Energy-Efficient Low-Latency Signed Multiplier for FPGA-Based Hardware Accelerators

Salim Ullah, Tuan Duy Anh Nguyen, and Akash Kumar, *Senior Member, IEEE*

***Abstract*—Multiplication is one of the most extensively used arithmetic operations in a wide range of applications, such as multimedia processing and artificial neural networks. For such applications, multiplier is one of the major contributors to energy consumption, critical path delay, and resource utilization. These effects get more pronounced in field-programmable gate array (FPGA)-based designs. However, most of the state-of-the-art designs are done for ASIC-based systems. Furthermore, a few field-programmable gate array (FPGA)-based designs that exist are largely limited to unsigned numbers, which require extra circuits to support signed operations. To overcome these limitations for the FPGA-based implementations of applications utilizing signed numbers, this letter presents an area-optimized, low-latency, and energy-efficient architecture for an accurate signed multiplier. Compared to the Vivado area-optimized multiplier IP, our implementations offer up to 40.0%, 43.0%, and 70.0% reduction in terms of area, latency, and energy, respectively. The RTL implementations of our designs will be released as an open-source library at https://cfaed.tu-dresden.de/pd-downloads.**

***Index Terms*— Accelerator architectures, artificial neural networks (ANN), fixed-point arithmetic, field-programmable gate arrays (FPGAs), multiplying circuits.**

I. Introduction

APPLICATIONS in the domain of digital signal processing and machine learning extensively use multiplication as one of the basic arithmetic operations. The architecture of a selected multiplier and its implementation directly affect the overall performance, resource utilization, and energy consumption of such applications. The FPGA synthesis tools tend to use DSP blocks for high-performance multiplication [1]. However, two points are worth noting concerning the DSP blocks utilization.

1) For many applications, such as artificial neural networks (ANNs), the 32-b floating-point precision is often not necessary for obtaining acceptable quality results. As discussed in Section III, our 8-b quantized implementation of an ANN reduces the classification accuracy only by 0.42% when compared with full-precision classification accuracy. For implementing multipliers for these low-precision numbers, the synthesis tools opt to use lookup tables (LUTs) instead of DSP blocks.
2) As noted by Ullah *et al.* [2] and Kuon and Rose [3], due to the nonuniform distribution of these DSP blocks across the FPGA, the critical path delay could be adversely affected when many of them have to be concatenated for large multiplication operations. Moreover, DSP resources are limited. On the other hand, the LUT resources are much larger. They also offer comparable performance with better energy-efficiency and flexibility than the DSP blocks for small-sized multipliers. Therefore, it is more advantageous to have the option to use the low-area, high performance, and energy-efficient LUT-based multiplier beside the DSP blocks. In this letter, we provide area-optimized, low-latency, and energy-efficient accurate signed multipliers for FPGA-based systems.

FPGA vendors, such as Xilinx and Intel, provide softcore LUT-based multipliers (signed and unsigned) as described in [4]. These multipliers can be either area or speed optimized. Booth's algorithm [5] is also a commonly used technique for multiplication because it reduces the total number of generated partial products by encoding the multiplier bits. The widely known related works are [7]–[9] and [11]. Kumm *et al.* [7] and Walters [8] have used Booth's algorithm to present area-efficient radix-4 multiplier implementations for Xilinx FPGAs. However, these implementations do not use compressor trees for adding the generated partial products and have large critical path delays. More importantly, Kumm *et al.* [7] has not discussed the implementations for signed numbers. Parandeh-Afshar *et al.* [11] have proposed a partial product compressor tree for Altera (now Intel) FPGAs. Nonetheless, their generalized parallel counters underutilize LUTs in two consecutive adaptive logic modules (ALMs). Their follow-up work, Parandeh-Afshar and Ienne [9] have used the Booth's and Baugh-Wooley's multiplication [6] algorithms for area-efficient multiplier implementation. However, in order to reduce the effective length of the carry chains, their design limits the length of the ALM to five, resulting in the underutilization of the FPGA resources.

On the other hand, Kakacak [12] and Kumm *et al.* [13] utilized smaller multiplier blocks for designing higher order multipliers. However, such techniques prove to be only useful for small bit-width multipliers; for higher bit-width multipliers, they consume more FPGA resources. For example, the logic-based implementation (using the "*" operation) of an accurate 8×8 multiplier on Virtex-7 FPGA in Xilinx Vivado, with default synthesis options, consumes 71 LUTs, whereas the modular implementation of an accurate 8×8 multiplier using accurate 4×4 multipliers consumes 82 LUTs.

