Systematic Design of Approximate Adder Using Significance Based Gate-Level Pruning (SGLP) for Image Processing Application

Sisir Kumar Jena(✉), Santosh Biswas, and Jatindra Kumar Deka

Department of CSE, Indian Institute of Technology Guwahati, Guwahati, India
sisir.jena@iitg.ac.in

Abstract. Approximate computing techniques emerged as a novel design paradigm that utilizes the error-resilience property of many applications and helps in reducing the power and area consumption with an expense of loss in accuracy of the result. In this paper, we introduce Significance-based gate-level pruning (SGLP) technique to design an approximate adder circuit whose accuracy can be controlled using an Error-Threshold provided by the application user. All previous method are nonsystematic and conceptually different from each other. Those methods can either apply to a chain-based adder (adders made up of a chain of full adders, e.g., Ripple Carry Adder) or unchain-based adder (e.g., Kogge-Stone Adder) but not both. SGLP follows a systematic approach to generate an approximate version of a Full Adder which is later used to produce multi-bit adder. By using the SGLP method, we can also realize an approximate variant of an unchained adder. This characteristic makes the SGLP more superior than the previous methods. To check the quality and reliability, we have tested our approach using a DCT architecture for image processing particularly image compression and found that our result is acceptable to human perception-behavior on image clarity.

Keywords: Approximate computing · Approximate circuit design · Approximate adder · Low power design · Gate-level pruning

1 Introduction

Approximate circuit design (ACD) paradigm is an emerging technique to realize future digital systems with less area and power consumption at a loss of a negligible amount of Quality of Result (QoR) or accuracy. Compared to contemporary IC design flow, ACD generates good enough result rather than an accurate result. There is a vast application area where these circuits can be used such as image processing, web search, machine learning, and many others those have some error resilience property. ACD can be used to design (i) Fundamental

B. Deka et al. (Eds.): PReMI 2019, LNCS 11942, pp. 561–570, 2019.
https://doi.org/10.1007/978-3-030-34872-4_62

arithmetic circuits like an adder, multiplier, and a divider, (ii) CPUs and GPUs, and (iii) Approximate accelerators. This paper mainly focuses on the design of an approximate adder circuit for image processing application. There are several ACD schemes proposed in the literature for designing approximate adder circuits [1–11]. We categorize them into three groups, shown in Fig. 1(a) based on either reducing the carry chain or redesigning a fundamental block. (1) *Block adder* [1–5]: This approach is also called full-adder approximation (AFA). Each FA is considered as a *block*, and the FAs present at the lower part of the multi-bit adder, are replaced with an approximate version of it. Figure 1(b) shows the overall idea of block adder technique. An approximate variant of an FA is built through redesigning either by substituting a gate with another (e.g., XOR is replaced with OR gate) or removing one or more component (gates or transistors). (2) *Segment Adder* [6–10]: This approach divides a given adder into equal/variable sized sub-adders called *segments*, shown in Fig. 1(c). One or more segments present in the lower part are executed inaccurately with no carry propagation. Hence, reducing the carry chain. The remaining segments being in the most significant portion will execute accurately and requite in producing good enough output. The two categories explained above, are often implemented on adders built from a chain of full adder cells (e.g., Ripple Carry Adder (RCA)), refer Fig. 1(a). So we introduce another category of ACD technique known as uncut adder that are applied on unchained adders to generate its approximate version. (3) *Uncut Adder* [11]: This approach neither divides an adder into blocks nor segments rather it directly prunes some components, and realizes an approximate variant of the adder (refer Fig. 1(d)). This technique mostly applied to adders like Kogge Stone Adder (KSA), Brent Kung Adder (BKA), etc. The pruning process, in this case, generally starts from the Least Significant Bits (LSBs) and moves towards the Most Significant Bits (MSBs). Unfortunately, Literature [11] is the only work published in this category, which is known as Gate-Level Pruning (GLP) technique. According to GLP, a circuit netlist is represented as a directed acyclic graph where nodes and edges represent gates and wires, respectively. The decision to prune a node depends on Significance-Activity Product (SAP) calculation. The nodes with the lowest SAP are pruned first. This process uses Error-Rate as the measure of approximation. GLP treats chained adders as a bad candidate for pruning process. In this paper, we propose an ACD technique known as Significance-based Gate-Level Pruning (SGLP) for designing adder circuits. If the above three categories are concerned, SGLP method comes under *uncut* as well as *block* adder category. i.e., we can apply this method to obtain an approximate version of chained (adders made up of a chain of full-adders, e.g., RCA), Unchained (e.g., KSA) and full adder block. As mentioned in [11], chained adders are not suitable for GLP approach but, with SGLP this is possible. In summary, SGLP approach can do the following that defines our contribution in this paper. (1)It is quite easy to get an FA approximation which can later be used in the lower part of a multi-adder to get a multi-bit approximate adder (Block adder category). (2) We can apply SGLP directly on the chain of FA (gate-level netlist) to realize its approximate variant.

