Accurus: A Fast Convergence Technique for Accuracy Configurable Approximate Adder Circuits

by

Vinamra benara, Suresh Purini

in

2016 IEEE Computer Society Annual Symposium on VLSI

Report No: IIIT/TR/2016/-1

Centre for Software Engineering Research Lab
International Institute of Information Technology
Hyderabad - 500 032, INDIA
January 2016

Accurus: A Fast Convergence Technique for Accuracy Configurable Approximate Adder Circuits

Vinamra Benara
International Institute of Information Technology
Hyderabad, India 500032
vinamra.benara@research.iiit.ac.in

Suresh Purini
International Institute of Information Technology
Hyderabad, India 500032
suresh.purini@iiit.ac.in

Abstract—**Approximate computing techniques have paved new paths to get substantial improvement in speed and power efficiency by making a trade-off with the accuracy of computations in inherently error tolerant applications, like from image and video processing domains. The accuracy requirements of various applications can differ from each other. Even within a same application different computations can have different accuracy requirements which can vary over time and upon user requirements. Accuracy configurable arithmetic circuits are essential for these reasons. Such techniques proposed earlier in the literature (ACA) work by improving the accuracy over several pipeline stages. However, those techniques suffer from the drawback that the corrections being made in the initial pipeline stages are small in magnitude as they are performed from the least significant bit position. In this paper, we propose a new correction technique -*Accurus* wherein we start from the most significant bit resulting in fast convergence of the result towards the accurate one. We used our approximate adder circuit in a Gaussian Blur filter which is then applied to an image. After one stage of correction, we achieved a peak signal to noise ratio of 40.90 dB when compared with 25.59 dB obtained using the previous well-known technique (ACA).**

I. Introduction

Many applications from domains such as machine learning, image and video processing can tolerate errors in their arithmetic operations without affecting the end user experience. This feature can be used to design arithmetic circuits which are faster, consume less power and area, while making a trade-off with the accuracy of the output. The focus of the current paper is on approximate adder circuits which have been studied earlier in various works [1]–[5]. These designs do not allow for configuring the accuracy of a circuit dynamically during the runtime. Dynamic reconfiguration have many applications such as when a mobile phone is low on battery we can switch to a very low power mode by sacrificing accuracy. Another example is that in a complex signal processing circuit, different component arithmetic circuits can be configured to different accuracy levels based on their impact on the final output of the DSP block.

For such applications, accuracy configurable adder circuits have been proposed earlier [6]–[9]. Verma et al. [8] proposed an approximate adder circuit which generates exact output on most input combinations, and incurs error detection and correction overheads on the rest resulting in variable circuit latency. Kahng and Kang [6] proposed an accuracy configurable adder circuit (ACA) in which the addition operation is performed by overlapping subadders of fixed width. It has error correction capabilities where correction happens in various stages. The stages can be pipelined for increasing the throughput of the circuit. Shafique et al. [7] proposed a more generalized version of ACA, called Generic Accuracy Configurable Adder (GeAr), which allows for flexible sub-adder and redundant overlapped bit widths. By configuring these parameters we can control the accuracy of the adder circuit although at the design time only. GeAr adder also provides error detection and correction facility which is similar to the one used in ACA.

Ye et al. [9] proposed an accuracy configurable adder circuit called Gracefully-degrading adder (GDA) which allows for reconfigurable sub-adder widths and carry chain considerations during runtime. It uses carry prediction units which can be both approximate or correct. It has no error detection unit. The correction techniques employed in above adders require more power and clock cycles to arrive at results with significant accuracy, which makes them perform poorer than correct adders when high accuracy is needed.

In this paper, we propose a new accuracy configurable adder circuit, *A*ccurus, which is similar to ACA in producing the first approximate result, but subsequently we improve the accuracy over pipeline stages starting from the Most Significant Bit (MSB) first. Due to this, we get tremendous improvement in accuracy just within the first few stages of the error correction pipeline.

In Section 2, we provide the required background and in Section 3 our main technique is presented. In Section 4 we model the probability of error. Section 5 contains the experimental results and finally we conclude with Section 6.

