SEDAAF: FPGA based Single Exact Dual Approximate Adders for Approximate Processors

Chandan Kumar Jha, Kailash Prasad, Arun Singh Tomar, and Joycee Mekie
Electrical Engineering Department
lndian Institute of Technology Gandhinagar, Gandhinagar, India
Email: {chandan.jha, kailash.prasad, arun.tomar@mtech2017.iitgn.ac.in, and joycee}@iitgn.ac.in

Abstract—**Approximate circuits for ASICs have gained immense traction in recent years due to the benefits obtained in both energy and performance with little or no loss in output quality. Approximation in FPGAs remain a challenge due to the higher level of granularity at which logic is implemented on FPGAs. The smallest configurable blocks in FPGAs used for implementing logic consists of the look up tables (LUTs). In this paper, we exploit the inherent structures available in the FPGAs to implement SEDAAF. SEDAAF is a runtime configurable approximate adder that can perform a one-bit exact addition or two-bit approximate addition using the same hardware. SEDAAF also has a maximum bounded error, i.e. for an n-bit adder if m-bits are approximated the maximum error is $2^m - 1$. SEDAAF consumes 25% lesser power and has a 17% lesser power delay product as compared to existing designs. SEDAAF outperforms the existing state of the art designs in terms of output quality for Sobel edge detection application and can be used in approximate processors for performing both exact and approximate additions.**

I. Introduction

Approximate circuits bank on the error resilience of applications to provide benefits in terms of both energy and performance [1]–[6]. While approximate arithmetic circuits have received a lot of attention in recent years most of the works have been limited to ASIC. Due to the widespread use of adders in various applications there has been a lot of focus on approximate adders [7]–[12]. Many error tolerant applications consist of both exact and approximate sections [13]–[17]. Thus, there is a need for the hardware to configure at runtime between exact and approximate modes. In ASIC there have been some works that introduce runtime configurability by utilizing voltage and frequency as a knob for approximation [18]–[22]. There have also been works that focus on runtime configurability by crafting hybrid hardware that can perform both exact and approximate computations [23]–[28].

While a lot of approximate designs have been explored for ASICs, FPGAs have received far less attention. Approximate circuits for FPGAs are a lot more complex to design due to the higher granularity of abstraction available to the user for programming the FPGAs. There have been some recent works that have looked into the design of approximate adders and multipliers in FPGAs [29]–[31]. In [29] the authors propose approximate adders that use the look up tables (LUTs) available in the FPGAs for approximation. The work proposed various approximate adders which were implemented by exploiting the structure of the LUTs. The work also showed that approximate adders tailored for FPGAs far outperform the ASIC approximate adder designs implemented on the FPGAs. While the work has shown that approximation can be done on the FPGAs and gives benefits in terms of area, delay and power there are still several concerns thatt need to be addressed for making approximation a viable option on the FPGAs.

A full adder has two outputs, *Sum* and *Carry* [32]. In [33], authors have shown that approximate adders which need to be used in processors cannot have an error in *Carry* as it may cause an overflow. This happens because approximation in *Carry* bits, even if it is in the least significant bits (LSBs), in a multi-bit adder can affect the most significant bits (MSBs) of the outputs. The limitations of the approximate adder designs proposed in [29] are as follows: 1) All the approximate adder designs proposed have errors in the *Carry* bits thus making them unsuitable for use in processors; 2) The approximate adder designs do not allow runtime configurability between approximate and exact modes; 3) The designs also do not provide knobs to vary the amount of approximation at runtime.

