BLASYS: Approximate Logic Synthesis Using Boolean Matrix Factorization

Soheil Hashemi
School of Engineering
Brown University
Providence, RI 02912
soheil_hashemi@brown.edu

Hokchhay Tann
School of Engineering
Brown University
Providence, RI 02912
hokchhay_tann@brown.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI 02912
sherief_reda@brown.edu

ABSTRACT

Approximate computing is an emerging paradigm where design accuracy can be traded off for benefits in design metrics such as design area, power consumption or circuit complexity. In this work, we present a novel paradigm to synthesize approximate circuits using Boolean matrix factorization (BMF). In our methodology the truth table of a sub-circuit of the design is approximated using BMF to a controllable approximation degree, and the results of the factorization are used to synthesize a less complex subcircuit. To scale our technique to large circuits, we devise a circuit decomposition method and a subcircuit design-space exploration technique to identify the best order for subcircuit approximations. Our method leads to a smooth trade-off between accuracy and full circuit complexity as measured by design area and power consumption. Using an industrial strength design flow, we extensively evaluate our methodology on a number of testcases, where we demonstrate that the proposed methodology can achieve up to 63% in power savings, while introducing an average relative error of 5%. We also compare our work to previous works in Boolean circuit synthesis and demonstrate significant improvements in design metrics for same accuracy targets.

ACM Reference Format:
Soheil Hashemi, Hokchhay Tann, and Sherief Reda. 2018. BLASYS: Approximate Logic Synthesis Using Boolean Matrix Factorization. In *Proceedings of ACM Conference (DAC'18).* ACM, New York, NY, USA, 6 pages.
https://doi.org/10.1145/3195970.3196001

1 INTRODUCTION

Approximate computing is an emerging paradigm that trades off accuracy with improvements in power consumption, hardware complexity, or design area. Approximate computing is effective in applications that have inherent resilience to errors, such as signal processing, machine learning, computer vision, and computer graphics. As data-rich applications continue to rise, the relevance and need for approximate computing will increase.

A key problem in approximate computing is how to generate or synthesize an approximate circuit given as inputs an existing circuit, presumably accurate. There are two lines of research in approximate synthesis. The first line of research has devised custom approximate designs for typical arithmetic building blocks (e.g., adders, multipliers [2–4, 15]). The second line has targeted approximation of more generic circuits either from gate-level (i.e., Boolean descriptions) [7, 9, 14, 17, 18], higher-level descriptions, such as RTL or behavioral descriptions [13], or even direct C to approximate hardware synthesis [6].

This paper seeks to devise a new direction for approximate boolean-level circuit synthesis. Our inspiration comes from Boolean Matrix Factorization (BMF) that factors a boolean matrix into two boolean matrices [10, 11]. BMF is a derivative of non-negative matrix factorization (NNMF), in which the elements of all input and output matrices are limited to the non-negative space [5]. 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 extends NNMF techniques to Boolean matrices, where matrix operations are carried in $GF(2)$ [10, 11]. The use of BMF as a technique for logic synthesis is a new direction in the field, and we show that it provides a solid foundation for approximate logic synthesis. We summarize our contributions as follows.

- We devise a new methodology for **B**MF-based **L**ogic **A**pproximate **SY**nthesi**S** (BLASYS) that is based on solid mathematical foundations, where Boolean Matrix Factorization (BMF) is used to generate approximate circuits with controllable trade-off between accuracy and circuit complexity.
- We modify existing BMF algorithms to incorporate the ability to work with different quality-of-results (QoR) functions, instead of the standard L_2 norm.
- To scale the factorization method to large circuits, we propose a circuit decomposition method to break down a given circuit into manageable subcircuits with limited number of inputs and outputs. We propose a design-space exploration heuristic to order the subcircuits to identify a good sequence for generating their approximate variants. Our technique results in a very smooth trade-off between accuracy and circuit complexity.
- We implement our approach and test it on a number of application circuits that are typically used in approximate computing domains. We show that our approach is able to trade-off accuracy with circuit area and power consumption as evaluated by an industry-strength synthesis tool. We also evaluate our methodology against a established approach in the literature (e.g., SALSA [18]) and show significant improvements.