978-1-4673-9039-2/16 $31.00 © 2016 IEEE
DOI 10.1109/ISVLSI.2016.58

II. Background

Without loss of generality, we have based our model on ACA adder [6], although it can be implemented with any configuration of approximate adders which have carry truncation and segmented addition as their basis.

A. Overview of the ACA adder

The ACA adder does approximate addition by truncating the carry chain. It uses several overlapping sub-adders to reduce errors from carry truncation as shown in 1. The architecture of an ACA adder circuit has two design parameters N and K. N is the length of addends and each sub-adder is $2K$ bits wide. Each sub-adder contributes K bits to SUM. Figure 1 shows the basic ACA adder for parameters $N = 20$ and $K = 4$. The total number of sub-adders are $M = (N/K) - 1$. The output of the adder circuit SUM is given as follows where $TEMP$ is used as an intermediary variable for clarity of expression.

$$TEMP[2k-1:0] = A[N-ik-1:N-(i+2)k] + B[N-ik-1:N-(i+2)k]$$

$$SUM[N-ik-1:N-(i+1)k] = TEMP[2k-1:k-1]. \quad i = 0,\cdots,N/k-2. \quad (1)$$

From now on, we use the following notation.

- S^i = i^{th} sub-adder.
- $SUM[x:y]$ = Final SUM bits from x downto y.
- $OUT^i[x:y]$ = Output bits of sub-adder S^i from x downto y.
- C^i_j = Carry from the j^{th} bit of sub-adder S^i.

Each sub-adder has K redundant bits (except the first one). So, when the actual carry chain is greater than K bits, then the sub-adder misses a carry and its output is wrong. Each sub-adder contributes K bits to the final SUM, except for the first sub-adder which contributes $2K$ bits. Let us consider the ACA adder shown in the Figure 1. It has 4 sub-adders S^0 to S^3 and each subadder is 8 bits wide. Here $SUM[19:16]$, which is taken from S^3, is incorrect when $OUT^3[3:0] = 1111_2$ and $C^2_4 = 1$, as this is the carry which it missed. To correct this error, the corresponding output should be incremented by 1. In a similar way, for i^{th} subadder $S^i (i \geq 1)$, it's output is incorrect when $OUT^i[3:0] = 1111_2$ and $C^{i-1}_4 = 1$ (assuming that all the previous sub-adders are already corrected).

This forms the basis for error detection and correction circuit in ACA. The error is detected by a simple *'and'* gate structure. Corrections start from fixing the sub-adders from right to left i.e., from *least-significant* sub-adder to the *most-significant* sub-adder in a sequential manner. After each correction, we improve upon the accuracy of the result. The result becomes completely accurate after $(M-1)$ correction stages where M was the total number of sub-adders.

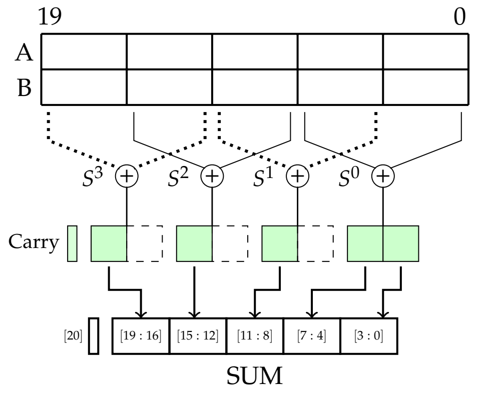


Fig. 1: Accuracy Configurable Adder for N=20 bits, K=4

The generic form of ACA adder is the GeAr Adder [7]. It has three parameters N, P and R. Each sub-adder contributes R bits to the final sum with P redundant bits for carry prediction, and N is the bit-width of the addends. The width of the sub-adder is $R + P$. By changing the parameters, we get different configurations of the adder. For ACA, $R = P = K$. GeAr adder also has a error detection and correction mechanism, similar to the ACA adder. Hence this adder also suffers from the same drawback that very little accuracy is gained in the early stages of correction.