A. Motivation for Signed Multipliers

For some signed numbers-based applications, it may still be possible to implement the required hardware accelerators utilizing unsigned multiplier designs. For example, we have quantized the trained parameters (weights and biases) of a

Manuscript received March 10, 2020; revised April 30, 2020; accepted May 10, 2020. This work was supported by the German Research Foundation (DFG) funded Project ReAp under Grant 380524764. This manuscript was recommended for publication by J. Hu. *(Corresponding author: Akash Kumar.)*

Salim Ullah and Akash Kumar are with the Department of Processor Design, Technische Universität Dresden, 01062 Dresden, Germany (e-mail: salim.ullah@tu-dresden.de; akash.kumar@tu-dresden.de).

Tuan Duy Anh Nguyen is with Xilinx Research Labs, Xilinx Inc., Singapore (e-mail: duyanhtu@xilinx.com).

Digital Object Identifier 10.1109/LES.2020.2995053

TABLE I
BOOTH ENCODING AND CORRESPONDING SE FOR PARTIAL PRODUCTS. b_{m+1}, b_m, AND b_{m-1} ARE MULTIPLIER BITS. (a) RADIX-4 BOOTH ENCODING. (b) SE

(a)

Inputs			Encoding	Output		
b_{m+1}	b_m	b_{m-1}	BE	s	c	z
0	0	0	0	0	0	1
0	0	1	1	0	0	0
0	1	0	1	0	0	0
0	1	1	2	1	0	0
1	0	0	$\overline{2}$	1	1	0
1	0	1	$\overline{1}$	0	1	0
1	1	0	$\overline{1}$	0	1	0
1	1	1	0	0	0	1

(b)

BE	MSB Multiplicand	SE
0	0	0
0	1	0
1	0	0
1	1	1
2	0	0
2	1	1
$\overline{2}$	0	1
$\overline{2}$	1	0
$\overline{1}$	0	1
$\overline{1}$	1	0

lightweight ANN to 8-b fixed-point numbers to implement the ANN on FPGA. These parameters are signed numbers. To implement the ANN hardware using unsigned multipliers, we require additional signed-unsigned converters to extract the sign bit from the operands and compute the final product sign. These converters receive 2's complement numbers and produce corresponding numbers in sign–magnitude format. After multiplication in sign–magnitude format, the result is converted back to the 2's complement scheme using signed–unsigned converter. These additional modules have increased the critical path delay of each multiplier by 2.061 ns and LUTs utilization by 24. Therefore, for the hardware implementations of applications utilizing signed numbers, it is always advantageous to have high performance signed arithmetic units.

B. Novel Contributions

Our contributions include the following.

1) *A Novel Architecture for Booth Multiplier:* Using 6-input LUTs and associated fast carry chains of modern FPGAs, we present an architecture for signed multipliers that provides better performance[1] than state-of-the-art designs.
2) *Parallel Generation of Partial Products:* We eliminate the need for sequential computation of the partial products and generate all Booth-encoded partial products in parallel; that significantly reduces the overall critical path delay of the multiplier.
3) *Efficient Partial Products Encoding:* Our partial product encoding technique reduces the length of the carry chain in each partial product to further reduce the critical path of the multiplier.

II. PROPOSED DESIGNS OF ACCURATE MULTIPLIERS

Using the concepts of radix-4 Booth's multiplication algorithm, we present our area-optimized, low-latency, and energy-efficient accurate signed multipliers. The correct sign of a partial product, in booth's encoding (BE)-based multiplier, is decided by the sign of the multiplicand (the MSB) and the corresponding value of BE. Table I(a) and (b) shows the list of required sign extensions (SEs) for all possible combinations of BE's values and MSB of the multiplicand. We have used Bewick's SE technique [16] to implement the correct sign of a partial product. Unlike state-of-the-art implementations, our proposed architecture computes all partial products in parallel and then adds the generated partial products using multiple 4:2 compressors and a ripple carry adder (RCA). The parallel generation of partial products significantly reduces the critical path delay of the multiplier. Our implementations provide optimized configurations for the 6-input LUTs and the associated carry chains in a logic slice of modern FPGAs such as Xilinx Virtex-7 series.

