Approximate Computing with Approximate Circuits: Methodologies and Applications

ESWEEK 2017 Tutorial

Lukáš Sekanina
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
sekanina@fit.vutbr.cz

Jie Han
Department of Electrical and Computer Engineering, University of Alberta
Edmonton, AB, Canada
jhan8@ualberta.ca

Approximate Computing with Approximate Circuits: Methodologies and Applications

Part II: Design automation methods

Lukáš Sekanina
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
sekanina@fit.vutbr.cz

Tutorial Outline – Part II.

- Introduction
- Design automation methods for approximate circuits
  - Classification and overview
  - Circuit parameter estimation
  - Error computation
  - Relaxed equivalence checking
  - Evaluation methodology
- Examples of design automation methods for approximate circuits
  - Minterm complements, SASIMI, AIG rewriting, ABACUS, GRATER
- Evolutionary algorithms, CGP and circuit optimization
- Applications of CGP-based approximation methods
  - Open-source library of approximate adders and multipliers
  - Approximate TMR
  - Approximate multipliers in neural networks
  - Symbolic error analysis using BDDs/SAT solving in CGP-based tools
  - Approximate image filters
- Conclusions

Sensitivity analysis

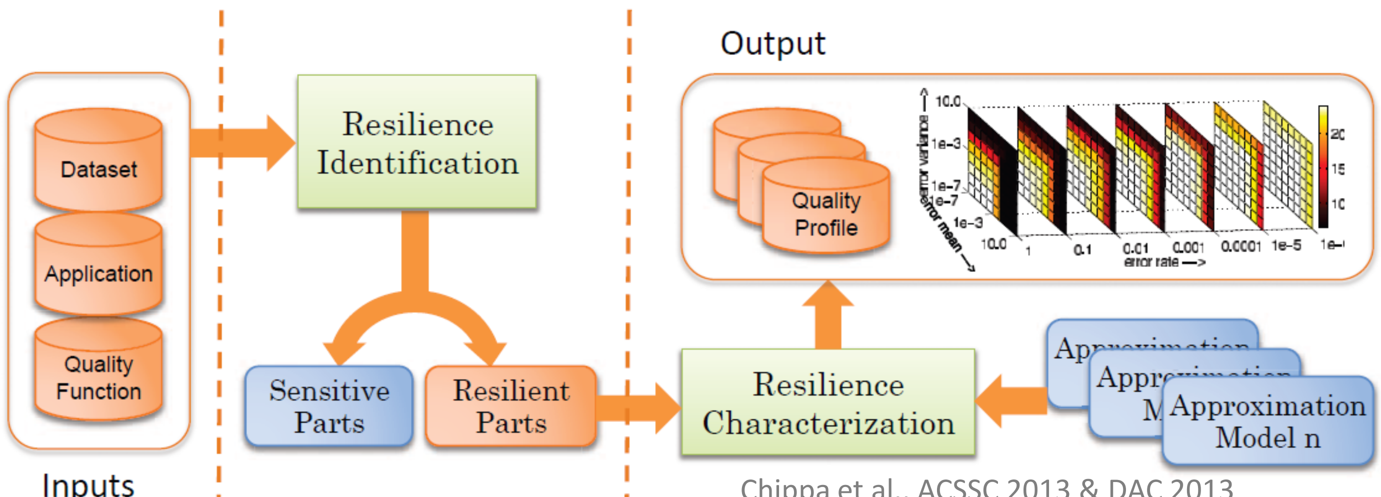
- The goal is to identify subsystems suitable for undergoing the approximation.
- Method: Random/guided modification of the original implementation and statistical evaluation of the impact on the quality of result.

In software

- precision of number representation
- data storage strategies
- code simplification
- relaxed synchronization
- unfinished loops
- skipped function calls

In hardware

- bit width reduction
- intentional disconnecting of components
- timing changes
- power supply voltage changes
- fault injection



Chippa et al., ACSSC 2013 & DAC 2013

Error metrics: Notation

- Arithmetic error metrics
  - The worst-case error (error magnitude, error significance)
  - Relative worst-case error
  - The average-case error (average error magnitude, mean error distance)
- Generic error metrics
  - Error probability (error rate)
  - Maximum Hamming distance (bit-flip error)
  - Average Hamming distance
- Application-specific error metrics
  - Distance error
  - Accumulated worst-case error and accumulated error rate

$$e_{wst}(f,\hat{f}) = \max_{\forall x \in \mathcal{B}^n} |\operatorname{int}(f(x)) - \operatorname{int}(\hat{f}(x))|$$

$$e_{rel}(f,\hat{f}) = \max_{\forall x \in \mathcal{B}^n} \frac{|\operatorname{int}(f(x)) - \operatorname{int}(\hat{f}(x))|}{\operatorname{int}(f(x))}$$

$$e_{avg}(f,\hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathcal{B}^n} |\operatorname{int}(f(x)) - \operatorname{int}(\hat{f}(x))|$$

$$e_{prob}(f,\hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathcal{B}^n} [f(x) \neq \hat{f}(x)]$$

$$e_{bf}(f,\hat{f}) = \max_{\forall x \in \mathcal{B}^n} \left(\sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \right)$$

$$e_{hd}(f,\hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathcal{B}^n} \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x)$$

f, $\hat{f}$ – original and approximate solution
n, m – the number of inputs and outputs
int – returns a decimal value from m bits

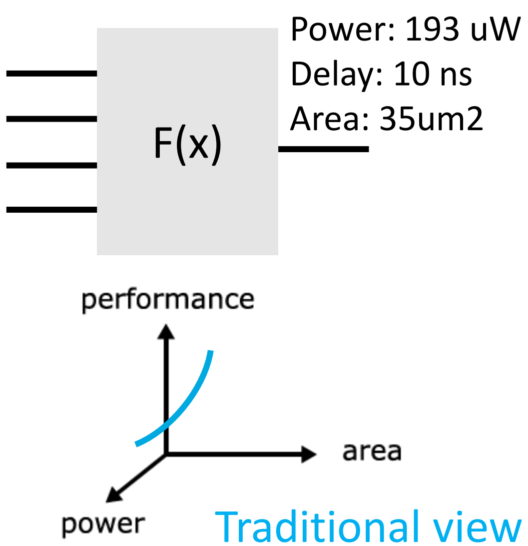
Approximation techniques - examples

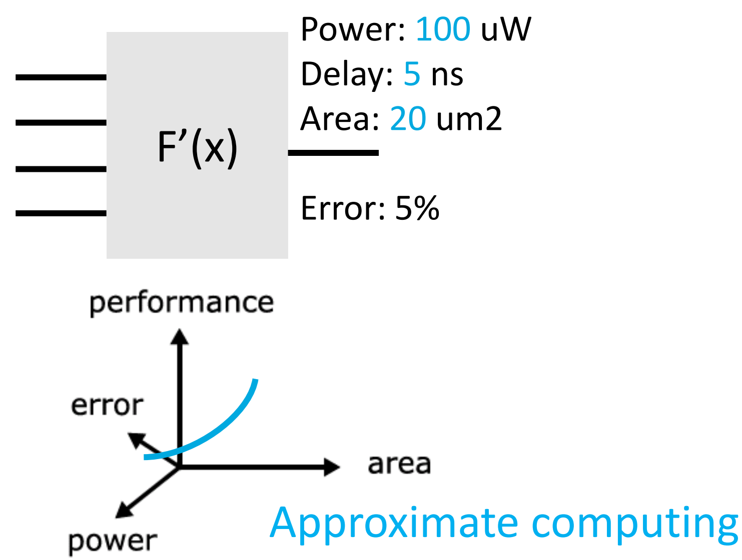
- precision scaling
- loop perforation
- load value approximation
- memorization
- task dropping/skipping
- memory access skipping
- data sampling
- using different program (circuit) versions
- etc.
- using inexact or faulty hardware
- voltage scaling
- refresh rate reducing
- inexact read/write
- reducing divergence in GPUs
- lossy compression
- use of neural networks.

Functional approximation

- Principle: Given F(x), implement a different function F'(x) that minimizes power, area and other circuit parameters, but satisfies the requirements on the quality of output.

Example:





Functional equivalence
is requested between the specification
and implementation at all levels.

Relaxed functional equivalence

Error as a design metric!

A complex multi-objective design/optimization problem!

Tutorial Outline – Part II.

- Introduction
- Design automation methods for approximate circuits
  - Classification and overview
  - Circuit parameter estimation
  - Error computation
  - Relaxed equivalence checking
  - Evaluation methodology
- Examples of design automation methods for approximate circuits
  - Minterm complements, SASIMI, AIG rewriting, ABACUS, GRATER
- Evolutionary algorithms, CGP and circuit optimization
- Applications of CGP-based approximation methods
  - Open-source library of approximate adders and multipliers
  - Approximate TMR
  - Approximate multipliers in neural networks
  - Symbolic error analysis using BDDs/SAT solving in CGP-based tools
  - Approximate image filters
- Conclusions

Languages supporting approximate computing

- EnerJ [Sampson et al., PLDI 2011]
  - An extension to Java that adds approximate data types. Approximate operations introduced by generating code with cheaper approximate instructions. The system can statically guarantee isolation of the precise program component from the approximate component.
- Rely [Carbin et al., OOPSLA 2013]
  - Programmer can mark both variables and operations as approximate. Rely works at the granularity of instructions and symbolically verifies whether the quality-of-result requirements are satisfied for each function. Rely requires programmer to provide preconditions on the reliability and range of the data.
- Axilog [Yazdanbakhsh et al., DATE 2015]
  - A set of language annotations that provide the necessary syntax and semantics for approximate hardware design and reuse in Verilog. Axilog's language semantics and the Relaxability Inference Analysis are independent of the approximate synthesis, i.e. Axilog can be used with virtually any approximate synthesis tool.
- ExpAX [Tech. Report GT-CS-14-05, Georgia Tech., 2014]
  - A static safety analysis is performed that uses the high-level (error) expectations to automatically infer a safe-to-approximate set of program operations
- Others: Chisel, …
- They require a hardware (CPU) supporting approximate computing.

```
@Approx int foo (
    @Approx int x[][],
    @Approx int y[]) {
  @Approx int sum := 0;
  for i = 1 .. x.length
    for j = 1 .. y.length
      sum := sum + x[i][j] * y[j];
  return sum;}
```

(a) EnerJ [21]

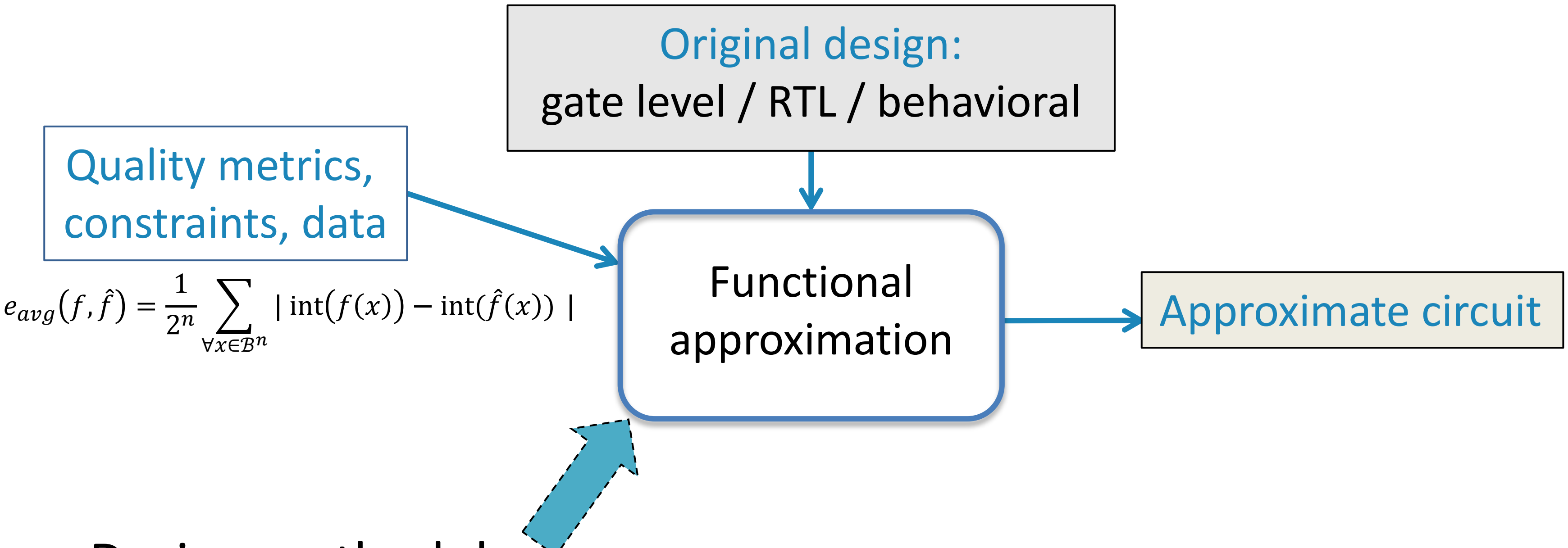
```
int <0.90*R(x, y)> foo (
      int <R(x)> x[][] in urel,
      int <R(y)> y[] in urel) {
  int sum := 0 in urel;
  for i = 1 .. x.length
    for j = 1 .. y.length
      sum := sum +. x[i][j] *. y[j];
  return sum;}
```

(b) Rely [4]

```
int foo (int x[][], int y[]) {
  int sum := 0;
  for i = 1 .. x.length
    for j = 1 .. y.length
      sum := sum + x[i][j] * y[j];
  accept magnitude(sum) < 0.10;
  return sum;
}
```

(c) ExpAX

Functional approximation of digital circuits



- Design methodology
  - Manual [Kulkarni et al.: J. Low Power Electronic 2011 and others]
  - Design automation methods (= some heuristics used)
    - SALSA (DAC 2012), SASIMI (DATE 2013), ABACUS (DATE 2014), ASLAN (DATE 2014), AIG-Rewriting (ICCAD 2016) …
    - CGP (ICES 2013, DDECS 2014, EuroGP 2015, IEEE Tr. on EC 2015, FPL 2016, GENP 2016, ICCAD 2017), ABACUS with NSGA-II (2017)
- Voltage over-scaling not covered in this tutorial.

Functional circuit approximation: Classification

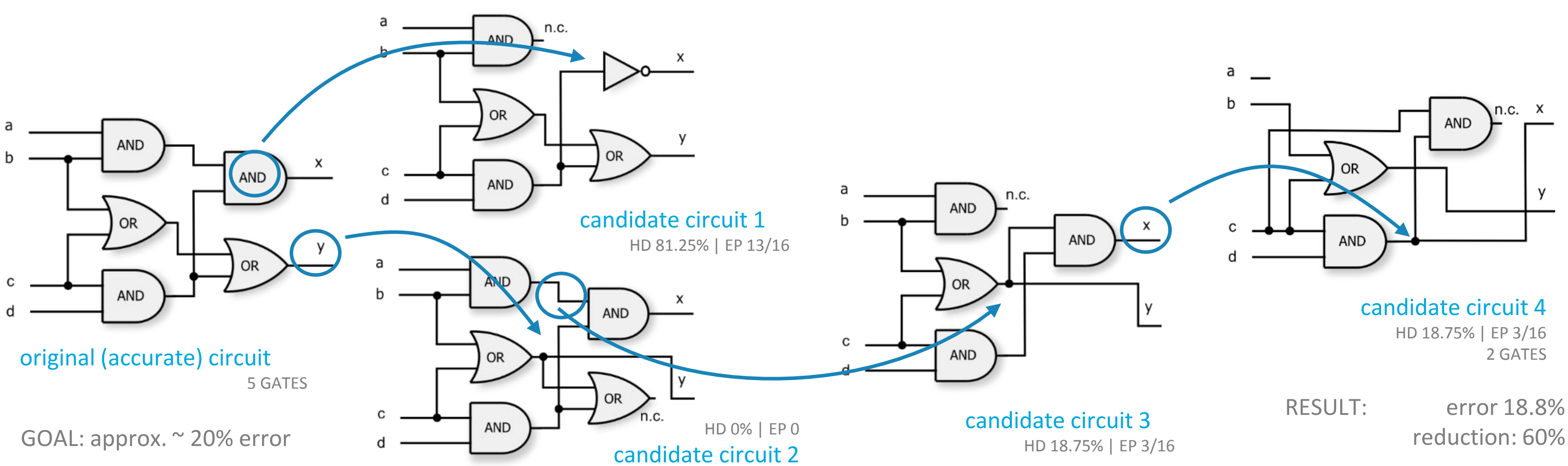
- Where is the approximation conducted?
  - Component (e.g. adder) / module (e.g. DCT) / application (e.g. video compression)
- What is the level of abstraction?
  - transistor, gate, RTL, behavioral, abstract representation (e.g. SoP, BDD, AIG …)
- How is the circuit approximated?
  - truncation
  - pruning
  - component replacement (using a library of approximate components)
  - re-synthesis
  - others
- How are candidate approximate circuits evaluated?
  - quality (at different levels of the application)
    - simulation/probabilistic/formal-based methods
  - electrical parameters
    - power, delay, area, …
- How is the approximation method evaluated?
  - The approximation methods are often heuristics! A proper statistical evaluation is requested (the best vs median value out of several independent runs).

