FAU: Fast and Error-Optimized Approximate Adder Units on LUT-Based FPGAs

Jorge Echavarria, Stefan Wildermann, Andreas Becher, Jürgen Teich and Daniel Ziener
Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
{jorge.a.echavarria, stefan.wildermann, andreas.becher, juergen.teich, daniel.ziener}@fau.de

***Abstract*—During the design of embedded systems, many design decisions have to be made to trade off between conflicting objectives such as cost, performance, and power. Approximate computing allows to optimize each objective, yet for the sake of accuracy. This means that a functional flaw is allowed to produce an error as long as this is small enough to maintain a feasible operation of the system or guarantee a certain accuracy of the results. In this paper, we propose a new technique for approximate addition optimized for LUT-Based FPGAs with segmented carry chains. Our optimized adder structure is able to a) best exploit artifacts of LUT-Based FPGAs such as unused inputs and b) provide a smaller average error than previously proposed approximate adder structures, as well as c) a reduced critical path delay than dedicated accurate logic in modern FPGAs. We present a novel stochastic error calculus that is able to take into account also non-uniform input distributions and present a detailed comparison of approximate adder structures proposed in literature with our novel LUT-Based approximate arithmetic structure.**

***Keywords*-Approximate Adder, Approximate computing; LUT-Based FPGAs; Low-latency Adders.**

INTRODUCTION

The amount of data to process, and computation needed in modern IT systems, is rapidly increasing day by day. Hardware technology is improving at a pace just as fast. However, there is still a gap to efficiently utilize available resources, e.g., for battery driven embedded systems, as battery technology cannot keep up with this development. Fortunately, applications are often tolerant to errors or intrinsically resilient to an imprecise computation. Multimedia processing, for example, often grants certain amount of accuracy relaxation, mostly because of human senses limitations. With approximate computing [3], it is possible to consciously perform data processing within a bounded accuracy of the computed results. This allows to trade off accuracy to reduce power consumption and/or delay of a circuit.

Adders are the basic building blocks in arithmetic circuits. Therefore approximate adders have been intensively investigated. However, exploiting reconfigurable logic such as FPGAs in particular, is still an emerging topic.

In this paper, we propose a methodology for building approximate adders that are tailored to modern LUT-based FPGAs.

Related work

Several approximate arithmetic designs have been proposed following different approaches. Approximate adders are generally focused to reduce resources, power consumption and latency.

One of the most common techniques is applied in ETA [6] which splits the addends in two parts to parallelize the addition process. To reduce the error distance, the authors propose to set the least significant bits to '1' when detecting a cut carry propagation. In [4], an approximate adder aimed to improve throughput and power consumption is proposed. The approach named ACA for accuracy-configurable adder, varies the accuracy of the result at runtime. However, the proposal is similar to ETA, with the slight, but important difference of introducing an extra sub-adder to calculate the middle part of the result. This idea increases the accuracy as it is more probable to predict an accurate carry for the most significant bits due to a given degree of *visibility* from lower bits. Finally, Shafique et. al. proposed a generic accuracy-configurable adder [7] which incorporates the overlapping idea from ACA. The major difference is the capability of integrating any given amount of sub-adders with variable degrees of *visibility*. However, their approach does not include any fixing technique within the same design, but extra circuitry with an expensive timing overhead.

DESIGN METHODOLOGY

Most approximate adder approaches split the carry chain to reduce circuit delay or to save resources. Also our methodology follows this approach, see Fig. 1. Splitting the carry chain allows to perform the addition with two parallel adders. We call those adders most and least significant parts (MSP and LSP, respectively). Without any error correction technique, this approach introduces an inaccuracy generally of 2^m if m is the carry chain splitting point. However, the circuit delay is significantly reduced. For example, an n-bit accurate carry-ripple adder has a critical path of n full adders. By splitting the carry chain at position m, the critical path (CP) is now proportional to the amount of full adders in the longest part.

