Multiply-Accumulate Enhanced BDD-based Logic Synthesis on RRAM Crossbars

Saman Froehlich Saeideh Shirinzadeh Rolf Drechsler
Cyber-Physical Systems, DFKI GmbH and Group of Computer Architecture, University of Bremen, Germany
froehlich@cs.uni-bremen.de saeideh.shirinzadeh@dfki.de drechsle@cs.uni-bremen.de

***Abstract*—Resistive random access memory (RRAM) is a non-volatile memory technology which allows to perform computations in both digital and analog circuits. Multiply-Accumulate (MAC) is an analog column-based operation enabled on RRAM crossbars providing high efficiency to perform complex matrix vector multiplications, which is attractive for neural network accelerators. However, the analog computational capability of RRAM devices has not been yet utilized for logic synthesis.**

In this paper, we show how a synthesis approach based on binary decision diagrams (BDD) can efficiently exploit efficient MAC computation enabled by RRAM. The proposed approach highly benefits from a symmetric structure of Boolean functions. Therefore, a design methodology is presented which optimizes and approximates BDDs under provided error thresholds to maximize efficiency of synthesized logic circuits under negligible loss of accuracy. In the experiments, we show that our proposed synthesis approach allows for an average reduction of up to 47% in the number of operations and up to 66% in the number of required devices compared to state-of-the art methods, even without approximation. Using approximation, we can further reduce the number of required devices.

I. Introduction

Resistive Random Access Memory (RRAM) is a non-volatile memory technology. Its low power consumption and inherent computing capabilities make RRAM a promising candidate as a basis for future computer architectures. However, in order to utilize the full potential of RRAM based computations, efficient synthesis approaches are needed. Graph-based representations provide a state-of-the art basis for logic synthesis. Structures such as *Binary Decision Diagrams* (BDDs), *And-Inverter Graphs* (AIGs) and *Majority-Inverter Graphs* (MIGs) have been successfully utilized. Thus, recent research focuses on synthesis approaches for BDDs [1], [2], AIGs [3] and MIGs [4]. Conventional approaches for BDD-based synthesis try to map the BDD nodes to the *implication* (IMP) or *majority* (MAJ) function, which can be implemented in RRAM efficiently (c.f. [2]). Besides digital computation, RRAM also allows for efficient computation of *Multiply-Accumulate* (MAC) which is the basis for RRAM-based neural network implementations (e.g. [5], [6]).

In this paper, we propose a novel BDD-based synthesis approach for RRAM. This approach utilizes the efficient MAC computation capabilities of RRAM in order to compute BDD nodes. BDD nodes can be represented by multiplexers, which can be directly mapped to MAC operations. Since a RRAM array can compute multiple MAC operations in parallel, it allows for the computation of multiple BDD nodes within a single cycle. In order to reduce the number of nodes in the respective BDD and thus the number of computational cycles in a RRAM-based implementation, we use an *Evolutionary Algorithm* (EA) to find a variable ordering, which minimizes the average number of nodes per level. To further boost the efficiency of the proposed approach, we combine the EA with symmetric BDD approximation, which further reduces the number of required devices. In the experiments we show that our approach can outperform state-of-the art BDD-based synthesis methods.

This work was supported by the German Research Foundation (DFG) under contract number DR 287/35-1.

II. Background

A. Basic Concepts

1) Resistive RAM: Resistive Random Access Memory (RRAM) is a nano-scale, two-terminal, non-volatile memristive device [7]. If the voltage V_{set} is applied, the device is put into a high resistive state. Likewise, if the voltage V_{reset} is applied, the state switches to a low resistive state. RRAM crossbars have successfully been utilized for both, digital and analog computing (e.g. [3], [8]).

2) Binary Decision Diagrams: Reduced Order Binary Decision Diagrams (BDDs) are a graph-based, canonical representation of Boolean functions, consisting of multiple nodes. The nodes are ordered on levels, each level being associated with a Boolean input of the function. Each node f in a BDD has two child nodes f_H and f_L. If the level of f is associated with the variable x_i, then f implements the function

$$f = x_i \cdot f_H + \bar{x}_i \cdot f_L \tag{1}$$

B. RRAM MAC Computation

For applications such as neural networks and neuromorphic computing which perform a lot of matrix multiplications, an efficient implementation of the MAC operation is needed. RRAM provides the basis for such an efficient implementation, by allowing to compute multiple MAC operations in parallel.