Functional circuit approximation: Design automation

First Auth., Conf/Journal, Tool	Method, description	Error comp.	Benchmarks
Shin, DATE10	Elimination in SoP	Exhaustive sim.	<16 inputs: rd73, sym10, rd73, clip, sao2, 5xp1, t481
Shin, DATE11	Greedy, fault injection	Simulation	c880, c1908, c3540, c5315, c7552
Venkataramani, DAC12, SALSA	Don't care simplification	SAT	32-bit+, 8-bit *, 8-bit MAC, SAD, BUT, FIR, IIR, DCT
Venkataramani,DATE13,SASIMI	Similar signal detection	Simulation	ISCAS85, 32-bit +, 8-bit *, MAC, SAD, …
Ranjan, DATE14, ASLAN	Sequential/heuristics	SAT	FIR, IIR, MAC, DCT, Sobel, SAD, BUT …
Nepal, DATE14, ABACUS	Greedy over AST	Simulation	FIR, FFT, perceptron, block matcher, …
Venkataraman, DATE15	Probabilistic pruning	Simulation	Filters, QRS in ECG
Li, DAC15	Replacement in HLS	Probabilistic	MediaBench, IIR, FIR, …
Soeken, ASPDAC16, ABM	Heuristics over BDD	BDD	6 ISCAS-85
Chandrasekharan, ICCAD16	Greedy, rewriting, AIG	BDD, SAT	LGSynth91, 8/16-bit +, 8 bit *, MAC, parity
Jain, DATE16	Logic isolation	Probabilistic	32-bit +, 12-bit *, 8-bit DCT, FFT, FIR, …
Lofti, DATE16, GRATER	Truncation, OpenCL	Simulation	Sobel, DCT, recurs. Gaussian, n-body, convolution
Sekanina, SSCI-ICES13	CGP	Exhaustive sim.	4 ISCAS85 circuits, adders
Vašíček, IEEE Tr. on EC, 2015	CGP	Simulation	Multipliers, 9/25-input median
Vašíček, GPEM, 2016	CGP	BDD	Selected circuits from LGSynth, ITC and ISCAS
Češka, ICCAD17	CGP	SAT	32-bit *, 128-bit +

Search-based synthesis of approximate circuits

- The optimization engine applies various transformation rules on a given circuit and gradually modifies the circuit with the aim to obtain its approximate version which satisfies a given condition (e.g. maximal error).
- See the CGP-based approximation in this Tutorial.



Vasicek, Sekanina. Evolutionary Approach to Approximate Digital Circuits Design. IEEE Transactions on Evolutionary Computation. 2015

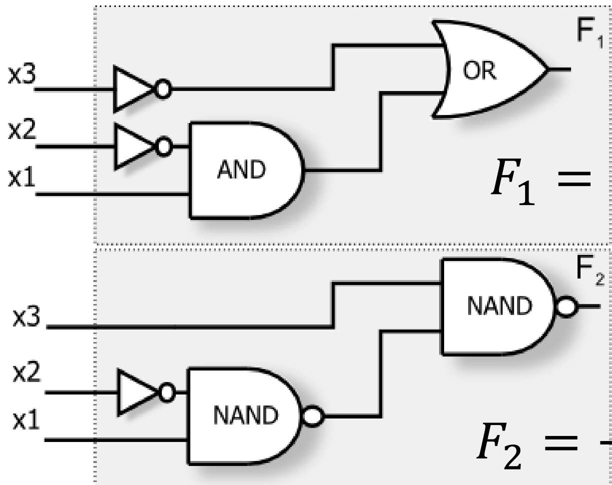
Circuit parameter estimation

- Basic circuit parameters: delay, area, power, …
- Professional CAD tools
  - Good quality
  - Slow if thousands of candidate approximate circuits have to be evaluated
- Simple methods
  - Fast, but could be inaccurate
    - Area = sum of the areas of the gates involved
    - Delay = delay along the longest path; the capacitive output load ~~not~~ ignored
    - Power = static (leakage) + dynamic (switching activity simulation)
    - Calibration is needed!
  - They are used during the approximation process.
  - The resulting approximate circuits have to be validated using professional tools.

Hrbacek, Mrazek, Vasicek. Automatic Design of Approximate Circuits by Means of Multi-Objective Evolutionary Algorithms. In: DTIS 2016, IEEE, 2016, pp. 239-244

How to determine the error?

Are F_1 and F_2 functionally equivalent?



$$F_1 = (x_1 \wedge \neg x_2) \vee \neg x_3$$

$$F_2 = \neg(\neg(x_1 \wedge \neg x_2) \wedge x_3)$$

x3	x2	x1	F1	F2
0	0	0	**1**	**1**
0	0	1	**1**	**1**
0	1	0	**1**	**1**
0	1	1	**1**	**1**
1	0	0	**0**	**0**
1	0	1	**1**	**1**
1	1	0	**0**	**0**
1	1	1	**0**	**0**

- Functional equivalence checking methods have been developed for decades.
  - They exploit the model canonicity, SAT solving, algebraic approaches, …
- Relaxed functional equivalence checking is a new topic!
  - How to prove the equivalence up to some bound?
- Scalability problem of (relaxed) equivalence checking!

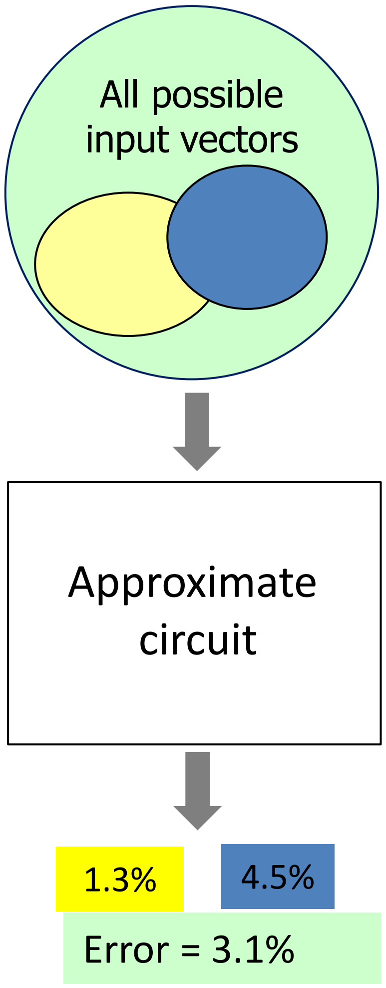
How to determine the error?

Error "estimation"

- (Functional) circuit simulation
- Probabilistic models, e.g. Li at al., DAC 2015

Exact error calculation

- Exhaustive simulation – small problem instances only
- Analysis of Binary decision diagrams
  - Average error, worst case, error rate …
    - M. Soeken et al., ASP-DAC 2016
  - Average Hamming distance:
    - Z. Vasicek and L. Sekanina. Gen. Prog. Evol. Mach., 17(2), 2016
  - Not scalable for some circuits such as multipliers
- Transforming to SAT problem
  - Worst case error
    - S. Venkataramani et al. : DAC 2012 (SALSA), A. Chandrasekharan et al. DAC 2016, M. Ceska et al., ICCAD 2017
  - Not suitable if counting the number of solutions is requested.

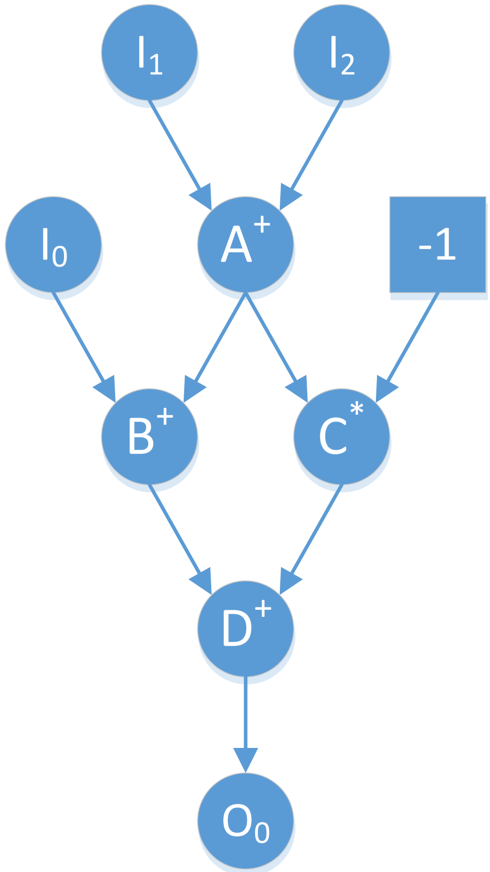


Error computation: Probabilistic methods

- For a given approximate circuit and the input data distribution, a probabilistic model is constructed and the error statistics are derived.
- Examples
  - The error statistics can be expressed as functions of the number of input bits, carry-chain length, number of overlapping prediction bits and number of sub-adders in the case of approximate adders [Mazahir et al. IEEE TC 66(3), 2017]
  - In the context of approximate HLS, an error of approximate adders and multipliers was characterized by its mean and variance. The mean is systematic and can be compensated. The overall computation precision is then determined by the variance which, after the constant compensation corresponds, to the Mean Squared Error [Li et al. DAC 2015].
- Advantages
  - Fast error computation
- Disadvantages
  - An error model has to be derived for all components which is time consuming and impractical for circuits different to adders and multipliers.
  - It is hard to provide formal guarantees in terms of the error bound.

Example [Li et al. DAC 2015]

- Assumptions
  - Error can be modeled as a random variable described by its mean and variance: $Mean(\varepsilon), Var(\varepsilon)$
  - Mean value of the error can be canceled out by a constant bias
  - First order model is sufficient
- Basic operations error model
  - $y = a + b \rightarrow y + \varepsilon_y = (a + \varepsilon_a) + (b + \varepsilon_b) + \varepsilon_+$
  - $y = a * b \rightarrow y + \varepsilon_y = ab + a\varepsilon_b + b\varepsilon_a + \varepsilon_* + \cancel{\varepsilon_a \varepsilon_b}$
- Pre-processing
  - Compute the error sensitivity ($ES_{O_i,Y}$) of output O_i to an error introduced by node Y.
  - Searching all paths from O_i to Y, using modified DFS traversal.
- Error evaluation
  - $Var(\varepsilon_{O_i}) = \sum_{y \epsilon Nodes} ES_{O_i,y} * Var(\varepsilon_y)$
- In this case
  - $Var(\varepsilon_{O_0}) = 1 * Var(\varepsilon_D) + 1 * Var(\varepsilon_B) + 1 * Var(\varepsilon_C)$
  - (the impact of component A is eliminated)

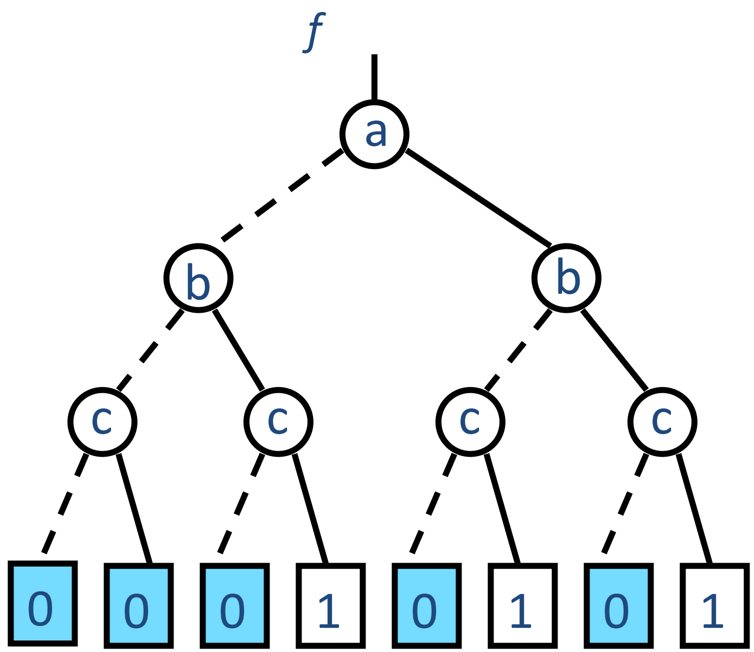


Binary Decision Diagrams

f = ac + bc

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

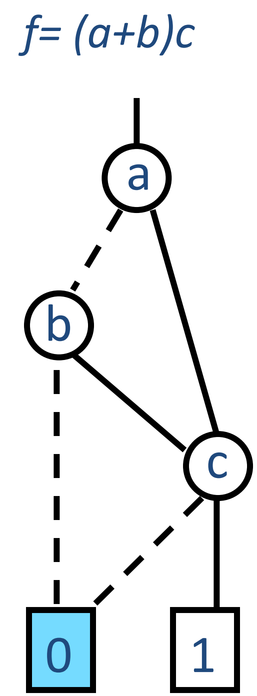
Truth table



Decision tree

1 edge

0 edge



Reduced Ordered BDD (ROBDD)

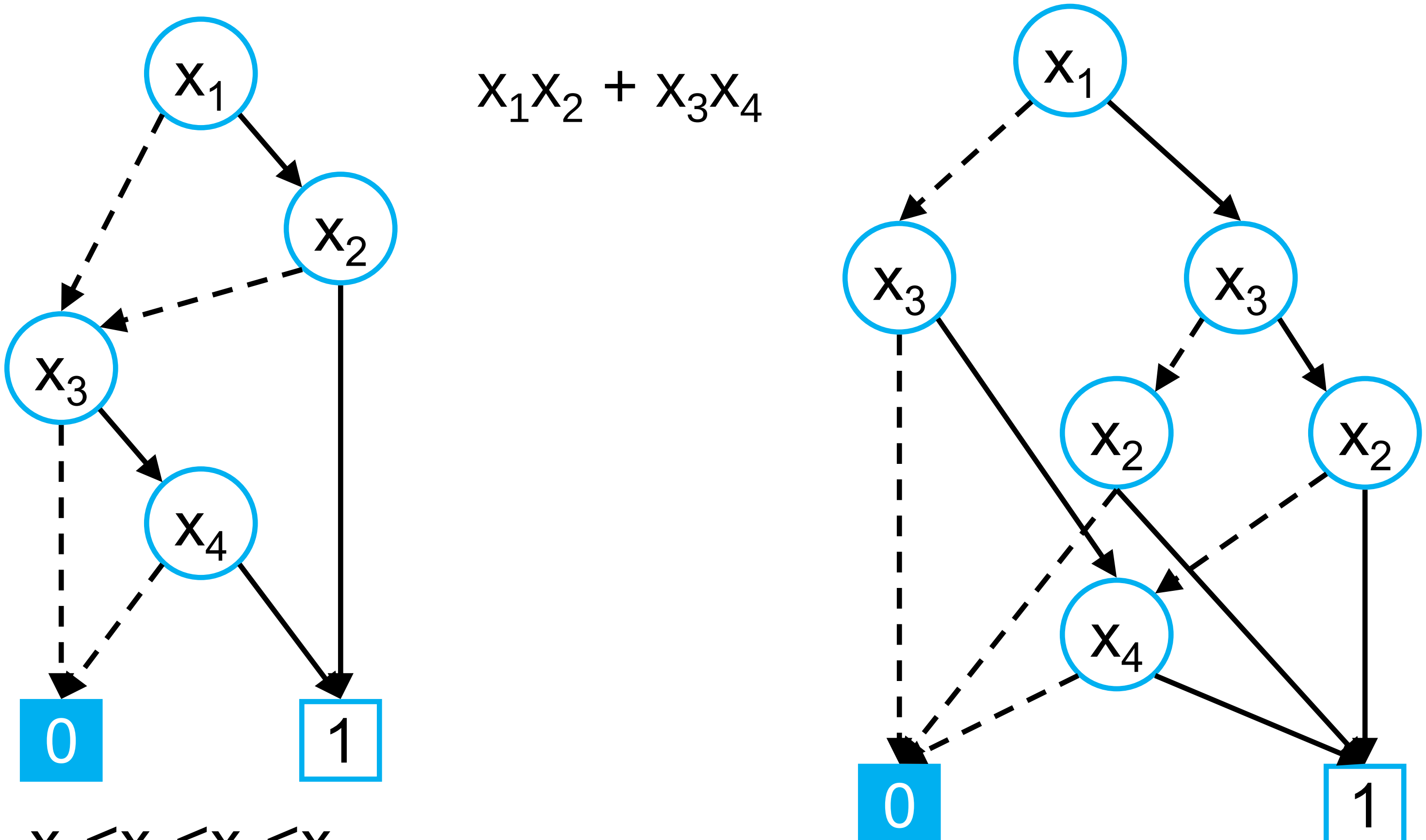
(canonical form)

Operations over (RO)BDDs implemented by many libraries, e.g. Buddy.

