Approximate Logic Synthesis Using BLASYS

(WOSET-2019)

Jingxiao Ma, Soheil Hashemi, and S. Reda
School of Engineering, Brown University, Providence RI 02912

***Abstract*—Approximate computing is an emerging paradigm where design accuracy can be traded for improvements in design metrics such as design area and power consumption. In this work, we overview our open-source tool, BLASYS, for synthesis of approximate circuits using Boolean Matrix Factorization (BMF). In our methodology the truth table of a given circuit is approximated using BMF to a controllable approximation degree, and the results of the factorization are used to synthesize the approximate circuit output. BLASYS scales up the computations to large circuits through the use of partition techniques, where an input circuit is partitioned into a number of interconnected subcircuits and then a design-space exploration technique identifies the best order for subcircuit approximations. BLASYS leads to a graceful trade-off between accuracy and full circuit complexity as measured by design area. Using an open-source design flow, we extensively evaluate our methodology on a number of benchmarks, where we demonstrate that the proposed methodology can achieve on average 48.14% in area savings, while introducing an average relative error of 5%.**

I. Introduction

Approximate computing techniques trade off accuracy with improvements in design area and power consumption [14]. Approximate computing is attractive in applications domains that are inherently tolerant to errors such as machine learning, computer vision, computer graphics, and signal processing. A central issue in approximate computing is how to automatically synthesize an approximate circuit from an input circuit [6]–[8], [10], [11], [13], [16], [17].

This paper overviews the public release of BLASYS, which is a tool to synthesize approximate circuits [3], [4]. BLASYS is based on Boolean Matrix Factorization (BMF), where the truth table of a circuit is represented as a matrix that is factorized into two smaller Boolean matrices that are then synthesized into the approximate circuit. BMF is a derivative of non-negative matrix factorization (NNMF) [5], [9]. The non-negativity constraints on the factorization arise in physical domains, such as computer vision and document clustering [20]. Recent advances in the mathematical community extend NNMF techniques to Boolean matrices, where matrix operations are carried in $GF(2)$ [9]. The use of BMF creates a solid foundation for approximate logic synthesis, and enables systematic trade-off between design complexity and Quality of Results (QoR). Since enumerating the truth table is limited to only circuits of moderate number of inputs, BLASYS partitions a large-input circuit into smaller subcircuits, that are then approximated individually using BMF. We devise a greedy design space exploration method that identifies a good ordering for the subcircuits to approximate together with the approximation degree for each subcircuit. Our tool uses a full stack of open-source tools, including LSOracle for circuit partitioning [12], Yosys for Verilog parsing [19], iVerilog for simulation and QoR estimation [18], and ABC for logic synthesis [2]. We evaluate our approach on a large a number of circuits from the EPFL benchmark set [1]. We show that our approach is able to trade-off accuracy with circuit area in a graceful manner.

The organization of this paper is as follows. We discuss the details of BLASYS in Section II, where we describe the basic idea of using BMF algorithms to approximate logic circuits, and then show how to scale our proposed method to larger circuits. We describe BLASYS tool-chain flow in Section III. We provide results from the public release of BLASYS in Section IV. Finally, we summarize our main conclusions and directions for future work in Section V.

II. Proposed Methodology

Non-negative matrix factorization (NNMF) is a factorization technique where a $k \times m$ matrix $\mathbf{M}$ is factored into two non-negative matrices: a $k \times f$ matrix $\mathbf{B}$, and an $f \times m$ matrix $\mathbf{C}$, such that $\mathbf{M} \approx \mathbf{BC}$ [5]. NNMF has been extended to Boolean matrices where elements of all matrices are restricted to Boolean values. In this case, multiplications can be performed using logical AND, and additions are performed using logical OR [9]. Figure 1 provides an example of BMF over $GF(2)$.