Consider the RRAM crossbar depicted in Fig. 1. The RRAM crossbar in this figure computes $I = Ax$, where $I = (i_1, \ldots, i_m)^T$, $x = (x_1, \ldots, x_n)^T$ and $A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}$.

The resistive values of the RRAM devices are set to $a_{j,k}^{-1}$. Applying the voltages $x_1, \ldots, x_n$ to the corresponding rows computes $i_j = \sum_{k=1}^{n} a_{j,k} \cdot x_k$. Thus, after writing the resistive values $a_{j,k}^{-1}$ to the corresponding devices, m MAC operations are computed in parallel within a single cycle, each consisting of n multiplications.

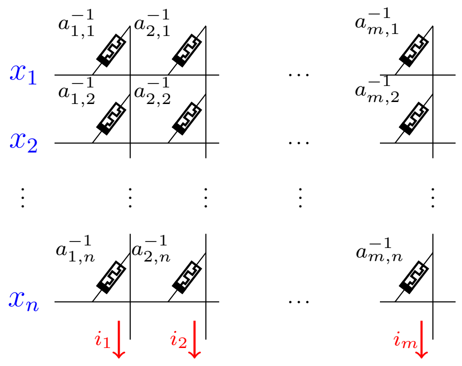


Fig. 1. MAC computation with RRAM

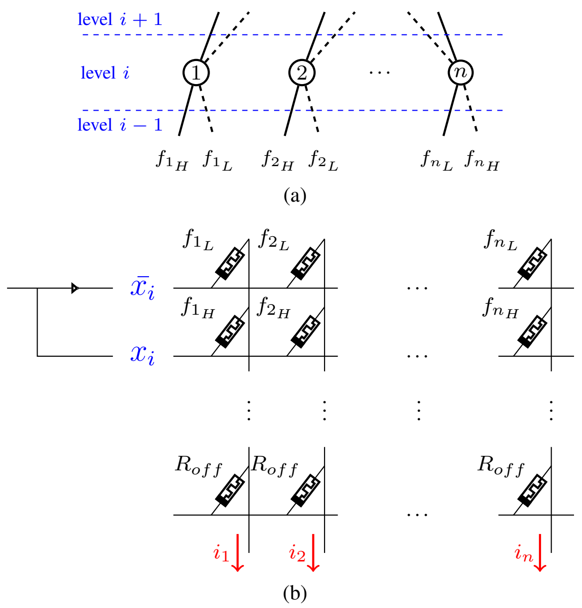
C. Symmetric BDD Approximation

Approximate Computing is a design paradigm, which trades off accuracy for complexity. Many approaches for approximate HW synthesis exist (e.g. [9], [10], [11], [12], [13]). The authors of [14] propose an approach which enhances the symmetry of a given BDD while introducing errors to the outputs. The error is measured in terms of the bit threshold B_t, which is a specialization of the error rate. The B_t counts the number of output bits, which differ in the truth table of the approximated function from that of the original function. The presented algorithm computes the symmetric approximation and the corresponding B_t of a given BDD in polynomial time. The number of bit flips needed in order to make a function fully symmetric with respect to the number of input variables is computed in the process. Thus, the B_t is computed implicitly while performing the symmetrization. Increasing the symmetry reduces the number of nodes for most benchmarks presented in [14]. If a BDD has more than one output, the algorithm uses a greedy approach in order to make the outputs fully symmetric while ensuring that the error bound is met. Partial symmetry (i.e., with respect to some of the input variables) is observed not to have a positive impact on the BDD size in general.

D. Related Work

In [1], the authors propose an implication based synthesis approach which maps BDD nodes to RRAM devices. Each BDD node is implemented by five RRAM devices and requires six steps for computation. In contrast, our approach requires only two RRAM devices per BDD node, needs only two steps for initialization of the devices and two write cycles per node.

The authors of [2] present a comprehensive approach for RRAM-based circuitry. The appraoches allow for AIGs, MIGs and BDDs as basic circuit representation. The authors use an EA for the optimization of the BDD size, by finding an optimal variable ordering. Besides the implication based approach of [1], the authors also use a majority based approach for synthesis. This approach needs six devices, while only needing five steps for computation. While we use an EA for the optimization of the BDD, we further use symmetric approximation. In addition, we use a novel synthesis approach to implement BDD nodes, which only needs two devices and less cycles for computation.



(b)

Fig. 2. BDD level computation on RRAM crossbar using multiply- accumulate (MAC) operation.