Pitfalls of Binary Decision Diagrams

- Variable ordering is important, may result in a more complex (or simple) BDD.

$x_1x_2 + x_3x_4$

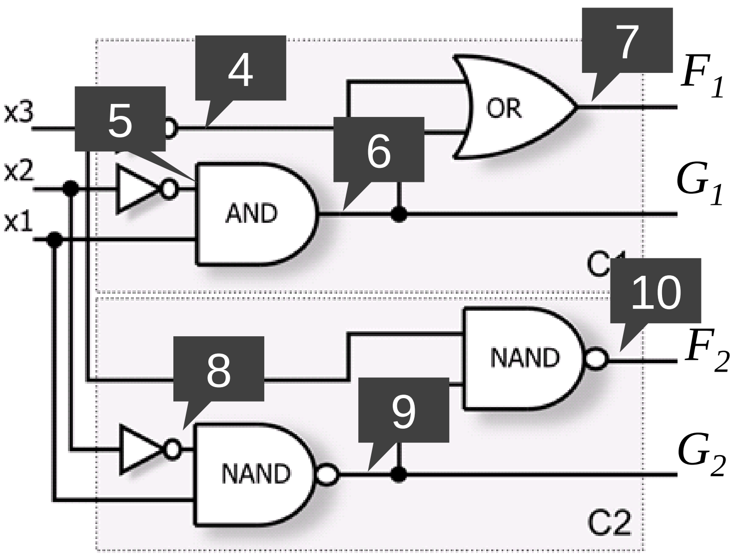


$x_1<x_2<x_3<x_4$
(optimal)

$x_1<x_3<x_2<x_4$

Equivalence checking using ROBDDs

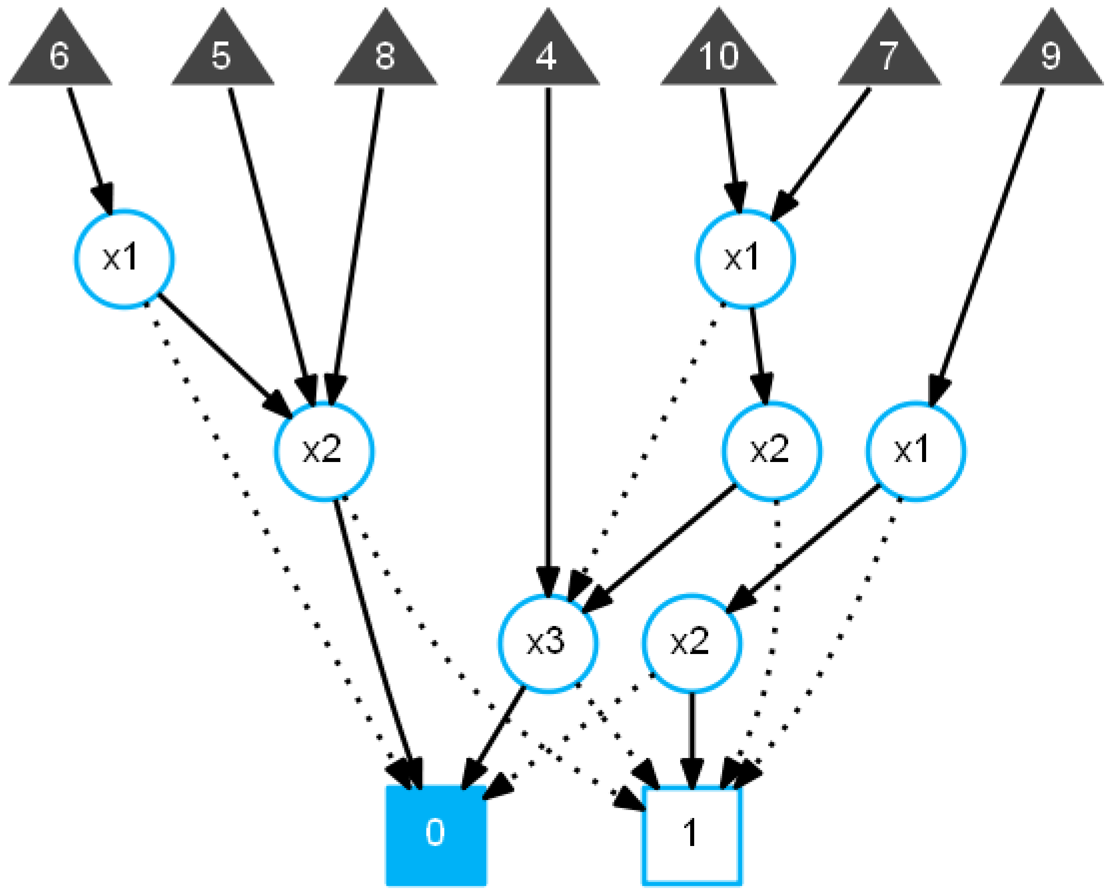
Are circuits C1 and C2 functionally equivalent?



The decision procedure is trivial and reduces to pointer comparison.

ROBDD construction:

Apply (*op*, *a*, *b*) – creates ROBDD representing logic function *op* over two ROBDDs *a* and *b*



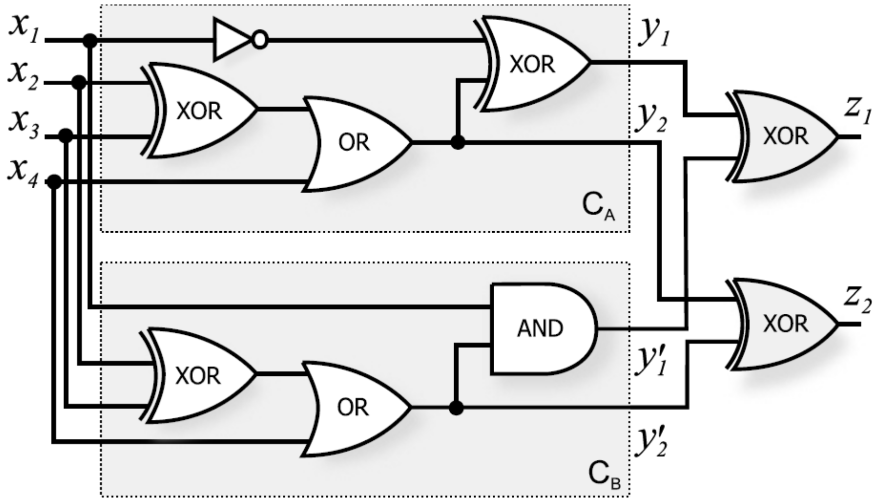
Other operations on ROBDDs

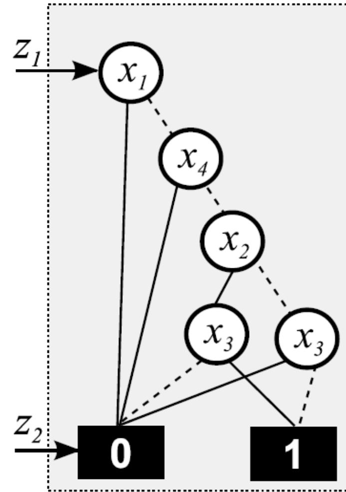
- Many logic operations can be performed efficiently on BDDs
  - usually in linear time
  - tautology and complement are constant time

Procedure	**Result**	**Time Complexity**
Reduce	G reduced to canonical form	$O(\lvert G\rvert \cdot \log\lvert G\rvert)$
Apply	f_1 <op> f_2	$O(\lvert G_1\rvert \cdot \lvert G_2\rvert)$
Restrict	$f\vert_{x_i=b}$	$O(\lvert G\rvert \cdot \log\lvert G\rvert)$
Compose	$f_1\vert_{x_i=f_2}$	$O(\lvert G_1\rvert^2 \cdot \lvert G_2\rvert)$
Satisfy-one	some element of S_f	$O(n)$
Satisfy-all	S_f	$O(n \cdot \lvert S_f\rvert)$
Satisfy-count	$\lvert S_f\rvert$	$O(\lvert G\rvert)$

Bryant R.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. on Comp. 1986

Average Hamming distance using BDDs





SatCount (*f*) – gives the number of input assignments for which *f* is '1'.

$SatCount(z_1) = 2$
$SatCount(z_2) = 0$

x_1	x_2	x_3	x_4	# combinations
0	0	0	0	1
0	1	1	0	1

- Create ROBDD for C_A, C_B and the XOR gates.
- Average Hamming distance:

$$e_{HD} = \frac{1}{2^{inputs}} \sum_{i=1}^{outputs} SatCount(z_i)$$

Average-case and worst-case error analysis

- Let $f: \mathcal{B}^n \rightarrow \mathcal{B}^m$ be a Boolean function that describes correct functionality and $\hat{f}: \mathcal{B}^n \rightarrow \mathcal{B}^m$ an approximation of it. The average-case error is defined as the sum of absolute differences in magnitude between the original and approximate circuit, averaged over all inputs:

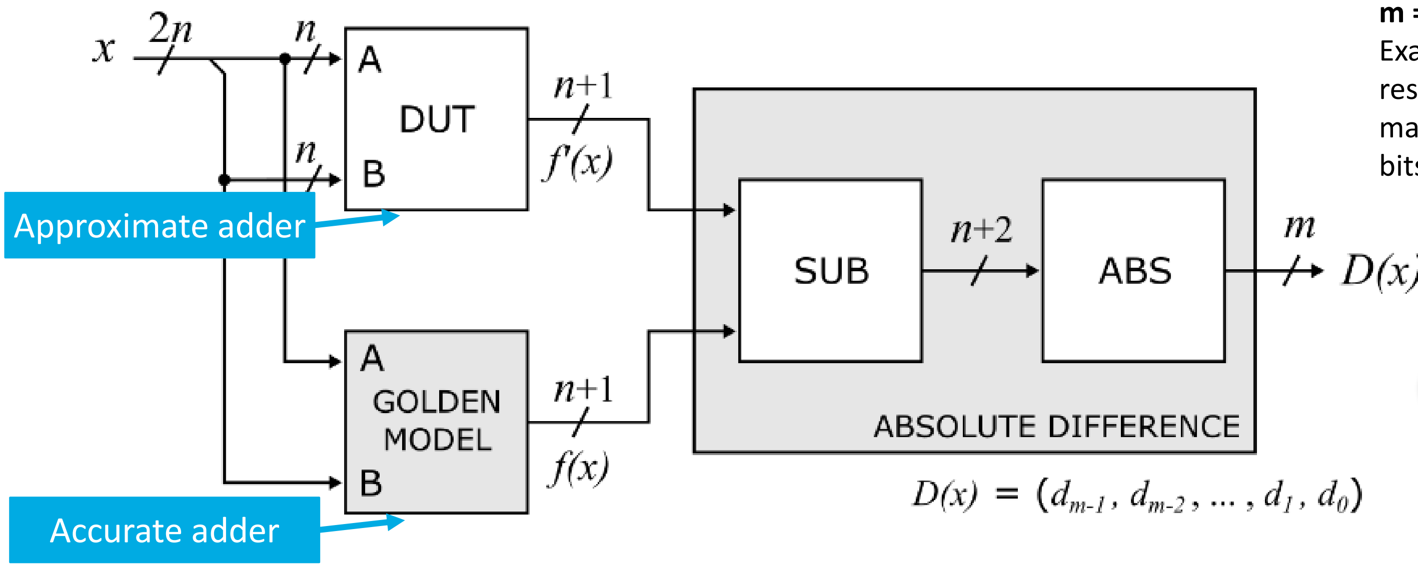
$$e_{avg}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathcal{B}^n} | \operatorname{int}(f(x)) - \operatorname{int}(\hat{f}(x)) |$$

  where $\operatorname{int}(x)$ represents a function returning a decimal value of the m-bit binary vector x.

- The worst-case error is defined:

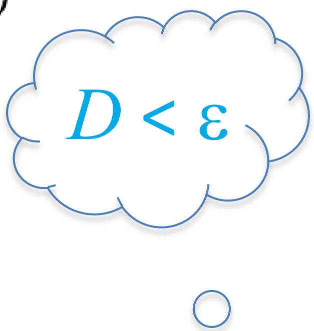
$$e_{wst}(f, \hat{f}) = \max_{\forall x \in \mathcal{B}^n} | \operatorname{int}(f(x)) - \operatorname{int}(\hat{f}(x)) |$$

Error analysis using BDD (adders)



m = n + 2

Example for n = 4: Because the result of SUB is -32 … +31, the max absolute value is 32 and 6 bits are needed for m.



Algorithm 2: average-case error analysis

Input: BDD representation of the virtual circuit (d)
Output: The average arithmetic error (ε_{avg})

$\varepsilon_{avg} \leftarrow 0$;
for $i \in \{m-1, m-2, \dots, 0\}$ **do**
  $\varepsilon_{avg} \leftarrow \varepsilon_{avg} + 2^{i-2n} \cdot satcount(d_i)$;
return ε_{avg};

Algorithm 1: BDD worst-case error analysis

Input: BDD representation of the virtual circuit (d)
Output: The maximum arithmetic error (ε_{max})

$\varepsilon_{max} \leftarrow 0$, $\mu \leftarrow true$;
for $i \in \{m-1, m-2, \dots, 0\}$ **do**
  if *satisfiable*($\mu \wedge d_i$) **then**
    $\mu \leftarrow \mu \wedge d_i$; $\varepsilon_{max} \leftarrow \varepsilon_{max} + 2^i$;
return ε_{max};

VASICEK Z., MRAZEK V., SEKANINA L.: Towards Low Power Approximate DCT Architecture for HEVC Standard. DATE 2017

Soeken et al. BDD Minimization for Approximate Computing ASPDAC 2016

BDD vs exhaustive simulation: Adders

The average time needed to perform the worst-case and the average-case error analysis for *w*-bit adders:

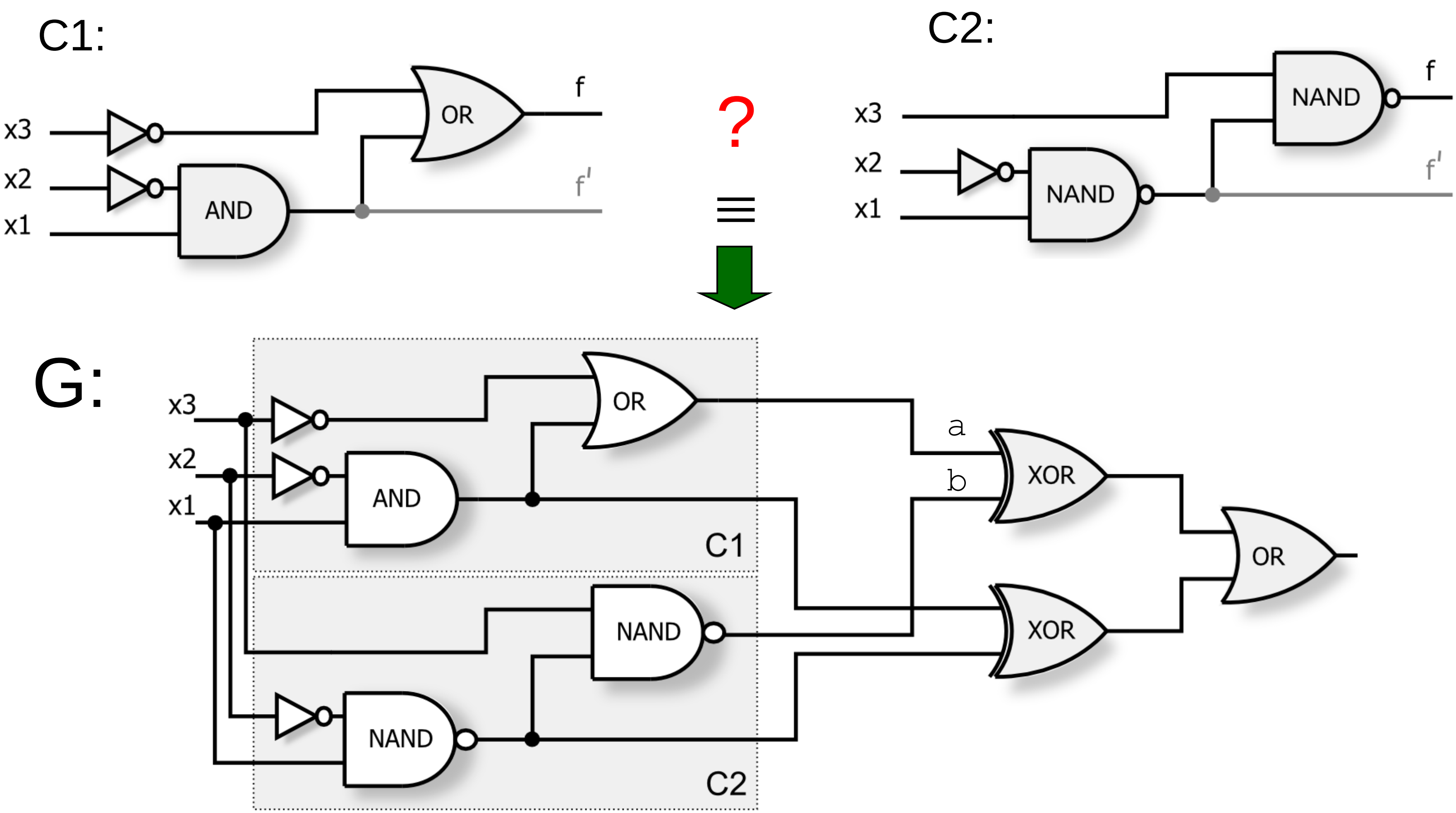
bit-width	*inputs*	*parallel simulation*	*BDD-based method*		*speedup*	
		$\varepsilon_{max} + \varepsilon_{avg}$	ε_{max}	ε_{avg}	ε_{max}	ε_{avg}
4-bit	8	4.5 us	10.3 us	14.0 us	0.43 ×	0.32 ×
8-bit	16	1.9 ms	3.5 ms	4.6 ms	0.54 ×	0.42 ×
12-bit	24	682.4 ms	127.9 ms	312.7 ms	5.33 ×	2.18 ×
16-bit	32	140.9 s	1.38 s	2.93 s	102.3 ×	48.09 ×

Notes
1) 100 randomly generated approximate adders were evaluated for each bit-width.
2) The time required to construct a BDD for the virtual circuit is included.

