

# Systematic Approaches to Approximate Logic Synthesis

Quo Vadis, Logic Synthesis Workshop 2019

---

SHERIEF REDA  
BROWN UNIVERSITY



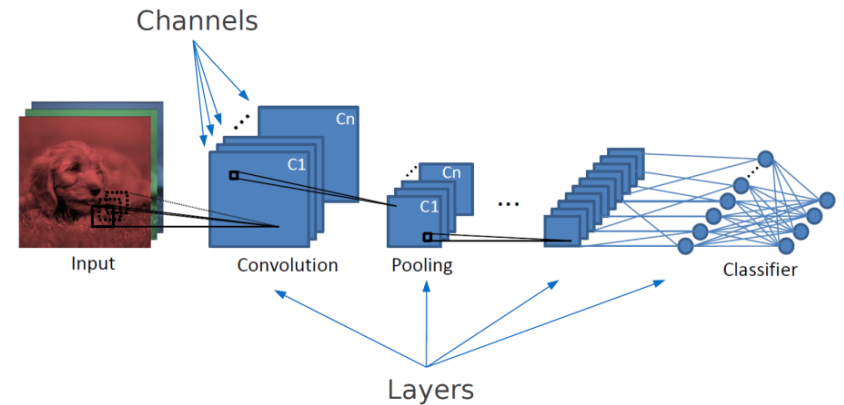
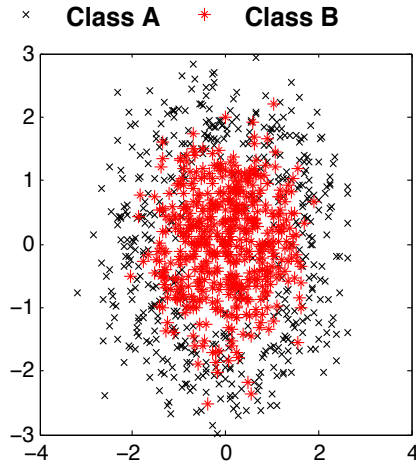
MARCH 2019

Based on

- S. Hashemi, H. Tann and S. Reda, "BLASYS: Approximate Logic Circuit Synthesis Using Boolean Matrix Factorization," DAC, 2018.
- S. Hashemi and S. Reda, "Generalized Matrix Factorization Techniques for Approximate Logic Synthesis," to DATE, 2019.

Ack: NSF grant #1814920

# The Case for Approximate Computing



Many algorithms and circuit implementations in machine learning, signal processing, computer vision and cognitive computing have inherent error resiliency:

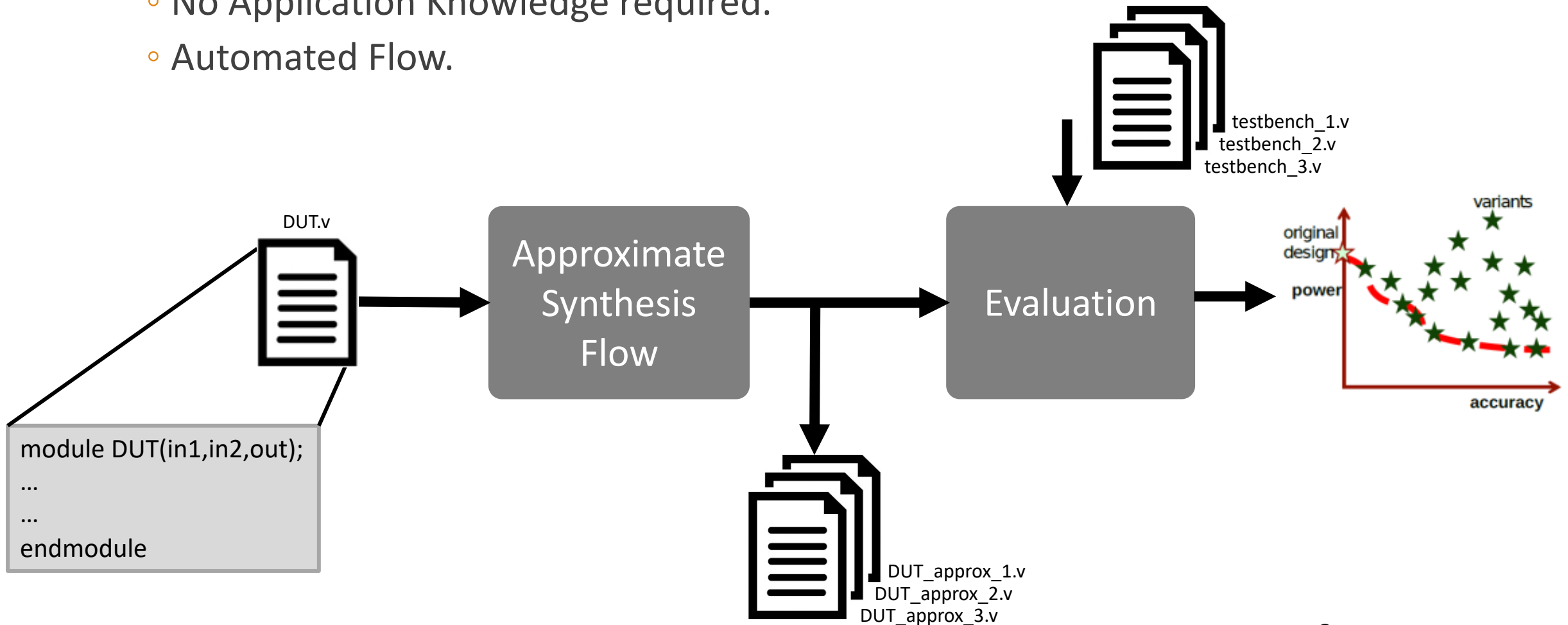
- Noisy inputs
- Approximate algorithms
- Loose constraint on output.

Leverage this tolerance to trade-off accuracy for hardware resources (e.g. Design area, power consumption, and delay).

# Goal of Approximate Synthesis

Generate Approximate Variants from any Arbitrary input circuit.

- No Application Knowledge required.
- Automated Flow.



# Boolean Matrix Factorization (BMF)

Special case of Non-negative matrix factorization (NNMF).

Factorizes a matrix to two smaller matrices such that:

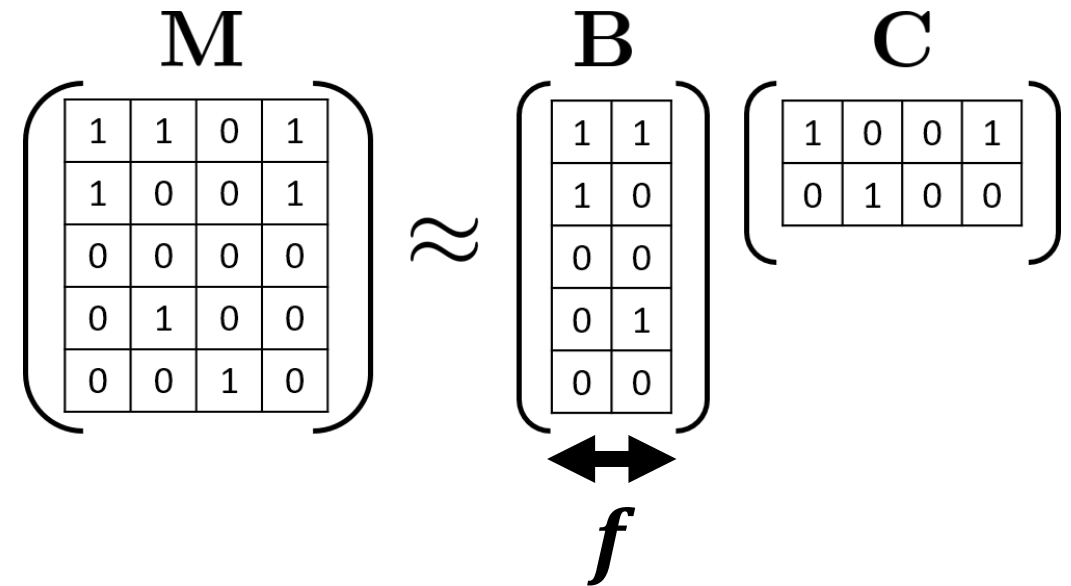
$$M \approx B \times C$$

$$\forall i, j \quad m_{i,j}, b_{i,j}, c_{i,j} \in \{0,1\}$$

*Boolean Arithmetic* (Multiply  $\rightarrow$  AND, and Addition  $\rightarrow$  OR)

Applications in:

- Low Dimension Data Representation
- Document Classification
- Language Processing