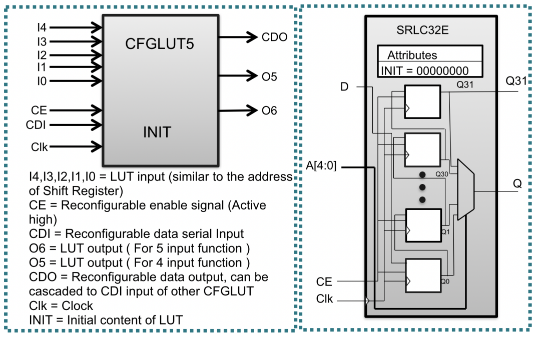
In this paper we propose *SEDAAF*, an approximate adder design that addresses all the issues of the previous work. Following are the contributions of our work:

- We propose *SEDAAF*, a single exact dual approximate adder for FPGAs. *SEDAAF* can perform a single n-bit exact addition or it can be used to perform two n-bits approximate additions using the same hardware.
- *SEDAAF* allows configurability to vary the amount of approximation at runtime making it ideal for processors.
- In *SEDAAF* the maximum error is bounded because it does not introduces any approximation in the *Carry* bits. For an n-bit adder, if m-bits are approximated the maximum error is bounded by $2^m - 1$.

II. Background

We have implemented the adder designs using *Xilinx Kintex-7 FPGA KC705* board [34], [35]. In the 7-series *FPGAs*, each slice has four *LUTs* [34], [35]. *LUTs* are the basic building blocks used to implement logic functions in the *FPGA*. The two elements used for the design of *SEDAAF* are as follows:

978-1-7281-3320-1/20/$31.00 ©2020 IEEE



CFGLUT5
I4
I3
I2
I1
I0
CE
CDI
Clk
CDO
O5
O6
INIT
I4,I3,I2,I1,I0 = LUT input (similar to the address of Shift Register)
CE = Reconfigurable enable signal (Active high)
CDI = Reconfigurable data serial Input
O6 = LUT output (For 5 input function)
O5 = LUT output (For 4 input function)
CDO = Reconfigurable data output, can be cascaded to CDI input of other CFGLUT
Clk = Clock
INIT = Initial content of LUT
SRLC32E
Attributes
INIT = 00000000
D
Q31
A[4:0]
Q
CE
Clk

Fig. 1: CFGLUT and SRLC32E [34], [35]

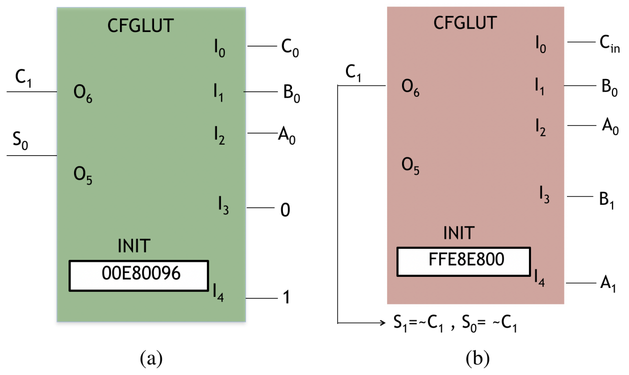


Fig. 2: (a) Exact (b) Approximate

A. *CFGLUT*

We used a 5-input dynamically reconfigurable look up table *(CFGLUT)* as shown in Fig. 1. The *CFGLUT* can be used to implement a 5-input *(I4:I0)* function or two 4-input functions. The truth table entries, also called *INIT* value, corresponding the logic to be implemented is stored in the *CFGLUT*. The output is taken either from *O6* or *O5* as shown in Fig. 1. In *CFGLUT* the *INIT* value can be changed dynamically at runtime. This is done with the help of *CE* and *CDI* pin. If *CE* is '1' the data present on the *CDI* pin is loaded into the *CFGLUT*. In 32 cycles the *INIT* value is replaced with the new *INIT* value, changing the function implemented by *CFGLUT*.

B. *SRLC32E*

SRLC32E is a variable length shift register implemented using an *LUT* as shown in Fig. 1. In *SEDAAF* we have two *SRLC32E* to store 32-bit *INIT* values for exact and approximate addition. It will be used to configure the value of *CFGLUT*. The *A* input is fixed to '11111', which configures *SRLC32E* as a 32-bit shift register. In 32 *Clk* cycles the data present in *SRLC32E* can be read out and loaded to *CFGLUT*.

