A Low Latency Generic Accuracy Configurable Adder

Muhammad Shafique[1], Waqas Ahmad[2], Rehan Hafiz[2], Jörg Henkel[1]
[1]Chair for Embedded Systems, Karlsruhe Institute of Technology, Germany
[2]National University of Sciences and Technology, Islamabad, Pakistan
Corresponding Authors: muhammad.shafique@kit.edu, rehan.hafiz@seecs.edu.pk

ABSTRACT

High performance approximate adders typically comprise of multiple smaller sub-adders, carry prediction units and error correction units. In this paper, we present a *low-latency generic accuracy configurable adder* to support variable approximation modes. It provides a higher number of potential configurations compared to state-of-the-art, thus enabling a high degree of design flexibility and trade-off between performance and output quality. An error correction unit is integrated to provide accurate results for cases where high accuracy is required. Furthermore, an associated scheme for error probability estimation allows convenient comparison of different approximate adder configurations without requiring the need to numerically simulate the adder. Our experimental results validate the developed error model and also the lower latency of our generic accuracy configurable adder over state-of-the-art approximate adders. For functional verification and prototyping, we have used a Xilinx Virtex-6 FPGA. Our adder model and synthesizable RTL are made open-source.

Keywords: *Approximate Computing, Configurable Accuracy, Arithmetic, Performance, Adder.*

1 INTRODUCTION

Relaxing the bounds of precise computing provides new opportunities that may bear orders of magnitude in performance/power benefits [1]-[5]. Recently, the research by Intel [1], Microsoft [2], and other groups [3][5] has shown that there is a large body of resource-hungry applications from prevalent and emerging application domains (like: data analytics and data mining, image/video processing, computer vision, mobile computing, artificial intelligence, etc. [5]-[7]) that is amenable to approximate computing due to applications' inherent resilience to approximation errors. The intrinsic resilience of these applications can be attributed to several factors like existence of noise and redundancy in the real-world input data (e.g., camera and other type of sensors), perceptual limitations of different application users, and error attenuation characteristics of processing algorithms employed in these applications (e.g., motion tracking and data/pattern classification) [1]-[5]. Therefore, even in the presence of approximation errors, a significant portion of functions/computations within these applications still produce output of useful and acceptable quality [1]-[7]. Approximate computing leverages this fact to tradeoff computation accuracy with circuit-delay/performance, power, and area.

Adders being one of the most common data operators in such applications have recently received a lot of interest for approximate computing [8]-[13]. Traditional Ripple Carry Adder (RCA) employs a single adder of length equal to that of the operands and hence the critical path is formed by the carry chain. *Nearly all of the approximate adders presented earlier rely on the idea that in most cases the longest carry propagation chain is less than the complete length of the adder*, e.g., for the case of a 64-bit addition the carry propagation chain of 64-bits is a very rare case. This allows the designers to break the carry chains of the adders by using multiple smaller disjoint or overlapping sub-adders [8]. Each sub-adder produces R number of resultant bits that contribute to the final summation and makes use of P 'previous bits' (or prediction bits) that are used to predict the carry.

The wide variety of adders poses a challenging decision to a designer on *how to select a particular adder that meets the design constraints* (such as delay, area, etc.) while still achieving the required accuracy level. State-of-the-art approximate adders such as Almost Correct Adder (ACA-I) [8], Error Tolerant Adder ETAII [9] and Accuracy Configurable Adder (ACA-II) [10] define a *fixed* scheme dictating the number of 'previous bits' as a function of sub-adder length. *The configurability of these adders is only limited to the variation in the bit width of the sub-adders*. The only exception being the recently proposed Gracefully-Degrading Adder (GDA) [13] which allows combination of fixed multiple sub-adder units with selectable lengths for carry prediction bits. It makes use of a hierarchical carry look-ahead structure for carry prediction and hence the *number of carry prediction bits is limited to the multiple of sub-adder length.* Moreover, GDA does not provide an error function that can predict the accuracy of the selected configuration nor it provides any error correction mechanism. In the following, we present our motivational experiments to illustrate the accuracy configurability design space of ACA-I [8], ETAII [9], ACA-II [10] and GDA [13] adders for 16-bit processing.

Motivational Analysis of State-of-the-Art Approximate Adders: Let *N=16* be the length of operands to be added. Fig.1 illustrates the design space for two cases (i.e., number of resultant bits *R=2* and *R=4*) by varying the number of previous carry prediction bits. As shown in Fig.1(a) for *R=2*, both ACA-II and ETAII support only one possible configuration with *P=2*. GDA on the contrary allows variable number of *P* bits to be used as carry prediction bits for *R=2*. Due to the particular architecture of GDA the number of carry prediction bits has to be a multiple of *R*. Similarly for the case of Fig.1(b) *R=4*, both ACA-II and ETAII support only one possible configuration with *P=4* while GDA allows configurations of *P* that are multiple of 4. ACA-I cannot be configured for these two cases since it has a fixed architecture and can be configured only for resultant *R=2* bit.

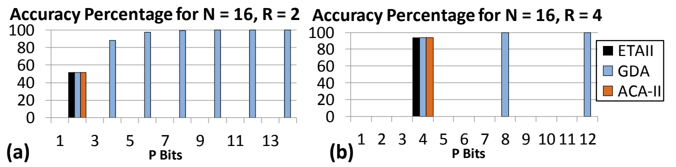


Fig.1: Design space comparison of ETAII, ACA-II and GDA for N=16, (a) R=2 and (b) R=4 bits and previous bits ranging from 1 to (N-R).