Practical experience: BDD-based analysis of multipliers is >10 times slower than simulation.

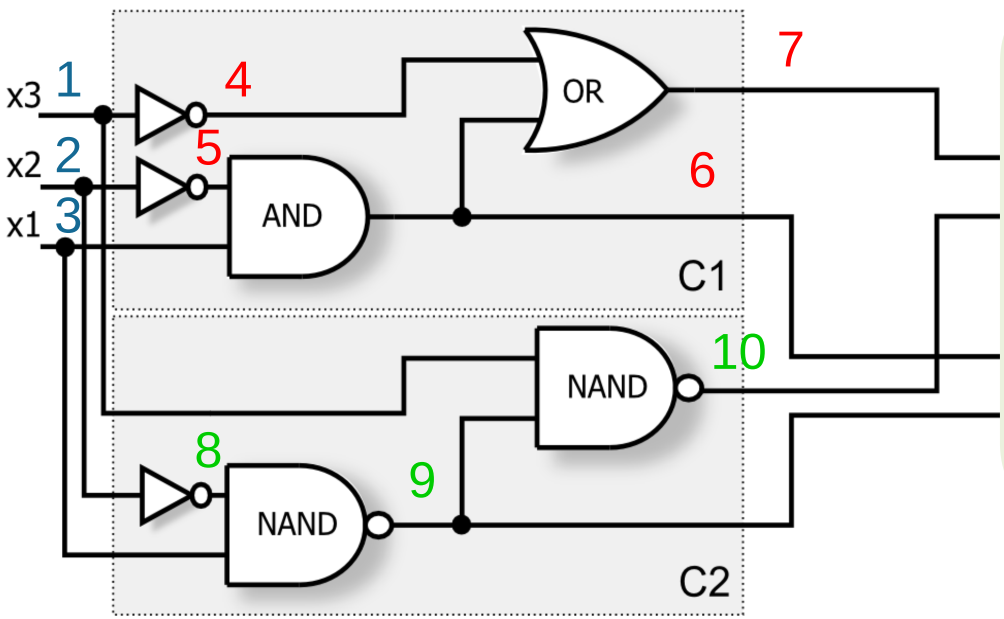
VASICEK Z., MRAZEK V., SEKANINA L.: Towards Low Power Approximate DCT Architecture for HEVC Standard. DATE 2017

Functional equivalence checking using SAT solvers



If C1 and C2 are not functionally equivalent then there is at least one assignment to the inputs for which the output of G is 1.

Tseitin transform used to create CNF for circuit G



Example: **y = not (x)**

CNF formula g(x, y) = 1 if the predicate y = OP(x) holds true

x	y	g
0	0	0
0	1	1
1	0	1
1	1	0

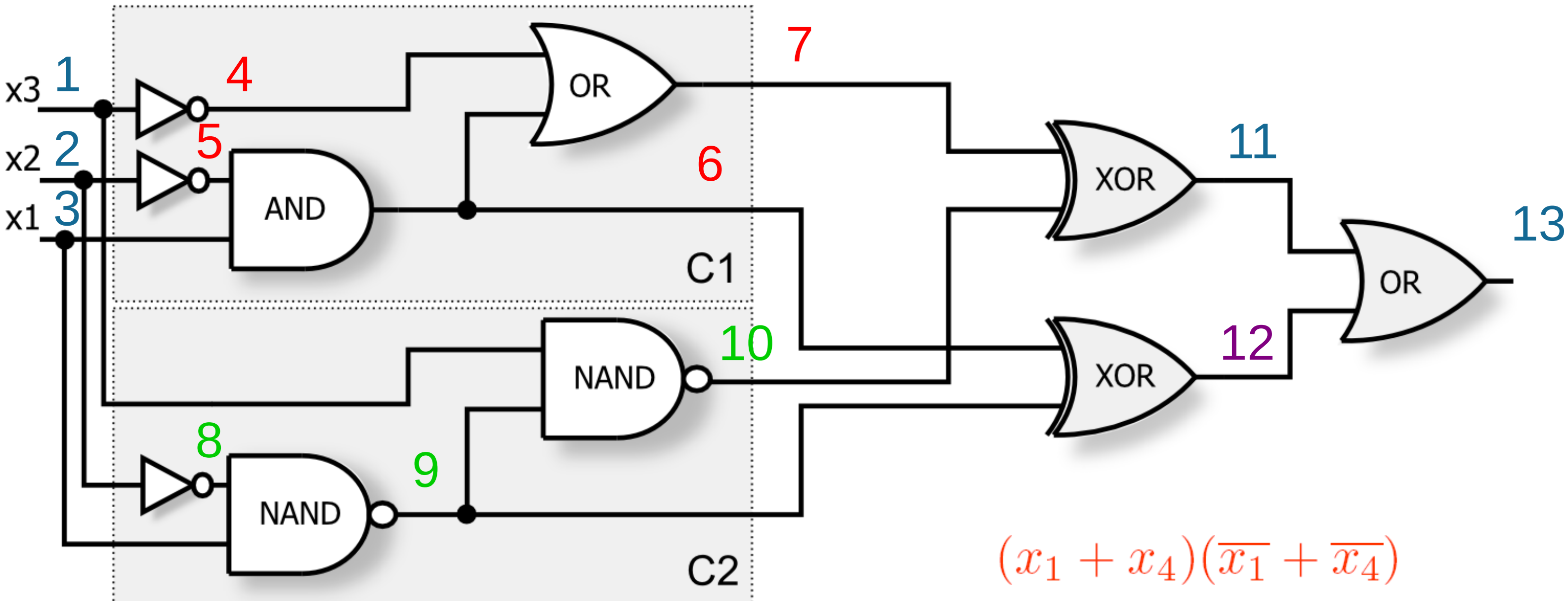
g = (~x ∨ ~y)(x ∨ y)

$$(x_1 + x_4)(\overline{x_1} + \overline{x_4})$$
$$(x_2 + x_5)(\overline{x_2} + \overline{x_5})$$
$$(x_5 + \overline{x_6})(x_3 + \overline{x_6})(\overline{x_5} + \overline{x_3} + x_6)$$
$$(\overline{x_4} + x_7)(\overline{x_6} + x_7)(x_4 + x_6 + \overline{x_7})$$

$$(x_2 + x_8)(\overline{x_2} + \overline{x_8})$$
$$(x_8 + x_9)(x_3 + x_9)(\overline{x_8} + \overline{x_3} + \overline{x_9})$$
$$(x_1 + x_{10})(x_9 + x_{10})(\overline{x_1} + \overline{x_9} + \overline{x_{10}})$$

$$(\overline{x_7} + x_{10} + x_{11})(x_7 + \overline{x_{10}} + x_{11})(\overline{x_7} + \overline{x_{10}} + \overline{x_{11}})(x_7 + x_{10} + \overline{x_{11}})$$
$$(\overline{x_6} + x_9 + x_{12})(x_6 + \overline{x_9} + x_{12})(\overline{x_6} + \overline{x_9} + \overline{x_{12}})(x_6 + x_9 + \overline{x_{12}})$$
$$(x_{11} + x_{12} + \cancel{\overline{x_{13}})}\cancel{(x_{13} + \overline{x_{11}})}\cancel{(x_{13} + \overline{x_{12}})}$$
$$\cancel{(x_{13})}$$

SAT solver in action



$(x_1 + x_4)(\overline{x_1} + \overline{x_4})$

$(x_2 + x_5)(\overline{x_2} + \overline{x_5})$

$(x_2 + x_8)(\overline{x_2} + \overline{x_8})$

SAT solver: MiniSAT

variables: 13, clauses: 30, time elapsed: 0.03ms

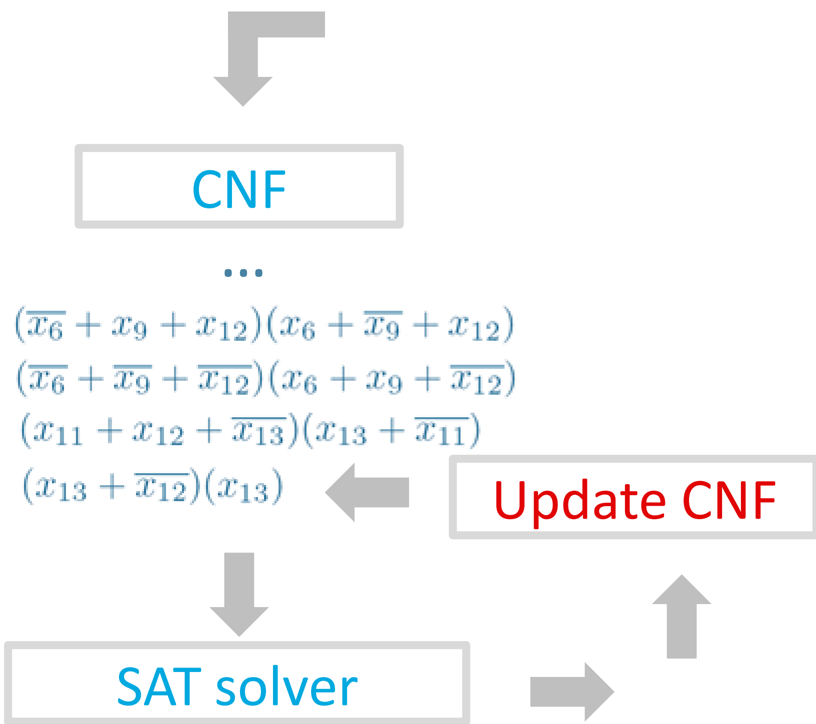
result: **SATISFIABLE / NONEQUIVALENT**

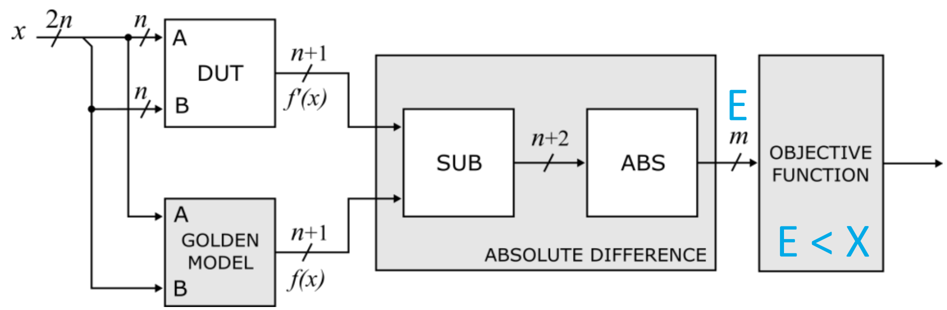
model / counter example: 0011111101011

$\cancel{(x_{13})}$

Worst-case error analysis using SAT solver

- The common approach is to use SAT-solver and binary search to find WCE (= X).
- Example: WCE for approximate n-bit adders





Algorithm 2: SAT worst-case error analysis

Input: SAT representation of the accurate circuit (f) and the approximate circuits (f') with m outputs

Output: The maximum arithmetic error (ε_{max})

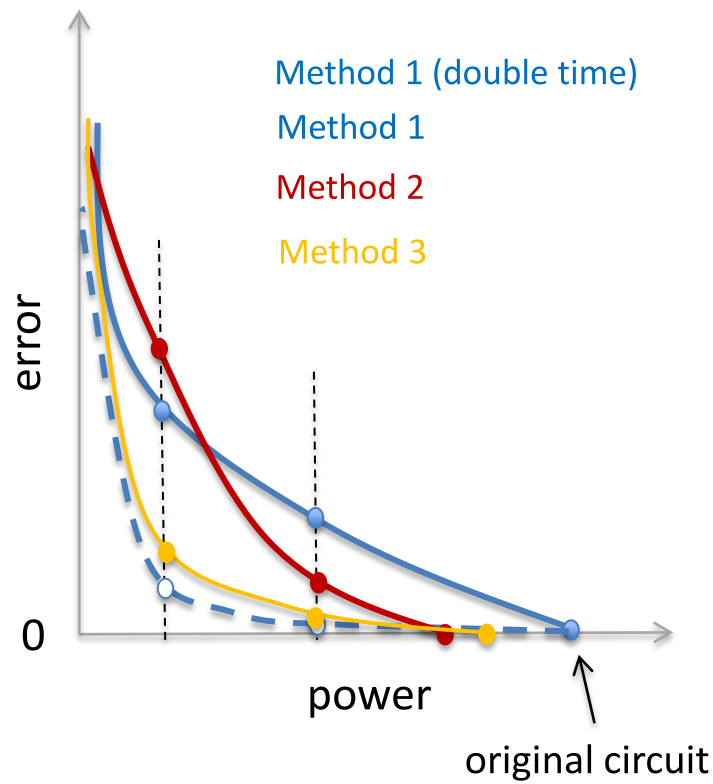
```
lbound ← 0;
ubound ← 2^m − 1;
while lbound < ubound do
    X ← ⌈(ubound+lbound)/2⌉;
    if satisfiable(ApproxMiter(|f − f′|, X)) then
        lbound ← X;
    else
        rbound ← X − 1;
ε_max ← lbound;
return ε_max;
```

Venkatesan et al. ICCAD 2011; Chandrasekharan et al. DAC 2016

On a fair comparison of automated approx. methods

- Common practice: The original circuit and approximate circuits created using a given method are compared -> not sufficient!
- A comparisons with other approximation methods is needed!
- Important assumptions for a fair comparison:
  - the original circuits are the same
  - the error is calculated using the same method (simulation vs. exact)
  - electrical parameters are calculated using the same tool and for the same technology library
  - the time/resources for the approximation methods under investigation are the same
  - the same statistically relevant values are reported (best, median, mean etc.)

8-bit multiplier approximation



Benchmarks for approximate computing

- Adders and multipliers
  - lpACLib Library
    - https://sourceforge.net/projects/lpaclib/
  - GeaR Library:
    - https://sourceforge.net/projects/approxadderlib/
  - Evoapprox8b Library
    - http://www.fit.vutbr.cz/research/groups/ehw/approxlib/

- Other
  - AxBench (GPU, CPU, Verilog)
    - http://axbench.org/
  - ApproxBench
    - http://approxbench.org/
  - AcHEe
    - http://www.scorpio-project.eu/wp-content/uploads/2016/06/CERTH_PP4REE@PPoPP_March2016.pdf

Tutorial Outline – Part II.

- Introduction
- Design automation methods for approximate circuits
  - Classification and overview
  - Circuit parameter estimation
  - Error computation
  - Relaxed equivalence checking
  - Evaluation methodology
- Examples of design automation methods for approximate circuits
  - Minterm complements, SASIMI, AIG rewriting, ABACUS, GRATER
- Evolutionary algorithms, CGP and circuit optimization
- Applications of CGP-based approximation methods
  - Open-source library of approximate adders and multipliers
  - Approximate TMR
  - Approximate multipliers in neural networks
  - Symbolic error analysis using BDDs/SAT solving in CGP-based tools
  - Approximate image filters
- Conclusions

Finding minterm complements to reduce # literals

- The objective is to obtain designs that have a minimum number of literals for a given error rate threshold.
- Method: Identify minterm complements that produce an approximate circuit version that has the smallest number of literals for a given error rate threshold.
- Exhaustive search for simple functions, a heuristics approach for more complex functions.