**Factorization Degree ( $f$ )  
determines the degree of  
approximations.**

# How does Boolean matrix factorization work?

- BMF is NP-Hard.
- Solutions are based on heuristics.
- Heuristics solving:  $\operatorname{argmin}_{B,C} |M \otimes (B \times C)|$

$$\begin{array}{c} \mathbf{M} \\ \left( \begin{array}{|c|c|c|c|} \hline 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline \end{array} \right) \approx \begin{array}{c} \mathbf{B} \\ \left( \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 0 \\ \hline 0 & 0 \\ \hline 0 & 1 \\ \hline 0 & 0 \\ \hline \end{array} \right) \end{array} \begin{array}{c} \mathbf{C} \\ \left( \begin{array}{|c|c|c|c|} \hline 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 \\ \hline \end{array} \right) \end{array}$$

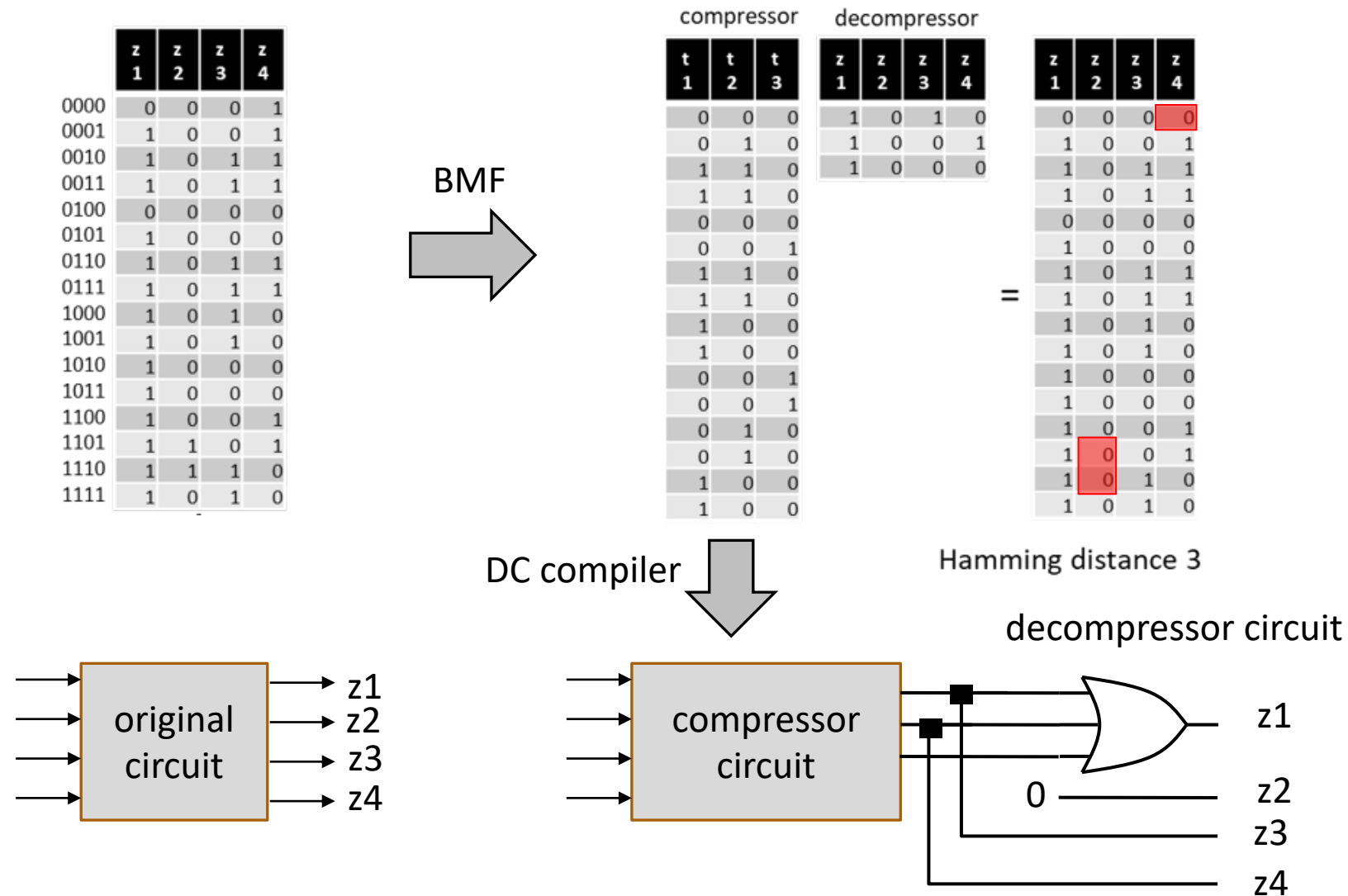
$\underbrace{\hspace{1.5cm}}_f$

0. Initialize  $B$  randomly

Iterate until converge:

1. Fix  $B$ , Find  $C$  such that  $\operatorname{argmin}_C |M \otimes (B \times \textcolor{red}{C})|$
2. Fix  $C$ , Find  $B$  such that  $\operatorname{argmin}_B |M \otimes (\textcolor{red}{B} \times C)|$

# Proposed approximate synthesis using BMF

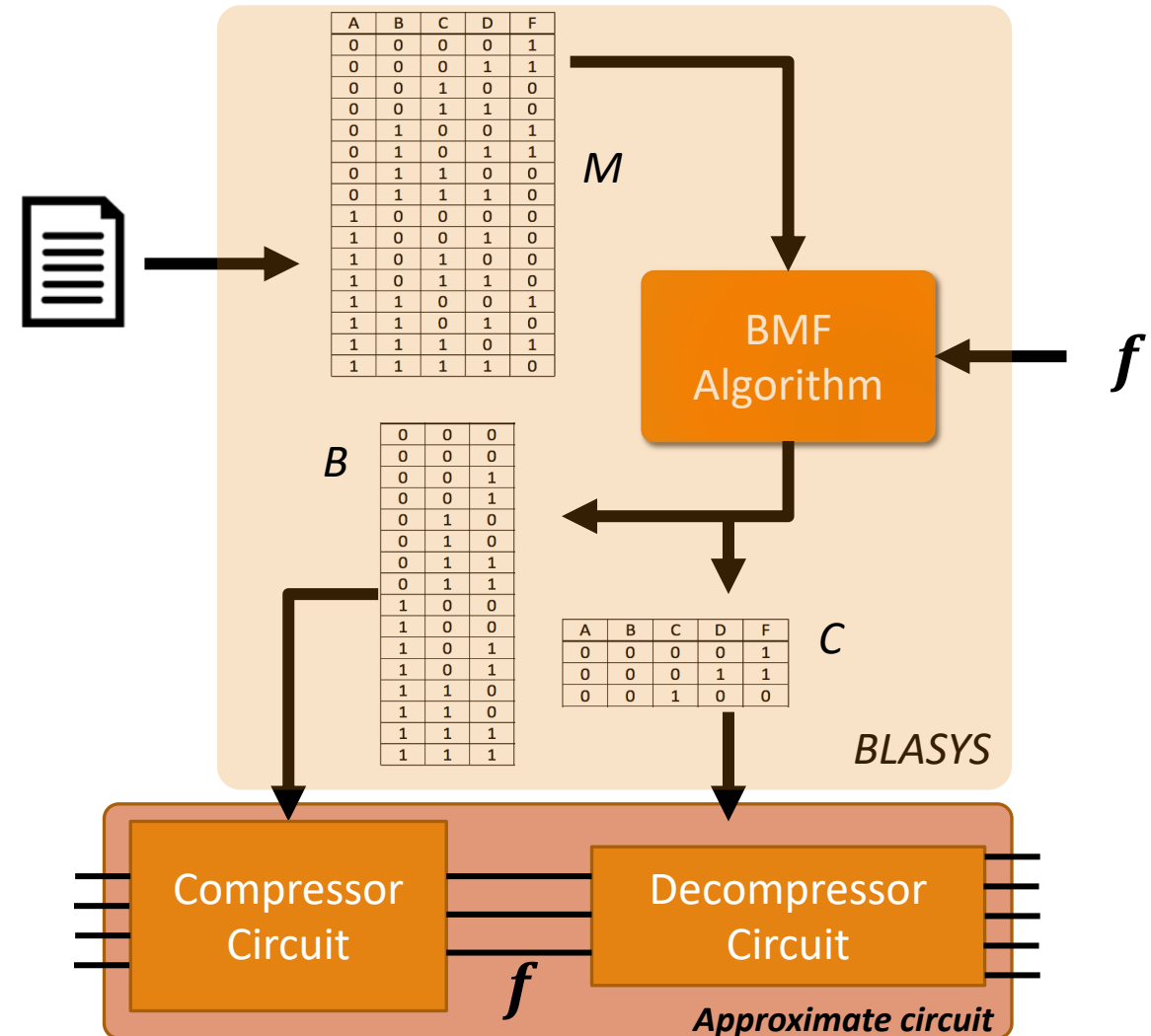


# Proposed Approximate Logic Synthesis Using BMF

Utilize BMF for generic approximate circuit synthesis.