Based on the idea of [10], most approximate adders provide an *all1* error reduction mechanism which sets least significant bits to '1' when an error occurs in the LSP. Our *all1* mechanism is based on the proposal from [1] which has the idea to set all output bits of the LSP to '1' when there is a carry out c_{m-1} at the splitting point. The maximum error of this approach is 2^m-1.

As a new idea to reduce the error distance, we propose the sharing of input bits from the LSP with the MSP. Note that if two input bits at any given position i are '1', then it is said that position i is *generating* (G_i) a carry c_i, if both input bits are '0', then position i *kills* (K_i) carry c_{i-1}, and if both input bits are different they *propagate* the carry, that is, $c_i = c_{i-1}$.

Obviously, augmenting the amount of 'shared' inputs from 1 to any given value $p \leq m$ will increase the probability of finding an input pair (a_i, b_i) *generating* a carry c_i with $i \in [m-p, m-1]$ that should be *propagated* to position m, thus, reducing the error rate. On the one hand, MSP and LSP can be implemented following any preferred addition technique depending on specific pursued goals. This obviously includes different techniques as carry look-ahead adders, for example, or even dedicated modules as carry logic units in modern FPGAs. On the other hand, LUTs have typically more than the 3 required inputs to implement a full adder. Our idea of sharing information between MSP and LSP can be easily applied by exploiting unused LUT inputs as illustrated in Fig. 1.

Moreover, available but unneeded inputs may be exploited by synthesis tools also to reduce the overall latency. Our experimental results give indication that place and route tools indeed utilize LUTs with only partially used inputs for the shared p inputs, thus avoiding any resource overhead.

ERROR ANALYSIS

In the following, we are presenting an error analysis for the proposed approximate addition technique. Many related works

978-1-5090-5602-6/16/$31.00

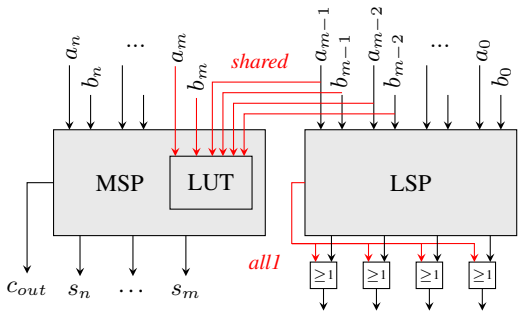


Fig. 1: Outline of our proposed approximate adder.

assume uniformly distributed inputs for their analysis. This is acceptable when no complex input statistics, as, e.g., assumed in [5], are available. However, even when assuming that inputs of modules are uniformly distributed, their output distribution which may serve as input of subsequent modules is no longer uniformly distributed in the general case.

Definition 1 (FAU Adder)**:** The proposed approximate adder with bit-width n is defined by the splitting point m of the carry chain and by the degree of *visibility* $p \in [0,m-1]$ used to predict carry c_m.

Definition 2 (Input Distributions)**:** Let $\rho_1(x)$ be the probability for an input bit x to be '1', and $\rho_0(x)=1-\rho_1(x)$ the probability for x being '0'. The *probability for generating* a carry at bit position i is $\rho(G_i) = \rho_1(a_i) \cdot \rho_1(b_i)$, whereas the *probability for propagating* a carry is $\rho(P_i)=\rho_1(a_i)\cdot\rho_0(b_i)+\rho_0(a_i)\cdot\rho_1(b_i)$.

Arithmetic Error Rate (AER)

Let $S = (s_n, \ldots, s_0)$ be the tuple of output bits of our approximate adder technique. The *arithmetic error rate* represents the probability of getting an erroneous output S.

Observe that our approximate adder will incur in an error if input bits at position $0\leq i\leq m-p-1$ *generate* a carry and all input bits at positions $i<j\leq m-1$ *propagate* the carry (refer to § II for details).