Original solution:

$\bar{x}_1 x_2 x_4 + x_2 x_3 x_4 + x_1 \bar{x}_2 \bar{x}_3 x_4$

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	0	1	1	0
10	0	0	0	0

(a)

Approximation 1: $x_2 x_4 + x_1 \bar{x}_3 x_4$

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	1	*c*	1
11	0	1	1	0
10	0	0	0	0

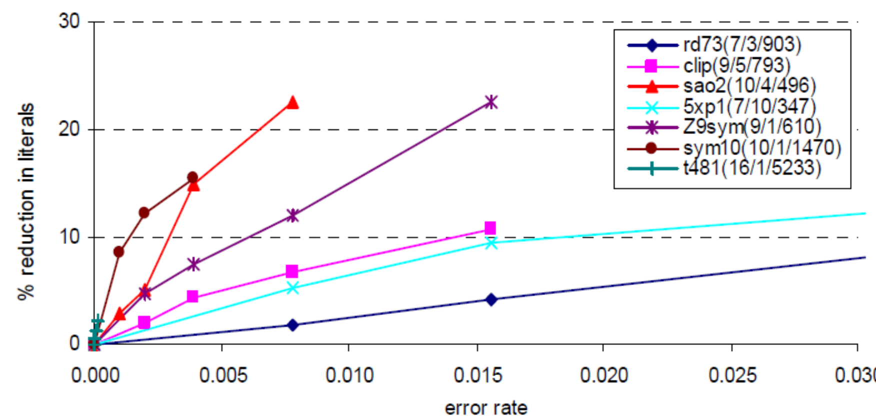
(b)

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	1	0	c
11	0	1	1	0
10	0	0	0	0

(c)

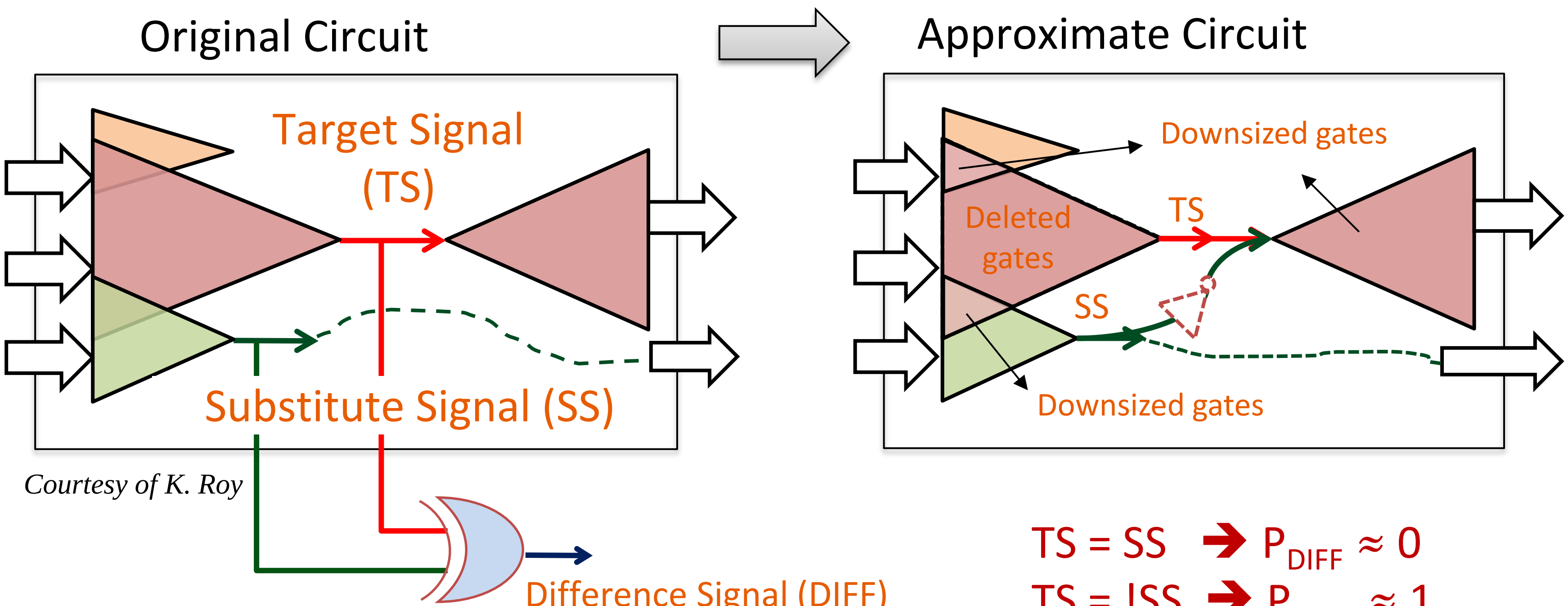
(*c*: complemented minterms)

Approximation 2: $\bar{x}_1 x_2 x_4 + x_2 x_3 x_4$



(legend: #of inputs/#of outputs/# of literals in original function.)

SASIMI: Substitute and Simplify



TS = SS ➔ $P_{DIFF} \approx 0$

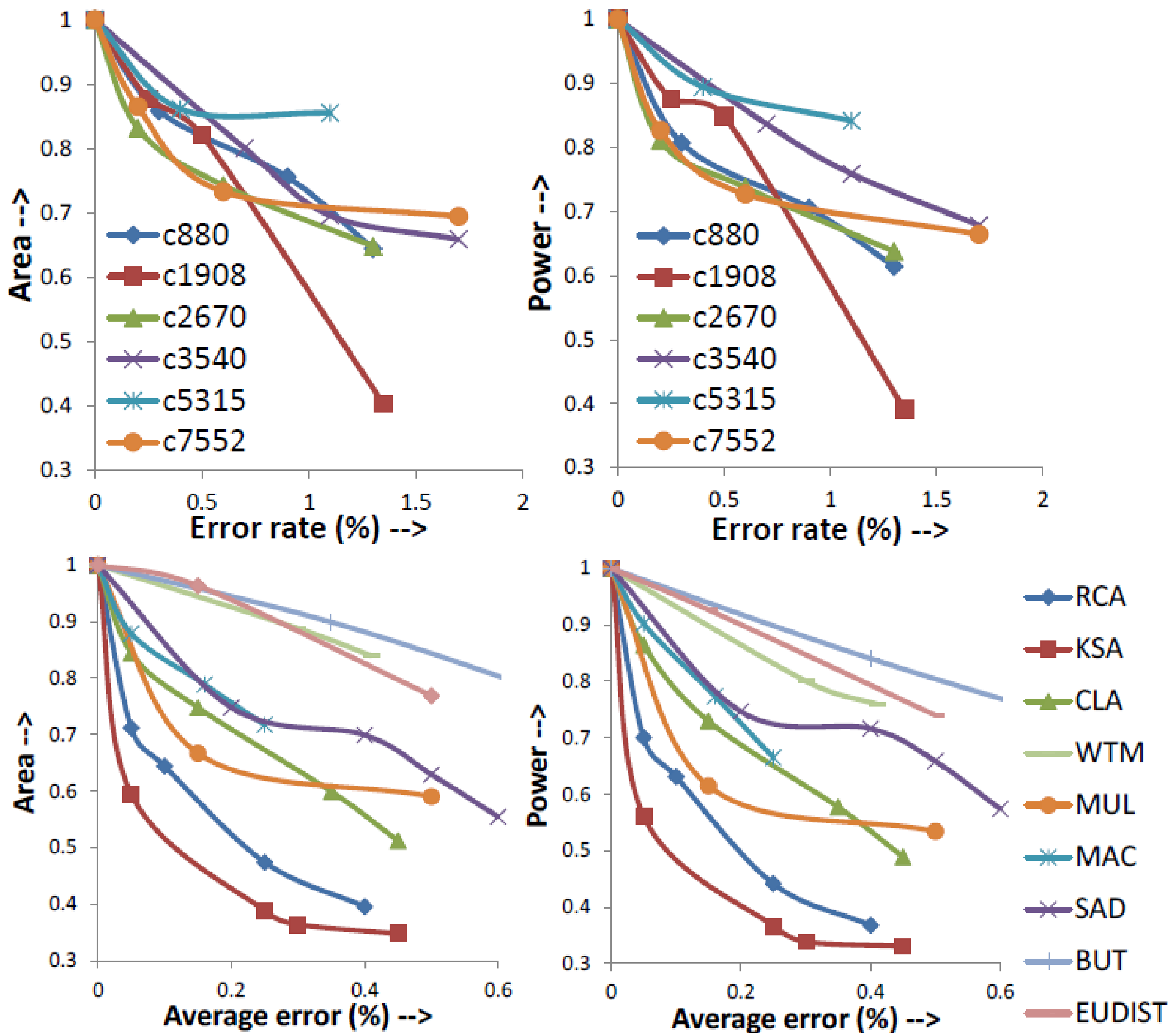
TS = !SS ➔ $P_{DIFF} \approx 1$

- Key Idea: Identify signal pairs (TS and SS) that are similar in functionality *i.e.* produce the same value for most of the inputs among signal pairs.
  - Substitute one in place of the other
    - Circuit becomes approximate
  - Simplify the circuit: Logic Deletion & Downsizing

The signal probability calculation engine in Synopsys Power Compiler was used to obtain difference probabilities

S. Venkataramani, K. Roy, and A. Raghunathan: Substitute-and simplify: a unified design paradigm for approximate and quality configurable circuits, DATE'13, pp. 1367–1372

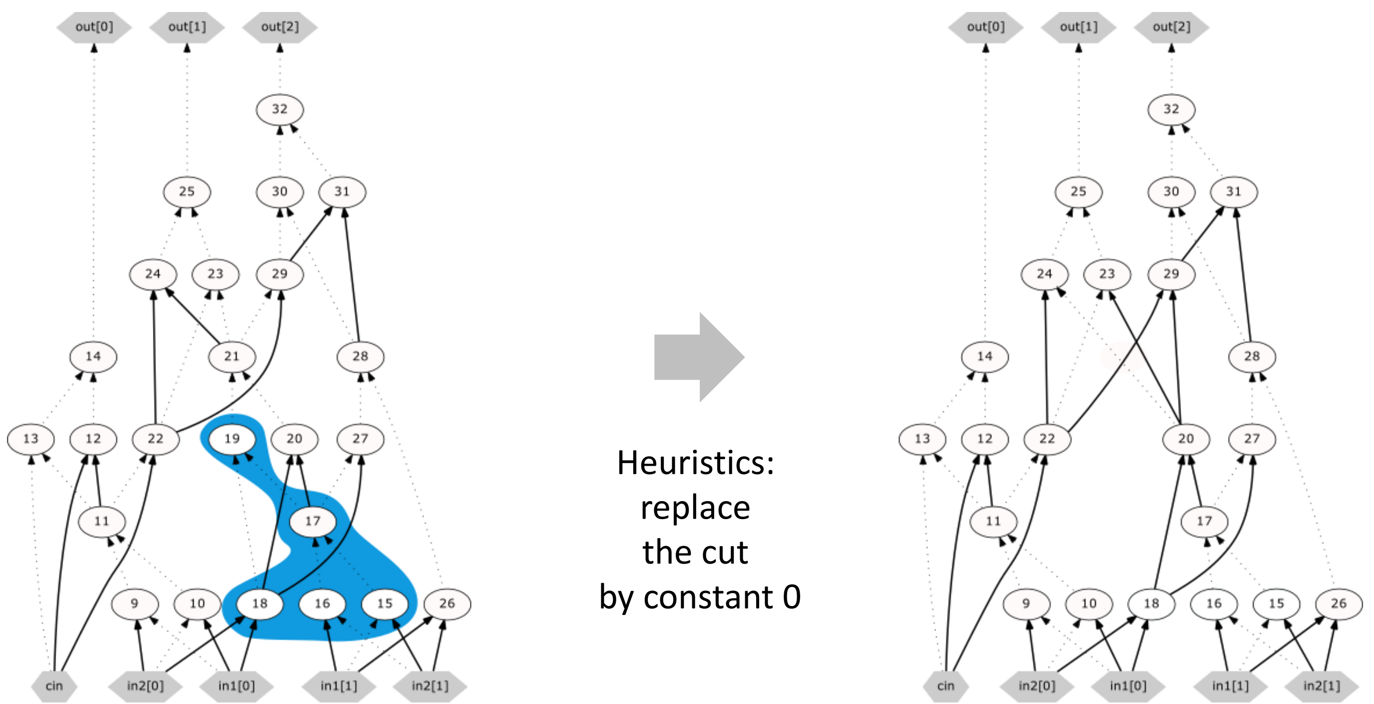
SASIMI: Substitute and Simplify



S. Venkataramani, K. Roy, and A. Raghunathan, “Substitute-and simplify: a unified design paradigm for approximate and quality configurable circuits, DATE’13, pp. 1367–1372

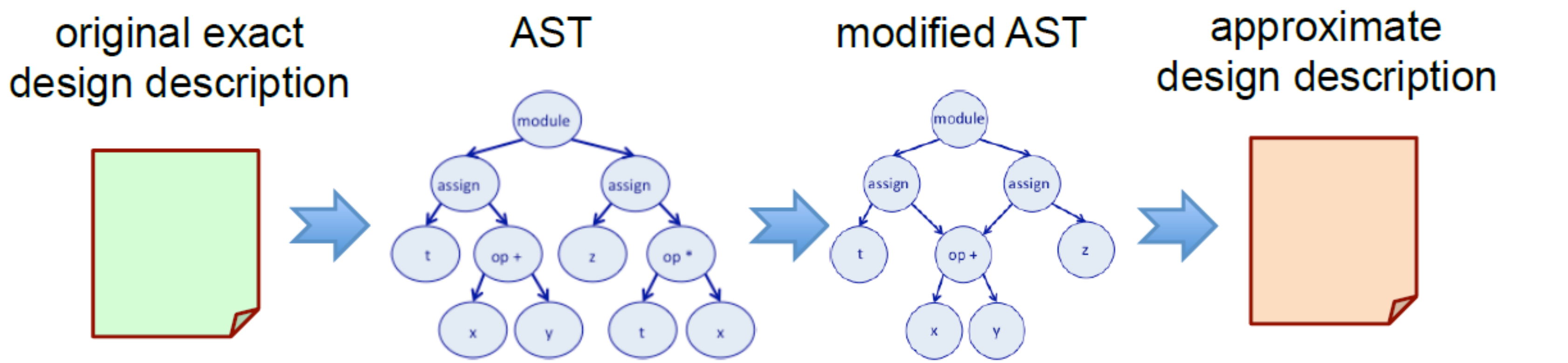
Approximation-aware Rewriting of AIGs

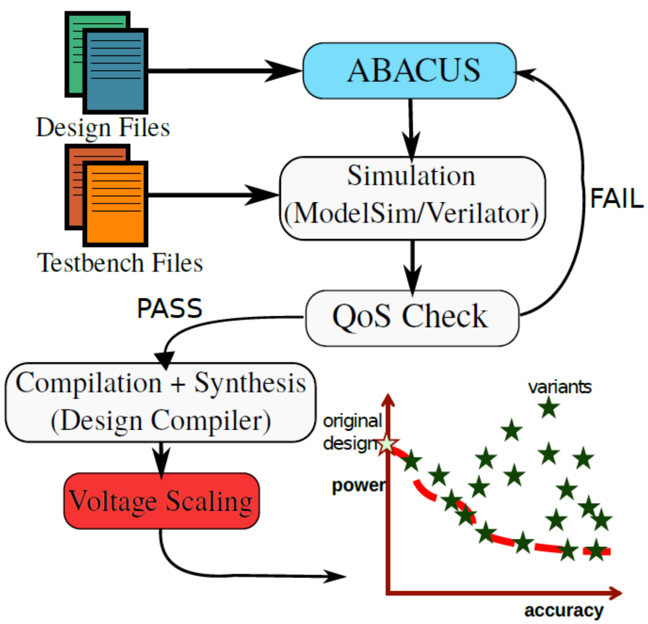
- Principle: allow AIG rewriting to change the functionality of the circuit without violating a predefined error bound.
- Rewriting (at the level of cuts on selected paths) takes a greedy approach.
- Worst-case error, bit-flip error and error rate determined exactly (formally).
- Evaluated: 8/16-bit adders, LGSYnth91, 8-bit multipliers, 32-bit parity, …



2-bit adder

ABACUS: Approximations at Behavioral RT-level





- Original file: Verilog
- Abstract Syntax Tree (AST) transformations (mutations)
  - Data type simplification
  - Operation transformations (e.g. + -> or)
  - Arithmetic expression transformation
  - Variable to Constant transformations
  - Loop transformations
- Search algorithm: Greedy / NSGA-II
- Fitness is obtained by circuit simulation and combines the error & power