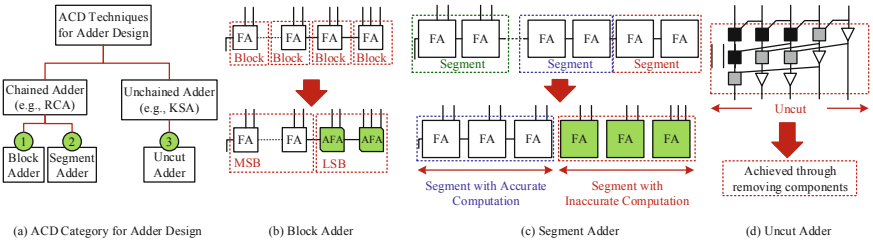

Fig. 1. ACD scheme for approximate adder

(3) Unlike GLP [11], SGLP can also be used to obtain the approximate version of uncut adders (e.g., KSA).

The other contribution of this paper includes: (1) We introduces a way of categorizing the ACD techniques for approximate adder design. To the best of our knowledge, this is the first paper that categorizes the ACD techniques. (2) We propose a systemic approach that removes gates and reduces the logic complexity at gate-level. (3) We demonstrate the benefits (in terms of power, area, and accuracy) of SGLP over previous approximate procedures and conventional adder design. (4) We built a DCT architecture using the approximate adders generated through SGLP for image compression application, and the result is outrightly acceptable.

2 SGLP Technique and Implementation

2.1 SGLP for FA Approximation

In this section, we describe the detailed procedure to generate an approximate FA block (AFA) and later part of this section describes how this AFA block is used to form a multi-bit adder circuit. We use RCA as the primary architecture upon which the AFA is implemented to build the required approximate multi-bit adder. SGLP follows a systematic approach and prune gates one by one and on every removal, we get one approximate version of the FA. Figure 2 shows the overall process. There are mainly two processes: (1) Significance Assignment (SA) and (2) Prune and Truth Table Analysis (PTA). The objective of SA is to assign an integer numeral to each gate of the given FA netlist. The purpose of PTA is to prune the gate having the lowest significance and analyze the effect in the truth table for all exhaustive set of inputs. One gate removal originates one approximate FA (AFA) which is again going through the SA and PTA method to produce another AFA. By connecting AFAs, we can generate several multi-bit approximate adders. Figure 3 shows the detail of the entire AFA generation started with the SA shown in Fig. 3(a). There are two output lines y and c_{out}, which is assigned with an initial value 2^0 and 2^1, respectively. Hence, the significance of the gates connected to y and c_{out} will be 1 and 2, respectively. The

significance of the remaining gates follows a reverse topological order and is calculated by adding the significance of its descendants.