The organization of this paper is as follows. First, we overview related work in Section 2. We discuss the details of our proposed

DAC'18, June 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196001

method in Section 3, 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 provide comprehensive results of our method's performance together with a comparison against a previous technique in Section 4. Finally, we summarize our main conclusions and directions for future work in Section 5.

2 PREVIOUS WORK

Recent work on approximate circuit synthesis can be divided into two general categories: Boolean or gate-level approaches and high-level synthesis approaches.

For Boolean and gate-level synthesis, a number of approaches have been proposed [9, 14, 16–18]. In SALSA, a systematic approach for approximate circuit synthesis is proposed [18]. The idea is to create a difference circuit that compares the QoR between the original circuit and the approximated circuit. The don't cares of the outputs of the approximate circuit – which are internal nodes in the difference circuit – with respect to outputs of the difference circuit can be used to simplify the approximate circuit using regular logic synthesis techniques. This approach has been extended in ASLAN [14] to model error arising over multiple cycles. ASLAN also uses a circuit block exploration method that identifies the impact of approximating the combinational blocks and then uses a gradient-descent approach to find good approximations for the entire circuit. In SASIMI [17], a technique is proposed to identify similar signals, such that their values agree over a large number of input test cases, and then substitute one for the other, simplifying the logic. A logic synthesis formulation proposed by Miao *et al.* uses a two-level logic synthesis approach that incorporates constraints on error deviation, and then a heuristic is used to solve the synthesis formulation [9]. Evolutionary techniques have been also explored [16].

For high-level logic synthesis, ABACUS seeks to generate variants of an input high-level Verilog description file by applying a set of possible transformations, such as bit width truncation, operand simplification and variable-to-constant substitution, to generate a set of mutant approximate circuit variants [13]. A multi-objective design space exploration technique is used to identify the best set of approximate variants. Recently, a new technique is proposed to raise the level of abstraction by synthesizing approximate circuit directly from C descriptions [6]. High-level synthesis in conjunction with approximations on the critical path can yield additional savings through voltage scaling [6, 12].

3 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]. The non-negativity constraints on the factorization enables the utilization of the factorization algorithm in physical domains, such as computer vision and document clustering. NNMF essentially compresses the data storage in an approximate way depending on the *factorization degree* (f) [20]. In the mathematical statistics community, the *factorization degree* determines the number of "features"

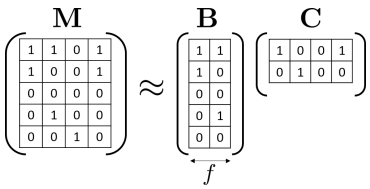

Figure 1: Boolean NNMF example.

that are computed [10]. Therefore, f, clearly represents a trade-off between quality of factorization and storage amount. Recently, 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 (for Boolean semi-ring implementations) and logical XOR (for Boolean field implementations) [10, 11]. Figure 1 provides an example of NNMF over GF(2).

3.1 Circuit Approximation using BMF

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 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 (for Boolean semi-ring implementations) or XOR gates (for Boolean field implementations), to generate the approximate m outputs. More specifically, a 1 in the (i, j) index of the decompressor matrix represents the existence of the f_i intermediate signal in the j-th output, effectively mapping each one to a OR (or XOR) gate. 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 (or XOR) gates. Figure 2 illustrates the proposed approach.

Figure 3 provides an illustrative example of a 4-input, 4-output arbitrary logic circuit. First, we present the original circuit with its truth table, and we synthesize it with Synopsys Design Compiler (DC) using 65 nm library. We then provide approximate variants

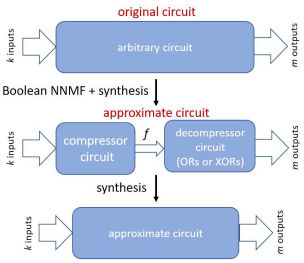

Figure 2: Generating approximate circuits using BMF.