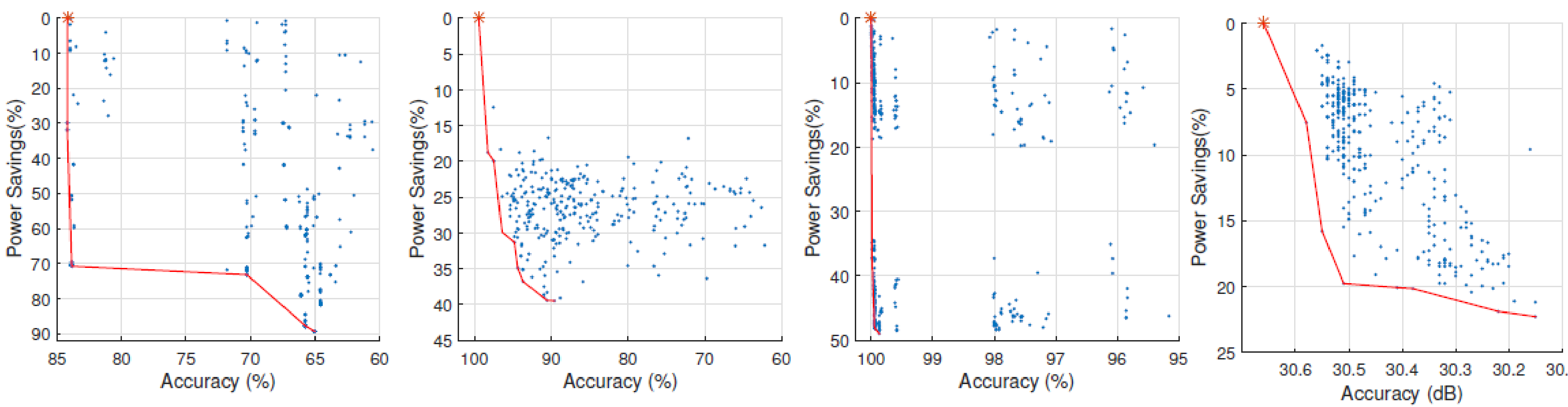
Nepal K. et al.: DATE 2014 and IEEE Trans. on Emerging Topics in Computing 2017

ABACUS: Results

Benchmark problems:

Design	Class of Application	#Lines	Area (um^2)	Power (mW)	Quality Measure	Quality
perceptron	Machine Learning	188	37775.16	2.74	classification error	82.9%
FIR filter	Signal Processing	265	40390.20	6.89	MSE	99.45%
FFT	Signal Processing	255	18480.96	2.07	MSE	100%
block matching	Computer Vision	1277	80272.44	30.42	PSNR	30.66 dB

Results of evolutionary approximation:

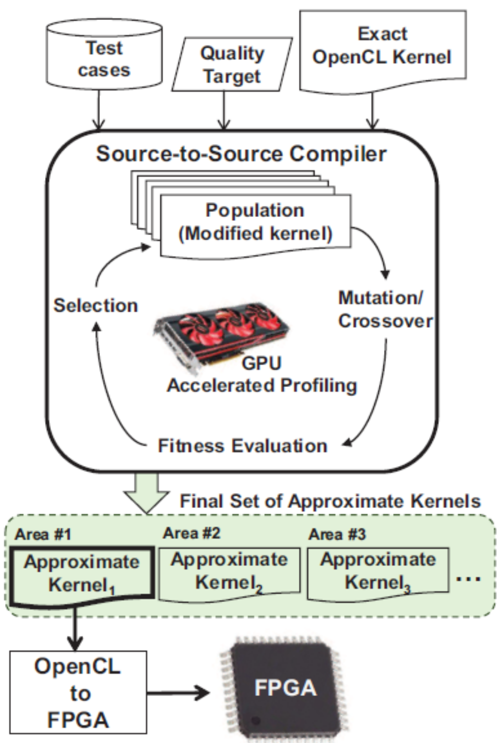


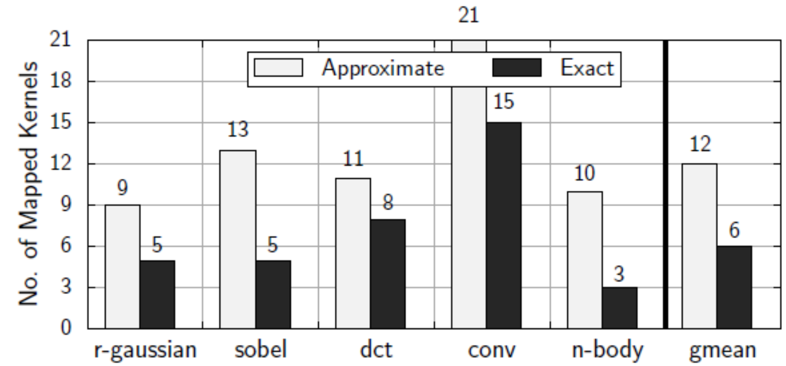
(a) Perceptron (b) FIR Filter (c) FFT (d) Block Matching

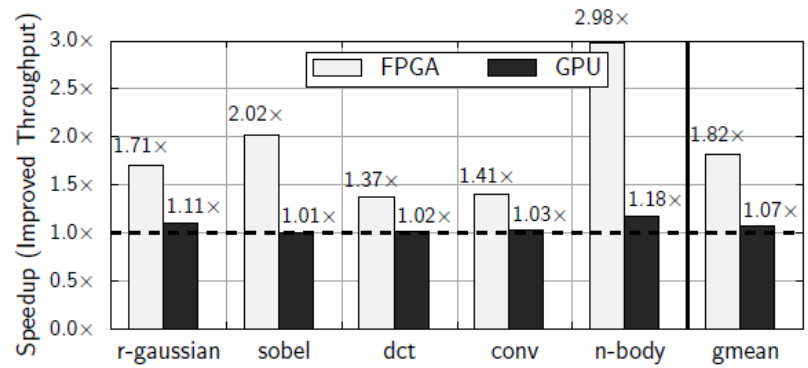
Nepal K. et al.: Automated High-Level Generation of Low-Power Approximate Computing Circuits, Trans. on Emerging Topics in Computing, 2017

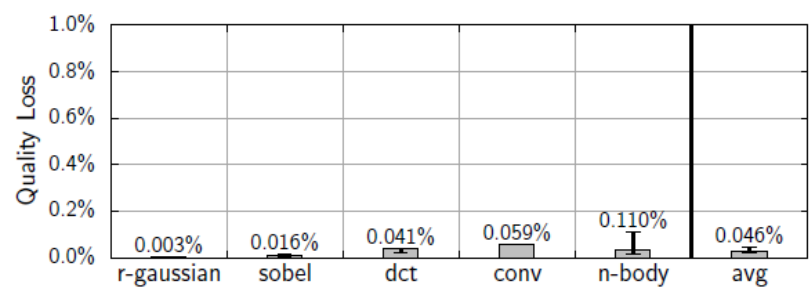
GRATER: GA-based optimization of data types

- Sensitivity analysis performed to find safe-to-approximate variables (AV) in OpenCL kernel.
- Encoding: *n* integers specifying precision (i.e. data type) of *n* variables from AV.
- Objective: to find an approximate kernel that minimizes the resource utilization on FPGA while meeting the target quality.









Lotfi A. et al.: GRATER: An Approximation Workflow for Exploiting Data-Level Parallelism in FPGA Acceleration. DATE 2016

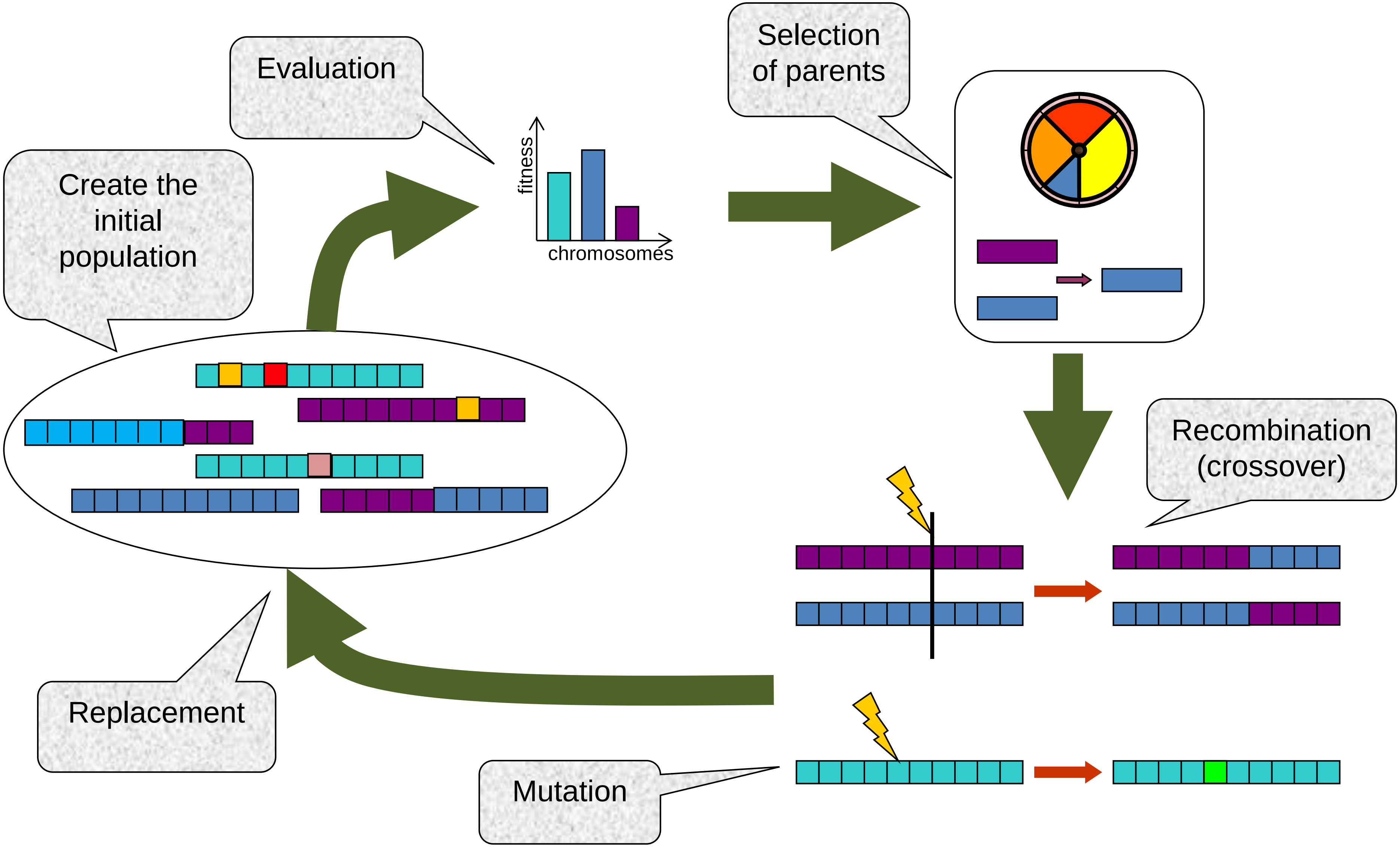
Tutorial Outline – Part II.

- Introduction
- Design automation methods for approximate circuits
  - Classification and overview
  - Circuit parameter estimation
  - Error computation
  - Relaxed equivalence checking
  - Evaluation methodology
- Examples of design automation methods for approximate circuits
  - Minterm complements, SASIMI, AIG rewriting, ABACUS, GRATER
- Evolutionary algorithms, CGP and circuit optimization
- Applications of CGP-based approximation methods
  - Open-source library of approximate adders and multipliers
  - Approximate TMR
  - Approximate multipliers in neural networks
  - Symbolic error analysis using BDDs/SAT solving in CGP-based tools
  - Approximate image filters
- Conclusions

Evolutionary algorithms: GA, ES, EP, GP, LGP, CGP, …

- The term Evolutionary Algorithm covers various search algorithms that have the following common features:
  - There is a population of candidate solutions (inherent parallelism).
  - New candidate solutions are created using operators inspired in genetics (crossover, mutation).
  - Nothing is expected about the objective (fitness) function.
- Main branches:
  - Genetic Algorithms – GA (Holland ~1973)
  - Evolution Strategies – ES (Rechenberg and Schwefel ~1964)
  - Evolutionary Programming - EP (Fogel ~1962)
  - Genetic Programming – GP (Cramer ~1985, Schmidhuber ~1987, Koza, ~1989)
  - and others such as differential evolution, grammatical evolution, Cartesian genetic programming etc.

Evolutionary algorithms: GA, GP, LGP, CGP, GE …



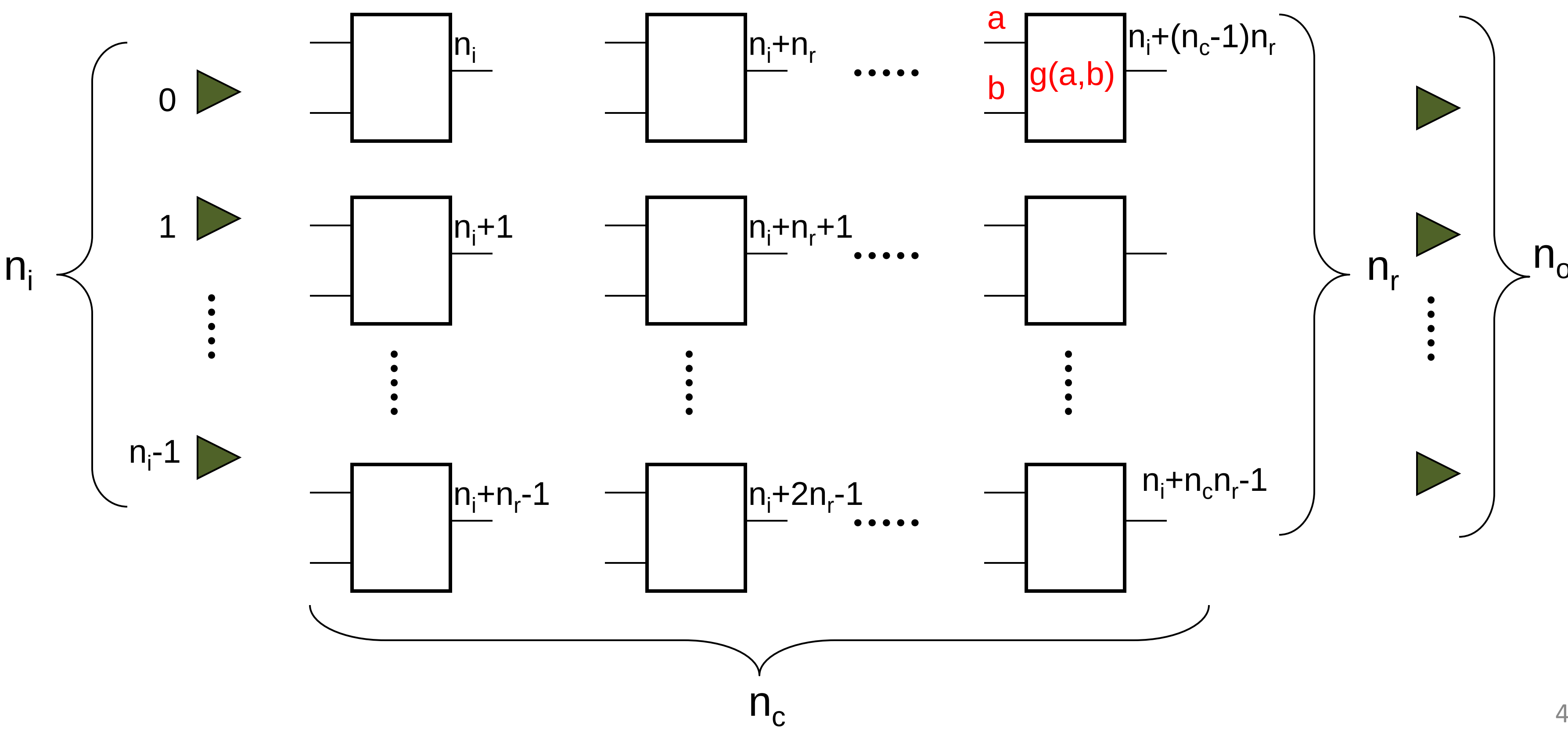
GA chromosome: **binary string**

Cartesian Genetic Programming (CGP) [Miller, 1999]

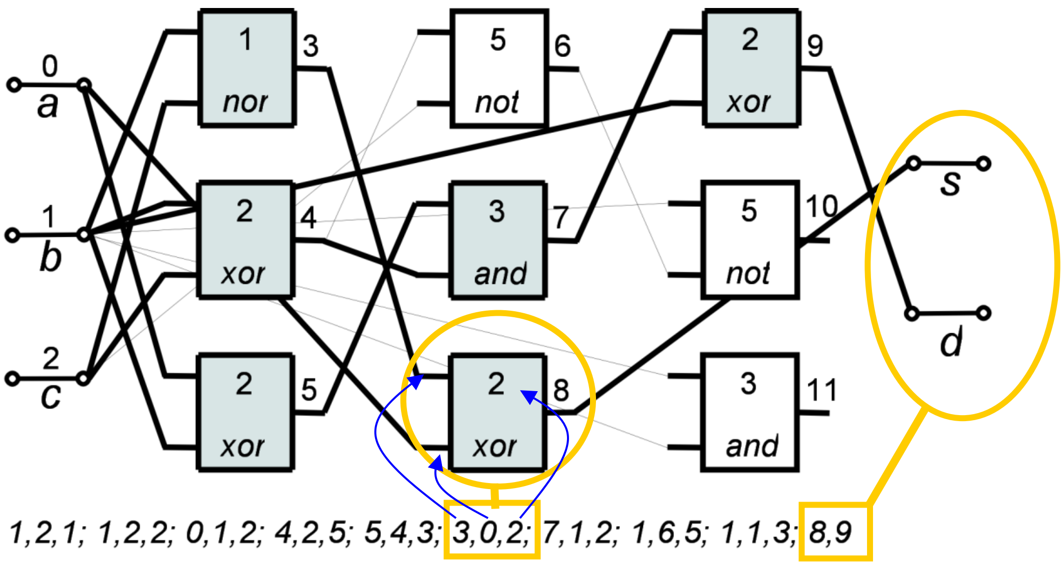
- n_i primary inputs
- n_o primary outputs
- n_c columns
- n_r rows

- n_a inputs of each node
- Γ function set
- L-back parameter

Nodes in the same column are not allowed to be connected to each other.
No feedback!