III. Approximate BDD-based Synthesis for RRAM Crossbar

We propose an approach for logic synthesis on regular RRAM crossbars which uses BDDs for efficient representation of arbitrary Boolean functions. The proposed approach particularly benefits from efficient realization of logic primitives, i.e., multiplexers in case of BDDs, using the MAC operation executed in RRAM crossbar columns. This section explains the procedure to compute a BDD on RRAM crossbar and discusses the cost functions and employed optimization schemes to tackle them.

To compute a BDD on a RRAM crossbar, every node has to be realized as a 2x1 multiplexer (mux) designating Boolean relation $\bar{s}{\cdot}x{+}s{\cdot}y$, where s is the select line and x and y are the two inputs. For BDD implementation, all multiplexers realizing nodes in the i_{th} BDD level use an identical input variable of the target function as select line. In a BDD representing a function with N input variables using an initial ascending variable ordering, i.e., $x_0 < x_1 < \cdots < x_{(N-1)}$, level i corresponds to input variable x_i. Using an arbitrary variable ordering, a BDD level of index i can correspond to any input variable with a different index j, while $j \in 0, \ldots, N-1$.

Assume the BDD level shown in Fig. 2(a). Level i including n nodes is eligible for computation when the low and high child nodes, i.e, $f_{i_L}, f_{i_H}, 1 \leq i \leq n$ are previously computed. In this case, each node can be computed by a single MAC operation while the low and high child nodes are stored in two successive rows within the same column as shown in Fig. 2(b). The output of MAC operation at each column under assumption that the rest of RRAM devices are under zero voltage and ideally do not pass any electrical current is equal to $\bar{x}_i \cdot f_{i_L} + x_i \cdot f_{i_H}$, representing the value of node i.

Using the MAC operation as explained above, a BDD can be computed on a RRAM crossbar starting from the bottom of the graph, i.e., 0 and 1 terminals. To compute each level, first the low and high child nodes have to be stored in the first and second row of the crossbar, respectively. Assuming that one word with a maximum length of r can be written into a row at each cycle, for a BDD level with a size of n_l, i.e., number of nodes, it takes $2\lceil\frac{n_l}{r}\rceil$ write cycles to store the child node values on two crossbar rows. For example, for a level with 18 nodes while using a write register with 16 bits it takes 4 write cycles to have the level inputs ready on the RRAM, as both low and high child nodes' values exceed the register size.

When the inputs of the BDD level are stored as shown in Fig. 2(b), the MAC operation can be carried out by applying the input variable corresponding to the level being computed. After computation, the currents of the columns are fed into analog to digital converters (ADC) and then written into the crossbar in resistance form in order to be used as the inputs of the next level. This procedure continues until all levels are computed.

The number of writes to the memory in the approach presented above depends on BDD level sizes, the write register length, and nonconsecutive fanouts/BDD node outputs which target levels not immediately after their origin level. Fig. 3(a) shows a BDD based on initial ordering which has a nonconsecutive fanout indicated in red targeting the second level, while originating from level four. Since the presented synthesis procedure uses the outputs of the nodes in the very next level without permanent copies, such fanouts have to be copied to be used at the fanout targets. The copy devices can be located nearby node devices in the remaining idle columns as shown in Fig. 3(b). In this case, the total number of write cycles required to compute a BDD on RRAM with our proposed approach is equal to

$$OP = \sum \left\lceil \frac{n_l}{r} \right\rceil \cdot 2 + \sum \left\lceil \frac{f_l}{r} \right\rceil,$$

where n_l is the level size, r is the length of the write register, and f_l in the number of nonconsecutive fanouts in the entire graph.

The devices for node computations are reused for the next level and therefore their total number depends on the maximum level size. However, the copy devices for nonconsecutive fanouts are not reused during implementation, so they are summed up. Accordingly, the total number of RRAM devices required is as follows,

$$R = \left(\max \left\lceil \frac{n_l}{r} \right\rceil \cdot 2 + \sum \left\lceil \frac{f_l}{r} \right\rceil \right) \cdot r.$$

An optimization approach has been developed which targets the number of write cycles and RRAM devices needed by the proposed synthesis approach which are defined above. For this purpose, we use a prioritized-ε-preferred EA [15] for variable ordering which has been efficiently applied to BDDs. We consider the latency a more important criterion and give a higher priority to it compared to the number of devices representing area.