[1]Collective performance considering the area, delay, and energy.

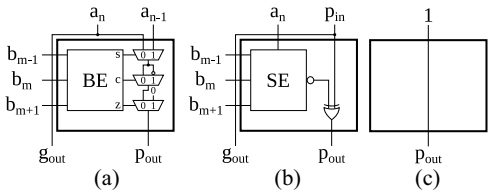

Fig. 1. Configuration of LUTs used in proposed design. (a) Type-*A*. (b) Type-*B*. (c) Type-*C*.

A. Accurate Signed Partial Products Generation

Fig. 1 shows the configurations of the 6-input LUTs used for the implementation of the proposed accurate multiplier. The BE is implemented by LUT Type-*A* configuration, as shown in Fig. 1(a). It receives five inputs, i.e., a_n and a_{n-1} (from multiplicand) and b_{m+1}, b_m and b_{m-1} (from multiplier). The LUT internally implements three MUXes. Based on the value of BE, the first MUX (controlled by *s* signal) decides whether a_n or a_{n-1} should be forwarded for partial product generation. The second MUX, controlled by *c* signal, manages the inversion of the output of the first MUX. Finally, the third MUX can make the partial product zero depending upon the value of the *z* signal. This information is forwarded to the associated carry chain as carrying propagate signal "p_{out}." The input a_n is used as the carry generate signal "g_{out}" for the carry chain.

Bewick's SE technique for each partial product row is implemented by LUT Type-*B* and LUT Type-*C* configurations, as shown in Fig. 1(b) and (c), respectively. The LUT Type-*B* receives five inputs, i.e., b_{m+1}, b_m, and b_{m-1} (from multiplier), a_n (the MSB of the multiplicand), and p_{in}. The p_{in} signal is constant "1" for the first row of partial products and for all other rows it is constant "0." The LUT computes the $\overline{SE}$ signal, performs the XOR operation on it and provides the result to the associated carry chain as the carry propagate signal p_{out}. The carry generate signal g_{out} is directly provided by the p_{in} signal. LUT Type-*C* is used to transfer the correct sign information of its respective partial product row to the following partial product row.

Utilizing LUTs of types *A*, *B*, and *C*, Fig. 2(a) shows the first row of partial products for an 8×8 multiplier. The rightmost LUT of Type-*A* in each partial product row is used for computing the required input carry. This input carry is applied for representing a partial product in 2's complement format. For an 8×8 multiplier, a total of four partial product rows will be generated. The last partial product row does not require an LUT of Type-*C*.

B. Optimizing Critical Path Delay

For an $N \times M$ multiplier, the length of the carry chain in each partial product row is $N+4$ bits. To improve the critical path delay of the multiplier, the length of the carry chain can be reduced to $N+1$ bits. A critical path delay-optimized implementation of our novel multiplier is shown in Fig. 2(b). The partial product terms $\text{pp}_{(x,0)}$ and $\text{pp}_{(x,1)}$, in each partial product row, require one and two bits of the multiplicand, respectively. These two partial product terms can be implemented by one single 6-input LUT "*A*1." Similarly, $\text{pp}_{(x,2)}$, in each partial product row, can be independently implemented using another 6-input LUT "*A*2." A separate 6-input LUT, "CG," can be used to compute the correct input carry for each partial product row. Fig. 3 shows the internal configurations of LUT types *A*1, *A*2,

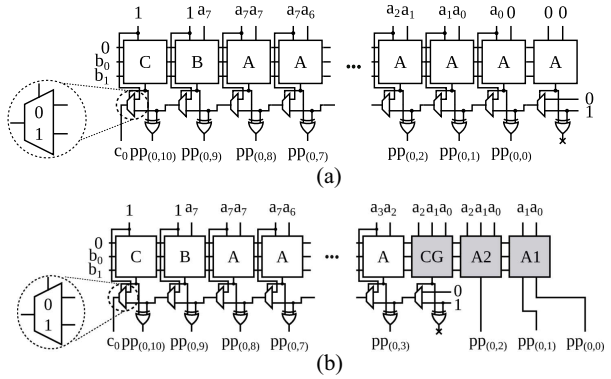

Fig. 2. First partial product row for an 8×8 multiplier. (a) First version of multiplier. (b) Optimized version of multiplier.