Now, the probability of *generating* a carry at position i while at the same time *propagating* it to position m can be calculated according to the expression $\rho(G_i)\cdot\prod_{j=i+1}^{m-1}\rho(P_j)$.

Thus, the arithmetic error rate can be calculated by accumulating all probabilities of generating and propagating from positions $i=0,1,...,m-p-1$ as shown in Eq. (1):

$$AER=\sum_{i=0}^{m-p-1}\left[\rho(G_i)\cdot\prod_{j=i+1}^{m-1}\rho(P_j)\right]. \tag{1}$$

While Eq. (1) represents the general case, Eq. (2) presents the result for uniformly distributed inputs, i.e., $\rho_1(a_i) = \rho_0(a_i)=1/2$ and $\rho_1(b_i)=\rho_0(b_i)=1/2$ for $i=0,1,...,n-1$.

$$AER=\sum_{i=0}^{m-p-1}\left[\frac{1}{4}\cdot\prod_{j=i+1}^{m-1}\frac{1}{2}\right]=\frac{1}{2}\left(2^{-p}-2^{-m}\right). \tag{2}$$

Maximum Error (ME)

Most of the approximate adders with split carry chain found in the literature propose an *all1* mechanism based on ETA [6]. However, their maximum error is generally 2^m-1, due to the carry ripples towards the LSB, making it impossible to fix previous bits. Or, their approach has zero visibility in the LSP to predict carry c_{m-1} [6]. In other cases, authors have proposed to share inputs [2], [7]. However, this is only done for neighboring bits [2], or the approach is forced to have more than one set of shared p inputs with a high overhead and without any fixing technique [7]. Thanks to the higher visibility in our design, and the fixing technique applied, we may greatly reduce the maximum error. In our proposed adder technique, ME depends on the amount of 'visible' input bits p and the splitting point m. The error can be calculated according to Eq. (3). Note that ME does not depend on the addends bit-width n.

$$ME=2^{m-p}-1. \tag{3}$$

Mean Error Distance (MED)

We need to consider all the erroneous results to calculate the average error for arbitrary input distributions as follows:

$$MED=\sum_{\delta\in[0,ME]}\delta\sum_{\substack{\forall\mathbf{a}[m-p-1:0]\\+\mathbf{b}[m-p-1:0]\\=2^{m-p}+\delta+1}}\rho(\mathbf{a}')\cdot\rho(\mathbf{b}')$$

Above, $\mathbf{a}'$ and $\mathbf{b}'$ represent $\mathbf{a}[m-p-1:0]$ and $\mathbf{b}[m-p-1:0]$, respectively. The first sum iterates over all possible error distances δ, the second sum iterates over all $\mathbf{a}$ and $\mathbf{b}$ leading to that error (see Eq. (5)). This approach has exponential complexity. However, for uniform distributions, the mean error distance can be calculated analytically, without having to evaluate all input combinations, as shown next. Let $\Delta_\delta=\{(x,y)|\Delta(\mathbf{a}=x,\mathbf{b}=y)=\delta\}$ represent the set of input combinations that lead to error distance δ. The mean error distance may be calculated by accumulating all possible error distances δ times their total amount of occurrences $|\Delta_\delta|$ and averaging it by the total amount of possible inputs:

$$MED=\frac{1}{2^{2n}}\sum_{\delta=1}^{2^{m-p}-1}|\Delta_\delta|\cdot\delta. \tag{4}$$

The number $|\Delta_\delta|$ of possible input combinations leading to error distance δ can be calculated as follows. An error occurs when the $m-p$ least significant bits (LSBs) produce a carry at c_{m-p-1} and when all p shared bits are propagating this carry. In this case, the $(m-p)$ LSBs of the two addends lead to an error distance of δ when their sum is:

$$\mathbf{a}[m-p-1,0]+\mathbf{b}[m-p-1,0]=2^{m-p}+\delta-1=\omega(\delta) \tag{5}$$