Approximate Synthesis Flow:

1. The truth table of the arbitrary input circuit is generated.
2. The truth table is passed to a BMF algorithm to generate two factorized matrices.
3. Two factorized matrices are mapped to hardware:
  - Compressor Circuit is the truth table of a simpler logic with fewer outputs.
  - Decompressor Circuit is a simple OR gate network.

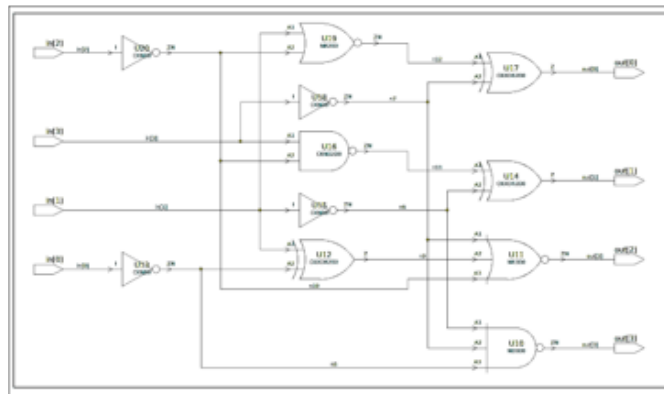


# An Example

original circuit

	z 1	z 2	z 3	z 4
0000	0	0	0	1
0001	1	0	0	1
0010	1	0	1	1
0011	1	0	1	1
0100	0	0	0	0
0101	1	0	0	0
0110	1	0	1	1
0111	1	0	1	1
1000	1	0	1	0
1001	1	0	1	0
1010	1	0	0	0
1011	1	0	0	0
1100	1	0	0	1
1101	1	1	0	1
1110	1	1	1	0
1111	1	0	1	0

area =  $22.3 \mu\text{m}^2$



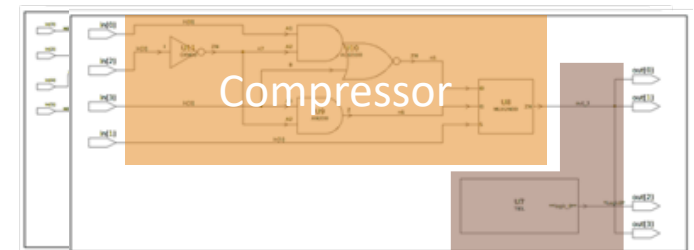
Approximation

$f = 1$

compressor decompressor

t 1	t 2	t 1	z 1	z 2	z 3	z 4
0	0	0	1	0	1	1
0	1	1	1	0	0	1
1	1	1	1	0	0	0
1	1	1				
0	0	0				
0	0	0				
1	1	1				
1	1	1				
1	0	1				
1	0	1				
0	0	0				
0	0	0				
0	1	1				
0	1	1				
1	0	1				
1	0	1				

Hamming distance 8



area =  $19.4 \mu\text{m}^2$



# Towards an effective scalable scheme

1. Field and semi-ring implementations.

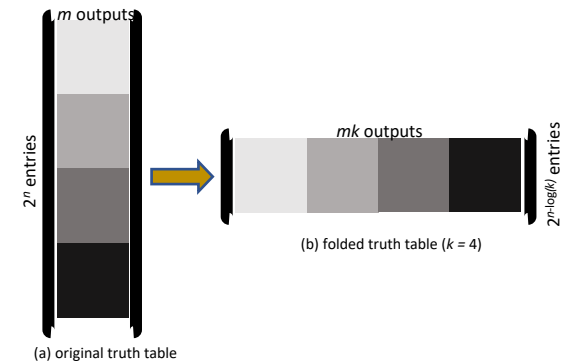
$$\begin{array}{ccc}
 \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} & \begin{array}{c} \text{||} \\ \text{||} \\ \text{||} \\ \text{||} \\ \text{||} \\ \text{||} \\ \text{||} \\ \text{||} \end{array} & \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix} \\
 \text{(a) input matrix} & & \text{(b) factorization using semi-ring Boolean algebra}
 \end{array}$$

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

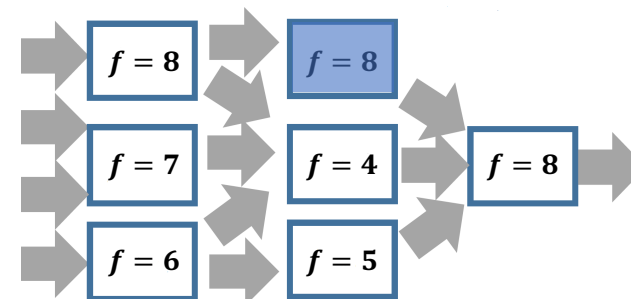
(c) factorization using field modulo-2 algebra

2. Truth table folding.

3. Arbitrary QoR.



4. Scalability via circuit breakdown and design space exploration.



# 1. GF(2) Semi-ring and field implementations

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

(a) input matrix

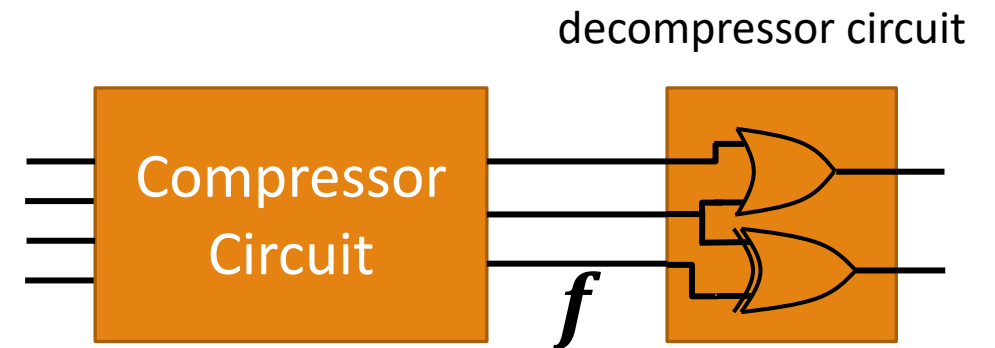
$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ \color{red}{0} & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ \color{red}{1} & 0 & 1 & 1 & 1 \end{pmatrix}$$

(b) factorization using semi-ring Boolean algebra

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & \color{red}{0} \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

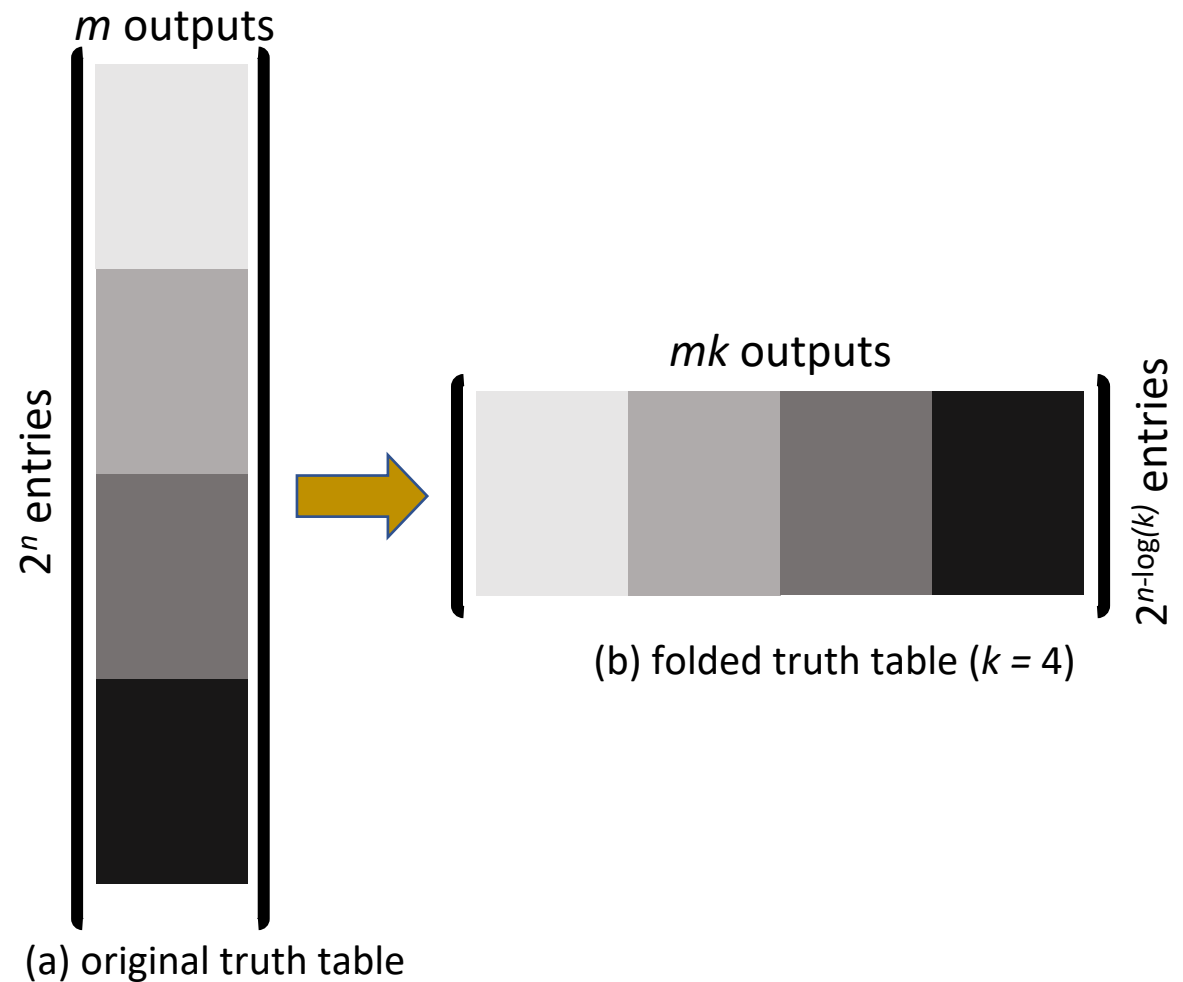
(c) factorization using field modulo-2 algebra