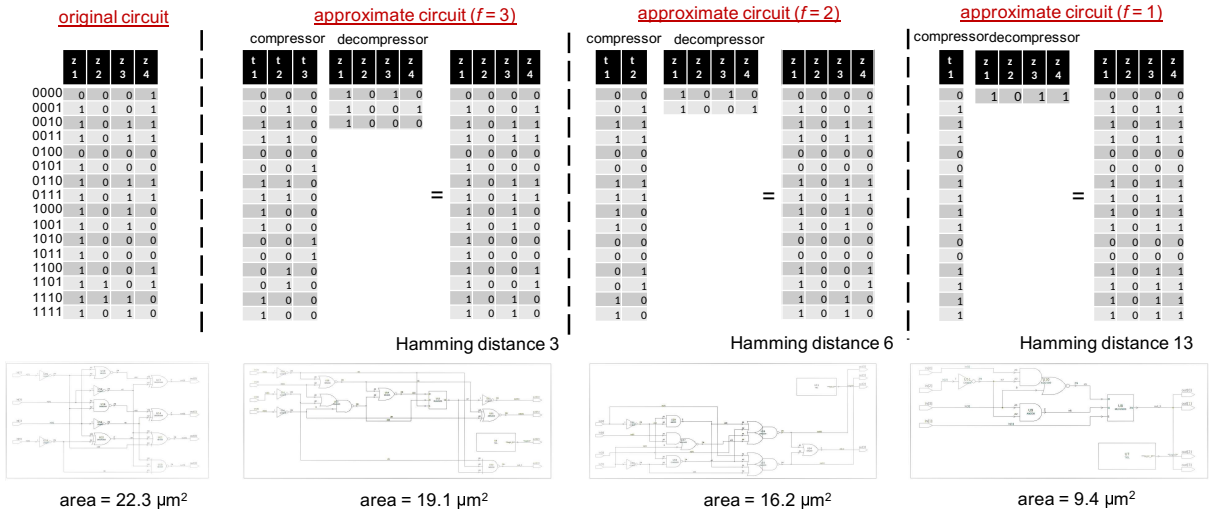

Figure 3: Results of proposed approximation method with various f on a simple circuit for illustration purposes. Circuits are synthesized using Synopsys DC using 65 nm technology library. A semi-ring implementation is used for Boolean NNMF.

for the circuit with $f = 1$, $f = 2$, and $f = 3$. We computed the truth tables for the compressor and decompressor using the ASSO NNMF algorithm [10, 11]. We provide both the quality of results as measured by the Hamming distance between the truth table of the original circuit and the approximate circuit as well as the design area reported by DC. For instance, when $f = 3$, we reduce the area of the circuit by 14.3%, while compromising the quality of results (QoR) by only 4.6% since the Hamming distance between the original and the approximate truth tables is equal to 3; that is out of the 64 entries in the truth table, only 3 entries flipped in the approximate circuit. With $f = 2$ and $f = 1$, we can reduce the area by 27.3% and 57.8% while compromising the QoR by 9.3% and 20.3% respectively.

Our approach leads to a new paradigm for creating approximate logic circuits in a controlled fashion that are based on solid mathematical foundations. There are two main challenges:

(1) NNMF algorithms use the L_2 norm to measure the quality of factorization. For Boolean matrices, L_2 translates to Hamming distance. In addition, we need to identify methods to factorize for other QoR metrics that are relevant for approximate applications.
(2) The basic idea is limited in scalability since the complexity of generating the truth table grows exponentially as a function of the number of circuit's inputs. Thus, we need to create factorization methods that can scale up for large circuits.

3.2 Factorization for Arbitrary QoR

The goal of the BMF algorithm is to minimize $||\mathbf{M} - \mathbf{BC}||_2$, which translates to Hamming distance in $GF(2)$. However, not all applications or circuits necessarily use this metric to assess QoR. For instance, in the case of circuit design, if an m bit signal is to be interpreted as an m bit number, Hamming distance is not really an accurate representation of the inaccuracies as mismatches in different bit indices contribute differently to the actual error.