In summary, *Fig.1 illustrates the sparse design space of accuracy configurability of state-of-the-art adders*. The design space of ACA-I, ETAII and ACA-II is very limited due to their fixed architecture.

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06…$15.00
http://dx.doi.org/10.1145/2744769.2744778

Even for GDA, the design space is limited due to the constraint on prediction bits to be multiple of the sub-adder length.

Required: *There is a need for a unified configurable adder model that can provide a wide range of accuracy configurability / approximation modes along with the error correction and prediction capabilities*. Moreover, such a unified design should also support design tradeoff points provided by existing adders to facilitate designer with the selection of an appropriate adder without changing the overall design.

1.1 Novel Contributions

In this paper we present a novel **Generic Accuracy Configurable (GeAr)** adder that supports:

1) *A generalized model and architectural designs for accuracy-configurable adders* that makes use of multiple sub-adder units of equal length to provide a wide-range of accuracy configurability and variable approximation modes.
2) *A configurable error correction unit* that enables computation of accurate results when required.

We also present *an error probability model for GeAr* to define the probability of approximation errors associated with each configuration (i.e. so-called approximation mode) of the adder. This model facilitates convenient comparison of different adder configurations without requiring the need to numerically simulate the adder.

Configuration Coverage for State-of-the-Art Adders: The generic nature of the *GeAr* model allows it to be configured as various state-of-the-art approximate adders like ACA-I, ACA-II, ETAII and even for the cases of GDA adders in which the number of carry prediction bits is same for all sub-adders. *GeAr* also enables error recovery for all these adders, though some of these (e.g., ETAII and GDA) do not originally provide it.

Open-Source Contributions: We release the open-source the synthesizable RTL and MATLAB model of our *GeAr* adder and other state-of-the-art adders at *http://ces.itec.kit.edu/GeAR/*. To the best of our knowledge, this is the first open-source library of approximate adders that facilitates reproducible comparisons and further research and development in this direction across various layers of design abstraction.

2 RELATED WORK

Verma et al. [8] proposed an approximate adder called ACA-I that exploits the idea that for most additions the longest carry propagation chain is smaller compared to the total number of bits. Hence by using multiple sub adders of the length of maximum carry propagation chain an accurate result can be produced. They proposed the use of multiple overlapping sub-adders with one resultant bit per sub-adder. Due to the fixed single-bit shift operation, sub-adder count increases along with the fan-out of the input, which in turn increases the area overhead. Error detection and correction was also proposed for ACA-I. Zhu et al. [9] proposed three variants of approximate adders. Error Tolerant Adder I (ETAI) reduces the maximum carry propagation by splitting the total number of bits into accurate and inaccurate parts. Normal addition method is used for accurate part and a special addition method is proposed for the inaccurate part. For small inputs ETAI produced inaccurate results as the inaccurate part of the adder was used. To handle the limitations of ETAI another adder ETAII was proposed. ETAII reduces the maximum carry propagation by splitting the total number of bits into multiple sets of bits resulting in sub-adders of non-overlapping windows. Each sub adder uses carry generated by previous carry generator unit. Hence, the maximum carry propagation for ETAII was reduced to twice the sub-adder length. For large inputs ETAII accuracy was degraded. To overcome the problems of ETAII, a modified version ETAIIM was proposed that connects the higher sub adders to increase the accuracy in MSBs. No error recovery scheme was proposed for any of the ETA adders. Kahng et al. [10] proposed an approximate adder ACA-II that uses multiple overlapping sub-adders with half of sub-adder length resultant bits per sub-adder. Error detection and correction scheme was also proposed for ACA-II. Rong et al. [13] proposed an approximate adder GDA that also used multiple non-overlapping sub adders and used multiplexers for carry selection from either previous sub-adder or from the carry-in prediction block. It uses a hierarchical carry look-ahead structure for carry prediction blocks. So, its prediction is dependent on the sub-adder length. No error recovery scheme was proposed for the GDA adder. Furthermore, no error probability calculation scheme was proposed for any of these adders.

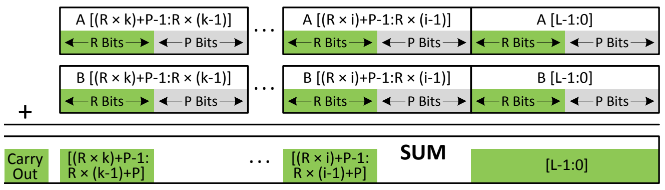
3 *GeAr* ADDER

3.1 Generic Model and Architecture of our *GeAr* Adder

In this section, we first provide the detailed description of our proposed *GeAr* adder model. Let N be the length of operands to be added. Our *GeAr* adder makes use of k *L-bit* sub-adders in parallel to perform the approximate addition, where $L<=N$. As compared to an *N-bit* adder the delay is reduced since the carry propagation is now limited to L bits only. Let R be the number of resultant bits contributing to the final sum and P be the number of previous bits used for carry prediction in each sub-adder. Each sub-adder produces *R-bits* result except the first sub-adder which produces *L-bits* result where $L=R+P$. For a given N, R, and P the number of required sub-adders k can be calculated by using Eq. 1.

$$k=\left(\left(N-L\right)/R\right)+1 \tag{1}$$

Our *GeAr* adder is completely defined by three parameters N, R, and P. Fig.2 shows the *N-bit* generic implementation of the *GeAr* adder. In Fig.2, two types of inputs to the sub-adder units (first sub-adder and subsequent sub-adders) and their corresponding outputs are shown. Apart from 1st sub-adder all subsequent sub-adders have the same configuration.