Outputs in the decompressor circuits do not need to be all implemented using XORs or OR, every output could be implemented independently by XORing or Oring the factors from the decompressor circuit that reduces that minimizes the error.

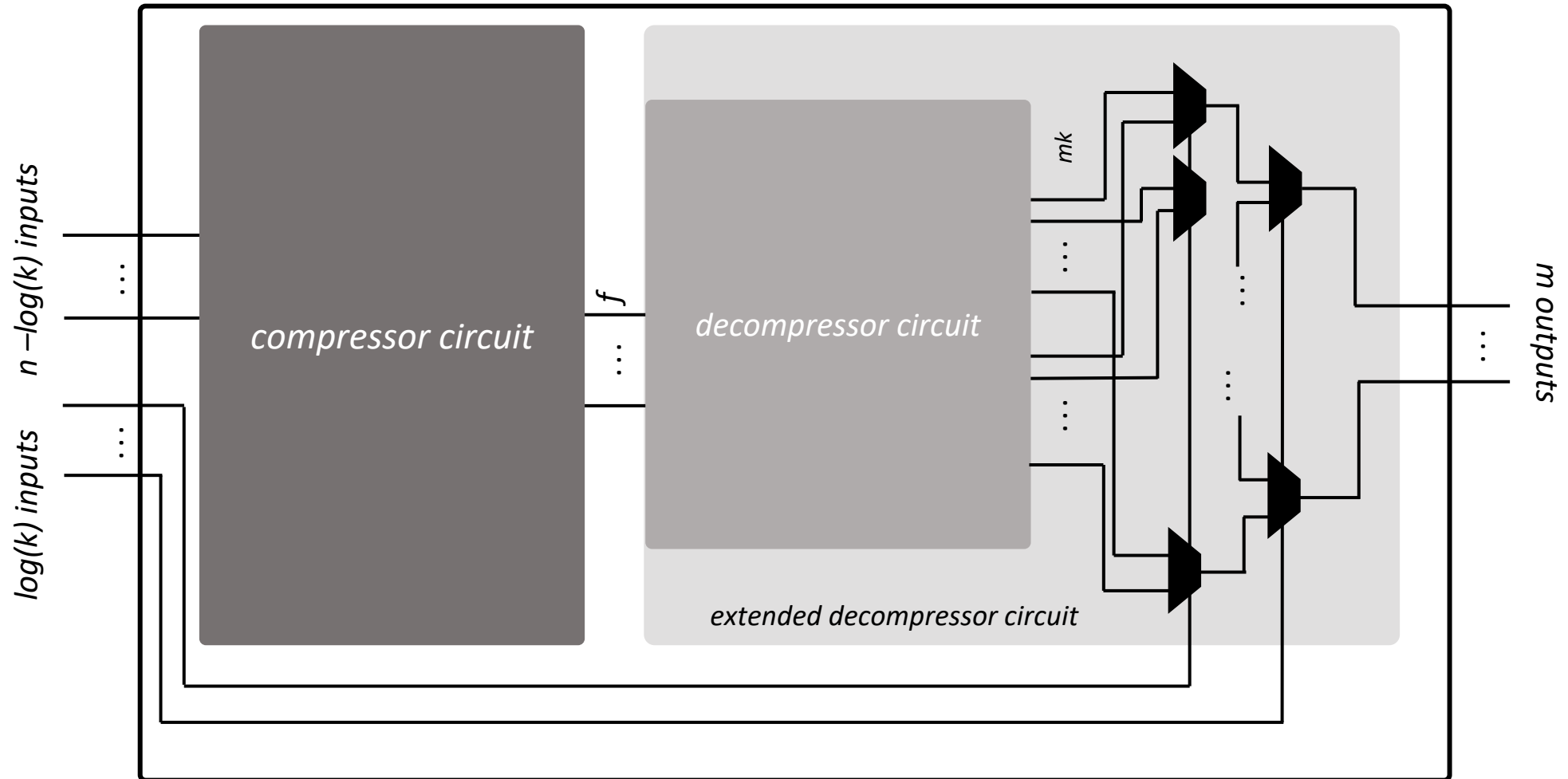


## 2. Truth table folding

- The number of rows in the truth table of a circuit grows exponentially as a function of the number of circuit inputs  $\rightarrow$  tall-and-skinny matrix, with possibly lack of common bases between different outputs.
- Fold a tall-and-skinny input matrix. By dividing the number of rows by half, by dividing the input matrix into two equal sub-matrices and concatenating the two sub-matrices in a column-wise fashion.



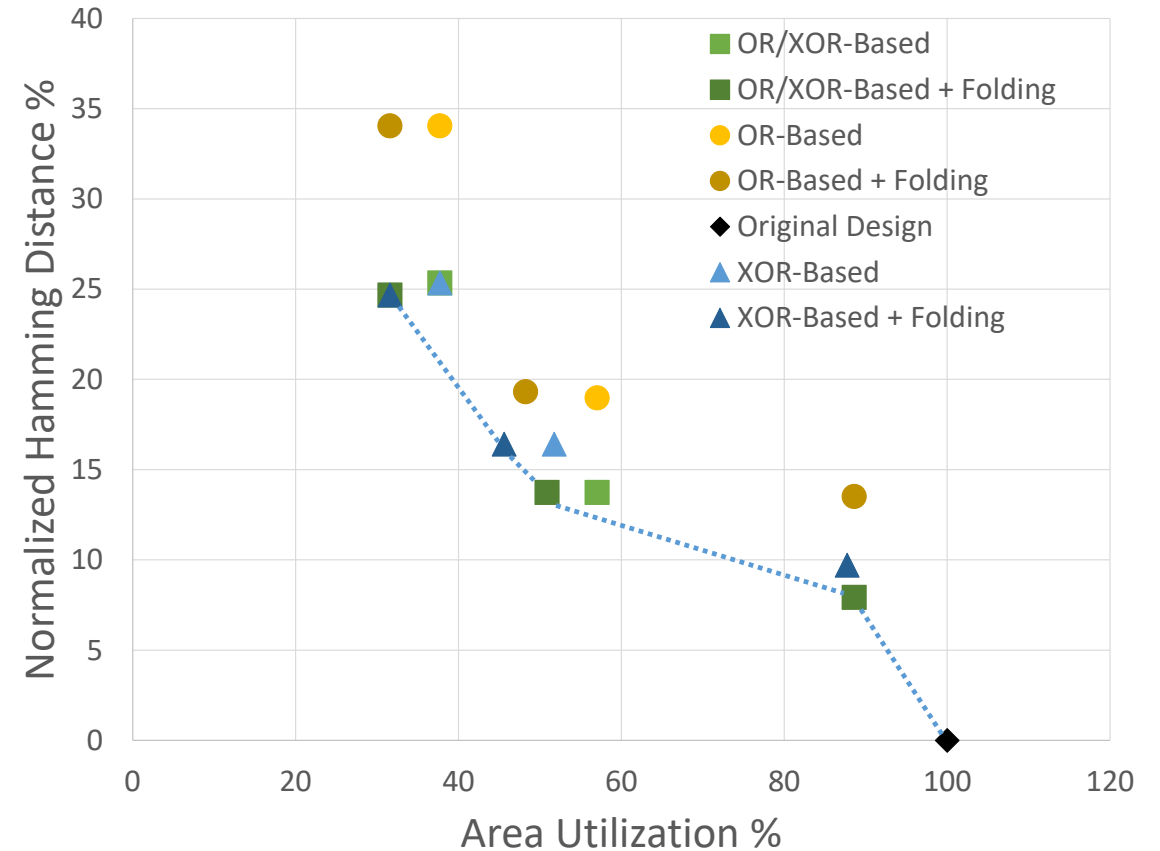
## 2. Modified compressor-decompressor circuit



# Impact of various approximation choices

Circuit: x2 – 10 inputs/7 outputs →  
truth table enumerated and error  
metric is Hamming distance between  
original circuit and approximate circuit

- OR decompressor
- XOR decompressor
- OR/XOR decompressor
- with and without Folding
- Factorization degree
- 12% reduction in area with 8% inaccuracy
- 50% reduction in area with 13% inaccuracy



### 3. Arbitrary QoR

- Many approximate circuits have outputs that should be interpreted numerically.
- Heuristics solving:

$$\operatorname{argmin}_{B,C} |M \otimes (B \times C)|$$

- Therefore uniform cost function.