As discussed before, both of the cost functions representing latency and area of the resulting implementations depend on the level sizes and number of nonconsecutive fanouts denoted by n_l and f_l, respectively. Indeed, a BDD with smaller levels and less nonconsecutive fanouts can be computed more

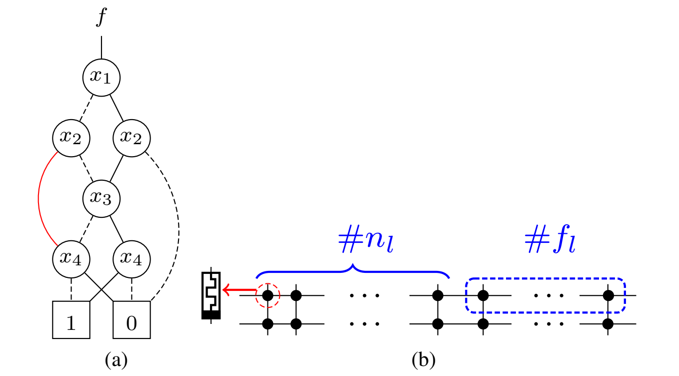


Fig. 3. (a) Example BDD with a nonconsecutive fanout indicated in red. (b) Storing nodes and nonconsecutive fanout values in RRAM crossbar.

efficiently on RRAM crossbar, requiring smaller number of write cycles and devices. Therefore, maximum value of n_l over all BDD levels as well as f_l have been considered as optimization criteria in a third priority level.

In order to increase efficiency, we also propose an approximation approach besides optimization which lowers the cost function further at a cost of negligible inaccuracy. We use the approximation technique proposed in [14] which aims to make BDDs as symmetric as possible by flipping some output bits under an error rate of 5%. Symmetrization is particularly beneficial for our proposed synthesis approach as it is known that BDDs representing totally symmetric functions grow in each level at most by one node [16]. Therefore, symmetrization keeps the level sizes close to each other and avoids extra costs imposed by levels extremely larger than others.

As the error threshold of 5% does not allow to make all BDDs fully symmetric, our proposed approach finds the highest symmetries possible. In this condition, some outputs of the the approximated function are symmetric with respect to the inputs but not all of them. Size of BDDs representing fully symmetric functions do not depend on ordering, while for partially symmetric functions different ordering vectors result in different costs. Therefore, in case of partial symmetrization we perform variable ordering by the EA to lower the costs further.

It should be noted that the required RRAM crossbar for the proposed approach needs a minimum number of rows equal to the sum of the largest level size of the target BDD and fanout count. The required dimensions can be smaller in case of using more than a single memory bank. Other hardware cost includes a light control unit, read and write circuitries, as well as analog-to-digital/digital-to-analog converters similarly to what needed by RRAM-based dot-product engines used in NN-accelerators. This makes it cheap for our proposed approach to co-exist on such systems without extra hardware overhead when a synthesis unit is required.

IV. Experimental Results

We have implemented the proposed approach using CUDD 3.0.0 [17]. We compare our results to those presented in [1] and [2], which were optimized for the number of operations in a BDD-based implementation. In order to have comparable results, we haven chosen benchmarks from the same benchmark set (23 circuits from LGSynth91 [18]) and

TABLE I
COMPARISON OF RESULTS TO CHAKRABORTI ET AL. [1] AND SHIRINZADEH ET AL. [2]