A [(R × k)+P-1:R × (k-1)]
A [(R × i)+P-1:R × (i-1)]
A [L-1:0]
R Bits
P Bits
B [(R × k)+P-1:R × (k-1)]
B [(R × i)+P-1:R × (i-1)]
B [L-1:0]
+
Carry Out
[(R × k)+P-1: R × (k-1)+P]
[(R × i)+P-1: R × (i-1)+P]
SUM
[L-1:0]

Fig.2: *GeAr* adder generic form: first sub-adder, subsequent adders (right to left).

The *N-bit* generic implementation of the *GeAr* adder can be described by Eqs.2-3. The result of the first sub-adder is calculated using Eq. 2.

$$Sum\left[L-1:0\right]=A\left[L-1:0\right]+B\left[L-1:0\right] \tag{2}$$

The result of the i^{th} sub-adder can be calculated using Eq. 3.

$$\begin{aligned}Sum\left[\left(R\times i\right)+P-1:R\times\left(i-1\right)+P\right]&=A\left[\left(R\times i\right)+P-1:R\times\left(i-1\right)\right]\\&+B\left[\left(R\times i\right)+P-1:R\times\left(i-1\right)\right]\end{aligned} \tag{3}$$

where $1<i\leq k$.

Two Example Architectural Configurations of *GeAr*: Fig.3 and Fig.4 show two different architectural designs/configurations (among many possible configurations) of our *GeAr* adder for $\{N,R,P\}=\{12,4,4\}$ and $\{12,2,6\}$, respectively. The maximum carry chain propagation in both cases is eight, i.e., the sub-adder length. For the case of Fig.3 two sub-adders of *8* bits are being used. In the first configuration all the sum bits of *sub-adder 1* contribute to the final sum. However, for *sub-adder 2* the lower *4* bits are only used to improve the carry prediction for the resultant bits of this sub-adder and hence the sum bits obtained by *P=4* previous bits are discarded (marked as red line in Fig.3). c_{pi} and c_{oi} are the carry predicted by the previous bits and the carry out of the ith sub-adder, respectively. These carry bits are generated by each sub-adder and are further explained in the next section. Similarly, for the adder of Fig.4, six previous bits are being used for carry prediction. Greater the number of *P bits*, greater is the accuracy achieved albeit at the cost of increased path delay and area due to increased number of sub-adders. This is illustrated for the example of Fig.3 where increasing *P* results in the requirement of an additional sub-adder unit (Fig.4). Hence by comparing the two configurations, the first configuration achieves the result with low delay, low area and average accuracy, while the second configuration achieves the results with average delay, average area and high accuracy.

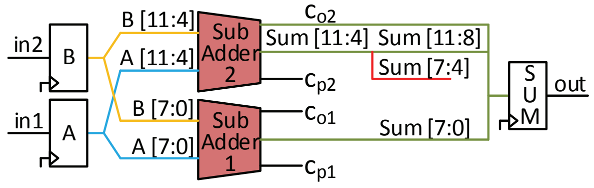


Fig.3: *GeAr* adder with N=12, R=4, P=4 and k=2.

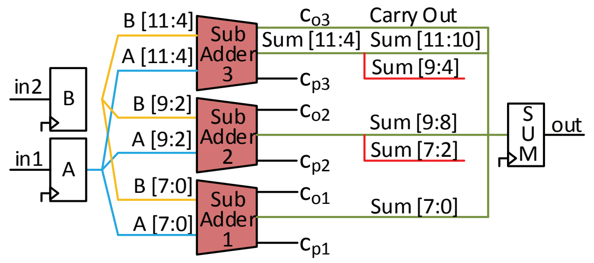


Fig.4: *GeAr* adder with N=12, R=2, P=6 and k=3.

GeAr's Configuration Coverage to support State-of-the-Art Approximate Adders: Since *GeAr* is highly configurable adder, it supports configurations to realize architectures of different state-of-the-art approximate adders by selecting different values of *R* and *P*. It can be configured as ACA-I [8] by setting the parameters *R=1* and *P=L – 1*. Similarly, it can also be configured as ACA-II [10] and ETAII [9] by setting the parameters *R=L/2* and *P=L/2*. *GeAr* adder can also provide the configuration for the cases of GDA adders in which the number of carry prediction bits is same for all sub-adders [13]. Due to the highly configurable nature of *GeAr* adder it has a large design space configurability to meet any adder requirements in terms of accuracy, path delay, area and fixed sub-adder length.

3.2 Modeling of Error Probability

By using approximate arithmetic units, approximation errors are introduced in the output results. Different applications allow certain level of accuracy tolerance depending upon their inherent error resilience. Thus, it is important to know beforehand the error probability provided by a certain approximate adder. Here, we provide the error model for our *GeAr* adder, which is applicable to the supported configurations for other state-of-the-art approximate adders as well.