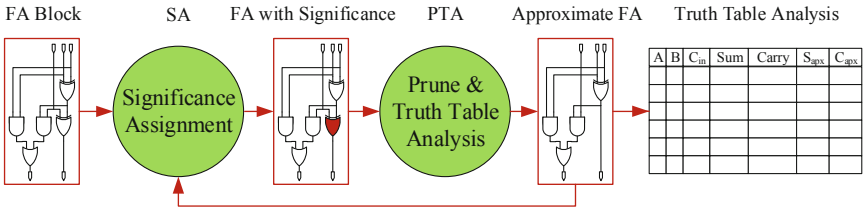

Fig. 2. Systematic process of SGLP for AFA

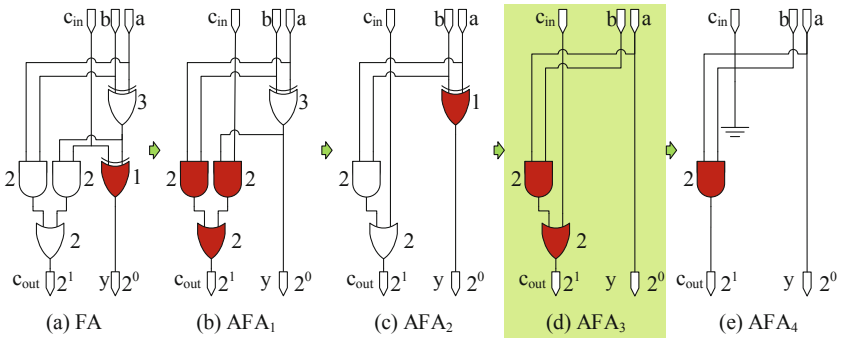

Fig. 3. Systematic generation of AFAs using SGLP

Figure 3(a) shows the significance of each gate calculated through the above approach. We can now prune the gates one by one to obtain AFAs starting with the gate having the lowest significance. Figure 3(b) shows the first AFA obtained by removing one gate from the original FA. After the removal, we recalculate the significance of the newly obtained AFA. The similar process continues to obtain the remaining AFAs. Figure 3(c)–(e) shows the entire AFAs obtained through the above approach. Table 1 shows the corresponding truth table analysis of each AFAs. The first five columns specify the input combinations (a, b and C_{in}) and the output of a conventional FA (Sum and C_{out}). The remaining columns show the outputs of all the AFAs (AFA_1 through AFA_4) where a tick mark (✓) is used to indicate the correct output and cross (×) to indicate incorrect output. Now, we can construct multi-bit adders using the AFAs depending on the application where it is used and accuracy of output the application needs.

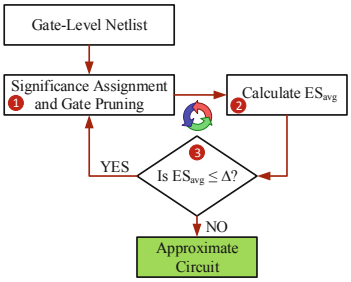

Fig. 4. Significance-based Gate-Level Pruning process

Table 1. Truth table analysis of approximate FAs

			FA		AFA_1		AFA_2		AFA_3		AFA_4	
a	b	C_{in}	S	C_o	S	C_o	S	C_o	S	C_o	S	C_o
0	0	0	0	0	0✓	0✓	0✓	0✓	0✓	0✓	0✓	0✓
0	0	1	1	0	0×	0✓	0×	1×	0×	1×	0×	0✓
0	1	0	1	0	1✓	0✓	1✓	0✓	0×	0✓	0×	0✓
0	1	1	0	1	1×	1✓	1×	1✓	0✓	1✓	0✓	0×
1	0	0	1	0	1✓	0✓	1✓	0✓	1✓	0✓	1✓	0✓
1	0	1	0	1	1×	1✓	1×	1✓	1×	1✓	1×	0×
1	1	0	0	1	0✓	1✓	0✓	1✓	1×	1✓	1×	1✓
1	1	1	1	1	0×	1✓	0×	1✓	1✓	1✓	1✓	1✓