III. SEDAAF

We have used 5-input *CFGLUT* in this work to allow dynamic configurability at runtime. As explained before the

TABLE I: Truth Table and INIT value for Exact Adder

A	B	Cin	S	INIT value	Cout	INIT value
1	1	1	1	9	1	E
1	1	0	0		1	
1	0	1	0		1	
1	0	0	1		0	
0	1	1	0	6	1	8
0	1	0	1		0	
0	0	1	1		0	
0	0	0	0		0	

TABLE II: Truth Table of (A1A0 + B1B0 + Cin) alongside approximation for *SEDAAF*

A1	B1	A0	B0	Cin	C1	INIT	S1=∼C1	S0=∼C1
1	1	1	1	1	1	F	0	0
1	1	1	1	0	1		0	0
1	1	1	0	1	1		0	0
1	1	1	0	0	1		0	0
1	1	0	1	1	1	F	0	0
1	1	0	1	0	1		0	0
1	1	0	0	1	1		0	0
1	1	0	0	0	1		0	0
1	0	1	1	1	1	E	0	0
1	0	1	1	0	1		0	0
1	0	1	0	1	1		0	0
1	0	1	0	0	0		1	1
1	0	0	1	1	1	8	0	0
1	0	0	1	0	0		1	1
1	0	0	0	1	0		1	1
1	0	0	0	0	0		1	1
0	1	1	1	1	1	E	0	0
0	1	1	1	0	1		0	0
0	1	1	0	1	1		0	0
0	1	1	0	0	0		1	1
0	1	0	1	1	1	8	0	0
0	1	0	1	0	0		1	1
0	1	0	0	1	0		1	1
0	1	0	0	0	0		1	1
0	0	1	1	1	0	0	1	1
0	0	1	1	0	0		1	1
0	0	1	0	1	0		1	1
0	0	1	0	0	0		1	1
0	0	0	1	1	0	0	1	1
0	0	0	1	0	0		1	1
0	0	0	0	1	0		1	1
0	0	0	0	0	0		1	1
Count					32		24	20

CFGLUT has 5 inputs and 2 outputs as shown in Fig. 1. To implement an exact adder we need to generate both *Sum* and *Carry*. Since there are two outputs available, we have implemented *Sum* and *Carry* using *CFGLUT* as shown in Fig. 2a. The I_0, I_1, I_2 are the actual inputs and remaining I_3, I_4 pins are tied to $logic-0$ and $logic-1$ respectively as shown in Fig. 2a. The 32-bit *INIT* value in hexadecimal is '$00E80096$'as shown in Table I. The '$00E8$'value corresponds to the exact *Carry* (*O6* pin) for a full adder and '0096'corresponds to the exact *Sum* (*O5* pin) for a full adder as also seen in Table I. For adding a 2-bits number i.e. $A_1A_0 + B_1B_0 + C_{in}$, we need to use 2 *CFGLUT*. This is because the number of output bits for 2-bit addition is 3 but there are only 2 output ports available in a single *CFGLUT*.

Even though a 2-bit exact adder cannot be implemented

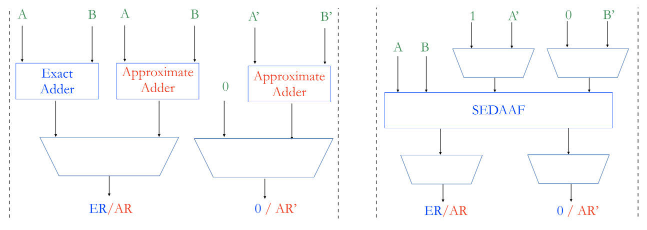


Fig. 3: (a) Baseline Design (b) *SEDAAF*

using a single *CFGLUT*, we can design a 2-bit approximate adder using one *CFGLUT*. We load the *CFGLUT* with the *INIT* value of '$FFE8E800$'. This value corresponds to the *Carry* output when $A1A0 + B1B0 + Cin$ are added as also shown in Table II. The addition will have a 3-bit output *C1, S1, S0*. The logic, 32-bit *INIT* value, for *C1* is implemented using the *CFGLUT*. *SEDAAF* introduces approximation in the *Sum* outputs *S1* and *S0*. *S1* and *S0* are equated to $\sim$*C1*. Thus, *S1* is equal to $\sim$*C1* 24 out of 32 times and *S0* is equal to $\sim$*C1* 20 out of 32 times as shown in Table. II. *S1* has a higher bit position value as compared to *S0*, and as a result *SEDAAF* introduces lesser error in *S1* as compared to *S0*. This will lead to lesser magnitude of error at the output as the most significant bit of the two will have lesser probability of error. Also, since $C1$ is not approximated the maximum error is bounded i.e. in an $n-$bit adder, if $m-$bits are approximated using *SEDAAF*, the maximum error is $2^m - 1$.

