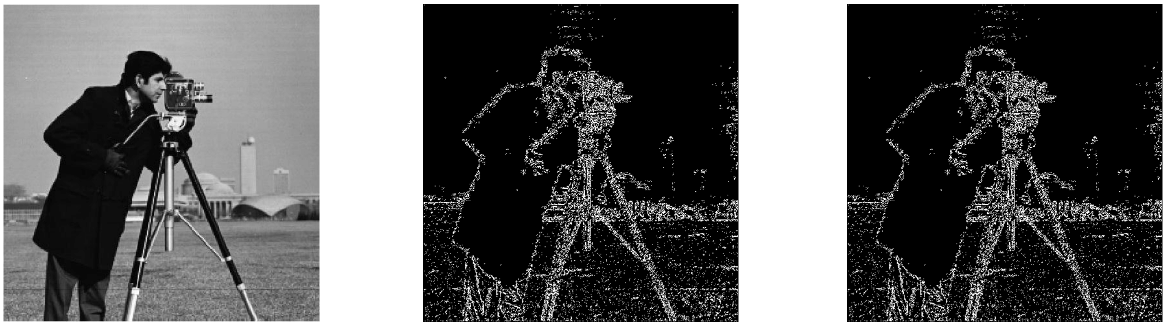


Fig. 5: Hardware Implementation of the *SOBEL* Filter



(a) (b) (c)

Fig. 6: (a) Input Image; (b) Computation of the *SOBEL* Filter when using Simulation; and (c) Computation of the *SOBEL* Filter when using the proposed Calculus Modeling

B. Modeling Multi-Operations Design

In the second investigation, we applied our methodology on a *SOBEL* filter [11] which identifies edges by detecting the minimum and maximum values in the first derivative of the image. Its implementation consists of Carry-Ripple Adders (CRAs), Squaring Units, Square Root, and Multiplexer, as shown in Fig. 5. The *SOBEL* filter was successfully modeled using the proposed Boolean to calculus conversions. On the other hand, since the *SOBEL* filter is a system that accepts more than 3 variables, representing the output in a plot is not feasible. Alternatively, we showcase the study by applying the *SOBEL* filter on the 'camera man' image, i.e., Fig. 6(a). The resulting computations when using simulation and the proposed calculus modeling are shown in Figs. 6(b) and 6(c), respectively. The numerical values obtained when using both techniques are identical. It must be noted that designs consisting of multi-operations, e.g., addition and multiplication, where only part of the design is approximated can be modeled in a hybrid mode. For instance, if the 16-bit CRA and the Square Root shown in Fig. 5 are exact operations while the other operations are approximated, then the final output can be written as $Output = \sqrt{In_1 + In_2}$ where In_1 and In_2 are the results from the squaring units, i.e., models of the approximate operations. Using such hybrid modeling will reduce the complexity of the equations and thus allowing the proposed technique to have a better scalability for more complex designs. Furthermore, the hybrid modeling will result in a faster quality assessment when compared to pure system modeling.

C. Extensibility of the Proposed Mathematical Modeling

In a different examination of the proposed mathematical modeling, we studied the extensibility to larger designs, i.e., 32-bit and 64-bit multipliers. The runtime to generate Boolean equations for given approximate 32-bit and 64-bit array multiplier configurations are shown in Figs. 7 and 8. The time consumed per row in the tree of the array multiplier and the cumulative runtime for the 32-bit array multiplier are shown in Fig. 7. The generations of Boolean equations for the 32-bit approximate multiplier topped at 1,453 seconds (∼24 minutes). On the other hand, as depicted in Fig. 8, the generation of the equation for the 64-bit multiplier presented an exponential growth in the runtime. As a result, we terminated the experiment after generating the equations for the first 22 rows, which required 25,560 seconds (∼7 hours). It must be noted that the generation of the Boolean equations grows exponentially. However, if the output equations were successfully generated, the subsequent operations would have been successful as they are of $\boldsymbol{O(1)}$. Furthermore, assessing the quality using the mathematical modeling for the 32-bit multiplier took 11 hours of computation time. This brings a total of 11.5 hours to analyze a 32-bit multiplier from