III. Error detection and Accuracy Enhancement using Accurus

The main problem associated with ACA/GeAr adders is that most of the correction stages have to be enabled to make enhancement in the more significant bits of the result. This makes them perform poorer than accurate adders like CLA even in a pipelined implementation. The other adder, GDA, suffers from two drawbacks. First, it has no error detection and correction mechanism, and second, GDA in its scope, cannot be pipelined, which results in greater delays to get results of higher accuracy.

The goal of correction part is to make the inaccurate result more closer to the accurate one after each stage of correction. Correcting the *least-significant* sub-adders first is not good because this makes a little difference in the accuracy of the output as it is still far away from the accurate result. This leads to poor efficiency. To overcome this problem we introduce an accuracy configuration method, *Accurus*, wherein the *most significant bits* are *enhanced* first. The direction of enhancement is from *most-significant* sub-adder to *least-significant* sub-adder. This way we get a significant enhancement in accuracy as compared to previous technique, while consuming almost the same amount of power and area, and having same delays. This is validated by the experimental results in Section VI. We will explain this technique in the rest of the section.

Consider the ACA design as in figure 1. To correct(enhance) output of S^3, we see if $OUT^3[3:0] = 1111_2$ and $C_8^1 = 1$, we increment $SUM[19:16]$ by 1, where $C_8^1 = 1$ is the carry out from previous $2K$ bits. By doing this, the subadder takes into account an extra $2K$ bits of carry chain. We are enhancing S^3 first rather than S^0. Similarly $\forall i > 1$, to enhance S^i, if $OUT^i[3:0] = 1111_2$ and $C_8^{i-2} = 1$, we increment the corresponding SUM output by 1. For i=1, we use C_4^0 for correction. This technique is scalable and can be applied for any parameter of generic ACA (GeAr) adder. Figure 2 gives a unified view of correction flow for the conventional and Accurus technique. Figure 3 gives the error detection and correction circuit for a subadder S^i.

This technique also allows us to enhance all adders in the same cycle in parallel. In doing so, the carry-outs of subadders will update after each iteration. To get a 100 percent correct output (which may be required in some cases) we have to iterate this process a fixed number of times. The hard bound on number of iterations and for the GeAr adder [7] it is given by

$$X = \frac{N-P}{R} \tag{2}$$

which is derived from the fact that after each iteration, the carry-chain taken into consideration by each subadder increases by $R+P$ bits. Here R is the number of bits each subadder contributes to the final SUM, and P are the redundant bits. For ACA, (P=R) =K. For ACA with N=20 and K=4, number of iterations X=4.

Now it is clear that there are two possible implementation schemes for this adder:

1) *Enhancing one sub-adder at a time, and*
2) *Parallel Enhancement*

Scheme 1 is more power aware. This uses almost same amount of power, delay and area as in ACA adder and performs many-fold better. Note that in this scheme, Accurus enhances the output result. It does not necessarily corrects the output.

Scheme 2 is more performance aware. This is more useful for applications where more precise additions are required. As shown earlier, if this technique is iterated X times (equation (2)), we get a 100% correct result.

IV. Modelling the Probability of Error

Here we give a general framework for finding the probability of error. This gives the probability of the output being incorrect. This can also be understood as error-rate. This could be useful for selecting the right configuration of adder at design time. The bit-width of adder is N and bit-width of subadder is $2K$. So, the total number of subadders are

$$M = \frac{N}{K} - 1 \tag{3}$$

The correctness of the output of a sub-adder is completely dependent on the carry-in to that sub-adder and the bits $OUT^i[3:0]$, as we discussed earlier. Let A_i denote the probability that output of i^{th} subadder goes wrong, $i \in [1, M]$. The event that a subadder goes wrong, is independent from other subadders. The Probability that the final result is given by the union of $A_i's$ as

$$P_{error} = P(\bigcup_{i=1}^{M} A_i) = \sum_{i=1}^{M} P(A_i) - \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} P(A_i \cap A_j) + \sum_{i=1}^{M-2} \sum_{j=i+1}^{M-1} \sum_{k=j+1}^{M} P(A_i \cap A_j \cap A_k) .... \tag{4}$$