The maximum possible value of such an $(m-p)$-bit addend is $\tau=2^{m-p}-1$. With this, the amount of all possible combinations of $(m-p)$-bit addends that fulfill Eq. (5) is given by:

$$P_{\text{LSB}}(\delta)=2\cdot\tau-\omega(\delta)+1.$$

Furthermore, the number of combinations for the p shared bits that propagate carry c_{m-p-1} is given by:

$$P_{\text{shared}}=2^p.$$

Finally, an error can be produced for any combination of the MSP's input bits. The number of the possible combinations is given by:

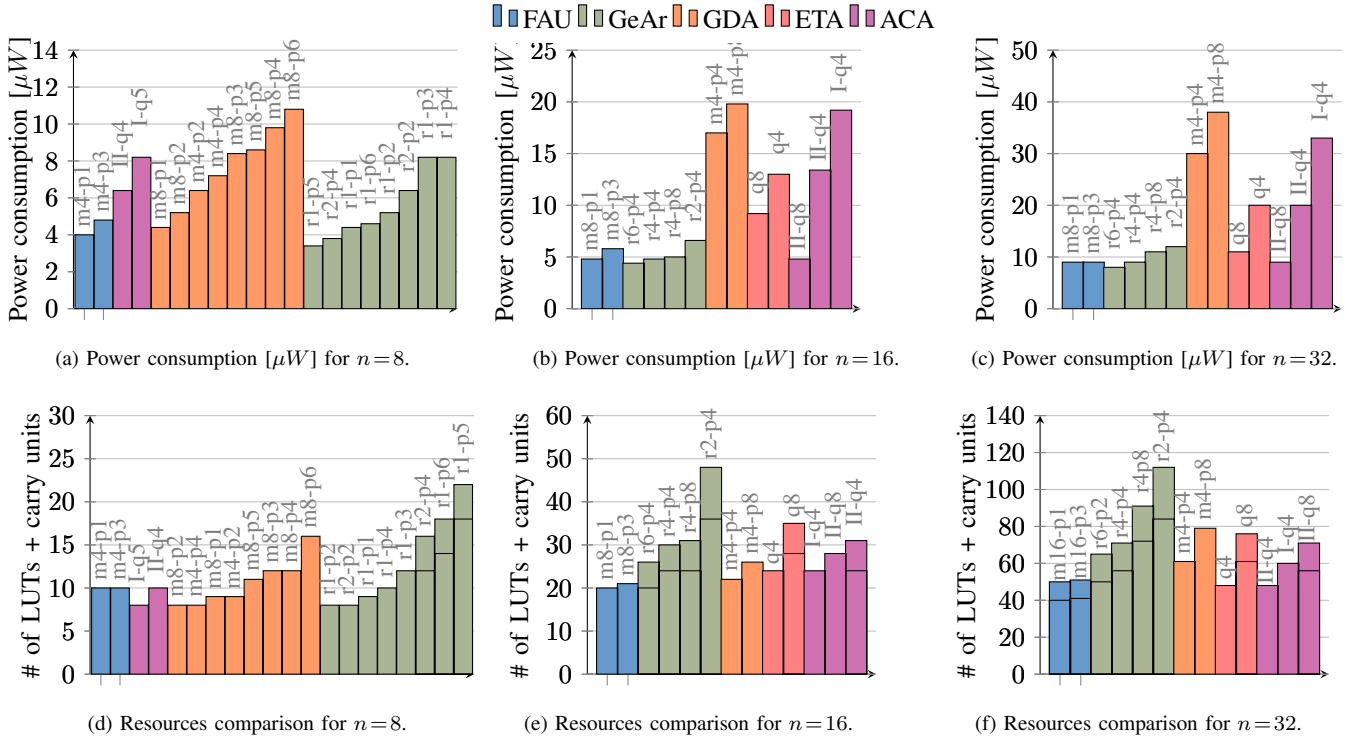
$$P_{\text{MSP}}=2^{2(n-m)}.$$

With this, $|\Delta_\delta|$ can be calculated as:

$$|\Delta_\delta|=P_{\text{LSB}}(\delta)\cdot P_{\text{shared}}\cdot P_{\text{MSP}}=2^{2(n-m)}\cdot(2^m-\delta^p).$$