For each of the sub-adder previous *P* bits are used to predict the carry-in for the corresponding resultant *R* bits. For the *i*th sub-adder carry c_{pi} can be calculated by Eq. 4.

$$c_{pi} = \prod_{j=0}^{P-1} Pr_i[j] = \prod_{j=0}^{P-1} \left(A_i[j] \oplus B_i[j] \right) \qquad (4)$$

where *Pr* is the bit propagation and *A* and *B* are input operands. In case c_{pi} and $c_{o(i-1)}$ both are *1* accuracy shall be compromised. Consider the configuration shown in Fig.3; the result of first sub-adder will always be accurate as there is no carry in. Error can only occur if the c_{p2} and c_{o1} are both one. We model the occurrence of *1* or a *0* for each bit location of the operands as an equally likely event. Let the propagate probability and generate probability be defined by $\rho[Pr]$ and $\rho[Gr]$ with probabilities *0.5* and *0.25*, respectively. Each sub-adder except the first sub-adder will generate R error generating events Z_m for $m=1,2,\ldots,R$. Hence for *k* sub-adders there will be $R\times(k-1)$ error generating events $Z_{m+(s\times R)}$ for $s=1,2,\ldots,(k-2)$. Then the probability of these error generating events can be calculated by Eq. 5.

$$\rho\left[Z_{m+(s\times R)}\right] = \rho[Gr]\times\prod_{o=1}^{L-m}\rho[Pr] \qquad (5)$$

The *generalized joint probability* of *n* events can be calculated using Eq. 6.

$$\rho\left[\bigcap_{i=1}^{n} Z_i\right] = \begin{cases} 0 & mutuallyExclusive \\ \rho[Gr]^n \times \rho[Pr]^\beta & else \end{cases} \qquad (6)$$

where $2 \le n \le R\times(k-1)$ and β is sum of propagate terms for the events jointly. Considering all the error generating cases, the probability of error associated with the adder can be generalized by using the probability of *n* events using Eq. 7 [15].

$$\rho[Error] = \rho\left[\bigcup_{i=1}^{R\times(k-1)} Z_i\right] = \sum_{j=1}^{R\times(k-1)} \rho\left[Z_j\right] - \sum_{j<k} \rho\left[Z_j \cap Z_k\right] + \ldots + (-1)^{R\times(k-1)+1}\rho\left[Z_1 \cap \ldots \cap Z_{R\times(k-1)}\right] \qquad (7)$$

Using the presented error model, probability of error for any *GeAr* architectural configuration can be calculated.

3.3 Configurable Error Detection and Correction

An error detection and recovery scheme is proposed that allows detection and correction of the introduced errors for cases where accurate result is required. The error detection is implemented with an *AND* gate. For error correction, the proposed method changes the inputs to the *i*th sub-adder rather than using the *incrementer unit* (as adopted in existing techniques like [10]) for correction. Both inputs to the *i*th sub-adder are passed through an *OR* gate, and their LSBs are set to *1*. Since, previously the *P* bits were in propagation; with this correction scheme the LSBs will generate the carry. Error correction requires one additional cycle for correction of single error. However for *k* sub-adders maximum *k-1* sub-adders can have error, which will require *k* cycles for correction.

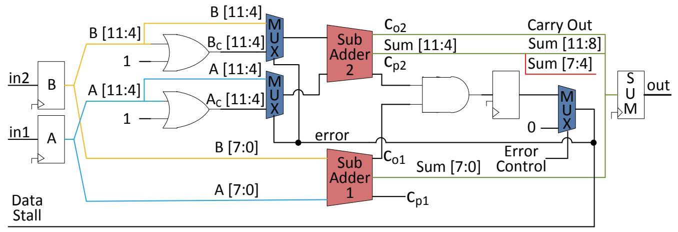


Fig.5: Error detection and correction for *GeAr* adder with N=12, R=4, P=4, k=2.

Fig.5 shows the error correction circuit for adder architectural configuration of Fig.3. Since there are two sub-adders without error correction it will require *1* cycle and with error correction it will require *2* cycles. In error correction stage, the lower inputs of the multiplexers are used which in turns generate the accurate results. Fig.6 shows the error correction circuit for adder architectural configuration of Fig.4. Since there are three sub-adders without error correction it will require *1* cycle and when only *sub-adder 2* or *sub-*

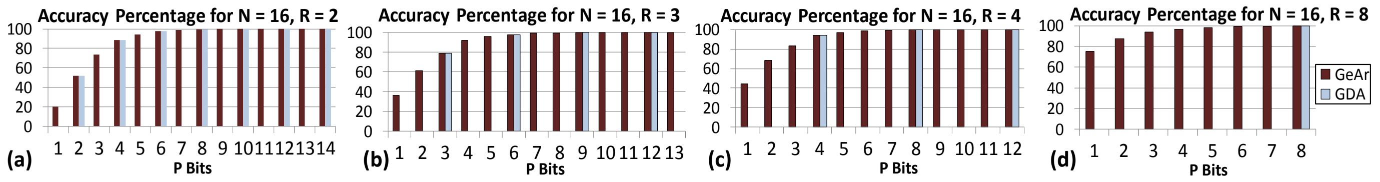


Fig.7: Probabilistic accuracy percentage with respect to change in previous/prediction bits for N=16: (a) R=2, (b) R=3, (c) R=4 and (d) R=8.