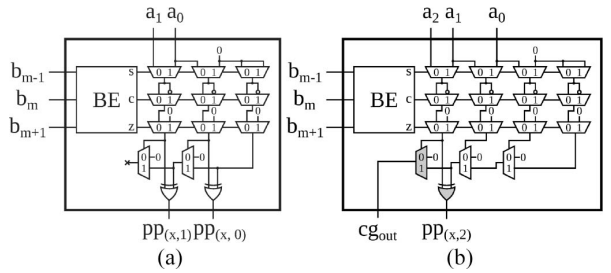

Fig. 3. Configuration of LUTs types *A*1, *A*2, and CG. (a) LUT *A*1. (b) LUT *A*2/CG.

and CG, respectively. LUT types *A*2 and CG only differ in the output signals $pp_{(x,2)}$ and cg_{out}. LUT type *A*2 utilizes $pp_{(x,2)}$ signal solely, whereas LUT type CG uses cg_{out} signal exclusively. For an $N \times M$ multiplier, the number of LUTs required to generate partial products is $(N+3) \times \lceil M/2 \rceil - 1$.

C. Accumulation of Generated Partial Products

For the reduction of generated partial products to compute the final product, binary adders, ternary adders, and 4:2 compressors [15] can be utilized. A 4:2 compressor is capable of reducing four partial product rows to two output rows. During our experiments, we observe that the deployment of ternary adders might reduce the overall resource utilization. However, they have higher critical path delays than binary adders. Therefore, in this letter, the 4:2 compressors and binary adders are used for the reduction of the generated partial products. We have used the 6-input LUTs and the associated carry chains to implement them.

III. Results and Discussion

We have used VHDL for the RTL implementations of all presented multipliers. The proposed designs have been synthesized and implemented using Xilinx Vivado 17.4 for the Virtex-7 xc7v585tffg1157-3 FPGA (unless stated otherwise). Power values are estimated by the simulator and power analyzer tools provided by Vivado.

We have compared the implementation results of our proposed multiplier with the Vivado's area/speed-optimized multiplier IPs [4], "R1" [8], "R5" [7], and "R7" [14].[2] Furthermore, the proposed design is also evaluated against the state-of-the-art approximate multipliers "R2" [17], "R3" [2], "R4" [18], and an 8×8 multiplier "R6" from [14].[3] For the

[2]A generic and open-source implementation for every size of multiplier is not available. Signed multiplier "mul8s_1KV8.v" from the library is used.

[3]For "R6" approximate "mult_000.v" from the library is used.

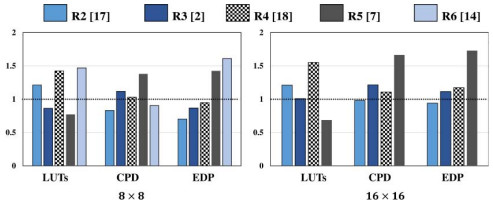

Fig. 4. Comparison of the proposed signed multipliers with the unsigned multipliers (without the signed–unsigned converters). The results are normalized to our proposed multipliers.

unsigned numbers-based architectures in "R1," "R2," "R3," "R4," and "R5," we have implemented signed–unsigned converters. To show a fair comparison, we have reported the performance results of the state-of-the-art multipliers with and without using these signed–unsigned converters.

A. Implementation Results

Table II presents the resource consumption (LUTs), CPD, and EDP requirements of our proposed design and different state-of-the-art accurate and approximate multipliers. In the table, the results for "R2," "R3," "R4," "R5," and "R6" multipliers are inclusive of the signed–unsigned converters. For "R6" multiplier, there is only one design point with the input bit-width of 8×8.

As shown in Table II, except for "R1" [8] and "R5" [7], our proposed multiplier always requires less number of LUTs than other state-of-the-art multipliers for different bit-widths. "R1" and "R5" multipliers utilize sequential computation of partial products to obtain area gains at the cost of high critical path delays. The area savings offered by our designs increase with the size of the multiplier, up to 16% when compared with Vivado 32×32 area/speed-optimized IP.

Our proposed multiplier provides higher performance than state-of-the-art accurate and approximate multipliers. For example, compared to the 8×8 Vivado speed-optimized multiplier IP, our multiplier reduces the critical path delay by 21%. "R1" and "R5" accurate multipliers have higher critical path delays among all presented multipliers.