By inserting this result into Eq. (4) and simplifying the equation, we finally obtain:

$$MED=\frac{1}{3}(2^{m-2p-1}-2^{-m-1}). \tag{6}$$



(a) Power consumption [μW] for $n=8$.

(b) Power consumption [μW] for $n=16$.

(c) Power consumption [μW] for $n=32$.

(d) Resources comparison for $n=8$.

(e) Resources comparison for $n=16$.

(f) Resources comparison for $n=32$.

Fig. 2: Power consumption comparison for (a) $n=8$, (b) $n=16$ and (c) $n=32$-bit approximate adders with different parameterizations (splitting point m, number of shared inputs p, q bits per partition size, and r resultant bits). Resources (amount of LUTs and dedicated carry logic) comparison for (d) 8, (e) 16 and (f) 32-bit approximate adders with different parameterizations.

Experimental Comparative Results

To demonstrate the advantages of our proposed approximate addition technique, we are comparing it to state of-the-art adders with respect to (a) required FPGA resources, (b) power consumption, as well as the (c) error metrics of AER, ME and MED (following Eq. (2), (3), and (6)). Subsequently, we present our findings regarding maximal clock frequency and power consumption of our technique when compared with d) fully accurate implementations, i.e., implementations exploiting DSP blocks with dedicated carry logic as this is known to be the fastest and most power efficient accurate implementation of adders on FPGAs. We synthesized all investigated adders using Xilinx XST and Vivado v2015.1 (64-bit) with default strategies for the Virtex-7 VC709 evaluation platform (xc7vx690tffg1761-2) as target device. For power and resources comparisons, we constrained the implementations to avoid any DSP blocks instantiations, however, we did include dedicated carry logic and LUTs.

Comparisons with state-of-the-art approximate adders

The *average power consumption* for 8, 16 and 32-bit adders is shown in Fig. 2a, 2b and 2c respectively. In order to give representative results, 5000 instances per arithmetic unit were implemented in one design, synthesized, and the power consumption of this design was then estimated using XPower of Xilinx. The results shown in each figure represent the average power of these 5000 analyzed units. Note that the toggle rate was left as Xilinx's default of 12.5%. As can be seen, FAU does not dominate the other adders regarding power consumption in all settings. However, it is evident that the difference with the most power efficient 32-bit design (32-bit GeAr adder parameterized with $r=6$ resultant bits and $p=4$) is negligible (less than $1\mu W$). Moreover, considering the error metrics reported in Table I, our design (32-bit FAU adder parameterized $m=16$ and $p=3$) has a maximum error of $ME=8191$ and a mean error distance $MED=170.7$, this value amounts to only 12.7% and 0.5%, respectively of GeAr's best implementation. From this, we can draw our first conclusion: FAU may provide profoundly more accurate results with a power efficiency as good as state-of-the-art implementations.

Resource usage is reported in Fig. 2d, 2e and 2f for 8, 16 and 32-bit approximate adders respectively. As can be seen, the resource usage advantage of FAU versus the other displayed adders increases as the bit-width n gets larger. Moreover, as can be seen in Fig. 2e, FAU is already dominating other proposals regarding LUT usage. Even though the total amount of resources (LUTs + carry logic units) is higher than some other 32-bit adders (ETA and ACA II parameterized $q=4$), it is noteworthy that FAU is also exploiting resources that many other designs do not use at all: The used carry logic is placed within those slices already containing the used LUTs. That means by using carry logic, FPGA resources can be utilized more efficiently.