To take into account the non-uniform nature of bit significance, we propose to modify the NNMF algorithms in the literature to account for bit indices. More specifically, instead of minimizing $||\mathbf{M} - \mathbf{BC}||_2$, we propose minimizing $||(\mathbf{M} - \mathbf{BC})\mathbf{w}||_2$, where $\mathbf{w}$ is a constant weight vector. For example, if numerical difference is the target QoR, then the $\mathbf{w}$ vector will be based on powers-of-two (e.g., 8, 4, 2, 1) therefore reflecting the fact that different bit positions lead to different numerical weights. In this work, we modify the ASSO [11] algorithm as such to penalize mismatches on the higher bit locations more than on the less significant bits. In Section 4.1, we demonstrate how such weighting scheme can improve the results compared to the uniformly weighted standard BMF algorithms.

3.3 Scaling Up for Large Circuits

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, 10, 11]. We propose a simple approach to scale BMF calculations for larger circuits. The basic idea is to decompose a large circuit into a number of subcircuits each with a maximum of k inputs and m outputs as afforded by the runtime of the factorization algorithm and memory requirements. Note that this approach is reminiscent *but yet fundamentally different* than FPGA mapping algorithms, where the goal is to map a circuit into logic elements, each with limited number of inputs [1]. Our motivation for decomposition is different because (1) we are mapping to subcircuits purely to address computational complexity, and (2) we apply the BMF on the truth tables of the subcircuits, and then we synthesize the factored circuits into *any* target ASIC or FPGA technology. Instead of using classical k-cut algorithms, e.g. [1], we propose to use $k \times m$-cut algorithms (e.g., KL algorithm [8]) to identify subcircuits with a maximum input of k and maximum output of m. Note that k and m are design choices mostly determined by the runtime and memory budgets.

Algorithm 1: BLASYS: Boolean Level Approximate Circuit Synthesis

```
   Input  : Accurate Circuit ACir, Error Threshold
   Output : Approximate Circuit Cir
1  subcircuits=Decompose input circuit using k × m decomposition
2  // Factorization profiling Phase
3  for each subcircuit s_i with m_i ≤ m outputs do
4      M=Construct truth table of s_i
5      // profile for every possible factorization degree
6      for f=1 to m_i-1 do
7          [B, C] = BMF(M, f)
8          T_{s_i,f}=Construct truth table of BC
9      end
10 end
11 // Circuit Space Exploration Phase
12 Cir=ACir;
13 Let f_i = m_i for all subcircuits s_i
14 while QoR(Cir) < threshold do
15     for each subcircuit s_i with f_i > 1 do
16         Cir'=Cir(s_i → T_{s_i,f_i-1})
17         Δerr_i = QoR(Cir') − QoR(Cir)
18     end
19     b = arg min_i(Δerr_i)
20     Cir = Cir(s_b → T_{s_b,f_b-1})
21     f_b = f_b − 1
22 end
23 Cir=Synthesize Best new Design
24 return Cir
```

Decomposing a large circuit into subcircuits of size $k \times m$ requires changing the way we evaluate the QoR. In particular one cannot evaluate the QoR of a subcircuit in isolation from the rest of the circuit, since a small error in the output of the subcircuit can propagate leading to larger errors. Thus, instead of evaluating the QoR of an original subcircuit against its approximate version, we have to evaluate QoR of the entire circuit $Cir(s_i \rightarrow T_{s_i,f_i})$, where $Cir(s_i \rightarrow T_{s_i,f_i})$ represents the approximate circuit created by substituting an accurate subcircuit, s_i, with its approximate version, T_{s_i,f_i}, using a f_i factorization degree. As evaluating the entire circuit for all possible inputs is infeasible, we use Monte Carlo sampling to estimate the QoR of the approximate version of the entire circuit.