CGP: Representation for logic networks



- CGP parameters
  - n_r=3 (#rows)
  - n_c = 3 (#columns)
  - n_i = 3 (#inputs)
  - n_o = 2 (#outputs)
  - n_a = 2 (max. arity)
  - L = 3 (level-back parameter)
  - Γ= {NAND$^{(0)}$, NOR$^{(1)}$, XOR$^{(2)}$, AND$^{(3)}$, OR$^{(4)}$, NOT$^{(5)}$}

Genotype (netlist):

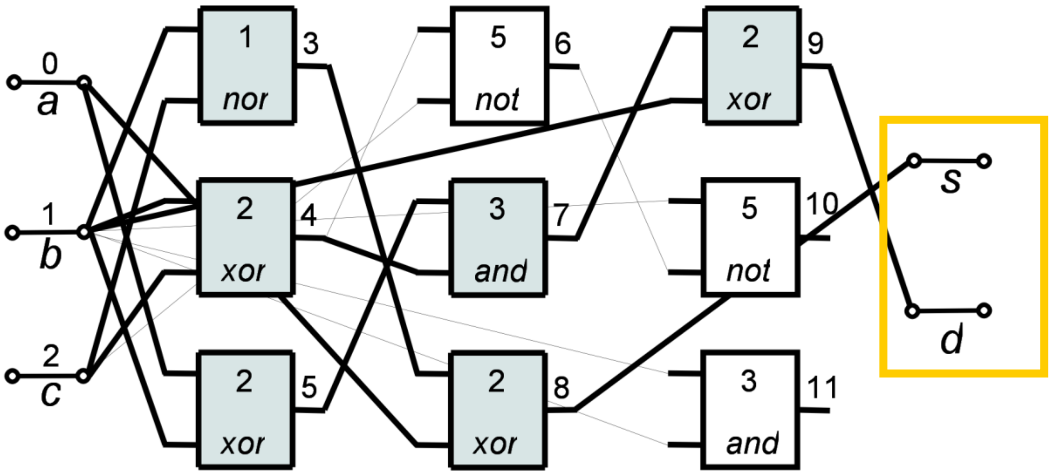
n_a+1 integers per node; n_o integers for outputs;

Constant size: $n_c n_r(n_a + 1) + n_o$ integers

Phenotype (directed acyclic graph $\Rightarrow$ circuit):

Variable size; unused nodes are ignored.

CGP: Fitness function for circuit design



1,2,1; 1,2,2; 0,1,2; 4,2,5; 5,4,3; 3,0,2; 7,1,2; 1,6,5; 1,1,3; 8,9

Specification (1-bit adder), target table:

a	b	c	d	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

=> fitness = 16

a	b	c	d	s
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=> fitness = 10

Typical fitness function (circuit functionality):

$$f = \sum_{i=1}^{K} HD(y_i, w_i)$$

The number of test vectors

Hamming distance (between circuit and desired response)

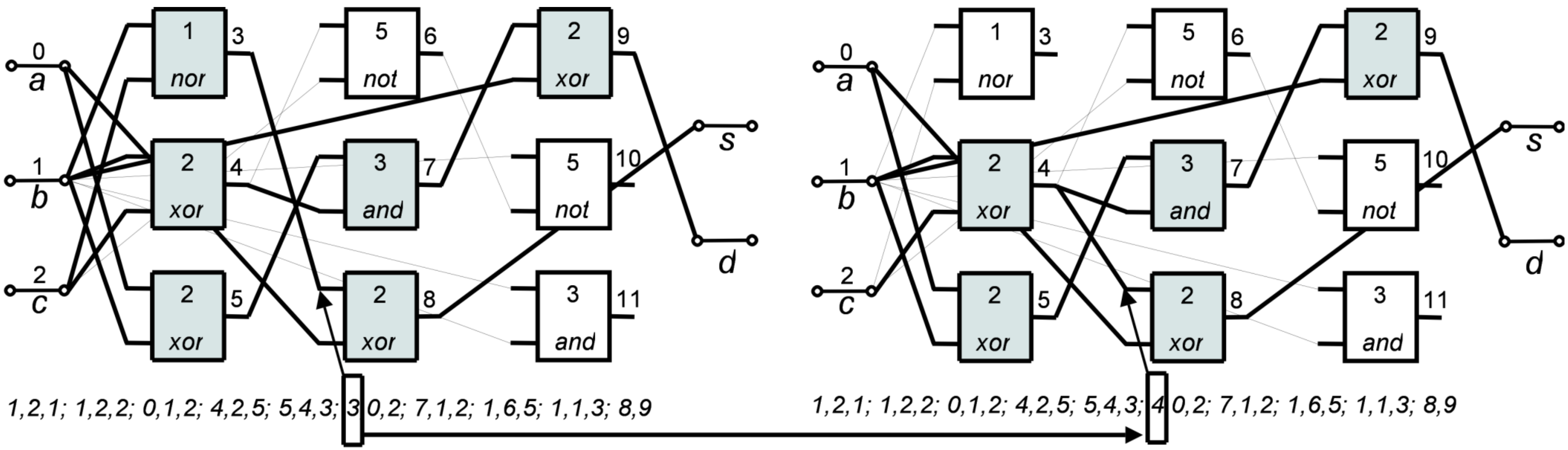
$K = 2^{\text{inputs}}$ for combinational circuits. Not scalable!!!

Additional objectives:

- area (the number of gates)
- delay
- power consumption etc.

CGP: Mutation-based search

- Mutation: Randomly select *h* integers and replace them by randomly generated (but legal) values:



(a) mutation (b)

a	b	c	d	s
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=> fitness = 10

a	b	c	d	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

=> fitness = 16
(for a full adder)

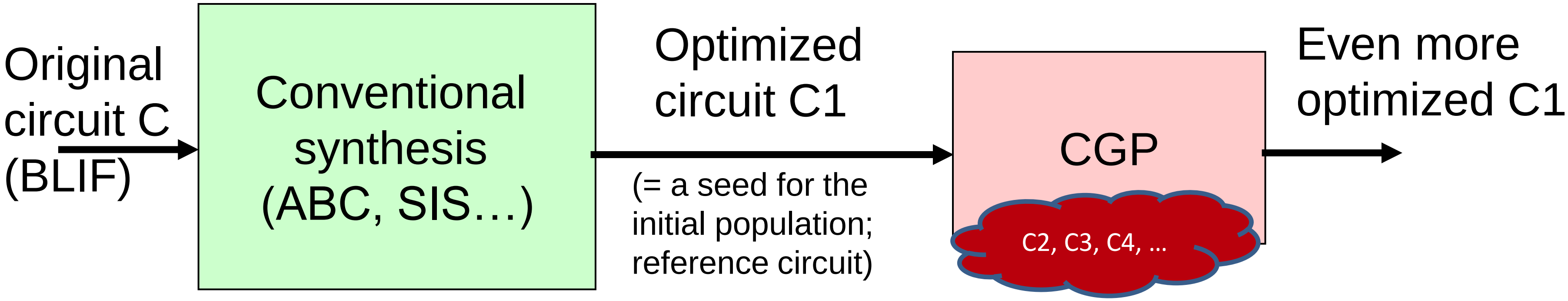
CGP: Search algorithm (1 + λ)

Algorithm 1: CGP

Input: CGP parameters, fitness function
Output: The highest scored individual p and its fitness

$P \leftarrow$ randomly generate population; // or use conventional designs
EvaluatePopulation(P); $p \leftarrow$ highest-scored-individual(P);
while ⟨*terminating condition not satisfied*⟩ **do**
    $\alpha \leftarrow$ highest-scored-individual(P);
    if *fitness*(α) $\geq$ *fitness*(p) **then**
        $p \leftarrow \alpha$;
    $P \leftarrow$ create λ offspring of p using mutation;
    EvaluatePopulation(P);
return p, fitness(p);

CGP for optimization of complex circuits

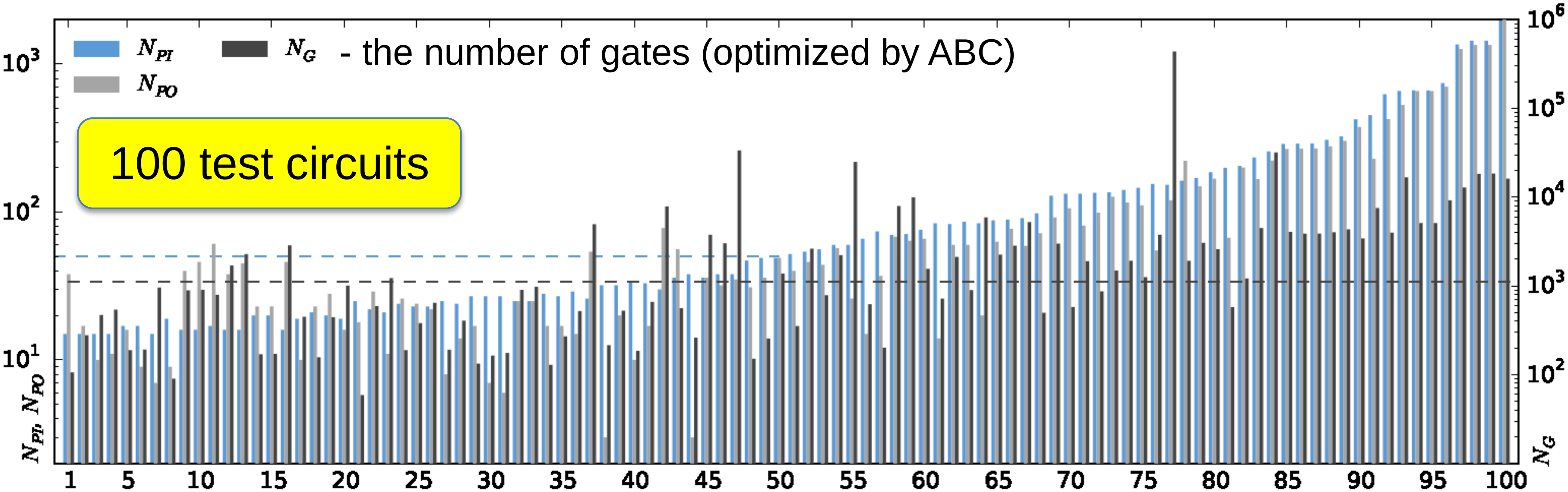


- SAT solver is used to decide whether candidate circuit C_i and reference circuit C1 are functionally equivalent.
  - If so, then fitness(C_i) = the number of gates in C_i;
  - Otherwise: discard C_i.

Vasicek, Sekanina: Genetic Programming and Evolvable Machines 12(3), 2011

CGP with SAT solver (no approximation)

SAT solver is called only if the circuit simulation performed for a small subset of vectors has indicated no error in the candidate circuit.



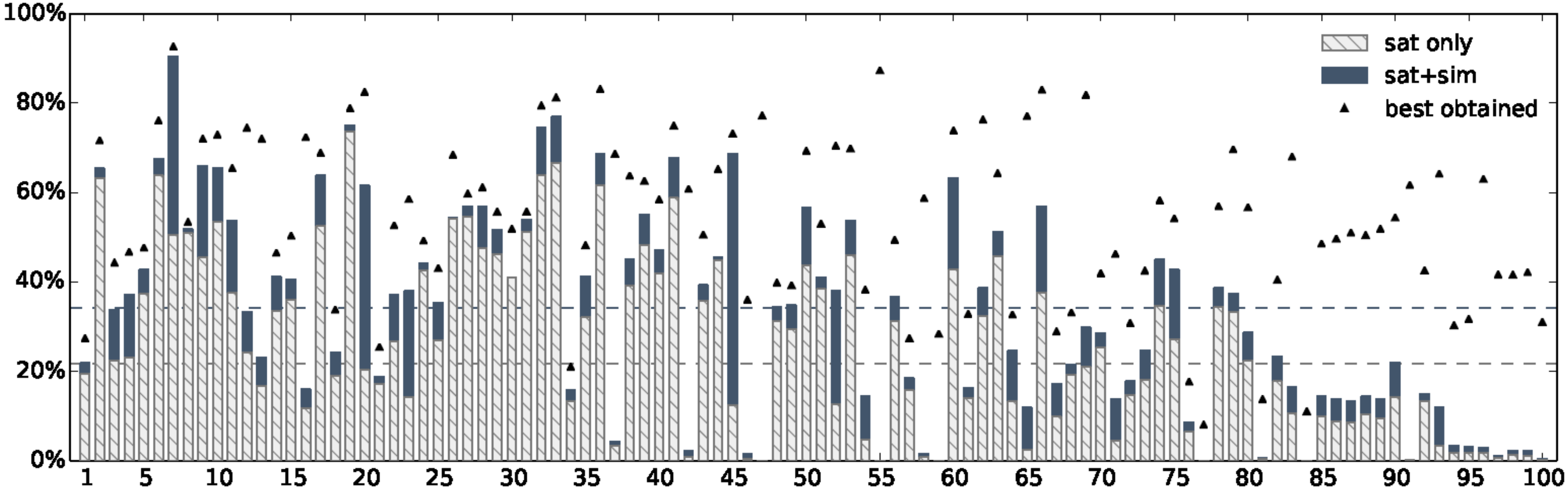
100 combinational circuits (≥15 inputs) - IWLS2005, MCNC, QUIP benchmarks

Heavily optimized by ABC

1: alcom (N_G = 106 gates; N_{PI} = 15 inputs; N_{PO} = 38 outputs)

100: ac97ctrl (N_G = 16,158; N_{PI} = 2,176; N_{PO} = 2,136)

CGP with SAT solver (no approximation)



CGP + SAT solver + circuit simulation
Y-axis: Gate reduction w.r.t. ABC after 15 minutes, 34% on average
▲ Gate reduction w.r.t. ABC after 24 hours

Properly optimize before doing approximations!

Vasicek Z.: EuroGP 2015

Tutorial Outline – Part II.

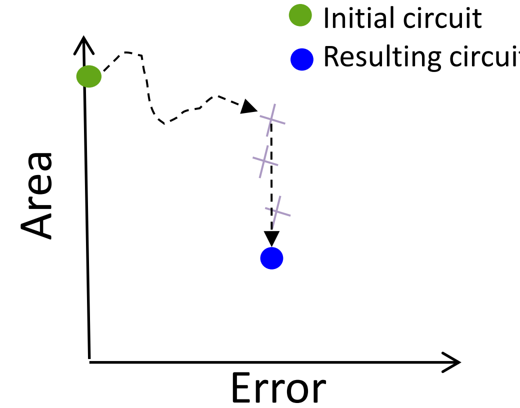
- Introduction
- Design automation methods for approximate circuits
  - Classification and overview
  - Circuit parameter estimation
  - Error computation
  - Relaxed equivalence checking
  - Evaluation methodology
- Examples of design automation methods for approximate circuits
  - Minterm complements, SASIMI, AIG rewriting, ABACUS, GRATER
- Evolutionary algorithms, CGP and circuit optimization
- Applications of CGP-based approximation methods
  - Open-source library of approximate adders and multipliers
  - Approximate TMR
  - Approximate multipliers in neural networks
  - Symbolic error analysis using BDDs/SAT solving in CGP-based tools
  - Approximate image filters
- Conclusions

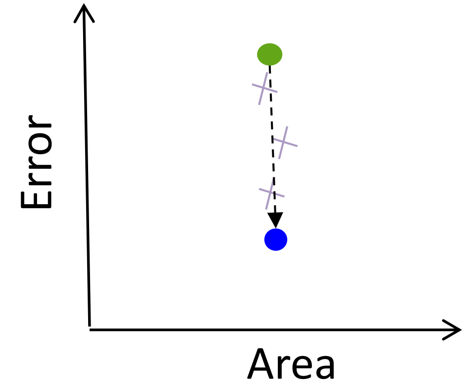
Why EA in approximate computing?