- Modify the algorithms to have a exponentially weighted cost function → Output bits at higher indices have higher weights

$$\operatorname{argmin}_{B,C} |(M \otimes (B \times C))w|$$

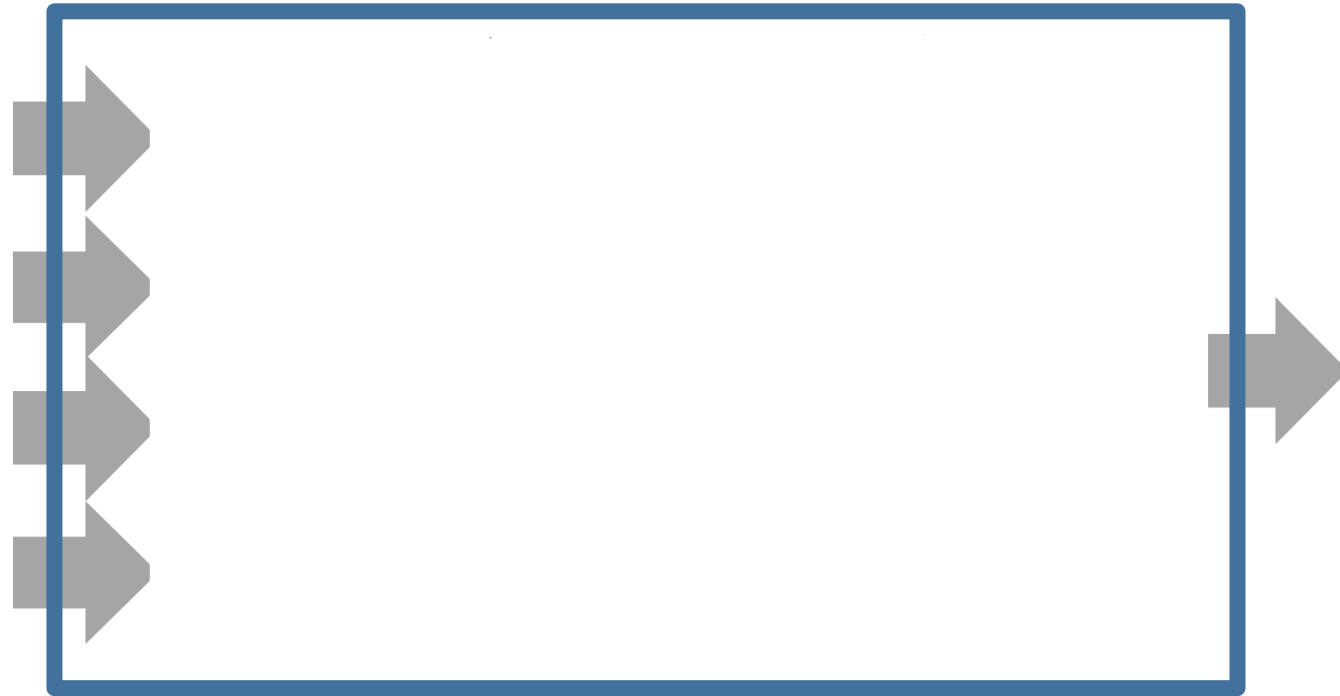
$$\begin{matrix} \mathbf{M} \\ \left( \begin{array}{|c|c|c|c|} \hline 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline \end{array} \right) \end{matrix} \approx \begin{matrix} \mathbf{B} \\ \left( \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 0 \\ \hline 0 & 0 \\ \hline 0 & 1 \\ \hline 0 & 0 \\ \hline \end{array} \right) \end{matrix} \begin{matrix} \mathbf{C} \\ \left( \begin{array}{|c|c|c|c|} \hline 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 \\ \hline \end{array} \right) \end{matrix}$$

$\xleftrightarrow{f}$

## 4. Circuit Breakdown

---

The size of the truth table grows exponentially as a factor of number of inputs → decompose the circuit into subcircuits with max input size.



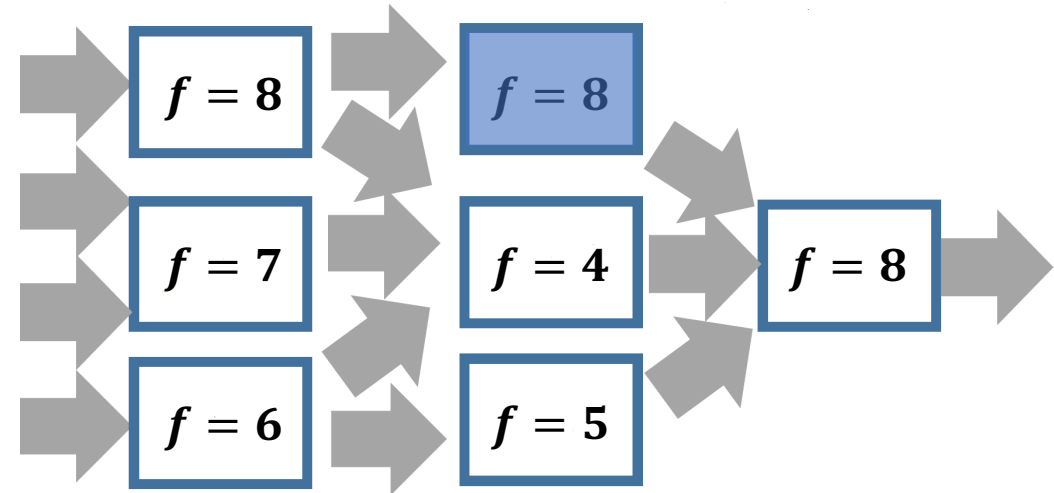
**Design Space Exploration Needed.**

## 4. Design Space Exploration Heuristic

---

A local search based heuristic:

- Start with all subcircuits at highest accuracy.
- One by one add 1 degree of approximation to each subcircuit.
- Select the best variant as the parent for the next generation.
- Repeat until the accuracy is lower than threshold.