In addition, the order of processing the subcircuits and the target factorization degree for each subcircuit is an important consideration. We devise Algorithm 1 to gradually approximate the circuit as guided by circuit accuracy. After identifying the subcircuits (Line 1), the first stage of the algorithm (lines 3-10) calculates the *potential* approximate versions for each subcircuit under various factorization degrees. The next stage (lines 12-22) seeks to explore the space of potential approximate subcircuits to identify a good approximation order. Lines 15-18 assess the reduction in accuracy of the entire circuit if the degree of factorization of each subcircuit is decremented. The subcircuit that leads to the smallest error is then chosen (line 19), and its more approximated version is then substituted in the main circuit (lines 20-21). The process is then repeated until the error is above the set threshold or all subcircuits are approximated to the highest degree possible.

4 EXPERIMENTAL RESULTS

In this section we evaluate our proposed BMF based approximation methodology. Similar to previous work [14, 18], we consider a number of arithmetic circuits (adder and multiplier) and a number of application circuits that are amenable for approximate computing such as a multiply-accumulate circuit (MAC), a butterfly network (BUT), a sum of absolute differences (SAD) circuit and finite impulse response (FIR) circuit. Table 1 summarizes the characteristics of the evaluated applications. Here we also give the number of input and output pins, and the design metrics of the accurate design. To evaluate design area and power consumption, we use Synopsys design compiler with an industrial 65 nm technology library in typical processing corner.

Table 1: The list of benchmarks evaluated using the proposed NNMF methodology.

Name	Function	I/O	Accurate Design Metrics		
			Area (um^2)	Power (uW)	Delay (ns)
Adder32	32-bit Adder	64/33	320.8	81.1	3.23
Mult8	8-bit Multiplier	16/16	1731.6	263.5	2.03
BUT	Butterfly Structure	16/18	297.4	80.6	1.79
MAC	Multiply and Accumulate with 32-bit Accumulator	48/33	6013.1	470.5	2.36
SAD	sum of absolute difference	48/33	1446.5	195.1	2.43
FIR	4-Tap FIR Filter	64/16	8568.0	466.3	1.56

For all our experiments, as discussed in 3.3, we first decompose each circuit to $k \times m$-cut subcircuits and then perform factorization. In our experiments we chose both $k = 10$ and $m = 10$. These numbers are simply chosen as they provide a balanced trade-off between truth table complexity and number of subcircuits. We use the modified ASSO algorithm for BMF [10, 11]. Further, for each subcircuit we perform a sweep on the factorization threshold in order to get the best accuracy. In order to evaluate the accuracy on the evaluated applications, we use a Monte Carlo simulation using one million randomly generated input test cases. We define average relative error as,

$$\text{Average Relative Error} = \frac{1}{N}\sum_{i=1}^{N}\frac{|R_i - R_i'|}{R_i}, \tag{1}$$

and average absolute error as,

$$\text{Average Absolute Error} = \frac{1}{N}\sum_{i=1}^{N}|R_i - R_i'|, \tag{2}$$

where N is the size of the test case, and R and R' are accurate and approximate results respectively.

Next, in the first subsection we show the impact of enabling arbitrary QoR functions, when compared to standard L_2 metric used in Boolean matrix factorization. In the second subsection, we show the trade-offs and Pareto Frontiers offered by our methodology for our applications. We also compare the results of our work to previous work.

4.1 Evaluation of QoR Impact

As described in Section 3, we modify the Boolean NNMF factorization algorithm, ASSO in this case, to enable weighted cost functions, where a bit error on higher bit indices results in a higher penalty compared to disparities on the lower significance bits.