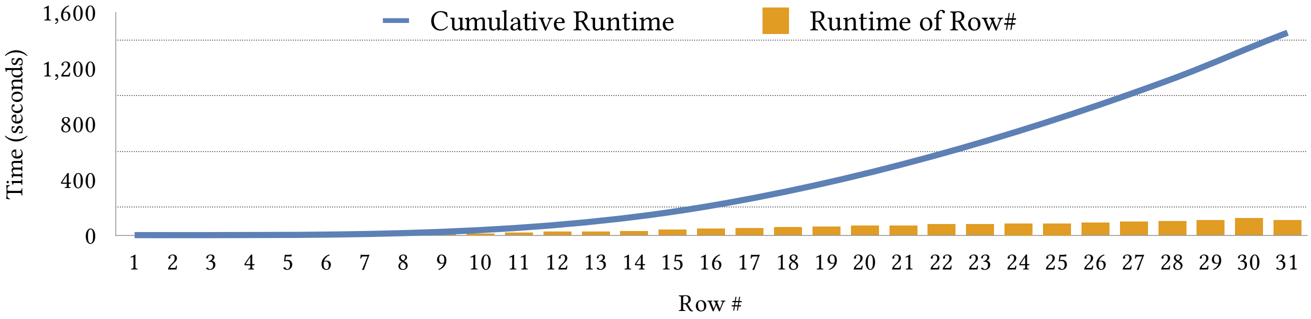


Fig. 7: Runtime to Generate the Output Equations for a 32-bit Array Multiplier

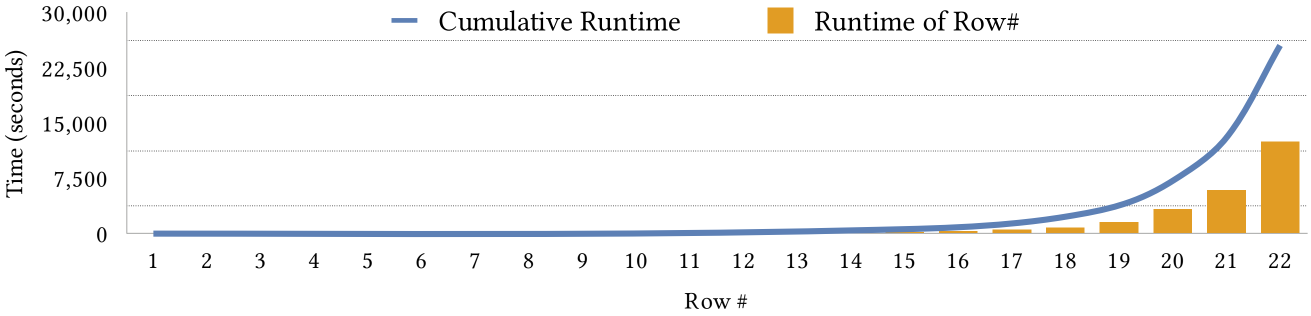


Fig. 8: Runtime to Generate Some of the Output Equations for a 64-bit Array Multiplier

start to finish. On the other hand, if the same design had to be analyzed using excessive simulation using an HPC, the projected runtime is estimated to be in the range of 3.4 to 4.3 $\times$ 10^{10} hours, i.e., thousands of decades.

Based on the investigations performed, we observe that using the proposed mathematical modeling provides a reduction of the time spent to perform the quality assessment. For instance, the quality assessment of the 224 approximate configurations of the 16-bit multiplier took less than 1 day. Similarly, the assessment time of the 32-bit multiplier took 11.5 hours instead of decades.

V. Conclusion

Approximate computing is gaining traction as an alternative to conventional computation units since they offer large savings in various physical aspects. Many techniques of approximation have been proposed so far yet most of them are tested in a limited set of configurations. Thus, a wide study using DSE tools is essential in order to discover the limitations and the outstanding configuration(s) of previously proposed techniques. In this paper, we proposed a modeling technique that uses calculus math to model arithmetic circuits. The proposed technique permits the quality analysis to be performed in math instead of simulation and thus resulting in a huge savings in analysis time. The proposed modeling has been tested in various aspects with the aim of determining its correctness, capabilities and limitations. The methodology has been tested for a hardware implementation that accepts multivariate and incorporates various operations, i.e., the *SOBEL* filter. The proposed modeling is able to model various approximate configurations and can support a wide range of error metrics. As a future work, we plan to integrate the proposed modeling technique in a DSE tool in order to provide a tool that efficiently finds the most suitable design.

References

[1] Dell Press Release, “New digital universe study reveals big data gap less than 1 of world s data is analyzed less than 20 is protected,” 2012. [Online]. Available: https://www.dell.com/en-us/dt/corporate/newsroom/announcements/2012/12/20121211-01.htm

[2] IDC, “Data creation and replication will grow at a faster rate than installed storage capacity, according to the IDC global datasphere and storagesphere forecasts,” 2021. [Online]. Available: https://www.businesswire.com/news/home/20210324005175/en/

[3] IDC Blog, “Future of industry ecosystems: Shared data and insights,” 2021. [Online]. Available: https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/

[4] Taiwan Semiconductor Manufactoring Company, “Fab capacity,” 2023. [Online]. Available: https://www.tsmc.com/english/dedicatedFoundry/manufacturing/fab_capacity

[5] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *European Test Symposium*. IEEE, 2013, pp. 1–6.

[6] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Approximate computing and the quest for computing efficiency,” in *Design Automation Conference*. ACM, 2015, pp. 1–6.

[7] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, “Managing performance vs. accuracy trade-offs with loop perforation,” in *European Conference on Foundations of Software Engineering*. ACM, 2011, pp. 124–134.

[8] 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*, vol. 32, no. 1, pp. 124–137, 2013.

[9] D. Hernández-Araya, J. Castro-Godínez, M. Shafique, and J. Henkel, “AUGER: A tool for generating approximate arithmetic circuits,” in *Latin American Symposium on Circuits & Systems*. IEEE, 2020, pp. 1–4.

[10] O. Keszocze, M. Soeken, and R. Drechsler, “The complexity of error metrics,” *Information Processing Letters*, vol. 139, pp. 1–7, 2018.

[11] I. Sobel and G. Feldman, “A 3×3 isotropic gradient operator for image processing,” *Pattern Classification and Scene Analysis*, pp. 271–272, 1973.

[12] Wolfram, “Wolfram Mathematica,” 2024. [Online]. Available: https://www.wolfram.com/mathematica/

[13] J. Goyvaerts, “The premier website about regular expressions,” 2021. [Online]. Available: https://www.regular-expressions.info/