SEDAAF allows for dynamically reconfigurability between exact and approximate mode. This is enabled as follows: (i) The *INIT* value of *CFGLUT* is changed from '$00E80096$' (*INIT* value for exact) to '$FFE8E800$' (*INIT* value for approximate) using the *CDI* input of the *CFGLUT*. We already have two *SRLC32E's*, one loaded with '$FFE8E800$' and the other with '$00E80096$'. It takes 32 cycles to change the *INIT* values of *CFGLUT* to and from approximate [34], [35]. (ii) To provide the input configurability, i.e. either exact and approximate inputs, MUXes are introduced in the design of *SEDAAF*. Thus depending upon the select line of the MUXes, it will either select *A1, A0, B1, B0* and *Cin* (inputs for approximate adder) or *A0, B0* and *Cin* (inputs for exact adder) inputs as shown in Fig. 3. Thus, *SEDAAF* can be configured at runtime using *SRLC32E* and *CFGLUT*.

IV. Evaluation

We have synthesised and implemented the designs using *Xilinx Vivado 2019.1* for the *Kintex-7 KC 705 Xilinx FPGA*. We have reported the results for power consumption, delay and LUT utilization. Since runtime configurable approximate adders designs do not exist for FPGAs for a fair comparison we have implemented a number runtime configurable adders, with properties similar to *SEDAAF*, using the existing designs proposed in [29]. *SEDAAF* can perform one exact addition or two approximate additions, thus we have implemented one

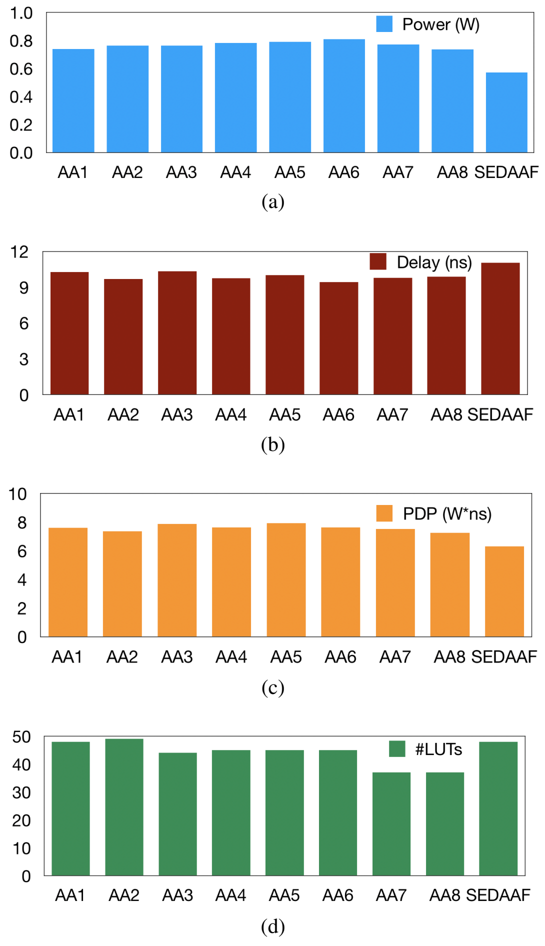
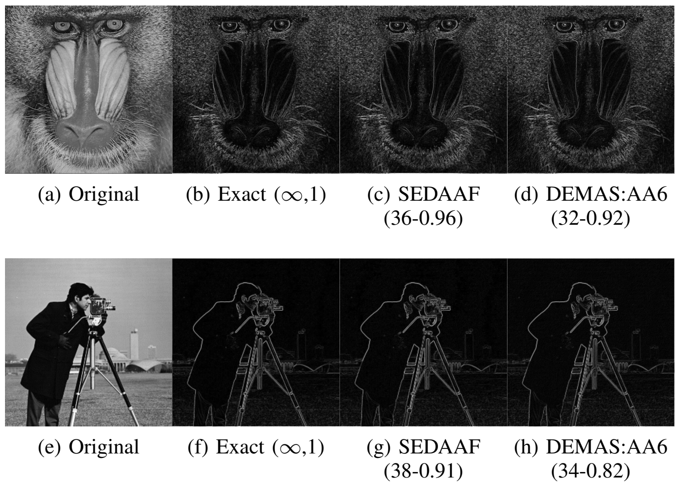