Table I Accuracy Comparison of Approximate Adders for an Image Processing Kernel (i.e., Image Integral).
(Italic filled entries denote the best cases, e.g., with the minimum area or path delay).

	RCA	ACA-I	ETAII	ACA-II	GDA(4,4)	GDA(4,8)	*GeAr*(4,2)	*GeAr*(4,4)	*GeAr*(4,6)	*GeAr*(4,8)
Path Delay	1.31E-09	1.30E-09	1.29E-09	1.19E-09	2.24E-09	3.19E-09	*1.16E-09*	1.19E-09	1.22E-09	1.25E-09
Area (LUTs)	16	30	28	*24*	35	37	*24*	*24*	30	*24*
MAA 100%	100	2.8496	14.1584	14.1584	14.1584	*88.1562*	3.47381	14.1584	51.764675	*88.1562*
MAA 97.5%	100	2.9128	14.1414	14.1414	14.1414	*88.1392*	3.4567875	14.1414	51.747666	*88.1392*
MAA 95%	100	3.138	14.1413	14.1413	14.1413	*88.1392*	3.4567875	14.1413	51.747666	*88.1392*
MAA 92.5%	100	3.3999	15.5885	15.5885	15.5885	*88.1392*	3.697705	15.5885	52.71455	*88.1392*
MAA 90%	100	3.6688	17.44	17.44	17.44	*88.1392*	4.0266816	17.44	55.814733	*88.1392*
ACC_{amp} (avg)	1	0.222	0.5129	0.5129	0.5129	*0.9334*	0.2062050	0.5129	0.8667216	*0.9334*
ACC_{inf} (avg)	1	0.7275	0.8524	0.8524	0.8524	*0.9896*	0.7853318	0.8524	0.9446139	*0.9896*
MED	0	4577	3496	3496	3496	*506.14*	4791.665	3496	764.14808	*506.14*
NED	0	0.2868	0.2233	0.2233	0.2233	*0.1711*	0.2941238	0.2233	0.0836727	*0.1711*
Delay x NED	0	3.71693E-10	2.88504 E-10	2.65504 E-10	5.00862 E-10	5.4598 E-10	3.4089 E-10	2.65504 E-10	*1.02415 E-10*	2.13704 E-10

adder 3 produces inaccurate results with error correction it will require *2* cycles. For the rare case when both of the sub-adders produce inaccurate results *3* cycles will be required. Additionally, there is *an error control select signal* to provide higher level of architectural support for configurable error correction to avoid correction overhead while trading accuracy, if allowable by the target system design depending upon the applications' resilience. This control signal is used to select the sub-adder that needs error detection and recovery. The error will only be corrected on those sub-adders where the *error control signal* is set. Hence, based on the application, we can select any desired sub-adder for error correction.

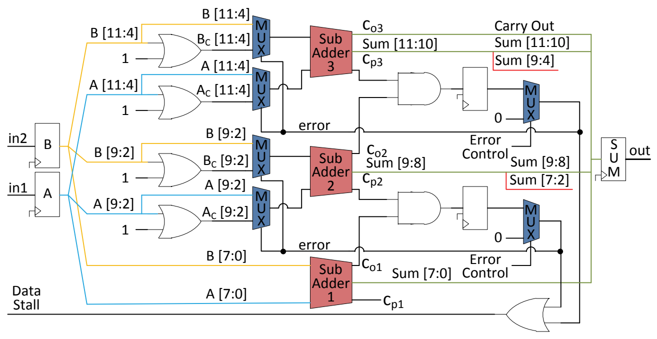


Fig.6: Error detection and correction for *GeAr* adder with N=12, R=2, P=6, k=3.

4 RESULTS AND DISCUSSIONS

In this section, we evaluate and compare different configurations of our *GeAr* adder and several state-of-the-art approximate adders (i.e., ACA-I [8], ETAII [9], ACA-II [10], and GDA [13]) for area, performance, and accuracy/error probability. We additionally illustrate the wide design space coverage of our *GeAr* adder by comparing it with the reconfigurable adder GDA [13]. We have implemented the adders in VERILOG hardware description language and synthesized using XILINX ISE. We have also developed equivalent MATLAB models for verification and error probability analysis. Moreover, for functional verifications and validation, we have prototyped on a XILINX Virtex 6 XC6VLX75T FPGA. These RTL and MATLAB models are made available for download at *http://ces.itec.kit.edu/GeAR/*.

4.1 Configuration Coverage Comparison to GDA [13]

To demonstrate the larger design space provided by our *GeAr* adder we have compared it with GDA in Fig.7 (a-d). Fig.7 shows that *GeAr* provides more design possibilities compared to the highly reconfigurable adder GDA. In Fig.7 (a-d), each figure has a fix *N* and resultant *R* bits. Previous *P* bits are increased till the sub adder length *L=N*. As for a fix *R*, increasing *P* bits in terms increases the sub-adder length *L*. Each resultant configuration has its own associated probability of accuracy. GDA on the other hand does not provide all these configurations as it puts a fixed limitation on carry prediction bits (multiple of sub-adder length). However, *GeAr* can have any previous bits for prediction. This enables it to provide configurations even GDA cannot provide.

Fig.7 (a-d) demonstrates that *GeAr* provides greater accuracy choices/design space as compared to GDA for an equal carry prediction unit. Using the error probability models each adder configuration's accuracy percentage was calculated. For *R=2* in Fig.7(a) GDA provides half the configurations *GeAr* provides. However, in Fig.7(b-d), GDA provides limited accuracy choices compared to *GeAr*. Later in the result section, we will also show how a similar configuration of *GeAr* achieves less delay and area compared to the GDA adder.