2.2 SGLP for Uncut Adder

This section describes the detailed procedure and algorithm of Significance based Gate-Level Pruning (SGLP) method for Uncut adder. Figure 4 shows the overall process flow of SGLP. The SGLP method comprehensively defined as the process of removing (pruning) the netlist component (in our case, gate) such that, on execution, the netlist may produce an error but within the threshold defined by the designer. It is an iterative process (refer Fig. 4), begins with (1) significance assignment and pruning the gate with lower significance followed by (2) average Error-Significance (ES_{avg}) calculation and finally, (3) checking whether ES_{avg} is less than or equal to the error-threshold (Δ). The process repeats until $ES_{avg} \leq \Delta$, and on completion, we get a pruned version of the netlist with less number of gates, which produce a result within the error-threshold. Algorithm 1 in Fig. 5 shows the detailed procedure of SGLP method. GN denote the given Gate-Level netlist with x_i and y_i as the input and output lines, respectively. We represent the circuit under consideration (GN) as a tuple set $GN = \left\{ \langle g_i, s(g_i) \rangle \right\}$ Where: g_i represents gate(s) in GN and $s(g_i)$ represents significance of each gate Our algorithm generates a pruned version of GN (PGN) such that, the following conditions hold:

1. $|PGN(g_i)| \leq |GN(g_i)|$
2. $|PGN(x_i)| \equiv |GN(x_i)|$ and $|PGN(y_i)| \equiv |GN(y_i)|$
3. $ES_{avg}(PGN) \leq \Delta$

i.e., PGN has less number of gate and an equal number of input/output lines compared to GN. Lastly, the result produced is less than or equal to the error-threshold (Δ). We use the following notations in our proposed algorithm: T_e: Number of exhaustive test patterns of a circuit which is 2^n, where n is the number of input lines. T_k: Set of sample test patterns. T_k is much less than T_e. T_k^j: Represent a test pattern in T_k. i.e., $T_k^j \in T_k$. Υ: Output produced by the netlist (GN or PGN). R and $R^\dagger$: Set of all output produced by GN and PGN, respectively.

Our algorithm starts with the calculation of the golden result produced by the given netlist (GN). For smaller circuits, we took exhaustive test patterns (T_e),

Algorithm 1: Significance based Gate-Level Pruning

Data: Gate-Level Netlist (GN), Error Threshold (Δ), Sample test pattern (T_k)
Result: Pruned Gate-Level Netlist (PGN)

```
    // Calculate the Golden result
    of GN and store it in a set R
1   foreach T_k^j ∈ T_k do
2       Υ = y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + ⋯ + y_0 2^0
3       R = R ∪ {Υ}
4   end
5   do
        //Assign Significance to
        each gate
6       foreach g_i ∈ GN do
7           if g_i has no successor then
8               s(g_i) = 2^m, m = 0,1,2, ...
9           else
10              s(g_i) = ∑ s(g_i^descendant)
11          end
12      end
        // Gate Pruning: Removing
        the gate from the netlist
        having lowest significance.
13      τ = findSmallest (GN) // Find
        smallest tuple w.r.t. s(g_i)
14      PGN = GN − {τ}
        // ES_avg calculation
15      Initialization: ES_sum = 0
16      foreach T_k^j ∈ T_k do
17          Υ = y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + ⋯ + y_0 2^0
18          R† = R† ∪ {Υ}
19      end
20      ES_sum = Σ_{i=1}^{|T_k|} (|R_i − R_i†|)
21      ES_avg = ES_sum / |T_k|
22      GN = PGN\GN
23  while ES_avg ≤ Δ
24  return PGN
```

Fig. 5. SGLP algorithm

and for a larger one, we randomly chose a subset (T_k). After applying them to GN, the output (Υ) is calculated using $\Upsilon = y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + ... + y_0 2^0$ and stored in a set R (refer line number 1 to 3 in Algorithm 1).