Figure 4 shows the accuracy vs. design area trade-offs offered for the approximate Mult8 design when comparing a factorization algorithm using standard L_2 QoR with uniform bit weighting against

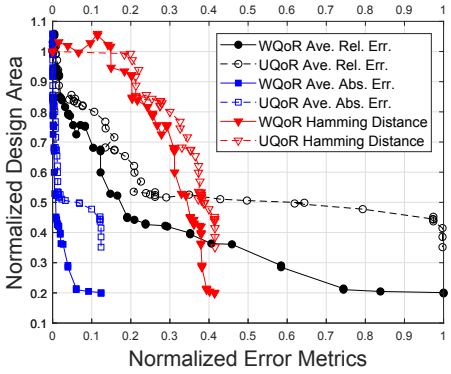

Figure 4: Comparison of the trade-offs offered using the proposed weighted QoR vs. the original factorization algorithm.

the proposed weighted QoR. We provide the trends in average relative error, normalized average absolute error, and the normalized Hamming distance. The results obtained from the weighted QoR (WQoR) are shown with solid lines while the dashed lines show the results for the original uniform algorithm (UQoR).

As shown in the figure, compared to the original algorithm, the weighted scheme provides consistent benefits in accuracy for the same design complexity for all three accuracy metrics. This result confirms the benefit of modifying the BMF algorithm to differentiate among inaccuracies in different indices. Furthermore, this figure highlights the necessity of an algorithm guiding the approximation process in the right direction as suboptimal points are commonly encountered. Next, we evaluate the trade-offs offered for all of our application circuits using our heuristic design space exploration and compare our results against SALSA [18].

4.2 Application Results

As previously described, for each application, first the circuit is decomposed into subcircuits with reduced number of inputs and outputs. Then, for each subcircuit and various values of f, each subcircuit is approximated and the approximate characteristics are stored. Next, the heuristic proposed in Algorithm 1 iteratively approximates the subcircuits while assessing the impact on the whole circuit.

Figure 5 shows the trade-offs offered by BLASYS for each of our six benchmarks. In our experiments as the inner workings of accuracy among different blocks is more difficult to model, we simulate the whole circuit while modeling the design metric. More specifically, for design space exploration purposes we assume the design metric, e.g. design area or power, of the large circuit is the sum of design metrics of the $k \times m$-cut subcircuits. For our experiments in order to simplify our design metric model, we use design area as it has less variation compared to power consumption when assembling the subcircuits into the larger circuit. Furthermore, our design area model is only a function of the subcircuits blocks being approximated, while registers and control paths are not considered. We plot the normalized combinational design area utilization as a function of average relative error (black plot and using the bottom x axis) and average normalized absolute error (red plot and using the top x axis). In the case of average absolute error, we normalize the values to the highest output possible. Further, to better show the trend, the average absolute error is plotted in log scale.

As shown in the figure, the proposed methodology enables the designer to choose among a wide range of fine-grain trade-offs. Intuitively, our design space exploration heuristic aims to find the lowest error possible for a specific degree of approximation where the degree of approximation is incremented by one in each generation. This insight explains the smooth trend of trade-offs for larger circuits while the smaller circuits can change in performance significantly in one iteration. Furthermore, note that while reducing the number of intermediate signals (f) generally reduces the complexity of the circuit, there are cases where the number of literals in the logic representation for one output can increase. This phenomenon, explains the temporary increases in design area observed in some of the trends.

The overall runtime of the algorithm is dominated by the accuracy simulation of the intermediate points. Therefore, the runtime is dictated by the Monte Carlo sample size, the threshold set for accuracy, and the tool chain utilized. For example, in our experiments and in the case of the Adder32, the simulation takes about 11 Seconds (using 1 million samples) for each design point, while the BMF algorithm for all the subcircuits takes 0.35 Seconds.

Table 2 summarizes all the design metrics of our 6 testcases and for two accuracy thresholds as synthesized at the end of the design space exploration. As shown in the table, significant reductions in design metrics are possible while insignificant errors are introduced to the circuit. Based on the application, benefits of approximately 8%-47% can be achieved for average relative errors of 5%.

We also compare our proposed methodology against the previous work SALSA [18]. Table 3 compares the results obtained using BLASYS against SALSA for given thresholds of 5% and 25%. As it can be seen from the table, in all cases, BLASYS delivers significant improvements in design area. We attribute the benefits to BLASYS' ability to approximate multiple outputs, up to m outputs, simultaneously, whereas SALSA approximates each output bit individually.