The energy efficiency of the presented multiplier designs is characterized by the EDP as illustrated in Table II. It can be drawn from the table that our proposed multipliers have better energy efficiency than state-of-the-art across different sizes. For example, our 16×16 multiplier delivers up to 23.6% reduction in EDP when it is compared against "R1."

To further elaborate on the efficacy of our proposed implementation, Table II shows the averages of the products of the normalized values of LUTs utilization, CPD, and EDP (Average [Norm. LUTs $\times$ Norm. CPD $\times$ Norm. EDP]) across different sizes of multipliers. All individual performance metrics of each multiplier have been normalized with respect to the corresponding performance metrics of Vivado area-optimized multiplier IP. Our proposed multiplier outperforms state-of-the-art implementations in the overall score.

We have also compared our proposed *signed* multipliers to the state-of-the-art accurate and approximate *unsigned* multipliers without deploying signed–unsigned converters. Fig. 4 presents the resource utilization, CPD, and EDP of 8×8 and 16×16 proposed implementations, "R2," "R3," "R4," "R5," and "R6" multipliers. These results have been normalized to the implementation results of our proposed implementations. Compared to the accurate 16×16 "R5" multiplier, our implementation provides 39.0% and 42.0% reduction in CPD and EDP, respectively.

TABLE II
IMPLEMENTATION RESULTS OF DIFFERENT MULTIPLIERS. "R2," "R3," "R4," "R5," AND "R6" MULTIPLIERS ARE IMPLEMENTED WITH THE SIGNED–UNSIGNED CONVERTERS. THE RESULTS WITH SHADING ARE THE LOWEST IN THEIR RESPECTIVE COLUMN

Design	4x4			8x8			16x16			32x32			Average Performance
	LUTs	CPD [ns]	EDP [zJs]	LUTs	CPD [ns]	EDP [zJs]	LUTs	CPD [ns]	EDP [zJs]	LUTs	CPD [ns]	EDP [zJs]	
Ours	*18*	*1.65*	*1.86*	*66*	*2.80*	*16.97*	*243*	*4.13*	*93.61*	*928*	*6.21*	*316.43*	0.347
R1: Walters [8]	10	2.00	1.82	36	3.65	16.57	136	6.56	122.53	528	13.09	930.54	0.427
Vivado IP (Speed) [4]	18	2.14	2.27	74	3.54	20.29	286	4.27	146.62	1103	5.81	861.45	0.532
R2: Kulkarni [17]	20	2.12	2.17	86	4.89	36.31	330	6.59	134.39	1257	8.92	303.06	0.885
R3: Ullah [2]	22	3.34	4.57	81	5.19	38.48	296	7.33	136.14	1121	9.65	354.33	1.02
R4: Rehman [18]	18	2.23	2.25	92	4.99	35.43	404	7.03	156.87	1512	9.57	322.10	1.05
R5: Kumm [7]	24	3.84	9.55	73	6.08	58.95	217	9.52	313.56	700	16.25	1631.78	1.931
R6: Mrazek [14] Approx.	-	-	-	121	4.21	27.18	-	-	–	-	-	-	-
R7: Mrazek [14] Accurate	-	-	-	79	3.28	34.25	-	-	–	-	-	-	-
Vivado IP (area) [4]	30	2.91	6.56	88	3.45	31.26	326	5.04	207.96	1102	6.79	1050.63	1

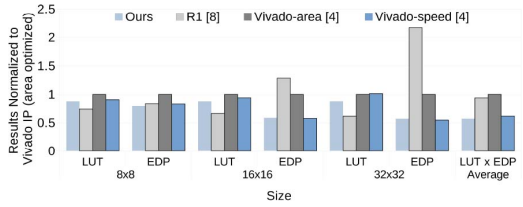

Fig. 5. Results of the neural network use-case. The LUT resources and EDP obtained for designs with different multiplier are normalized to Vivado-area.

B. Case Studies

1) Artificial Neural Network's Inference With Small-Size Multiplier: We have also applied our multiplier for the inference stage of a neural network [19]. The network is used for the classification of handwritten digits from MNIST database. The inference accuracy of the ANN for 10 000 images with 8-b fixed point numbers and our 8×8 multiplier is 96.67%. The original accuracy with 64-b number and multiplier is 97.09%. The loss in classification accuracy for the multiplier is negligible. If the network was implemented on the FPGA with our proposed accurate multiplier instead of the Vivado's area-optimized multiplier, the estimated LUT saving is 17.5%.