Fig.7 (a-d) also show that for a fix *N* and *R* bits or a fixed *L* bits, increasing *P* bits can reduce the error probability or can subsequently increase the accuracy percentage. This is due to the fact that by increasing previous bits more accurate estimate of carry can be produced. However with the increase in accuracy the associated path delay and area also increase. From Fig.7(a) we can infer that by using a *4* bit adder (*R=2*, *P=2*) accuracy percentage 51% can be achieved, while for an *8* bit adder (*R=2*, *P=6*), it increases to 97%. We can also compare Fig.7(a) and Fig.7(b) together for the case of same sub-adder lengths. From Fig.7(b) we can infer that by using an *8* bit adder (*R=4*, *P=4*) we can achieve accuracy percentage of 94%, which is lower compared to (*R=2*, *P=6*). However the area and path delay of (*R=4*, *P=4*) is less compared to (*R=2*, *P=6*).These figures also establish how instead of a *16*-bit adder, different combinations of our *GeAr* adder can be used to meet the required accuracy criteria.

4.2 Performance Evaluation

In Table I, a comparison of different adders for a 16-bit *1D Image Integral* application has been discussed. 4-bit resultant bits are selected for the comparison. Each adder has to be configured differently to meet the criteria. ACA-I can only generate 1 bit result so for its configuration a 4 bit sub-adder is used. ETAII will also use 4 bit sub-adder, using 4 bit each for carry generation and sum generation units. ACA-II will use 8 bit sub-adder as it generates 4 bit result. For GDA, we have selected two configurations where sub-adder size M_B is 4-bit in each case while M_C i.e., the carry in prediction bits are 4 and 8 bits. Similarly, to provide a fair comparison we have used $R=4$ bits and $P=2, 4, 6$ *and* 8 bits for *GeAr*. Although *GeAr* unlike others could be configured for any configurations of previous bits from 1, 2, ..., 12 bits. Other cases are intentionally left out to reduce the table size. As Ripple Carry Adder (RCA) is an ideal adder, we consider it as a benchmark for comparison. We have used many parameters for comparison of adders like path delay, area (in terms of LUT's used of FPGA), minimum acceptable accuracy (MAA) [9], accuracy of amplitude (ACA_{amp}) [10], accuracy of information (ACC_{inf}) [9], mean error distance (MED), normalized error distance (NED), and Delay x NED.

First we examine the path delays associated with each adder in Table I. *GeAr* achieves the minimum path delay along with ACA-II, while adders like ACA-I, and ETAII still achieve path delays close to RCA. GDA is the only adder that takes more time than the RCA. This is due to the fact that all adders except GDA can utilize any type of fastest available adder for sub-adder and prediction for previous bits while GDA utilizes carry look-ahead (CLA) adder for prediction. In terms of minimum area associated with the adders, *GeAr* and ACA-II still achieves the minimum benchmark after RCA. Area requirements of ETAII and ACA-I are slightly more while still better in comparison with GDA. In terms of accuracy GDA and *GeAr* both cases are identical, and produce results with higher accuracy in comparison with other adders. The mean error distances and normalized error distances of GDA and *GeAr* are also minimal. *GeAr* achieves the minimum Delay x NED factor while ACA-I, ACA-II and ETAII achieve slightly more. GDA gets the lowest score in Delay x NED factor due to the extra delay associated with CLA.

4.3 Detailed Performance Comparison with GDA

Since GDA adder can also be configured in many different configurations, to highlight the effectiveness of our *GeAr* adder, we have provided a detailed comparison with GDA in Table II for 8-bit data processing. For comparison we used a sub-adder size $M_B=1$ and 2-bit, while the carry in prediction bits $M_C=1, 2, 3, 4, 5$ *and* 6 bits for GDA while we used $R=1$ and 2 bits and $P=1, 2, 3, 4, 5$ *and* 6 bits for *GeAr*. In Table II, we have compared the path delays, area, NED and Delay x NED factor associated with different adder configurations of both GDA and *GeAr*. Table II highlights that the same configuration of GDA takes more area and path delay in comparison with *GeAr*. Table II also signifies the design possibilities/configurations of GDA and *GeAr* for an 8-bit adder. Fig.8 shows how each configuration of *GeAr* achieves better Delay x NED factor compared to GDA.

4.4 Evaluating Error Probabilities

To experimentally verify the error model and for error probability computations, simulations were performed assuming a uniform distribution for input operands for 10000 input patterns. Table III provides a comparison of probability of error calculated by the formula and by running simulations on adder's different configurations.

The major benefit of introducing the error probability is to calculate the execution time of an application without the need of actually simulating the adder on the application. Also with the addition of error recovery scheme, we can calculate the timings for partially or

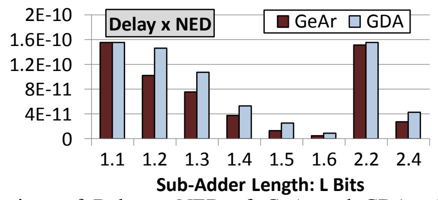


Fig.8: Comparison of Delay x NED of *GeAr* and GDA with sub-adder configuration [R.P]

Table II Comparison of GDA and *GeAr* for Designing 8-bit adder.