# Benchmarks

---

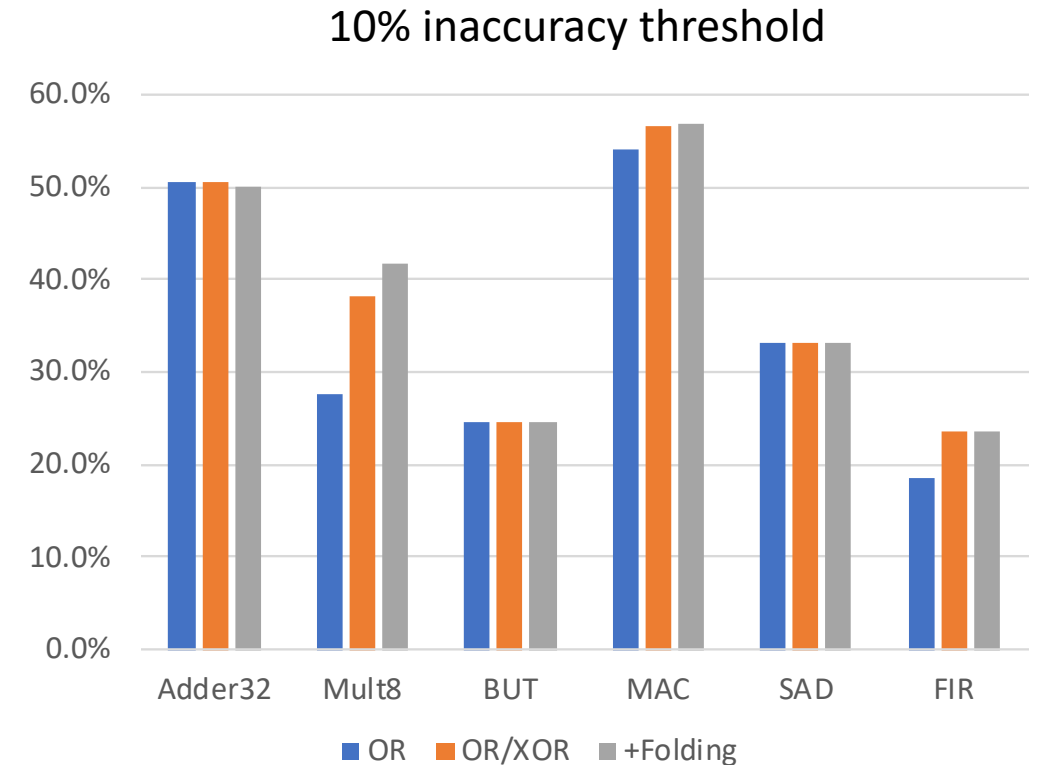
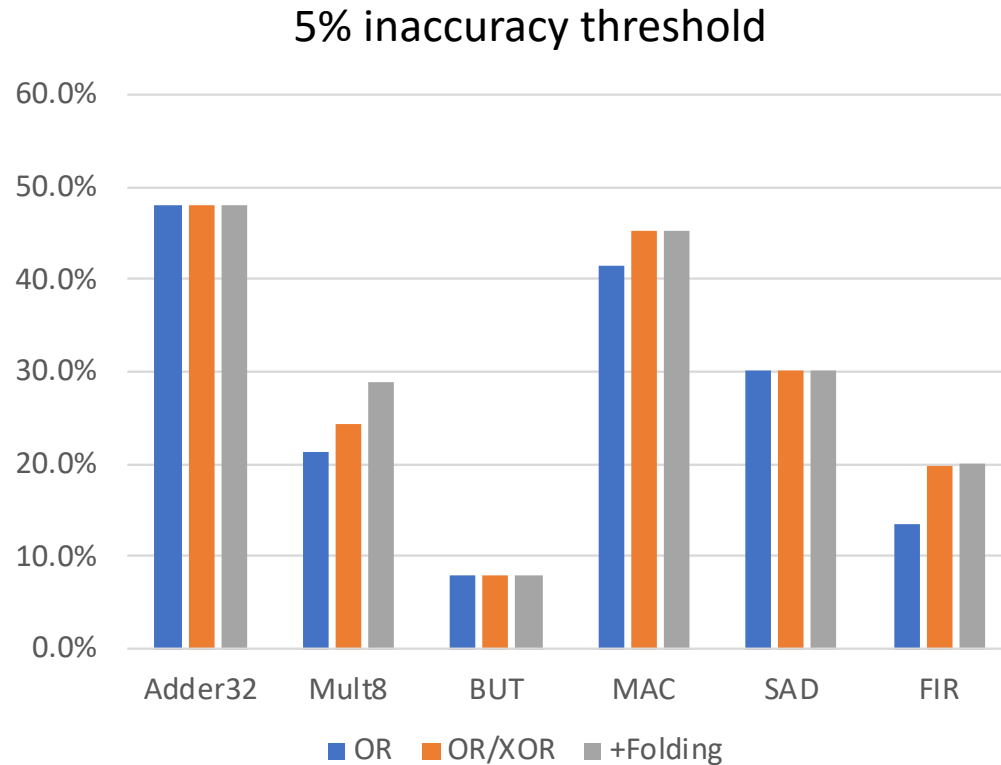
6 benchmarks ranging in number of inputs/outputs and hardware metrics.

Designs are coded in Verilog and synthesized using Synopsys DC compiler with an industrial 65-nm standard cell library at the typical process corner.

Error metric: average relative error (average )

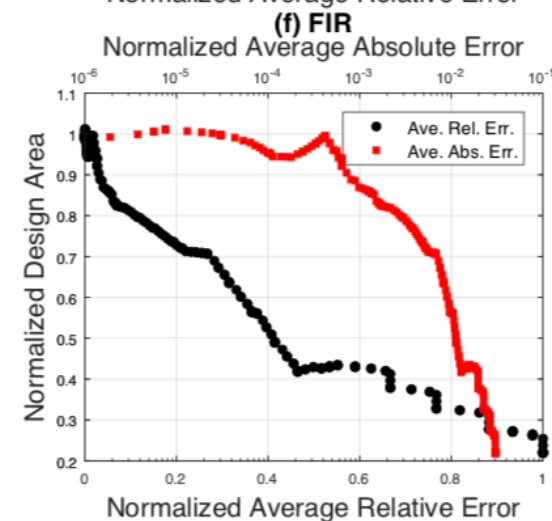
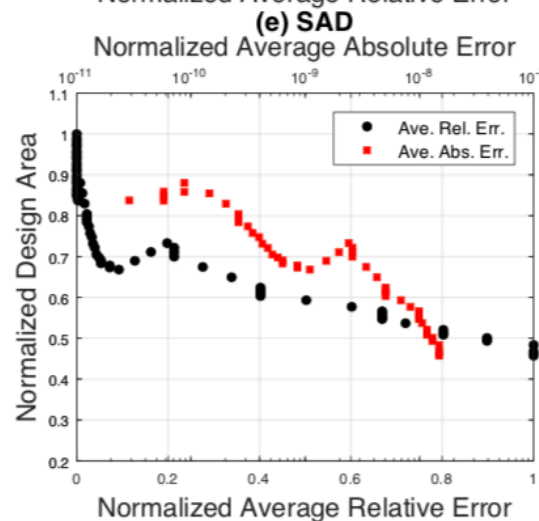
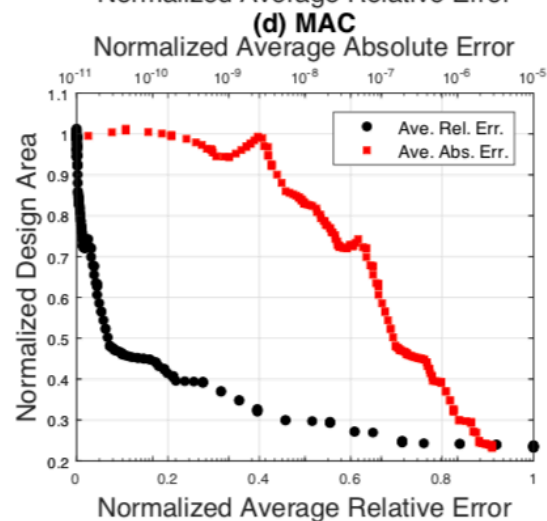
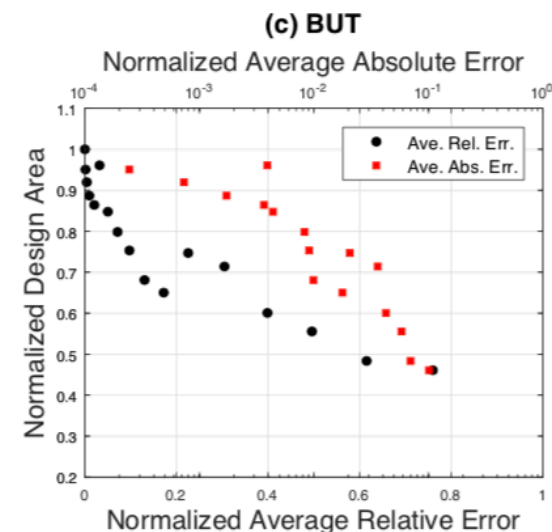
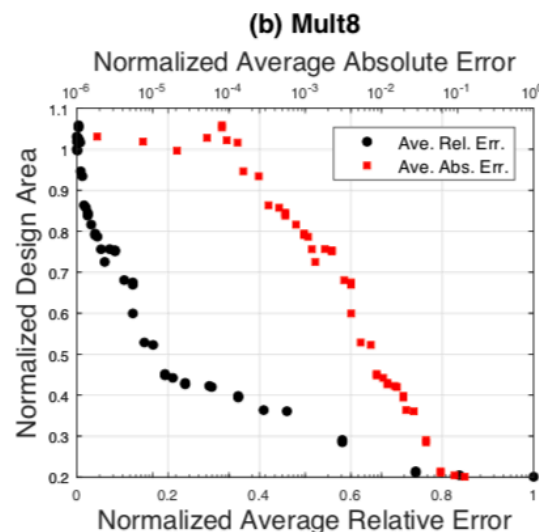
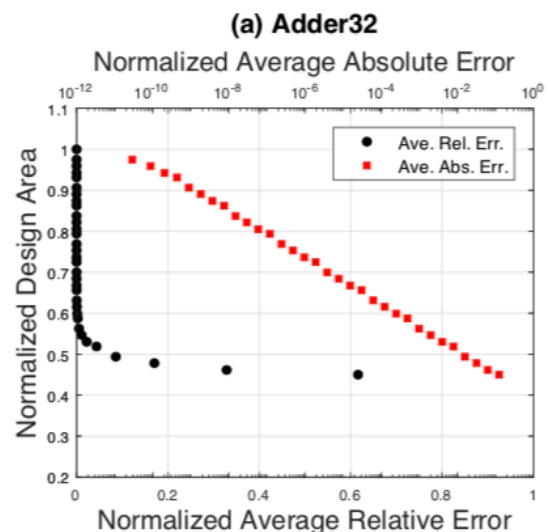
Name	Function	I/O	Accurate Design Metrics		
			Area (um2)	Power (uW)	Delay (ns)
Adder32	32-bit Adder	64/33	320.8	81.1	3.23
Mult8	8-bit Multiplier	16/16	1731.6	263.5	2.03
BUT	Butterfly Structure	16/18	297.4	80.6	1.79
MAC	Multiply and accumulate with 32-bit accumulator	48/33	6013.1	470.5	2.36
SAD	Sum of absolute differences	48/33	1446.5	195.1	2.43
FIR	4-Tap FIR Filter	64/16	8568.0	466.3	1.56