The next task of our algorithm is to assign significance to each gate, which helps in identifying the first one to be removed. The process starts with the lowest level gates connected to output line and having no successor. It takes the form 2^m (assigns from the Least significant bit) where $m = 0, 1, 2...$, represents the number of lines present in the output. Then a reverse topological traversal is performed to assign significance to the remaining gates present in the circuit. Significance calculation is carried out using $s(g_i) = \sum s\left(g_i^{descendant}\right)$ (refer line number 6 to 12 in Algorithm 1). Where: $s\left(g_i^{descendant}\right)$ is the significance of the immediate descendant of gate i.

After successfully assigning the significance to each gate of the circuit the pruning process is carried out. Line number 13 and 14 of our algorithm is doing this job. The function findSmallest(GN) is used to search the entire set and finds the gate having the lowest significance. This gate is pruned from the GN in line number 14 to get the PGN. Again, the same set of test-pattern (T_k) is applied on PGN to calculate the result. The output produced is stored in the set $R^\dagger$ (refer line number 16 to 19 in Algorithm 1). We have two sets of result R and $R^\dagger$, obtained from GN and PGN, respectively. We subtract them element-by-

element using $ES_{sum} = \sum_{i=1}^{|T_k|} \left(|R_i - R_i^{\dagger}| \right)$ to get our error-significance sum (ES_{sum}) in line number 20. Line number 21 calculates the average error-significance, and in line number 23 it is compared with the Error-Threshold (Δ). The process repeats only if $ES_{avg} \leq \Delta$, else our algorithm returns the PGN shown in line number 24. In case it reiterates then, the PGN obtained in the last iteration, is treated as the GN for the current iteration (refer line number 22).

3 Experimental Evaluation

In this section, we describe the detail implementation of the 16-bit adder circuit for FA approximation. Due to size limitation, we are not showing the detail implementation of the approximate uncut adder. But we can follow the same procedure as FA approximation to get our desired result. A simple Ripple Carry Adder (RCA) is considered in our case which is implemented using Verilog HDL in Xilinx Vivado 2018.1 environment (Vivado System Generator for DSP 2018.1). Compatible version Matlab 2017b is used to run our design. A system with Core i5 processor and 8 GB RAM is used to execute our experiments.

For FA approximation, we divided the entire circuit into two segments not necessarily equal. We replace the LSBs with the proposed AFAs, and the remaining MSBs uses the regular FA. We use the Design Compiler (DC) EDA tool from Synopsys with 45-nm open cell library to transform the RTL design into Gate-Level netlist. The detail of gain in terms of area, power, and delay is shown in Table 2. Here l_b represents the number of FA replaced with AFAs from LSB. For instance, $l_b = 4$ means we divide the entire circuit into two segments one segment contains 4 FAs from LSB, and other contains 12 FAs from MSB. We replace the 4 FAs from LSB with 4 AFAs.

After obtaining all these approximate 16-bit Adders (AFA-16), we generate the black box for each of them using Vivado System Generator Tool. These black boxes will be further used to implement the DSP application for image processing. We have generated black boxes for four AFA-16 with $l_b = 8$ which replaces AFA_1 through AFA_4 in each of these AFA-16.

DCT Application: DCT is a computationally unblemished component for image processing application. For our experiment, we took 8×8 pixel blocks DCT. Several DCT architecture has been proposed in the literature [12–14]. The conventional method requires 64 multiplication and 56 addition operations which is a substantial number and hence cannot solve our goal. The scope of our paper needs a multiplier-less DCT architecture. This paper does not present any new DCT rather we are using an existing multiplier-less state-of-the-art architecture to test our proposed method. One such architecture is presented in [14] which is a multiplication-free transform suitable for image compression commonly referred BAS-2011 in literature. The proposed hardware architecture of BAS-2011 has total 18 addition operations represent a 1D DCT. Using two 1D DCT block along with a transpose buffer, we can realize a complete 2D-DCT transform.