In our proposed approach, a multi-output logic circuit with k inputs and m outputs is first evaluated to generate its truth table. The truth table, represented by $\mathbf{M}$, is then given as input to a BMF algorithm together with the target factorization degree $1 \leq f < m$, to produce the two factor matrices $\mathbf{B}$ and $\mathbf{C}$. Matrix $\mathbf{B}$ is then given as the input truth table to a logic synthesis tool to generate a k input, f output circuit, which we refer to as the *compressor circuit*. Note that the compressor matrix is simply the truth table of a circuit with the same number of inputs as the original circuit but with fewer (f to be exact) output signals. Therefore, it

$$\begin{bmatrix}1&1&1&0&1\\1&0&0&1&0\\1&0&1&0&1\\0&1&0&0&1\\0&0&1&1&1\end{bmatrix}$$

(a) input matrix

$$\begin{bmatrix}1&1&0\\0&0&1\\1&0&0\\0&1&0\\1&0&1\end{bmatrix}\begin{bmatrix}1&0&1&0&1\\0&1&0&0&1\\0&0&0&1&0\end{bmatrix}=\begin{bmatrix}1&1&1&0&1\\0&0&0&1&0\\1&0&1&0&1\\0&1&0&0&1\\1&0&1&1&1\end{bmatrix}$$

(b) factorization using BMF

Fig. 1. BMF example.

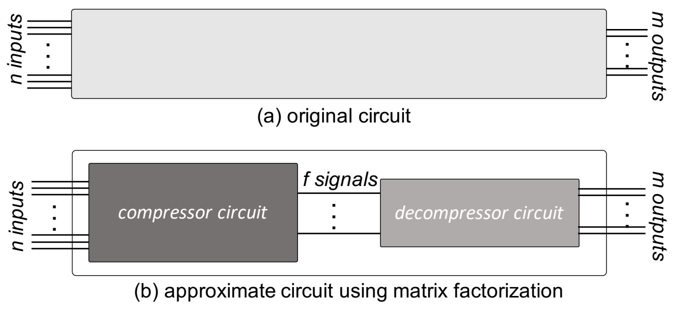

Fig. 2. Generating approximate circuits using BMF.

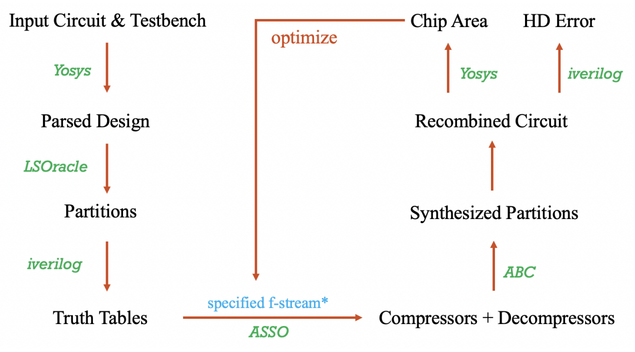

Fig. 3. BLASYS flow.

can easily be mapped to logic. These f outputs from the compressor circuit are then combined by the *decompressor circuit* according to the $\mathbf{C}$ matrix using a network of OR gates to generate the approximate m outputs as illustrated in Figure 2. The compressor and decompressor circuits can then be optimized by logic tools as one circuit to remove any logic redundancies. Using this methodology, any arbitrary circuit can be approximated by forcing the circuit to compress as much information as possible in f intermediate signals and then decompress them using simple OR gates.

Calculating the BMF is limited by computational complexity as one needs to generate the truth table for every possible input and state combination. Furthermore, BMF is a NP-hard problem, and most algorithms in the literature are heuristics [5], [9]. To address scalability, we partition a large circuit into a number of subcircuits, each with a maximum number of k inputs. k is chosen to make the runtime of the factorization algorithm tractable and to meet memory requirements. In our case we use a modified approach of the hypergraph partitioning tool KaHyPar [1] as part of LSOracle []. We then proceed to approximate each subcircuit individually. To evaluate the QoR of an approximate subcircuit, we first substitute a partitioned subcircuit by its approximate counterpart, and then evaluate the QoR of the entire modified circuit using the supplied testbenches. In our testbenches, we use Monte Carlo sampling to estimate the QoR of the approximate version of the entire circuit.

Partitioning a large circuit can yield tens of interconnected subcircuits. The order of processing the subcircuits and the target factorization degree for each subcircuit impact the results. BLASYS performs iterative design space exploration to identify a good ordering. At each iteration, BLASYS evaluates the impact of approximating each subcircuit by reducing its factorization degree by one, and assessing the resulting reduction in total circuit area and loss in QoR. The approximate subcircuit that leads to the smallest ratio of error to area reduction is then chosen and incorporated into the full circuit. The iterative process is then repeated as long as the error is above the set threshold or all subcircuits are approximated to the minimum possible factorization degree, i.e., $f = 1$.