Now A_i, the probability of a subadder S^i going wrong is given by $P(E_1 \cap E_2)$ where event E_1 is $OUT^i[3:0] = 1111$ and event E_2 is $C_8^{i-2} = 1$. $P(E_1)$ is given by $16/256 = 0.0625$. For event E_2, the carry C_8^{i-2} is coming from previous $i*k$ bits of addends. The probability that two addends of $i*k$ bits produce a Carry out is given by

$$P(E_2) = \frac{2^{ik}(2^{ik}+1)}{2^{2ik+1}} \tag{5}$$

Since the events E_1 and E_2 are independent of each other, $P(A_i)$ is given by

$$P(A_i) = P(E_1 \cap E_2) = P(E_1)P(E_2) \tag{6}$$

Since all subadders are also independent, $P(A_i \cap A_j) = P(A_i)P(A_j)$ and so on. Now after applying 1 stage of correction using conventional method, S1 is corrected, and the P_{error} is given by $P(A_2 \cup A_3 \cup A_4 ...)$. For Accurus, A_M is enhanced in stage 1, and P_{error} is given by $P(A_M \cup A_{M-1} .... \cup A_1)$, where $P(A_M)$ is given by $P(E_1 \cap E_3)$. $P(E_3)$ is given by replacing i by $i-2$ in equation (5).

Using these, for N=20 and K=4, the probability of error comes out to be 0.9387 (93.87%) for conventional method and 0.9398 (93.98%) for Accurus, which are almost the same. This shows that Accurus doesn't loses on the pass rate. This is also validated in results section.

V. Results

A. Setup

For doing the functional analysis and calculating the comparison metrics of the two designs, we simulated the circuit behaviour for a 20 bit adder in C for 250 million

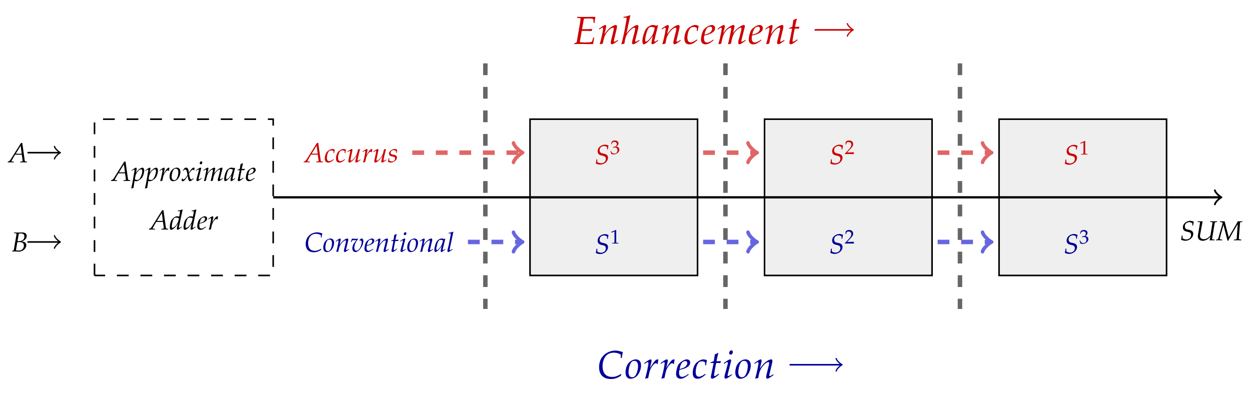


Fig. 2: Unified Pipelined Correction Flow

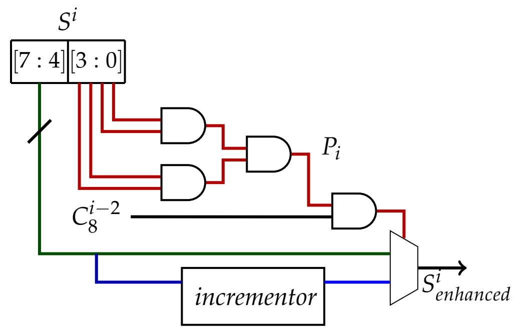


Fig. 3: Error Detection and Correction Circuit

random test cases. For verifying the calculating the hardware metrics, we wrote the designs in Verilog and synthesized it using *Cadence Encounter(R) RTL Compiler* [10], using the Faraday Technology Corp standard 180nm cell library. For functional verification, we simulated the designs using *ModelSim* [11]. For showing the benefits of *Accurus* in real life application, we demonstrated it in a Gaussian blurring application. All the results are shown in the section below.