				Chakraborti et al. [1]		Shirinzadeh et al. [2] IMP		Shirinzadeh et al. [2] MAJ		Proposed Approach Exact				Proposed Approach Approximated, $B_t \leq 5\%$			
Benchmark	PI/PO	R	OP	R	OP	R	OP	R	OP	R	OP	Red R	Red OP	R	OP	Red R	Red OP
5xp1_90	7/10	112	19	84	73	65	42	62	42	32	14	71.4%	26.3%	32	14	71.4%	26.3%
alu4_98	14/8	864	180	642	334	1014	77	1069	77	352	94	59.3%	47.8%	272	73	68.5%	59.4%
apex1	45/45	8016	1177	1626	705	3040	277	3185	232	1056	244	86.8%	79.3%	672	182	91.6%	84.5%
apex4	9/19	912	147	2073	447	2224	62	2588	53	720	123	21.1%	16.3%	672	112	26.3%	23.8%
apex6	135/99	3600	621	770	1169	220	813	235	678	1760	390	51.1%	37.2%	1568	760	56.4%	-22.4%
apex7	49/37	2176	358	290	437	190	328	151	279	512	130	76.5%	63.7%	448	252	79.4%	29.6%
b9	41/21	496	115	125	298	77	267	92	226	240	95	51.6%	17.4%	208	127	58.1%	-10.4%
clip	9/5	224	42	120	89	93	63	107	54	64	20	71.4%	52.4%	64	20	71.4%	52.4%
cm150a	21/1	130848	16412	56	127	28	127	30	106	96	46	99.9%	99.7%	96	46	99.9%	99.7%
cm162a	14/5	176	37	46	102	38	89	43	75	80	31	54.5%	16.2%	80	31	54.5%	16.2%
cm163a	16/5	192	42	42	116	31	108	32	92	80	35	58.3%	16.7%	80	35	58.3%	16.7%
cordic	23/2	64	48	32	149	26	140	29	117	64	48	0%	0%	48	47	25.0%	2.1%
misex1	8/7	80	19	83	69	79	50	91	42	48	17	40.0%	10.5%	32	12	60.0%	36.8%
misex3	14/14	528	185	444	185	681	86	781	72	304	83	42.4%	55.1%	144	37	72.7%	80.0%
parity	16/1	32	32	23	113	6	112	7	96	32	32	0%	0%	32	32	0%	0%
seq	41/35	57040	19099	1566	692	1207	248	1398	207	848	231	98.5%	98.8%	528	159	99.1%	99.2%
t481	16/1	144	39	26	107	16	100	30	84	144	39	0%	0%	144	39	0%	0%
table5	17/15	1344	390	580	168	1346	107	1511	90	336	105	75.0%	73.1%	192	60	85.7%	84.6%
too_large	38/3	2624	996	282	232	182	229	212	191	384	114	85.4%	88.6%	176	73	93.3%	92.7%
x1	51/35	1408	292	230	398	186	333	217	282	688	159	51.1%	45.5%	416	212	70.5%	27.4%
x2	10/7	144	29	60	80	45	65	52	55	48	21	66.6%	27.5%	48	21	66.7%	27.6%
x3	135/99	3296	716	770	1169	215	813	252	678	1744	389	47.1%	45.7%	1632	762	50.5%	-6.4%
x4	94/71	1968	363	401	642	209	573	333	479	720	233	63.4%	35.8%	992	388	49.6%	-6.9%
AVG		9403.8	1798.0	450.9	343.5	487.7	222.1	543.8	187.3	450.1	117.1	55.3%	41.5%	372.9	151.9	61.3%	35.3%

PI/PO: number of primary inputs/number of primary outputs
R: number of RRAM devices, OP: number of operations, Red: Reduction achieved, relative to naive implementation

the same parameters for the EA as the authors of [2]. We have applied the EA in five independent runs to each benchmark and evaluated the best out of them, i.e., the smallest number of operations. We assumed the register size to be 16 bit.

Table I shows the results for the best of the five runs. The first four columns show the characteristics of the benchmarks. The first column shows the name of the respective circuit, the second column the number of primary inputs and outputs and the columns three and four the number of devices (R) and operations (OP) needed, if the natural variable ordering is applied and the proposed synthesis approach is used. The fifth and sixth column show the results of [1], where again R is the number of RRAM devices needed and OP is the number of operations. Next, the results of [2] are shown (columns seven to ten). First, columns seven and eight show the results for an IMP-based implementation of the BDD nodes, while columns nine and ten show the results if MAJ is used. Finally, the columns eleven and twelve show the absolute results of the EA, if no approximation is used, while columns thirteen and fourteen show the reduction relative to the initial results where the natural variable ordering is used in $\%$. The last four columns show the results if the circuit is approximated before the EA is performed. During approximation, we assume an error bound of 5% for the B_t.

We can see that our proposed approach significantly reduces the number of devices and operations needed compared to [1] and [2]. Compared to [1], it needs almost the same number of devices, but reduces the number of operations by about 226.4 (which is a reduction of about 66%) on average. Compared to [2] our proposed approach needs 37.6 (8%) less devices if IMP is used and 93.7 (17%) less devices if MAJ is used for the implementation of [2]. Further, on average, our approach uses 105.0 (47%) and 70.2 (37%) less operations if IMP or MAJ are used for the implementation of [2], respectively.