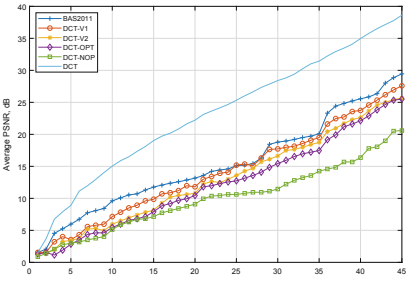

Fig. 6. Average PSNR for several compression ratios

Table 2. Area, Power, and delay characteristics

Adder	Matrix	$l_b = 2$	$l_b = 4$	$l_b = 8$	$l_b = 16$
RCA	Area	68			
	Power	0.54			
	Delay	0.8			
AFA_1	Area	66	64	61	54
	Power	0.52	0.51	0.48	0.43
	Delay	0.78	0.76	0.72	0.64
AFA_2	Area	64	62	55	41
	Power	0.50	0.46	0.42	0.32
	Delay	0.77	0.72	0.63	0.48
AFA_3	Area	59	55	46	27
	Power	0.49	0.45	0.37	0.21
	Delay	0.74	0.68	0.56	0.32
AFA_4	Area	61	54	40	14
	Power	0.47	0.42	0.30	0.10
	Delay	0.72	0.64	0.48	0.16
Area→$[nm^2]$, Power→$[mW]$, Delay→$[ns]$					

FPGA Implementation: Initially, each 1D DCT is modeled by replacing the conventional adders with the proposed adder circuit (implemented as black box using system generator tool) and then linked to form a comprehensive 2D transform. The entire design is realized using Vivado System Generator tool. By this process, we get four different 2D DCT model named as DCT_{v1}, DCT_{v2}, DCT_{opt}, and DCT_{nop}. The realized models are physically built using Xilinx Virtex-6 XC6VLX240T field programmable gate array (FPGA) and connected to the host computer running Matlab Simulink version 2017b. Image processing activity is carried out by estimating the DCT of sample images acquired from each model. The transformed image is then fed into Inverse DCT function to obtain the compressed image. Finally, we calculate the quality measure PSNR (peak signal-to-noise ratio) of the original and compressed image for image degradation using $PSNR = 10\log_{10}\left(\frac{MAX^2}{MSE}\right)$. Where MAX represents the maximum possible pixel value and MSE is the mean square error: the cumulative squared error between the original image I and obtained compressed image $\hat{I}$ using $MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=0}^{N}[I(i,j) - \hat{I}(i,j)]^2$.

Image Compression and Result Analysis: To show that our proposed approach does not provide any unreasonable output, we conducted the image compression experiment described in [15] and carried by [16–18]. We considered 45 512×512 grayscale images obtained from a public image library [19]. Each image is divided into 8×8 blocks and submitted to the 2D transformation similar to [17]. For a particular transformation, each block furnished 64 coefficients in the approximate transform domain. Following the standard zigzag sequence [20] reconstruction of the image is done by employing $r(1 \leq r \leq 45)$ initial coefficient to each block and zero to the remaining coefficient. Finally, we obtain

the compressed image by applying the actual inverse transformation. We then compare the original image with the compressed one for image degradation using PSNR as the quality measure. Figure 6 shows the average PSNR plot obtained by the experiment. Analyzing the result, we conclude that the proposed testing method does not produce any unreasonable output and quite competent for image compression.

Our objective is not to compare our result with other existing DCT algorithm. The main purpose of the experiment is to determine that approximate testing of circuit generates a tolerable result. Hence we also present a visual quality evaluation of our experimental result applied to the standard Lena image for r = 25. Figure 7 shows the effects of the experiment and supports our claim acquainted in introduction section.

Fig. 7. Lena image produced with (a) DCT, (b) BAS-2011, (c-f) Proposed method DCT_{v1}, DCT_{v2}, DCT_{opt}, DCT_{nop}

4 Conclusion

Approximate Computing technique is a novel design paradigm provides several benefits in terms of area, power consumption, and delay. In this paper, we have presented an ACD technique named as SGLP which can be used to reproduce adder circuits for chained and unchained adders. With the previously developed technique, this is not possible that shows the novelty of our work. We have tested our approach using a DCT architecture for image processing particularly image compression and found that our result is acceptable to human perception-behavior on image clarity.