Table 2: The hardware characteristics of the approximate testcases for an accuracy threshold of 5%.

Design	Area Savings (%)	Power Savings (%)	Delay Reduction (%)
Adder32	44.78	63.79	12.07
Mult8	28.77	26.87	12.32
BUT	7.87	11.25	2.23
MAC	47.55	55.58	64.41
SAD	32.80	41.47	69.14
FIR	19.52	22.26	12.18

Table 3: The design area savings at error thresholds 5% and 25% for the applications evaluated with comparison to SALSA [18].

	Threshold 5% Area Savings (%)		Threshold 25% Area Savings (%)	
	BLASYS	SALSA	BLASYS	SALSA
Adder32	44.9	20.5	48.2	23.2
Mult8	28.8	1.8	63.2	8.9
BUT	7.9	5.0	26.4	24.7
MAC	47.6	1.7	65.9	8.2
SAD	32.8	3.3	38.1	15.8
FIR	19.5	3.2	34.0	15.8

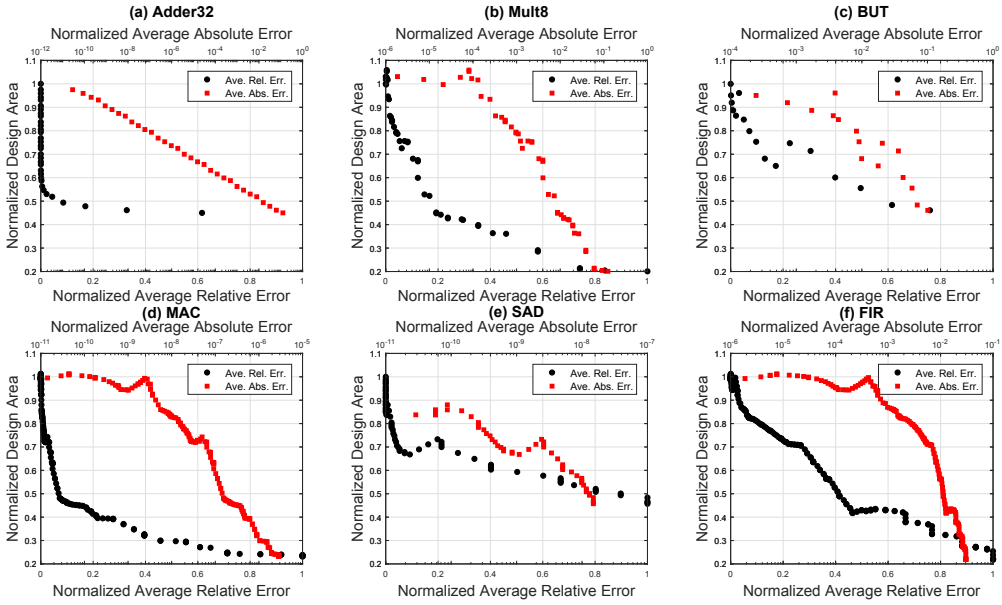

Figure 5: The trade-offs offered for each application. (a) Adder32, (b) Mult8, (c) BUT, (d) MAC, (e) SAD, and (f) FIR.

5 CONCLUSIONS

In this paper we proposed a new direction for approximate logic synthesis using boolean matrix factorization. Our proposed methodology, BLASYS, leads to a systematic approach to trade-off accuracy with circuit complexity. To scale our approach into large circuits, we proposed a circuit decomposition heuristic together with a processing order for the subcircuits. Our algorithm results in a very smooth way to trade-off the complexity of entire large circuits with accuracy. We also investigated ways to incorporate different QoR metrics into the circuit factorization algorithm. Our experimental results show solid improvements over state-of-the-art techniques.

The proposed approach opens many doors for investigation in logic synthesis. Future work include improved techniques for BMF including literal aware approximations, direct incorporation of the QoR metric into the numerical optimization of the factorization algorithm, improved $k \times m$ circuit decomposition, and improved design space heuristics for fewer design point evaluations. Finally, we have integrated our proposed methodology into the open-source YOSYS synthesis framework [19] available on Github [1].