# Power reductions from proposed method



On average 27% and 38% reductions in total power

# Results from design space exploration




# Comparison to Previous Work

	Threshold 5%		Threshold 25%	
	Area Savings (%)		Area Savings (%)	
	BLASYS	SALSA [1]	BLASYS	SALSA [1]
Adder32	48.10%	20.5%	52.20%	23.2%
Mult8	29.00%	1.8%	65.60%	8.9%
BUT	7.90%	5.0%	31.90%	24.7%
MAC	45.20%	1.7%	63.60%	8.2%
SAD	30.20%	3.3%	33.20%	15.8%
FIR	20.10%	3.2%	35.80%	15.8%



Average improvement:  
~ 24%



Average improvement:  
~ 29%

[1] S. Venkataramani, et al., "SALSA: Systematic logic synthesis of approximate circuits," *DAC Design Automation Conference*, 2012.

# Conclusions

---

- We proposed to use Boolean matrix factorization (BMF) for an automated approach to the approximate synthesis problem.
- Examined both modulo-2 and semi-ring implementations.
- Truth table reshaping for more balanced compressor-decompressor architecture.
- Introduced weighted cost functions for better error metric
- Circuit breakdowns and design space exploration methods.
- For an error bound of **5%**, our methodology delivers an average of **27%** power reduction.

# Thank you...

---

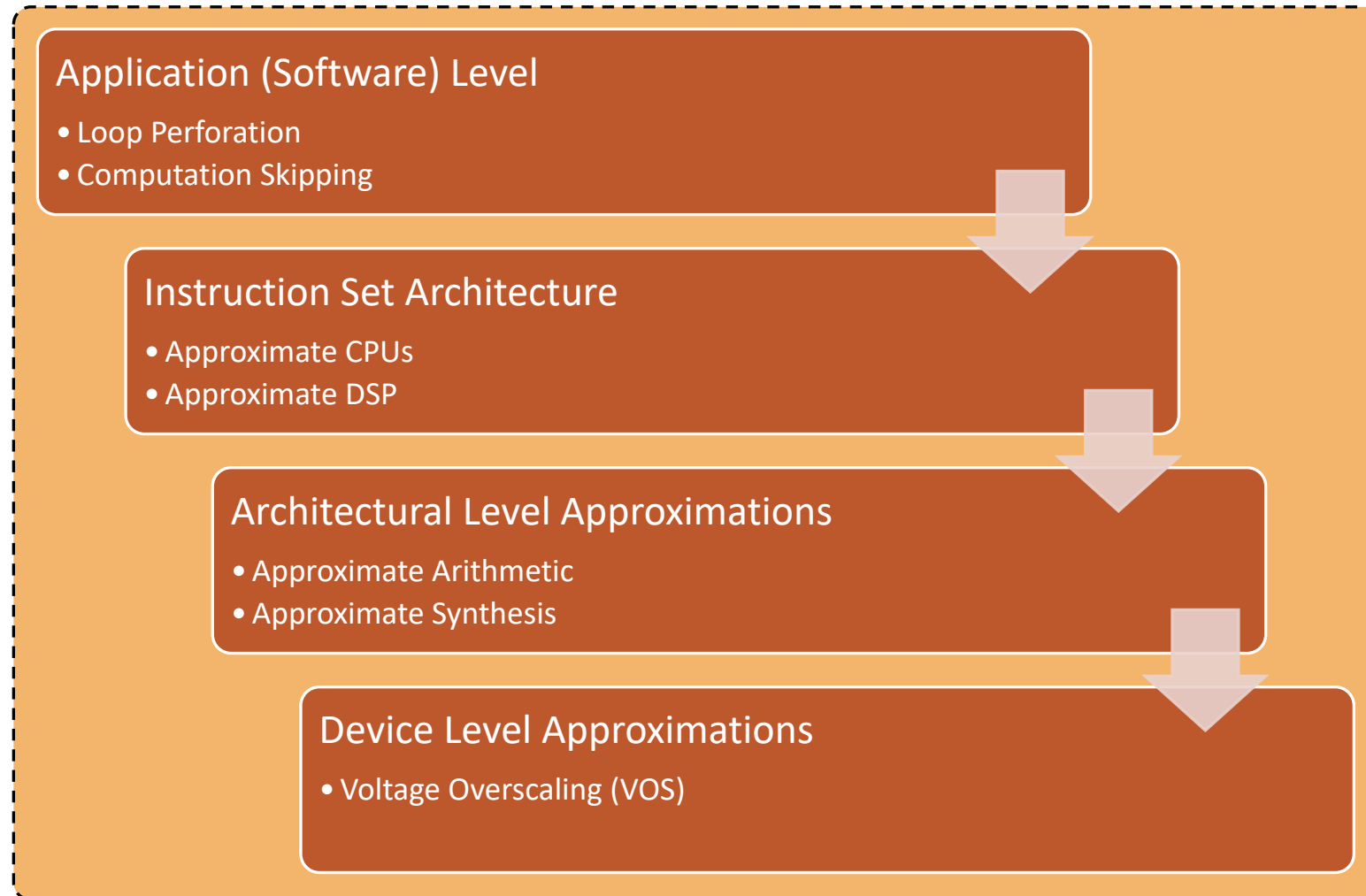
QUESTIONS?