2) Artificial Neural Network's Inference Implementation on FPGA: The target FPGA is Xilinx Zynq Ultrascale xczu3eg used in the Ultra96 evaluation platform. The network has one fully connected layer. Inside each neuron, beside the MAC unit, there are also activation and quantization modules. The activation function is ReLU. The quantization module converts the MAC results (which are represented in a wider bit width) back to the original fixed-point format.

First, we implement the network with the Vivado's speed-optimized multiplier with as many number of neurons as possible in three different input sizes, 8×8, 16×16, and 32×32. The timing constraint is 4 ns. After that, the same setups are applied for the Vivado-area multiplier, R6 [9], and ours. The results are presented in Fig. 5. In the combined LUTxEDP average across all input sizes, ours offers the best results. Our multiplier is 5%, 43%, and 37% better than Vivado-speed, Vivado-area, and R6 [9], respectively. While R6 [9] has the lowest LUT counts, its EDP is the worst among all when the input size increases. In comparison with Vivado-speed, ours is comparable in EDP but requires an average of 8% less number of LUTs. These results imply that with the same amount of fixed FPGA resources, more of our multipliers can be instantiated to further exploit the available parallelism of the application with only a slight increase in energy (if any). Our multiplier also fits well with various modern Xilinx FPGA architectures.

IV. CONCLUSION

This letter presented a novel area-optimized, low-latency, and energy-efficient accurate signed multiplier architecture for FPGA-based systems. We have also evaluated the benefits of our multipliers in neural network applications. The RTL models of our designs will be released as an open-source library at https://cfaed.tu-dresden.de/pd-downloads.

REFERENCES

[1] *7 Series DSP48E1 Slice*, document UG479, Xilinx, San Jose, CA, USA, 2018.
[2] S. Ullah *et al.*, "Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators," in *Proc. ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6.
[3] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.
[4] *LogiCORE IP Multiplier v12.0*, document PG108, Xilinx, San Jose, CA, USA, 2015.
[5] A. D. Booth, "A signed binary multiplication technique," *Quart. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236–240, 1951.
[6] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Comput.*, vol. C-22, no. 12, pp. 1045–1047, Dec. 1973.
[7] M. Kumm, S. Abbas, and P. Zipf, "An efficient softcore multiplier architecture for Xilinx FPGAs," in *Proc. IEEE Symp. Comput. Arithmetic (ARITH)*, 2015, pp. 18–25.
[8] E. G. Walters, "Array multipliers for high throughput in Xilinx FPGAs with 6-input LUTs," *Computers*, vol. 5, no. 4, p. 20, 2016.
[9] H. Parandeh-Afshar and P. Ienne, "Measuring and reducing the performance gap between embedded and soft multipliers on FPGAs," in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, 2011, pp. 225–231.
[10] *7 Series FPGAs Configurable Logic Block*, document UG474, Xilinx, San Jose, CA, USA, 2016.
[11] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, 2009, pp. 242–249.
[12] A. Kakacak, "Fast multiplier generator for FPGAs with LUT based partial product generation and column/row compression," *Integr. VLSI J.*, vol. 57, pp. 147–157, Mar. 2017.
[13] M. Kumm, J. Kappauf, M. Istoan, and P. Zipf, "Resource optimal design of large multipliers for FPGAs," in *Proc. IEEE Symp. Comput. Arithmetic (ARITH)*, 2017, pp. 131–138.
[14] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2017, pp. 258–261.
[15] M. Kumm and P. Zipf, "Efficient high speed compression trees on Xilinx FPGAs," in *Proc. Methods Description Lang. Model. Verification Circuits Syst. (MBMV)*, 2014, pp. 171–182.
[16] G. W. Bewick, "Fast multiplication: Algorithms and implementation," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, USA, 1994.
[17] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Int. Conf. VLSI Design*, 2011, pp. 346–351.
[18] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proc. EEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2016, p. 80.
[19] *MNIST-cnn*. (2016). [Online]. Available: https://github.com/integeruser/MNIST-cnn