GDA				
(M_B,M_C)	**(1,1)**	**(1,2)**	**(1,3)**	**(1,4)**
Path Delay (ns)	8.29E-10	1.36E-09	1.83E-09	1.95E-09
Area (LUTs)	9	16	21	20
NED	0.1875	0.1076	0.0585	0.0273
Delay x NED	1.55438E-10	1.46497E-10	1.07461E-10	5.3375E-11
(M_B,M_C)	**(1,5)**	**(1,6)**	**(2,2)**	**(2,4)**
Path Delay (ns)	2.21E-09	2.25E-09	1.32E-09	1.84E-09
Area (LUTs)	25	18	12	13
NED	0.0117	0.0039	0.1171	0.0234
Delay x NED	2.5875E-11	8.76953E-12	1.55156E-10	4.31016E-11
GeAr				
(R,P)	**(1,1)**	**(1,2)**	**(1,3)**	**(1,4)**
Path Delay (ns)	8.29E-10	9.47E-10	1.30E-09	1.36E-09
Area (LUTs)	9	9	14	17
NED	0.1875	0.1076	0.0585	0.0273
Delay x NED	1.55438E-10	1.01934E-10	7.59375E-11	3.72969E-11
(R,P)	**(1,5)**	**(1,6)**	**(2,2)**	**(2,4)**
Path Delay (ns)	1.16E-09	1.17E-09	1.29E-09	1.16E-09
Area (LUTs)	18	14	12	12
NED	0.0117	0.0039	0.1171	0.0234
Delay x NED	1.36406E-11	4.58594E-12	1.51172E-10	2.71641E-11

completely accurate results. Using the error probability function and error recovery schemes, we have calculated the time required for pixel-processing in the *Image Integral*, *Sum of Absolute Difference* (SAD) and *Low Pass Filters* (LPF) applications for a full-HD image. For *Image Integral*, $N=20$ and the sub-adder length $L=10$ bits. Similarly for *SAD*, $N=16$ and $L=8$ bits and for *LPF*, $N=12$ and $L=8$ bits.

Table III Comparison of Probability of error.

Adder Configuration (N,R,P,k)	**Probability of Error**	**Probability of Error by Simulation**
(12,4,4,2)	2.9297%	2.9480 %
(16,4,8,2)	0.1831%	0.1830 %
(32,8,8,3)	0.3891%	0.3830 %
(48,8,16,5)	0.0023%	0.003 %

Table IV provides the delay, error probability of the adders and their execution timings on the *Image Integral* application. Current FPGAs make use of dedicated carry chains for efficient implementation of RCAs. However, if such an adder is used as the sub-adder for *GeAr*, then c_{pi} cannot be accessed due to its internal carry chains. Thus, for the implementation of *GeAr* we make use of additional logic to generate c_{pi}. There are four types of timings shown in the Table IV. *Approximate result time* is the execution time when no error recovery is used. When all error cases are considered to have error in only one sub-adder the corresponding resultant time is termed as the *best execution time*. Similarly when all error cases are considered to have error in half and all of the sub-adders the resultant time is termed as *average or worst execution time*, respectively. Note that since error probability model was not previously defined for these adders, we have applied our error recovery and error probability schemes on these adders to populate the table since our developed error

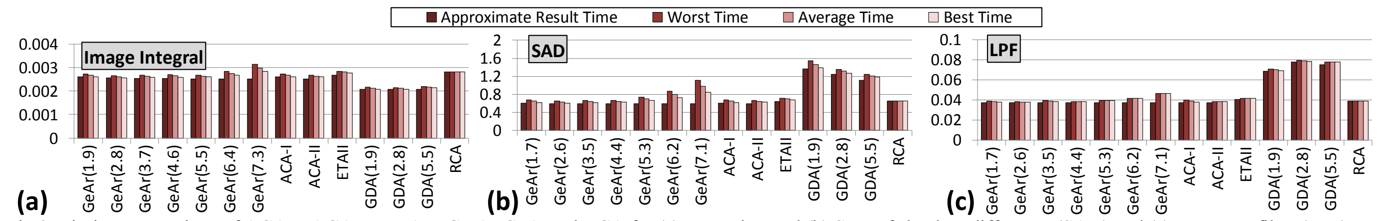


Fig.9: Timing comparison of ACA-I, ACA-II, ETAII, GDA, *GeAr* and RCA for (a) Image integral (b) Sum of absolute difference (SAD) and (c) Low pass filter (LPF).

Table IV Resource Comparison Path Delay, associated Error Probability and time for execution of GeAr, ACA-I, ACA-II, ETAII, GDA and RCA Adders (Italic filled entries correspond to the cases where our *GeAr* adder performs timing-wise better compared to the Ripple Carry Adder).