III. BLASYS Tool Chain

Figure 3 prvoides the flow of BLASYS. To begin with, an input circuit is parsed with Yosys [19] to estimate initial chip area, where ASAP 7nm design library is used. Using the provided testbench, BLASYS simulates the input circuit and store original results for QoR estimation. The simulation tool we used is Icarus Verilog [18].

Next, LSOracle [12] is used to partition the input circuit to multiple subcircuits, each of which has a similar size. BLASYS partitions an input circuit until all partitions have less than 16 inputs. BLASYS then creates a testbench that generates the truth table for each subcircuit. We use the ASSO algorithm [9] to perform BMF on each truth table based on a vector called *f-stream*, which consists of factorization degree for each subcircuit, which is determined by the design space exploration method as discussed in Section II. As a result, each truth table is factorized into a compressor and decompressor. BLASYS calls ABC [2] to synthesize the compressor matrix to a circuit and uses a network of logic OR to represent decompressor. Thus, an approximated version of the input circuit can be obtained by recombining all approximated subcircuits. Afterwards, BLASYS calls Yosys to estimate the chip area of the approximate circuit and executes a simulation using the input testbench. From the original and approximated simulation results, QoR can be estimated by measuring the Hamming Distance Error, which is the number of output bit flips in the results of the testbench simulation divided by the total number of output bits. The area reduction ratio and QoR are used to optimize f-stream iteratively and greedily as mentioned in Section II. To bring the the runtime of BLASYS to practical levels for large circuits, we implement a parallel mode, which allows our flow to work on multiple designs simultaneously using multiple cores.

IV. Experimental Results

In our experiments we consider a number of benchmarks from the EPFL benchmark set [1], and set $k \leq 16$. These numbers are simply chosen as they provide a balanced trade-off between truth table complexity and number of subcircuits. To generate the testbenches, we use Monte Carlo simulation with 10,000 randomly generated input test vectors. For QoR, we measure the normalized Hamming Distance Error by ratio of flipped bits. Figure 4 shows trade-off between accuracy

TABLE I
IMPROVEMENT IN DESIGN AREA AS A FUNCTION OF ALLOWED MAXIMUM HAMMING DISTANCE (HD) ERROR.

benchmark	I/O	Number of Partitions	Area Imp. (%) 5% HD	Area Imp. (%) 10% HD
Adder	256/129	20	11.09	21.71
Alu control unit	7/26	5	51.71	71.03
Barrel shifter	135/128	25	49.41	55.95
Coding-cavlc	10/11	50	71.67	90.56
i2c controller	147/142	100	63.86	78.66
int2float converter	11/7	5	35.32	47.27
Max	512/130	150	53.90	59.33
Average			48.14	60.64

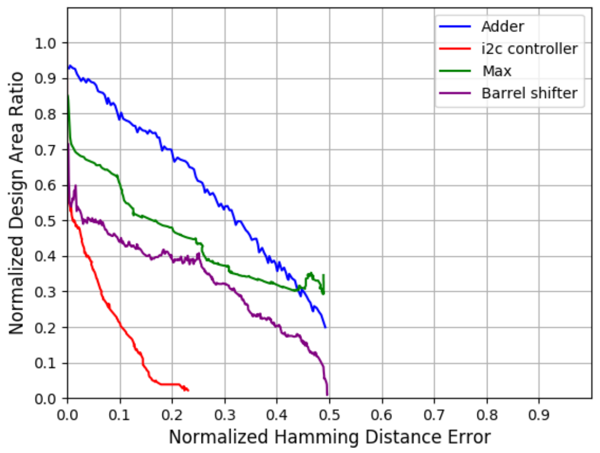

Fig. 4. Trade-off between design area and QoR for 4 benchmarks.

and design area on four benchmarks based on our design-space exploration technique. Although global optimum is not guaranteed, our results show significant area reduction within small HD error interval. This is beneficial since relatively small amount of error makes more sense in practice. We tabulate the results from all our evaluated benchmarks in Table I, where we report area reduction ratio at error threshold 5% and 10% in Table I. The second column in the table provides I/O information about benchmarks, and the third column provide the number of partitioned subcircuits. Based on the circuit, benefits of 11%-71% can be achieved within 5% HD error, which leads to 48.14% chip area reduction on average. The number increases to 60.64% with 10% HD error metric. As the figure and table demonstrate, BLASYS achieves a large amount of area reduction within small HD error.