The *maximum error* ME, *mean error distance* MED and *arithmetic error rate* AER are reported in Table I. We compared our design to four approximate adders from literature which are available in a open source library gathered in [7]. To obtain the error statistics, we simulated each unit using Monte Carlo. As can be seen, our both analyzed variants of FAU have the smallest average (MED) and maximum error (ME) from the entire collection. Only some implementations like GeAr [7] with $r=1$ and $p=6$, or GDA St [9] with $m=4$ and $p=6$, have also low error rates. For these examples, they achieve to generate less than 0.5% of errors for uniformly distributed inputs, while our model generates one error 3% of the times (FAU, $m=4$ and $p=3$). However, if we compare their maximum and average errors with our approach, we observe that both ME and MED are 99.3% smaller when compared with GDA, and 98.5% smaller when compared with GeAr. As our second experimental conclusion: FAU offers the best balance between the average error and the error for which this is generated.

Error metrics proof and advantages

In § III, we presented analytical models to calculate the error metrics of our FAU adder in order to avoid applying any expensive simulative evaluations like Monte Carlo techniques.

n	Name	m	r	p	q	ME	MED	AER
8	FAU	4		1		7	0.656	0.219
		4		3		1	0.031	0.031
	ACA I				5	128	74.667	0.047
	ACA II				4	64	40	0.188
	GDA St	4		2		64	40	0.188
		4		4		64	64	0.023
		8		1		168	52.364	0.602
		8		2		144	51.533	0.301
		8		3		128	60	0.125
		8		4		128	74.667	0.047
		8		5		128	96	0.016
		8		6		128	128	0.004
	GeAr		1	1		168	52.364	0.602
			1	2		144	51.533	0.301
			1	3		128	60	0.125
			1	4		128	74.667	0.047
			1	5		128	96	0.016
			1	6		128	128	0.004
			2	2		64	40	0.188
			2	4		64	64	0.023
16	FAU	8		1		127	10.667	0.248
		8		3		31	0.667	0.061
	ACA I				4	34944	5985.1	0.342
	ACA II				8	4096	2173	0.059
					4	17472	4281.2	0.479
	GDA St	4		8		4096	4096	0.002
		4		4		4096	2173	0.059
	GeAr		6	4		1024	1024	0.031
			4	4		4096	2173	0.059
			4	8		4096	4096	0.002
			2	4		16640	4474.1	0.116
	ETA II				8	4096	2173	0.059
					4	17472	4281.2	0.479
32	FAU	16		1		32767	2730.667	0.249
		16		3		8191	170.667	0.062
	ACA I				4	69872	11267	0.364
	ACA II				8	65536	23357	0.087
					4	69888	15335	0.533
	GDA St	4		8		61440	34326	0.004
		4		4		65536	23357	0.087
	GeAr		6	4		64536	33503	0.060
			4	4		65536	23357	0.087
			4	8		61440	34326	0.004
			2	4		66496	14835	0.137
	ETA II				8	65536	23357	0.087
					4	69888	15335	0.533

Generic **A**ccuracy Configu**r**able	M. Shafique et.al.	[7]
Gracefully-**D**egrading **A**dder	R. Ye et.al.	[9]
Almost **C**orrect **A**dder	A. K. Verma et.al.	[8]
Error-**T**olerant **A**dder	N. Zhu et.al.	[10]
Fast **A**pproximate Adder **U**nit	*Our approach*	

TABLE I: Error statistics of n-bit approximate adders with splitting point m, p shared inputs, q bits per partition size and r resultant bits.

In the following experiments, we evaluate the overhead incurred by such techniques when no analytical models are available. Moreover, we simulated our approximate adder technique with normally distributed inputs and we were able to nicely corroborate the error metrics from § III. Fig. 3 presents the number of simulations required until finding a steady approximation of MED, AER, and ME for 15 continuous samples within a 10% tolerance. As can be seen, the overhead grows exponentially with the bit-width of the adder, showing the importance of providing an analytical error model when dealing with approximate arithmetic units.