Fig. 4: 8-bit Adders (a) Power (b) Delay (c) *PDP* (d) Area

exact adder and two approximate adders (proposed in [29]). A, B are inputs for exact addition and A, B, A', B' are inputs for approximate addition[1] as shown in Fig. 3a. The ER is the exact result and AR, AR' are the approximate results in Fig. 3a. There were eight approximate adder designs proposed in [29] named *AA1* to *AA8*. We have named the baseline designs same as the name of the approximate adder used in the design. We have compared *SEDAAF* against all the designs. The corresponding design for *SEDAAF* is shown in Fig. 3b. The power, delay, power delay product (*PDP*), and area (number of LUTs) results for 8-bit approximate adders are shown in Fig. 4. On average as compared to the existing designs we obtain 25% reduction in power and 17% reduction in *PDP*. *SEDAAF* has a slight increase in area and delay overhead of 9% and 12% respectively due to the additional configuration logic. Even though we have done the comparisons with the designs proposed in [29], *SEDAAF* is the one suitable for use in processors. In this work, we have limited our analysis to 8-bit adders but it can be easily extended to 16-bit, 32-bit and 64-bit adders.

[1]Carry input is also available but has been ommited in the figure for simplicity


(a) Original (b) Exact (∞,1) (c) SEDAAF (36-0.96) (d) DEMAS:AA6 (32-0.92)

(e) Original (f) Exact (∞,1) (g) SEDAAF (38-0.91) (h) DEMAS:AA6 (34-0.82)

Fig. 5: Sobel Edge Detection Output for 4-bit approximation (PSNR-SSIM): SEDAAF and DEMAS (Best Design)

We applied *SEDAAF* for *Sobel Edge Detection* and compared it against the best design proposed in [29]. While applying the *sobel* filter the addition operation was done using the approximate adder. The output images are shown in Fig. 5. We compared *SEDAAF* against the best design in terms of error metric of *DEMAS* [29], *AA6*. We have used *structural similarity index (SSIM)* and *peak signal to noise ratio (PSNR)* as a metrics for the evaluation of output image quality as it is a widely used [7]–[11], [36]. *SEDAAF* outperforms *AA6* in terms of the output image quality as shown in Fig. 5. It has a higher *PSNR* and *SSIM* values as compared to *AA6*.

V. *SEDAAF* FOR APPROXIMATE PROCESSORS

In this section, we will discuss how the proposed approximate adder can be used in an approximate processor. In a recent work based on approximate ASIC adder [23], it was shown that approximate adders that can perform multiple approximate computations using the same hardware, can be integrated into a processor. It reduces the number of cycles [2] used while performing approximate additions. We use the same methodology proposed in [23] to show how *SEDAAF* can be integrated into a processor. We elaborate on the process of implementing *SEDAAF* in approximate processors through the example of *superscalar processors* [37]. Considering there are 8 operands $I1 - I8$, each of 8-bits, available in the *reservation station* (instructions are kept here before being executed) and 4 addition operations are to be performed as shown in Fig. 6a. The following operations $I1 + I2$, $I3 + I4$, $I5 + I6$, and $I7 + I8$ need to be performed. In Fig. 6a we can see that it will require 4 cycles to perform the addition. If these additions were to be performed approximately it would require only 2 cycles to perform the addition operation as shown in Fig. 6b. This is because *SEDAAF* can perform two