Acknowledgments: This work is partially supported by NSF grant 1420864.

REFERENCES

[1] Jason Cong and Yuzheng Ding. 1994. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. on CAD of Integrated Circuits and Systems* 13 (1994), 1–12.

[2] Soheil Hashemi, R. Iris Bahar, and Sherief Reda. 2015. DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '15)*. IEEE Press, Piscataway, NJ, USA, 418–425.

[3] Mohsen Imani, Daniel Peroni, and Tajana Rosing. 2017. CFPU: Configurable Floating Point Multiplier for Energy-Efficient Computing. In *Proceedings of the 54th Annual Design Automation Conference 2017 (DAC '17)*. ACM, New York, NY, USA, Article 76, 6 pages.

[4] A. B. Kahng and S. Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *DAC Design Automation Conference 2012*. 820–825.

[5] D. D. Lee and H. S. Seung. 1999. Learning the Parts of Objects by non-negative matrix factorization. *Nature* 401 (1999), 788–791.

[6] S. Lee, L. K. John, and A. Gerstaluer. 2017. High-Level Synthesis of Approximate Hardware under Joint Precision and Voltage Scaling. In *Design, Automation and Test in Europe*.

[7] C. Li, W. Luo, Sachin S. Sapatnekar, and Jiang Hu. 2015. Joint precision optimization and high level synthesis for approximate computing. In *Design Automation Conference*. 104:1–6.

[8] Osvaldo Martinello, Renato Perez Ribas Felipe Marque, and Andr?? Reis. 2010. KL-Cuts: A New Approach for Logic Synthesis Targeting Multiple Output Blocks. In *Design Automation Test in Europe*. 777–782.

[9] Jin Miao, Andreas Gerstlauer, and Michael Orshansky. 2013. Approximate Logic Synthesis Under General Error Magnitude and Frequency Constraints. In *Proceedings of the International Conference on Computer-Aided Design*. 779–786.

[10] P. Miettinen and J. Vreeken. 2011. Model order selection for boolean matrix factorization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 51–59.

[11] P. Miettinen and J. Vreeken. 2014. MDL4BMF: Minimum Description Length for Boolean Matrix Factorization. *ACM Transactions on Knowledge Discovery from Data* 8, 4 (2014), 18:1–31.

[12] K. Nepal, S. Hashemi, C. Tann, R. I. Bahar, and S. Reda. 2016. Automated High-Level Generation of Low-Power Approximate Computing Circuits. In *IEEE Transactions on Emerging Topics in Computing*.

[13] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. 2014. " ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits. In *Design, Automation and Test in Europe*. 1–6.

[14] Ashish Ranjan, Arnab Raha, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. 2014. ASLAN: Synthesis of Approximate Sequential Circuits. In *Design, Automation & Test in Europe Conference*. 1–6.

[15] Semeen Rehman, Walaa El-Harouni, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2016. Architectural-space Exploration of Approximate Multipliers. In *Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD '16)*. ACM, New York, NY, USA, Article 80, 8 pages. https://doi.org/10.1145/2966986.2967005

[16] Zdenek Vasicek and Lukas Sekanina. 2016. Evolutionary design of complex approximate combinational circuits. *Genetic Programming and Evolvable Machines* 17, 2 (01 Jun 2016), 169–192.

[17] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. 2013. Substitute-and-simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits. In *Design, Automation and Test in Europe*. 1367–1372.

[18] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. 2012. SALSA: Systematic logic synthesis of approximate circuits. In *DAC Design Automation Conference 2012*. 796–801.

[19] C. Wolf. [n. d.]. Yosys Open Synthesis Suit. http://www.clifford.at/yosys/.

[20] W. Xu, X. Liu, and Y. Gong. 2003. Document clustering based on non-negative matrix factorization. In *ACM SIGIR conference on Research and development in informaion retrieval*. 267–273.

[1] https://github.com/scale-lab/blasys