The runtime of our tool is dominated by simulation process. For example, it takes around 9.4 seconds to simulate Max circuit on 10,000 test cases with iVerilog. Due to the greedy optimization scheme, total times of simulation grows quadratically with number of partitions. Thus, choosing a proper partitioning is vital. Our modified ASSO algorithm works efficiently on subcircuits with $k \leq 16$. With 16 inputs, ASSO takes around 0.6 second to perform BMF.

V. CONCLUSIONS

In this paper we overviewed the public release of BLASYS. We have demonstrated that BLASYS can synthesize approximate version for arbitrary large circuits in Verilog. BLASYS partitions the circuits using hypergraph partitioning techniques and then proceeds to approximate the individual circuit partition, while taking into account the impact on global area and QoR. BLASYS leads to a systematic approach to trade-off accuracy with circuit complexity. Our tool is available at http://github.com/scale-lab/blasys.

Acknowledgments: This work is partially supported by NSF grant 1814920 and DoD ARO grant W911NF-19-1-0484.

REFERENCES

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. *Proceedings of the 24th International Workshop on Logic Synthesis (IWLS)*, 2015.
[2] Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pages 24–40. Springer, 2010.
[3] Soheil Hashemi and Sherief Reda. ”generalized matrix factorization techniques for approximate logic synthesis. In *IEEE/ACM Design Automation Test in Europe*, pages 1289–1292, 2019.
[4] Soheil Hashemi, Hokchhay Tann, and Sherief Reda. Blasys: approximate logic synthesis using boolean matrix factorization. In *Proceedings of the 55th Annual Design Automation Conference*, page 55. ACM, 2018.
[5] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.
[6] S. Lee, L. K. John, and A. Gerstaluer. High-level synthesis of approximate hardware under joint precision and voltage scaling. In *Design, Automation and Test in Europe*, 2017.
[7] C. Li, W. Luo, Sachin S. Sapatnekar, and Jiang Hu. Joint precision optimization and high level synthesis for approximate computing. In *Design Automation Conference*, pages 104:1–6, 2015.
[8] Jin Miao, Andreas Gerstlauer, and Michael Orshansky. Approximate logic synthesis under general error magnitude and frequency constraints. In *Proceedings of the International Conference on Computer-Aided Design*, pages 779–786, 2013.
[9] P. Miettinen and J. Vreeken. Mdl4bmf: Minimum description length for boolean matrix factorization. *ACM Transactions on Knowledge Discovery from Data*, 8(4):18:1–31, 2014.
[10] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. ” ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits. In *Design, Automation and Test in Europe*, pages 1–6, 2014.
[11] Kumud Nepal, Soheil Hashemi, Hokchhay Tann, R. Iris Bahar, and Sherief Reda. Automated high-level generation of low-power approximate computing circuits. In *IEEE Trans. Emerging Topics Comput.*, volume 17(1), pages 18–30, 2019.
[12] W. Lau Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon. Lsoracle: a logic synthesis framework driven by artificial intelligence. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, November 2019.
[13] Ashish Ranjan, Arnab Raha, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Aslan: Synthesis of approximate sequential circuits. In *Design, Automation & Test in Europe Conference*, pages 1–6, 2014.
[14] S. Reda and M. Shafique (Eds). *Approximate Circuits: Metehodologies and CAD*. Chapman & Hall/CRC, 2019.
[15] J. Stainstrup and W. Wolf. *Hardware/software co-design: principles and practice*. Springer.
[16] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *DAC Design Automation Conference 2012*, pages 796–801, June 2012.
[17] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation and Test in Europe*, pages 1367–1372, 2013.
[18] Stephen Williams. Icarus verilog. http://iverilog.icarus.com/, 2006.
[19] Clifford Wolf. Yosys open synthesis suite. http://www.clifford.at/yosys/, 2016.
[20] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273, 2003.