[2]In processors adders may not be in the critical path, thus adder delay does not impact the cycle time [37]

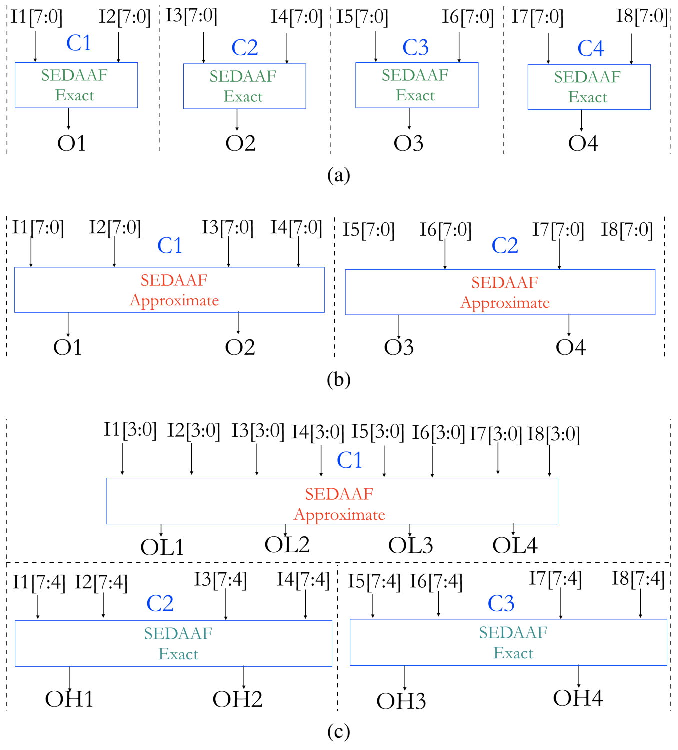


Fig. 6: (a) Exact Computation (b) 8-bit approximation (c) 4-bit approximation

approximate computations per cycle. Thus we can gain 2× improvement in performance. Fig. 6c we show that *SEDAAF* can also be configured to introduce 4-bit approximation. Since all the operands are available in the reservation station the 4 least significant bits (LSBs) of each operand can be added using *SEDAAF* in approximate mode. While in the other two cycles, two additions per cycle, we can perform the exact additions. We can either use the other exact adders present in the execution pipeline to perform the exact addition or can reconfigure *SEDAAF* at runtime to configure between exact and approximate modes.

VI. CONCLUSION

In this work, we proposed *SEDAAF*, a runtime configurable approximate adder for FPGAs. We exploited the structures present in the FPGA to design an approximate adder with bounded maximum error. On average we gain 25% in power and 17% in *PDP* as compared to existing state of the art. We also showed that *SEDAAF* can be included inside a processor to further gain benefits in performance while running approximate applications. In future we would like to extend the design of SEDAAF for implementing other widely used arithmetic circuits.

ACKNOWLEDGMENTS

This work was supported by Intel India PhD Fellowship and MeitY, India under Grant Visvesvaraya PhD scheme and SMDP-C2SD.

References

[1] H. Jiang, L. Liu, F. Lombardi, and J. Han, "Approximate arithmetic circuits: Design and evaluation," in *Approximate Circuits*, pp. 67–98, Springer, 2019.

[2] X. Yi, H. Pei, Z. Zhang, H. Zhou, and Y. He, "Design of an energy-efficient approximate compressor for error-resilient multiplications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2019.

[3] E. Adams, S. Venkatachalam, and S.-B. Ko, "Energy-efficient approximate mac unit," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, IEEE, 2019.

[4] S. Venkatachalam, E. Adams, and S.-B. Ko, "Design of approximate restoring dividers," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2019.

[5] D. Ray, N. V. George, and P. K. Meher, "Analysis and design of approximate inner-product architectures based on distributed arithmetic," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2019.

[6] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 667–673, IEEE, 2011.

[7] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[8] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate xor/xnor-based adders for inexact computing," in *Nanotechnology (IEEE-NANO), 2013 13th IEEE Conference on*, pp. 690–693, IEEE, 2013.