# EXTRA SLIDES:

---

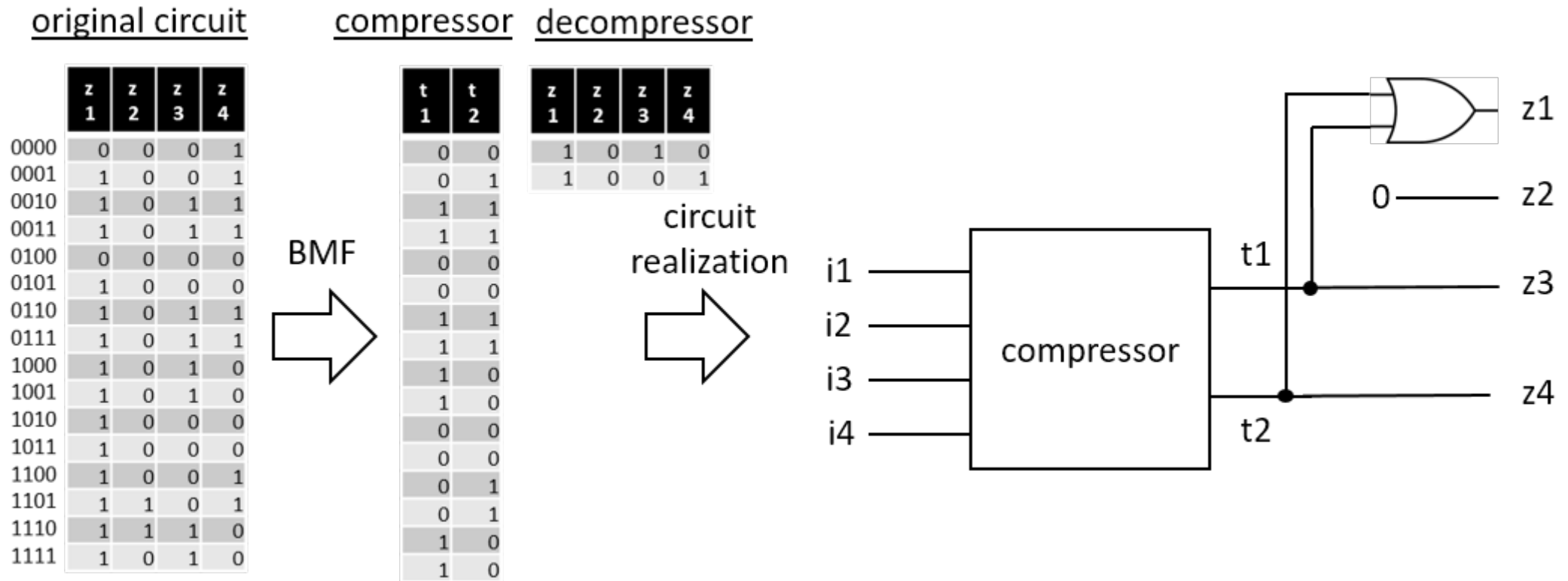
# Approximation Stack

---





# Mapping Decompressor to Hardware

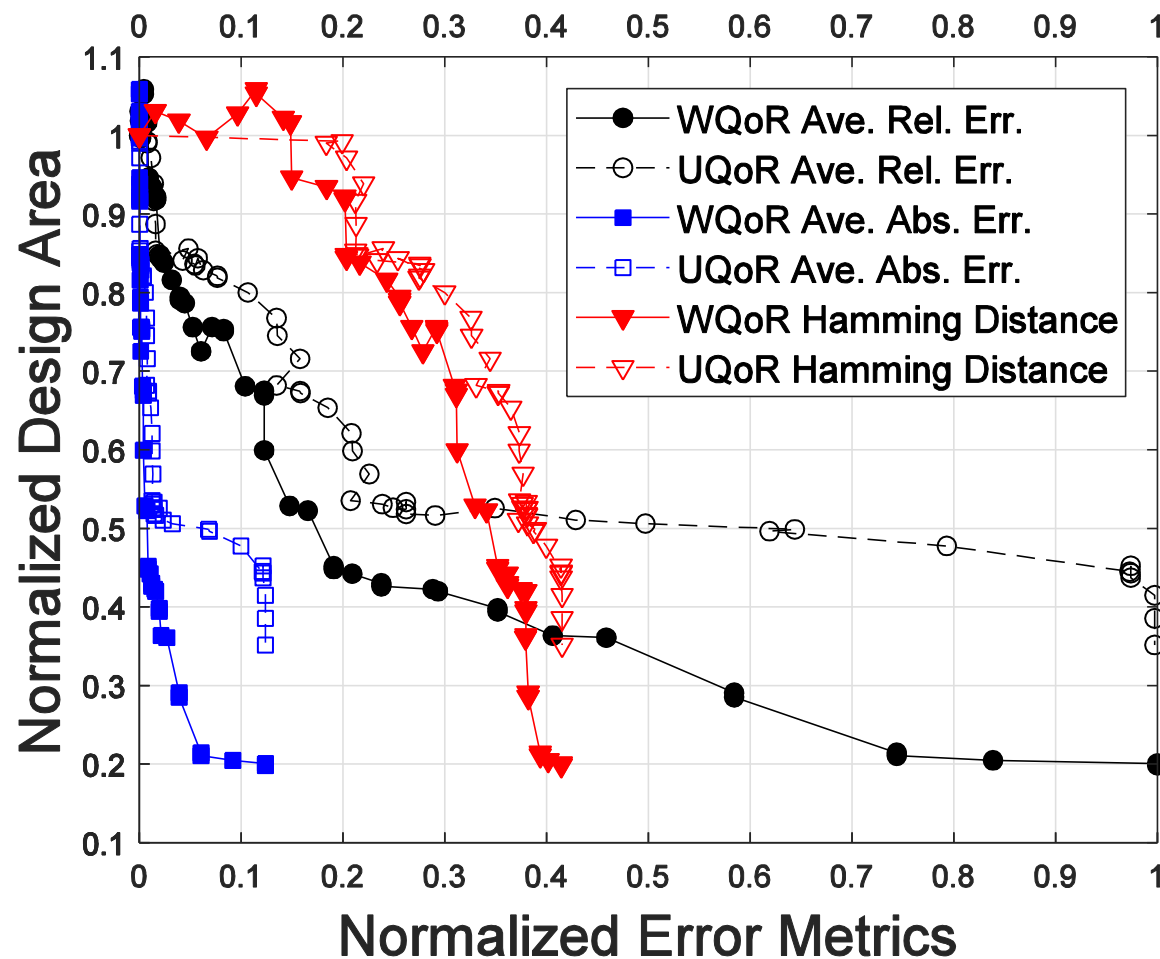


# Weighted QoR Results

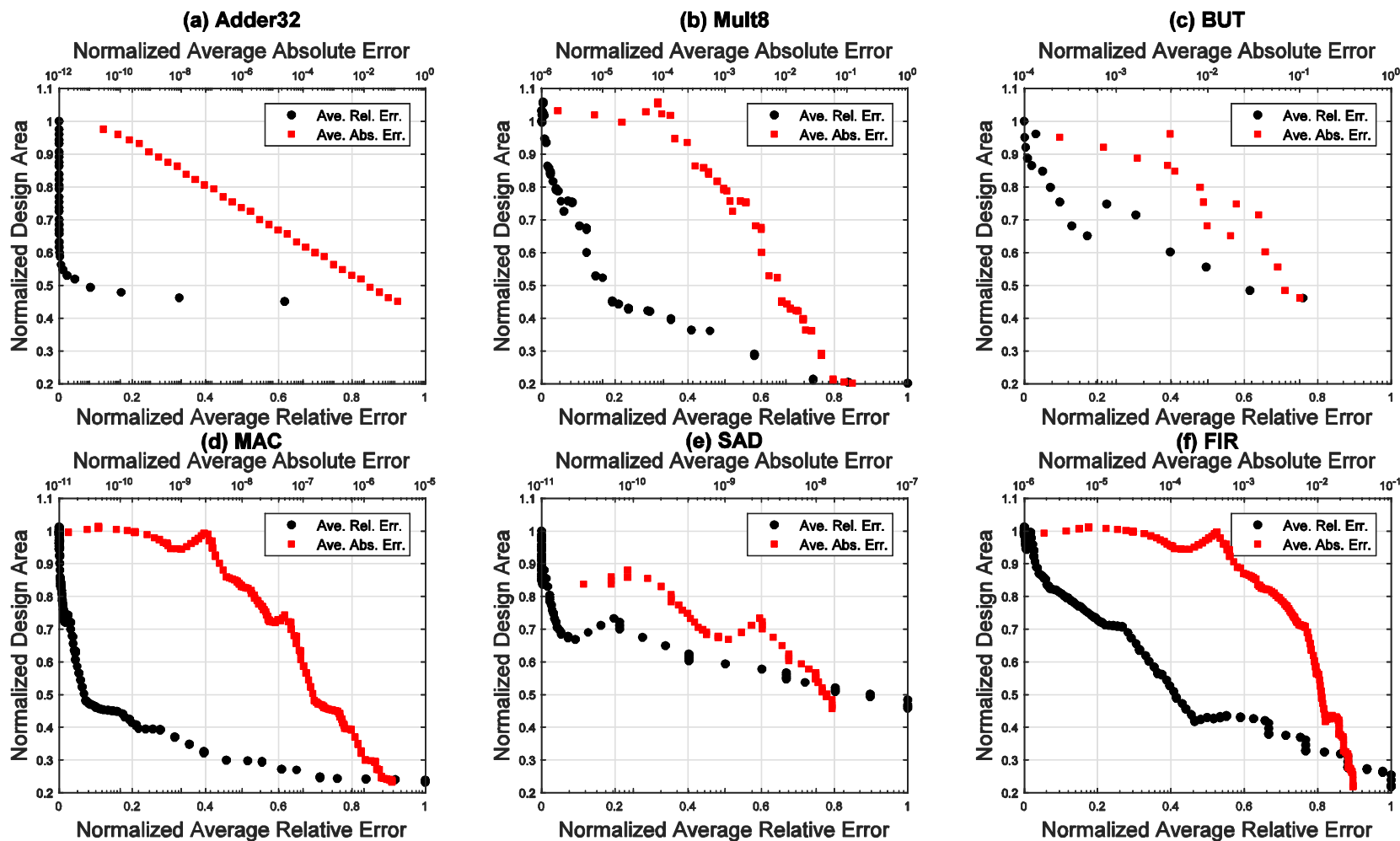
The results for Mult8 comparing:

- Uniform QoR (UQoR)
- Weighted QoR (WQoR)

**Consistent Benefits In All Error Metrics.**



# Cost-Accuracy Trade-offs



# Runtime

---

Runtime dominated by the accuracy simulation of the intermediate points.

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.