In the tables below, we refer to the previous correction technique as *conventional*, and our technique as *Accurus*. A 20 bit adder with K=4 has been used for obtaining the results.

B. Results

1) Pass Rate: Pass rate is defined as the percentage of results that have no error at all. Pass rates for both 1 and 2 stages of correction using both techniques have been shown in Table I. The simulation results are compared with the probabilistic results which we got from section IV. We simulated the adder in C. As it can be seen, Accurus shows slight improvement in terms of pass rate.

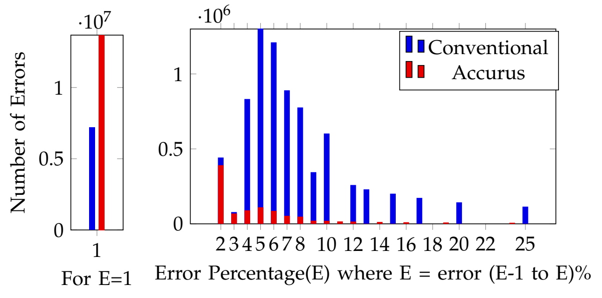
Technique	Simulation Results		Probabilistic Results	
	Stage 1	Stage 2	Stage 1	Stage 2
Conventional	93.95%	96.87%	93.87%	96.88%
Accurus	94.13%	97.06%	93.98%	96.98%

TABLE I: Pass Rate

2) Average accuracy in amplitude Acc_{amp} and information Acc_{inf}: Here we compare the two techniques using the metrics Acc_{amp} and Acc_{inf} as in [6]. Accuracy in amplitude or Acc_{amp} is defined as $R_o/R_c * 100$, where R_o is the obtained result and R_c is the correct result. We can say error amplitude $Err_{amp} = 100 - Acc_{amp}$. Figure 4a shows a error histogram of results obtained after stage1 correction by the two techniques. Each point E on the x-axis represents that the output error magnitude is in range E-1 to E%, (E-1 exclusive), and the y-axis shows the number of errors. For example, the bar plotted on x=2, gives the number of errors with magnitude 1 to 2% and so on.

As can be seen clearly from the graph, for Accurus, most of the errors have magnitude of 0 to 1%, and there are fewer errors with higher magnitude, as compared to the conventional technique, where most errors lie in higher magnitude side. This demonstrates the efficiency of *Accurus*. In the figure, the red bars are overlapped over the blue ones for facilitating clarity in graph. Similarly, figure 4b shows the results after stage2 of correction. Here, for Accurus, almost all of the results lie in 0 to 1%, while very little improvement is seen for conventional technique. Table II shows the average values of Acc_{amp}.

The second metric is information accuracy, Acc_{inf}, which depends on the hamming distance between obtained and correct results. Hamming distance depends only on the number of bits correct or not and has nothing to do with position of the bit. So, both techniques produce similar outputs and here Accurus has no added advantage over conventional technique. Figure 5 and table II shows the results.