Comparisons with accurate FPGA-based adder designs

The maximum frequency of two variations of FAU is compared with an adder implemented in DSP blocks (without pipelining) is shown in Fig. 4 from 16 to 128-bit adders. As can be seen, the FAU implementations provide a considerably higher maximal frequency than DSP blocks with dedicated carry logic – with only a small accuracy trade-off acc. to Table I. One important observation is that DSP blocks have a bit-width limit which leads to the use of carry propagation when the length of the carry chain exceeds this limit. This limit in Xilinx's 7 series gets evident in Fig. 4, where a drastic drop in maximal frequency may be observed for bit sizes greater than $n=32$ bits.

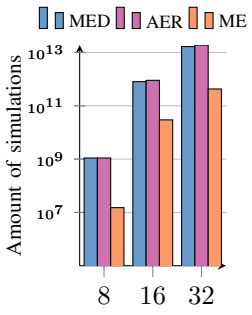


Fig. 3: Amount of simulations (y in log scale).

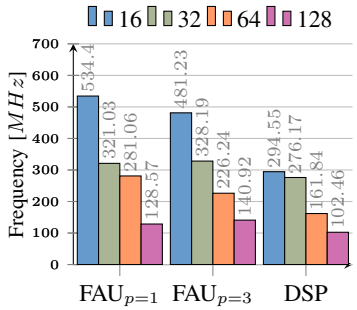


Fig. 4: Maximum frequency.

Conclusions

In this paper, we proposed a new approximate adder technique called FAU including several design optimizations for a LUT-Based FPGA implementation with a) an improved critical path and b) substantially less error margins than related work on approximate adders. We first elaborated these benefits mathematically in closed form, and then experimentally verified the results against accurate and existing approximate adder designs. We were able to outperform state-of-the-art approximate adders by a) approximating the carry bit and b) sharing information between least and most significant parts in approximate adders by using unused LUT inputs in order to reduce the likelihood of errors while improving latency. Compared to existing work, our proposed approximate adder shows a considerable improvement regarding accuracy. Also, great improvements were achieved regarding critical path delay when compared with accurate adders built from DSP blocks inside the FPGA.

References

[1] A. Becher, J. Echavarria, et al. A LUT-Based Approximate Adder. In *Proceedings of the 24th Annual IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2016)*. IEEE, 2016.

[2] S. Dutt, H. Patel, et al. Exploring Approximate Computing for Yield Improvement via Re-design of Adders for Error-Resilient Applications. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pp. 134–139. Jan 2016.

[3] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pp. 1–6. May 2013. ISSN 1530-1877.

[4] A. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pp. 820–825. June 2012. ISSN 0738-100X.

[5] S. Lee, D. Lee, et al. Statistical quality modeling of approximate hardware. In *2016 17th International Symposium on Quality Electronic Design (ISQED)*, pp. 163–168. March 2016. ISSN 1948-3295.

[6] Z. Ning, L. Wang, et al. Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. *IEEE Trans. VLSI Syst.*, 18(8):1225–1229, 2010.

[7] M. Shafique, W. Ahmad, et al. A Low Latency Generic Accuracy Configurable Adder. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pp. 86:1–86:6. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3520-1.

[8] A. K. Verma, P. Brisk, et al. Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design. In *2008 Design, Automation and Test in Europe*, pp. 1250–1255. March 2008. ISSN 1530-1591.

[9] R. Ye, T. Wang, et al. On reconfiguration-oriented approximate adder design and its application. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pp. 48–54. Nov 2013. ISSN 1092-3152.

[10] N. Zhu, W. L. Goh, et al. An enhanced low-power high-speed Adder For Error-Tolerant application. In *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, pp. 69–72. Dec 2009. ISSN 2325-0631.