Increasing the symmetry of a given BDD in terms of the approximation approach of [14] results in balancing the number of nodes per level. Since the number of needed devices in the presented approach scales with the maximum number of nodes per level, using approximation further reduces the number of devices. However, as reported in [14] the approach does not always reduce the total number of nodes and thus the number of operations needed for the computation. On average, it allows to reduce the number of needed devices at the cost of computation time and accuracy. However, for ten out of the 23 benchmarks, the approximate implementation needs less operations than the exact implementation, while only performing worse for six benchmarks. Further, on average, the results of the approximation still need less operations than [2] and [1].

V. Conclusions

In this paper, we have presented a novel BDD-based synthesis approach for RRAM crossbars. This approach utilizes the MAC computation capabilities of RRAM devices, which allows for the computation of multiple BDD nodes in parallel. We use an EA to optimize the variable ordering of the BDD in order to reduce the size of the final implementation. In the experiments we show that our approach outperforms state-of-the art BDD synthesis methodologies. We further boost the performance of our synthesis approach by additionally applying approximation to the given BDDs.

References

[1] S. Chakraborti, P. Chowdhary, K. Datta, and I. Sengupta, “BDD based synthesis of Boolean functions using memristors,” in *IDT*, 2014, pp. 136–141.

[2] S. Shirinzadeh, M. Soeken, P. Gaillardon, and R. Drechsler, “Logic synthesis for rram-based in-memory computing,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1422–1435, 2018.

[3] S. Frerix, S. Shirinzadeh, S. Froehlich, and R. Drechsler, “Comprime: A compiler for parallel and scalable reram-based in-memory computing,” in *NanoArch*, 2019.

[4] P.-E. Gaillardon, L. G. Amarù, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, “The programmable logic-in-memory (PLiM) computer,” in *Design, Automation & Test in Europe*, 2016, pp. 427–432.

[5] M. K. Mahadevaiah, E. Perez, C. Wenger, A. Grossi, C. Zambelli, P. Olivo, F. Zahari, H. Kohlstedt, and M. Ziegler, “Reliability of cmos integrated memristive hfo2 arrays with respect to neuromorphic computing,” in *2019 IEEE International Reliability Physics Symposium (IRPS)*, March 2019, pp. 1–4.

[6] W. Haensch, “Analog computing for deep learning: Algorithms, materials architectures,” in *2018 IEEE International Electron Devices Meeting (IEDM)*, Dec 2018, pp. 3.2.1–3.2.4.

[7] L. O. Chua and Sung Mo Kang, “Memristive devices and systems,” *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, Feb 1976.

[8] W. Zhang, H. Wu, P. Yao, B. Gao, and H. Qian, “A compact model of analog rram for neuromorphic computing system design,” in *2018 China Semiconductor Technology International Conference (CSTIC)*, March 2018, pp. 1–3.

[9] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, “Aslan: Synthesis of approximate sequential circuits,” in *Design, Automation and Test in Europe*, 2014, pp. 364:1–364:6.

[10] F. Vaverka, R. Hrbacek, and L. Sekanina, “Evolving component library for approximate high level synthesis,” 2016, pp. 1–8.

[11] S. Hashemi, R. I. Bahar, and S. Reda, “Drum: A dynamic range unbiased multiplier for approximate applications,” in *International Conference on Computer-Aided Design*, 2015, pp. 418–425.

[12] S. Lee, L. K. John, and A. Gerstlauer, “High-level synthesis of approximate hardware under joint precision and voltage scaling,” in *Design, Automation and Test in Europe*, 2017, pp. 187–192.

[13] Y. Lai, C. Lin, C. Wu, Y. Chen, and C. Wang, “Efficient synthesis of approximate threshold logic circuits with an error rate guarantee,” in *Design, Automation and Test in Europe*, March 2018, pp. 773–778.

[14] A. Bernasconi, V. Ciriani, and T. Villa, “Approximate logic synthesis by symmetrization,” in *Design, Automation and Test in Europe*, March 2019, pp. 1655–1660.

[15] S. Shirinzadeh, M. Soeken, D. Große, and R. Drechsler, “An adaptive prioritized ϵ-preferred evolutionary algorithm for approximate BDD optimization,” in *Genetic and Evolutionary Computation Conference*, 2017, pp. 1232–1239.

[16] C. Scholl, D. Moller, P. Molitor, and R. Drechsler, “Bdd minimization using symmetries,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 81–100, Feb 1999.

[17] F. Somenzi, “CUDD: CU Decision Diagram package-release 3.0.0,” University of Colorado at Boulder, 2015.

[18] S. Yang, “Logic synthesis and optimization benchmarks user guide version 3.0,” 1991.