(a) Err$_{amp}$ after 1 Level of Correction

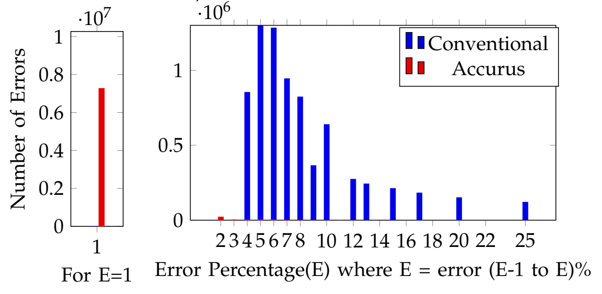

(b) Err$_{amp}$ after 2 Levels of Correction

Fig. 4: Err_{amp} Plots

Technique	ACC_{amp}		ACC_{inf}	
	Stage 1	Stage 2	Stage 1	Stage 2
Conventional	95.09%	95.38%	99.43%	99.71%
Accurus	**98.71%**	**99.48%**	99.46%	99.74%

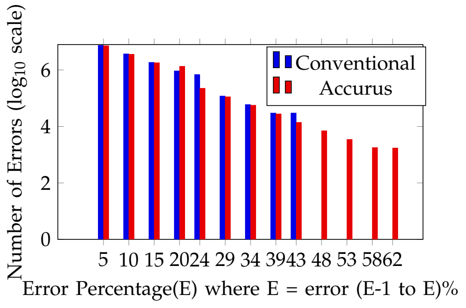
TABLE II: Average Amplitude and Information accuracy

3) Signed Addition Reliability: This metric represents the number of cases where the most significant bit or the signed bit is incorrect. In case of signed additions, the MSB bit is very crucial for the accuracy of result. We did a C simulation for obtaining the values and generated random numbers using rand() function. Table III below shows the number of errors in a simulation of 250 Million random test cases.

Technique	No. of errors
Conventional	488721
Accurus	1752

TABLE III: Signed Addition Reliability

4) MAA: Minimum Acceptable Accuracy (MAA) metric as introduced in [4] gives information about the fraction of outputs whose accuracy is greater than the specified value. Table IV below shows the pass rates for both adders for a given MAA for Stage 1 & Stage 2 corrections. Accurus has a better pass rate than the conventional technique for all values of MAAs. As the accuracy requirement is slightly relaxed, we see that accurus has a pass rate greater than 0.99 (99%), which is not seen for conventional technique. For both Stage 1 &


(a) Err$_{inf}$ after 1 Level of Correction

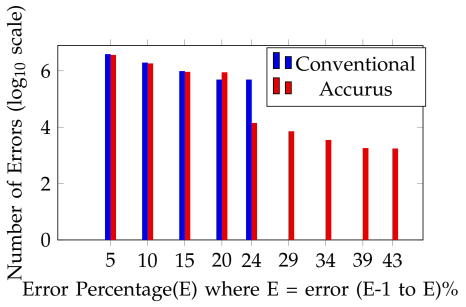

(b) Err$_{inf}$ after 2 Levels of Correction

Fig. 5: Err_{inf} Plots

Stage 2 columns, we see that for MAA $\leq$ 99%, The pass rate is almost 1 (100%).

MAA (%)	Stage 1		Stage 2	
	Accurus	Conventional	Accurus	Conventional
100	0.9414	0.9395	0.9706	0.9687
99	0.9961	0.9683	0.9997	0.9687
98	0.9976	0.9700	0.9998	0.9687
97	0.9979	0.9704	0.9998	0.9687
96	0.9983	0.9737	0.9998	0.9721
95	0.9987	0.9795	0.9999	0.9782
94	0.9990	0.9843	0.9999	0.9833
93	0.9993	0.9879	0.9999	0.9871
92	0.9994	0.9910	0.9999	0.9904
91	0.9995	0.9924	0.9999	0.9919
90	0.9996	0.9948	0.9999	0.9945

TABLE IV: MAA for 2 levels of Correction

5) PSNR Comparison: We used our proposed methodology on a real life application which uses a Gaussian Smoothing filter on an image. To perform Gaussian filtering, a Gaussian kernel, shown below,