Adders	R	P	Sub Adder Length	Path Delay [ns]	Probability of Error	Approximate Result Time [sec]	Error Correction Result Timings [sec]		
							Worst Time	Average Time	Best Time
GeAr	1	9	10	1.256	4.882813 E-03	*2.604442* E-03	*2.731612* E-03	*2.674385* E-03	*2.617159* E-03
	2	8	10	1.233	7.324219 E-03	*2.556749* E-03	*2.650380* E-03	*2.612927* E-03	*2.575475* E-03
	3	7	10	1.229	13.661861 E-03	*2.548454* E-03	*2.687721* E-03	*2.635496* E-03	*2.583271* E-03
	4	6	10	1.224	21.929741 E-03	*2.538086* E-03	*2.705065* E-03	*2.649406* E-03	*2.593746* E-03
	5	5	10	1.219	30.273438 E-03	*2.527718* E-03	*2.680764* E-03	*2.642502* E-03	*2.604241* E-03
	6	4	10	1.219	60.80246 E-03	*2.527718* E-03	2.835101 E-03	*2.758256* E-03	*2.681410* E-03
	7	3	10	1.219	120.389938 E-03	*2.527718* E-03	3.136342 E-03	2.984186 E-03	2.832030 E-03
ACA-I			10	1.256	4.882813 E-03	2.604442 E-03	2.731612 E-03	2.674385 E-03	2.617159 E-03
ACA-II			10	1.219	30.273438 E-03	2.527718 E-03	2.680764 E-03	2.642502 E-03	2.604241 E-03
ETAII			10	1.296	30.273438 E-03	2.687386 E-03	2.850098 E-03	2.809420 E-03	2.768742 E-03
GDA	1	9	10	3.069	4.882813 E-03	6.363878 E-03	6.674615 E-03	6.534783 E-03	6.394952 E-03
	2	8	10	2.344	7.324219 E-03	4.860518 E-03	5.038516 E-03	4.967317 E-03	4.896118 E-03
	5	5	10	2.969	30.273438 E-03	6.156518 E-03	6.529276 E-03	6.436087 E-03	6.342897 E-03
RCA			16	1.365	0	2.830464 E-03	2.830464 E-03	2.830464 E-03	2.830464 E-03

probability model is applicable on these adders. Figure 9 shows the probabilistic timing results of adders on image integral, SAD and LPF applications. *GeAr* configurations achieve better timing results compared to other adders and many cases achieve even better timings compared to RCA (highlighted in the table). Also note that our *GeAr* model is not specific to any particular sub-adder implementation. Thus, unlike FPGAs if for an ASIC implementation an *n-bit* CLA is considered faster as compared to an RCA, sub-adder unit of the *GeAr* may comprise a CLA.

5. CONCLUSION

In this paper, we proposed a highly configurable *GeAr* adder model and architectural design, error recovery scheme and error probability model. Experimental results demonstrate that the adder provides a better accuracy, area and speed tradeoff as compared to previously developed approximate adders. Also we have demonstrated that the *GeAr* adder provides a higher number of selectable architectural configurations/approximation modes compared to the state-of-the-art approximate adders that provide a trade-off between performance and accuracy choices. Furthermore, the error recovery circuit enables the adder to be used for applications requiring high accuracy. The associated error probability models allows the execution time of an application to be predicted without the running of simulations.

REFERENCES

[1] A. K. Mishra, R. Barik, S. Paul, "iACT: A Software-Hardware Framework for Understanding the Scope of Approximate Computing", Workshop on Approximate Computing Across the System Stack (WACAS), 2014.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming", International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[3] V. Chippa, S. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing", Design Automation Conference (DAC), 2013.

[4] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency", Design Automation Conference (DAC), 2010.

[5] G. Pekhimenko, D. Koutra, K. Qian, "Approximate computing:Application analysis and hardware design", Online available: www.cs.cmu.edu/~gpekhime/Projects/15740/paper.pdf.

[6] Xing Mei, Xun Sun, Mingcai Zhou, shaohui Jiao, Haitao Wang, Xiaopeng Zhang, "On building an accurate stereo matching system on graphics hardware", International Conference on Computer Vision Workshops (ICCV Workshops), vol., no., pp.467,474, 6-13 Nov. 2011.

[7] Ke Zhang, Jiangbo Lu, Lafruit, G., "Cross-Based Local Stereo Matching Using Orthogonal Integral Images", IEEE Transactions on Circuits and Systems for Video Technology, vol.19, no.7, pp.1073,1079, 2009.

[8] A. K. Verma, P. Brisk, P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design". Design, Automation and Test in Europe (DATE), 2008.

[9] N. Zhu, W.-L. Goh, K.-S. Yeo, "An enhanced low-power high-speed Adder for Error-Tolerant application", 12th International Symposium on Integrated Circuits (ISIC), 2009.

[10] A. B. Kahng, S. Kang, "Accuracy-configurable adder for approximate arithmetic designs", Design Automation Conference (DAC), pp.820-825, 2012.

[11] J. Miao, K. He, A. Gerstlauer, M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders", International Conference on Computer Aided Design (ICCAD), pp. 728-735, 2012.

[12] V. Gupta, D. Mohapatra, A. Raghunathan, K. Roy, "Low-Power Digital Signal Processing Using Approximate Adders", IEEE Transaction on CAD of Integrated Circuits and Systems 32(1): 124-137, 2013.

[13] R. Ye, T. Wang, F. Yuan, R. Kumar, Q. Xu, "On reconfiguration-oriented approximate adder design and its application", International Conference on Computer-Aided Design (ICCAD), pp.48-54, 2013.

[14] O. Veksler, "Fast Variable Window for Stereo Correspondence Using Integral Image", IEEE CS Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 556-561, 2003.

[15] A. L. Garcia, "Probability, Statistics and Random Processes for Electrical Engineering", 3rd Edition.