References

1. Shin, D., Gupta, S.K.: A re-design technique for datapath modules in error tolerant applications. In: 2008 17th Asian Test Symposium, pp. 431–437. IEEE (2008)
2. Xu, S., Schafer, B.C.: Exposing approximate computing optimizations at different levels: from behavioral to gate-level. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **25**(11), 3077–3088 (2017)
3. Almurib, H.A.F., Kumar, T.N., Lombardi, F.: Inexact designs for approximate low power addition by cell replacement. In: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 660–665. IEEE (2016)

4. Gupta, V., Mohapatra, D., Park, S.P., Raghunathan, A., Roy, K.: Impact: imprecise adders for low-power approximate computing. In: Proceedings of the 17th IEEE/ACM International Symposium on Low-Power Electronics and Design, pp. 409–414. IEEE Press (2011)
5. Gupta, V., Mohapatra, D., Raghunathan, A., Roy, K.: Low-power digital signal processing using approximate adders. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **32**(1), 124–137 (2012)
6. Verma, A.K., Brisk, P., Ienne, P.: Variable latency speculative addition: a new paradigm for arithmetic circuit design. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1250–1255. ACM (2008)
7. Zhu, N., Goh, W.L., Wang, G., Yeo, K.S.: Enhanced low-power high-speed adder for error-tolerant application. In: 2010 International SoC Design Conference, pp. 323–327. IEEE (2010)
8. Zhu, N., Goh, W.L., Zhang, W., Yeo, K.S., Kong, Z.H.: Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **18**(8), 1225–1229 (2009)
9. Mahdiani, H.R., Ahmadi, A., Fakhraie, S.M., Lucas, C.: Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. IEEE Trans. Circuits Syst. I: Regul. Pap. **57**(4), 850–862 (2009)
10. Dalloo, A., Najafi, A., Garcia-Ortiz, A.: Systematic design of an approximate adder: the optimized lower part constant-or adder. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **26**(8), 1595–1599 (2018)
11. Schlachter, J., Camus, V., Palem, K.V., Enz, C.: Design and applications of approximate circuits by gate-level pruning. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **25**(5), 1694–1702 (2017)
12. Karakonstantis, G., Banerjee, N., Roy, K.: Process-variation resilient and voltage-scalable DCT architecture for robust low-power computing. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **18**(10), 1461–1470 (2009)
13. Cintra, R.J., Bayer, F.M.: A DCT approximation for image compression. IEEE Signal Process. Lett. **18**(10), 579–582 (2011)
14. Bouguezel, S., Ahmad, M.O., Swamy, M.N.S.: A low-complexity parametric transform for image compression. In: 2011 IEEE International Symposium of Circuits and Systems (ISCAS), pp. 2145–2148. IEEE (2011)
15. Haweel, T.I.: A new square wave transform based on the DCT. Signal Process. **81**(11), 2309–2319 (2001)
16. Lengwehasatit, K., Ortega, A.: Scalable variable complexity approximate forward DCT. IEEE Trans. Circuits Syst. Video Technol. **14**(11), 1236–1248 (2004)
17. Bouguezel, S., Ahmad, M.O., Swamy, M.N.S.: Low-complexity 8×8 transform for image compression. Electron. Lett. **44**(21), 1249–1250 (2008)
18. Bouguezel, S., Ahmad, M.O., Swamy, M.N.S.: A fast 8×8 transform for image compression. In: 2009 International Conference on Microelectronics-ICM, pp. 74–77. IEEE (2009)
19. The USC-SIPI image database, University of Southern California, Signal and Image Processing Institute. http://sipi.usc.edu/database/
20. Wallace, G.K.: The JPEG still picture compression standard. IEEE Trans. Consum. Electron. **38**(1), xviii–xxxiv (1992)