[9] H. A. Almurib, T. N. Kumar, and F. Lombardi, "Inexact designs for approximate low power addition by cell replacement," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pp. 660–665, IEEE, 2016.

[10] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Block-based carry speculative approximate adder for energy-efficient applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019.

[11] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Rap-cla: A reconfigurable approximate carry look-ahead adder," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 8, pp. 1089–1093, 2018.

[12] C. K. Jha, A. Nandi, and J. Mekie, "Quality tunable approximate adder for low energy image processing applications," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 642–645, Nov 2019.

[13] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: a cache for approximate computing," in *Proceedings of the 48th International Symposium on Microarchitecture*, pp. 50–61, ACM, 2015.

[14] R. Boyapati, J. Huang, P. Majumder, K. H. Yum, and E. J. Kim, "Approx-noc: A data approximation framework for network-on-chip architectures," in *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 666–677, ACM, 2017.

[15] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *ACM SIGPLAN Notices*, vol. 46, pp. 164–174, ACM, 2011.

[16] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2016.

[17] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "Accept: A programmer-guided compiler framework for practical approximate computing," *University of Washington Technical Report UW-CSE-15-01*, vol. 1, no. 2, 2015.

[18] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ACM SIGPLAN Notices*, vol. 47, pp. 301–312, ACM, 2012.

[19] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited - cross-layer approximate computing: From logic to architectures," in *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, (New York, NY, USA), pp. 99:1–99:6, ACM, 2016.

[20] B. Moons and M. Verhelst, "Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 237–242, IEEE, 2015.

[21] B. Moons, D. Bankman, and M. Verhelst, "Circuit techniques for approximate computing," in *Embedded Deep Learning*, pp. 89–113, Springer, 2019.

[22] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 779–786, IEEE, 2013.

[23] C. K. Jha and J. Mekie, "Seda - single exact dual approximate adders for approximate processors," in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, (New York, NY, USA), pp. 237:1–237:2, ACM, 2019.

[24] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 1352–1361, April 2017.

[25] M. Imani, R. Garcia, A. Huang, and T. Rosing, "Cade: Configurable approximate divider for energy efficiency," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 586–589, IEEE, 2019.

[26] H. Saadat, H. Javaid, and S. Parameswaran, "Approximate integer and floating-point dividers with near-zero error bias," in *Proceedings of the 56th Annual Design Automation Conference 2019*, p. 161, ACM, 2019.

[27] M. Imani, D. Peroni, and T. Rosing, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, (New York, NY, USA), pp. 76:1–76:6, ACM, 2017.

[28] M. Imani, R. Garcia, S. Gupta, and T. Rosing, "Rmac: Runtime configurable floating point multiplier for approximate computing," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ISLPED '18, (New York, NY, USA), pp. 12:1–12:6, ACM, 2018.

[29] B. S. Prabakaran, S. Rehman, M. A. Hanif, S. Ullah, G. Mazaheri, A. Kumar, and M. Shafique, "Demas: An efficient design methodology for building approximate adders for fpga-based systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 917–920, IEEE, 2018.

[30] S. Ullah, S. S. Murthy, and A. Kumar, "Smapproxlib: library of fpga-based approximate multipliers," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[31] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, "Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators," in *Proceedings of the 55th Annual Design Automation Conference*, p. 159, ACM, 2018.

[32] D. Harris and N. Weste, "Cmos vlsi design," *Pearson Education, Inc*, 2010.

[33] C. K. Jha, S. N. Ved, I. Anand, and J. Mekie, "Energy and error analysis framework for approximate computing in mobile applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2019.

[34] XILINX, "Vivado design suite 7 series fpga libraries guide," 2012.

[35] XILINX, "7 series fpgas configurable logic block," 2016.

[36] C. Jha and J. Mekie, "Design of novel cmos based inexact subtractors and dividers for approximate computing: An in-depth comparison with ptl based designs," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pp. 174–181, Aug 2019.

[37] J. P. Shen and M. H. Lipasti, *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.