$$\frac{1}{273}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}$$

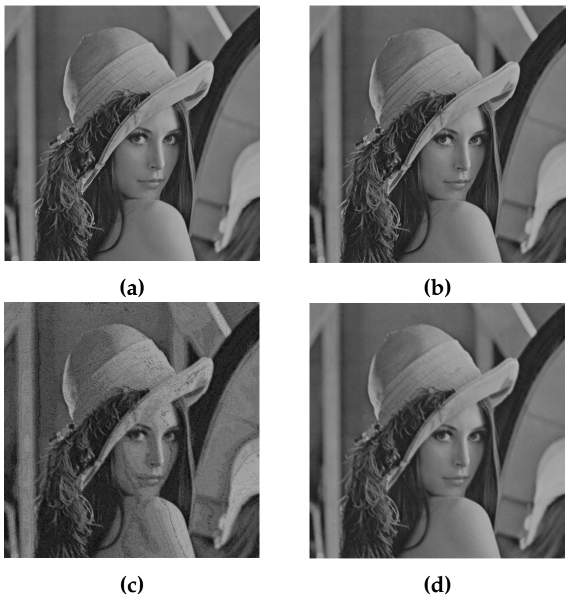


Fig. 6: Applying Gaussian filter on image: (a) Original image; (b) Accurate adder; (c) Stage1, Conventional, PSNR: 25.59 dB; (d) Stage1, Accurus, PSNR: 40.90 dB.

is convolved with the original image using ACA adder. We use the two different correction techniques for approximate additions performed in convolution. Here we used 16-bit adders because the computations for 8 bit image won't need more than 16 bits. The division $\frac{1}{273}$ is done using accurate adder in all cases. Figure 6 shows the images for visual inference. The Peak Signal to Noise Ratio (PSNR) values were obtained with respect to an image filtered using 100% accurate adder. From figure 6, the difference in quality of images (c) and (d) can be clearly seen. The image obtained using conventional technique has a PSNR of 25.59 dB while that obtained using Accurus has a PSNR of 40.9 dB, which is a huge difference. We obtained the images and PSNR values using GNU Octave [12].

6) Hardware Implementation: All the above results are arrived at by functional analysis and simulations of the two designs. Now to verify the actual hardware metrics, we wrote the designs in Verilog. We synthesized the design using *Cadence Encounter(R) RTL Compiler* [10], using the Faraday Technology Corp standard 180nm cell library. To verify the functionality on hardware, we also simulated the designs using *ModelSim*. ACA is used as the underlying adder. The Table V below shows the comparative matrices of both designs for 20 bit adder and $K = 4$ for stages 1 & 2. Both designs are pipelined. As can be seen from the table, an Approximate adder with Accurus and conventional correction technique uses almost same amount of power, area and has same minimum clock period.

With Stage-1 correction pipeline enabled			
Technique	Clock Period (ns)	Area (μm^2)	Power (mW)
Conventional	0.661	1450	0.226
Accurus	0.660	1510	0.230
With Stage-1&2 correction pipelines enabled			
Conventional	0.661	1882	0.318
Accurus	0.660	1938	0.309

TABLE V: Hardware metrics

VI. Conclusion

In this paper, we proposed a new approach for designing accuracy configurable adder circuits. The main idea is that by performing arithmetic from MSB first to LSB, we get very good estimates of the result within the initial stages of the computational pipeline. The idea although simple yields substantial benefits over the existing approaches. We believe this technique can be extended to other arithmetic operations such as multiplications.

References

[1] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, Aug 2011, pp. 409–414.

[2] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, Mar 2004.

[3] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Integrated Circuits, ISIC '09. Proceedings of the 2009 12th International Symposium on*, Dec 2009, pp. 69–72.

[4] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 8, pp. 1225–1229, Aug 2010.

[5] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *VLSI Design (VLSI Design), 2011 24th International Conference on*, Jan 2011, pp. 346–351.

[6] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 820–825. [Online]. Available: http://doi.acm.org/10.1145/2228360.2228509

[7] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 86:1–86:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2744778

[8] A. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Design, Automation and Test in Europe, 2008. DATE '08*, March 2008, pp. 1250–1255.

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

[10] "Cadence encounter(r) rtl compiler," http://www.cadence.com/products/ld/rtl_compiler.

[11] "Modelsim pe student edition."

[12] J. W. Eaton *et al.*, "Gnu octave." [Online]. Available: http://www.octave.org