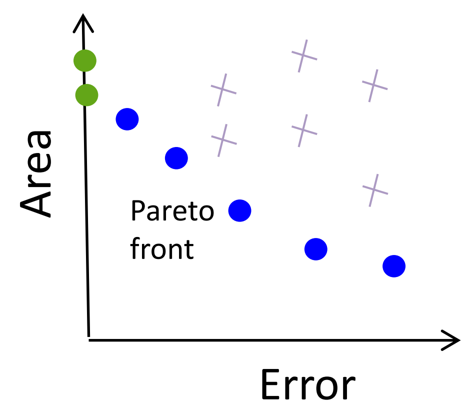
- In approximate computing, partially working solutions are sought.
- In EA, partially working solutions are improved.
- EAs are excellent in multi-objective design and optimization.
- Constraints can easily be handled.
- EA can be seeded with the original code (circuit).
- EA is easy to implement and parallelize.

CGP for circuit (functional) approximation

- Error-oriented (single-objective) method
  - CGP gradually degrades a fully functional circuit until a circuit with a <u>required error</u> is obtained. Then, the area (and so power consumption) is minimized for this error.
- Resources-oriented (single-objective) method
  - CGP is used to minimize the error, but only limited resources (components) are provided, insufficient for constructing a fully functional circuit.
- Multi-objective optimization
  - All target parameters are optimized together.







Library of approximate 8 bit adders and multipliers

- Parallel multi-objective CGP:
  - CGP + Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Hrbáček, GECCO 2015]
  - Parallel implementation: vectorized, multi-threaded, multiple islands (computer cluster employed)
- Constraints: worst case error, worst case relative error
- Initial population: a set of fully working conventional circuits
- Fitness: mean relative error, power consumption, delay

$$f_{\mathrm{mre}} := \frac{\sum_{\forall i} \frac{\left| O^{(i)}_{\mathrm{orig}} - O^{(i)}_{\mathrm{approx}} \right|}{\max(1, O^{(i)}_{\mathrm{orig}})}}{2^{N_{\mathrm{i}}}}$$

$O^{(i)}$ is the i-th circuit output

$i = 1 \ldots 2^{Ni}$

Target circuits - Inputs: N_i = 16; Outputs: N_o = 9 (adders), 16 (multipliers)

CGP parameters

- Population size: 500 candidate circuits
- Generations: 100k
- Mutation: 5%
- Parallel CGP: 10 islands exchanging circuits every 1000 generations (120 cores)
- CGP array: 1 x 200 nodes (adders), 1 x 1000 nodes (mult.)
- CGP function set (180/45 nm technology library):
  - BUF, INV, AND2, OR2, XOR2, NAND2, NOR2, XNOR2, NAND3, NOR3, MUX2, AOI21,OAI21, Full Adder, Half Adder
  - 3-input/2-output nodes used

CGP: Initial population

Architecture	Power	Area	Delay
Ripple-Carry Adder	**100.00%**	**100.00%**	**100.00%**
Carry-Select Adder	201.18%	174.78%	61.15%
Carry-Lookahead Adder	414.74%	334.78%	61.99%
HVTA (Brent-Kung)	286.00%	201.74%	68.52%
HVTA (Han-Carlson)	286.00%	201.74%	68.52%
HVTA (Kogge-Stone)	371.48%	257.39%	59.77%
HVTA (Sklansky)	305.07%	215.65%	60.45%
TA (Brent-Kung)	282.99%	201.74%	67.25%
TA (Han-Carlson)	295.74%	212.17%	61.87%
TA (Knowles)	362.25%	257.39%	59.94%
TA (Kogge-Stone)	342.20%	243.48%	57.68%
TA (Ladner-Fischer)	282.99%	201.74%	67.25%
TA (Sklansky)	298.34%	212.17%	57.84%

13 conventional 8-bit adders

TA = Tree Adder

HVTA = Higher Valency Tree Adder

Architecture	Power	Area	Delay
Ripple-Carry Array	**100.00%**	**100.00%**	**100.00%**
Carry-Save Array using RCA	102.30%	100.00%	71.16%
Carry-Save Array using CSA	108.42%	106.16%	62.03%
Wallace Tree using RCA	104.29%	107.39%	68.91%
Wallace Tree using CLA	116.10%	148.48%	51.26%
Wallace Tree using CSA	120.12%	122.35%	53.28%

6 conventional 8-bit multipliers

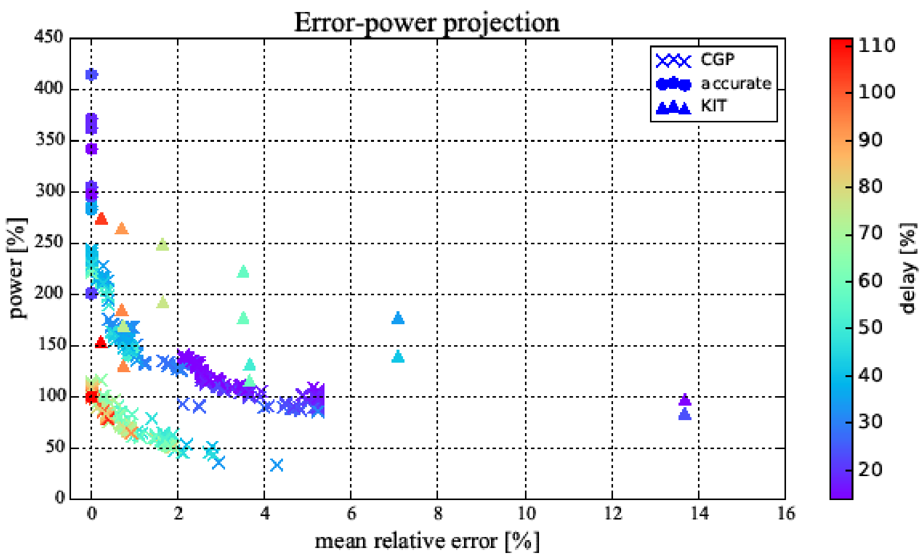
RCA = Ripple-Carry Adder

CSA = Carry-Save Adder

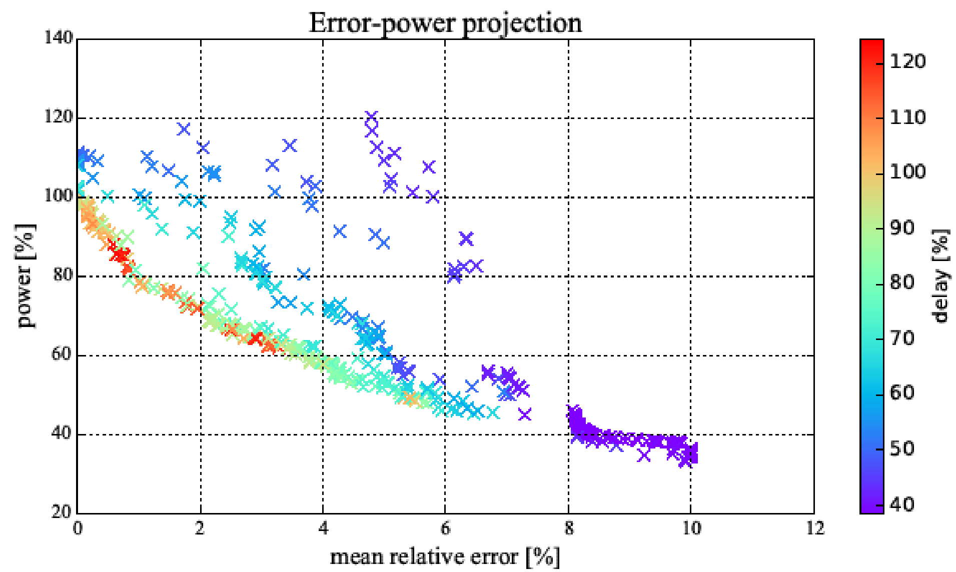
CLA = Carry-Lookahead Adder

Library of 8-bit approx. adders and multipliers

- Comprehensive library of approximate arithmetic circuits
  - 430 non-dominated adders (evolved from 13 accurate adders)
  - 471 non-dominated multipliers (evolved from 6 accurate multipliers)



Approximate adders
(100% is Ripple-Carry Adder)



Approximate multipliers
(100% is Ripple-Carry Array Multiplier)

V. Mrazek, R. Hrbacek, Z. Vasicek, L. Sekanina: EvoApprox8b: Library, DATE 2a017, p. 1-4
KIT: M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel: A low latency generic accuracy configurable adder, DAC 2015, pp. 86:1–86:6.

Library of 8-bit approx. adders and multipliers

Approximate adders (430), exact adders (43)

Circuit	Est. area	Est. delay	Est. power	Nodes	HD	MAE	MSE	MRE	WCE	WCRE	EP	OPS
add8_000	820 μm^2	1.314 ns	194.31 μW	10	138496	1.71875	6.00000	0.88 %	7	100 %	71.875 %	Verilog C Matlab
add8_001	2040 μm^2	0.718 ns	681.20 μW	42	0	0.00000	0.00000	0.00 %	0	0 %	0.000 %	Verilog C Matlab
add8_002	836 μm^2	1.282 ns	194.75 μW	13	140448	1.69531	5.85938	0.88 %	7	100 %	71.484 %	Verilog C Matlab
add8_003	912 μm^2	0.379 ns	266.66 μW	20	192640	9.64844	138.25000	5.21 %	24	100 %	96.875 %	Verilog C Matlab
add8_004	708 μm^2	1.213 ns	205.54 μW	9	134528	1.37500	3.25000	0.75 %	5	200 %	76.562 %	Verilog C Matlab

....

Approximate multipliers (471), exact multipliers (28)

Circuit	Est. area	Est. delay	Est. power	Nodes	HD	MAE	MSE	MRE	WCE	WCRE	EP	OPS
mul8_000	9224 μm^2	3.015 ns	4933.22 μW	137	176134	98.52710	27520.00000	1.99 %	820	560 %	86.490 %	Verilog C Matlab
mul8_001	5200 μm^2	3.566 ns	2524.84 μW	91	310752	239.95550	108908.84375	5.36 %	1671	100 %	98.169 %	Verilog C Matlab
mul8_002	6715 μm^2	2.086 ns	2789.47 μW	132	339806	329.88147	207883.35278	6.70 %	2193	700 %	98.482 %	Verilog C Matlab
mul8_003	4172 μm^2	1.963 ns	1816.06 μW	79	376002	624.46875	679898.57422	10.00 %	2911	700 %	98.984 %	Verilog C Matlab
mul8_004	5034 μm^2	1.944 ns	1893.73 μW	104	382402	639.22653	709554.15625	9.76 %	3143	253 %	99.071 %	Verilog C Matlab

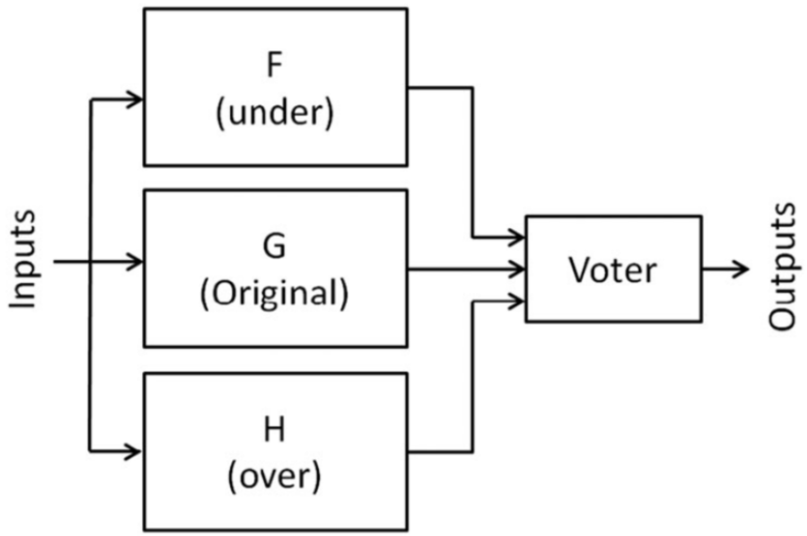
....

Synthesis results for 45 nm and 180 nm technology (Synopsys Design Compiler)
7 error metrics
New: 12-bit multipliers online, 16 – 32-bit multipliers completed

http://www.fit.vutbr.cz/research/groups/ehw/approxlib/

Click here to DOWNLOAD

Approximate circuits in TMR

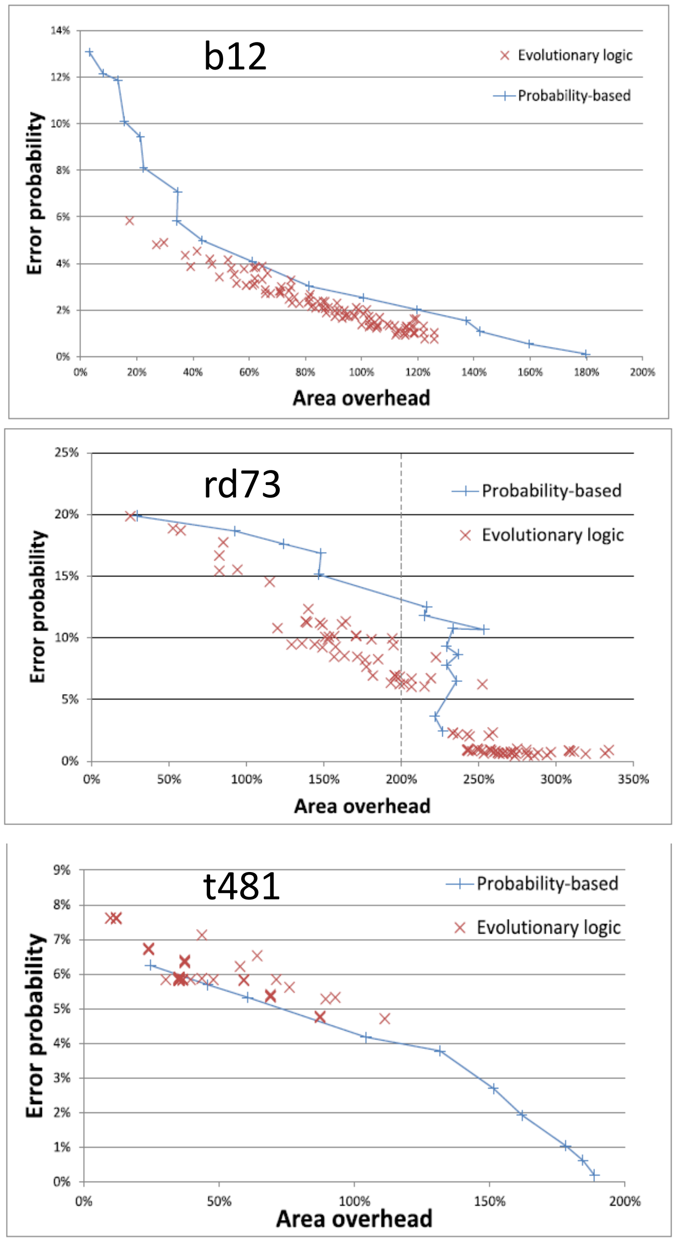


Incorrect subspace: The subset of input vectors for which the correct circuit and approximate circuit produce different outputs.

F (under-approximation):
Incorrect subspace is a subset of the on-set. 1 → 0 errors are produced

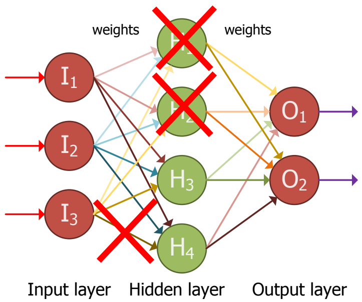
H (over-approximation)
Incorrect subspace is a subset of the off-set. 0 → 1 errors are produced

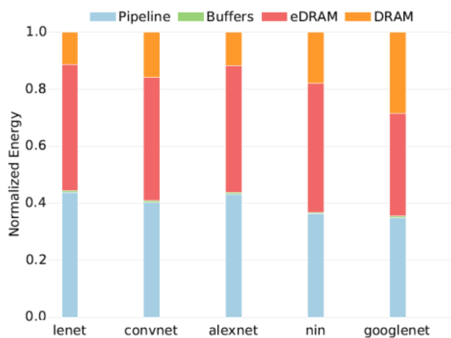
At most one of the circuits is allowed to produce an incorrect output for any input vector.



SÁNCHEZ-CLEMENTE, A., J., ENTRENA, L., HRBACEK, R. a SEKANINA, L. Error Mitigation using Approximate Logic Circuits: A Comparison of Probabilistic and Evolutionary Approaches. IEEE Transactions on Reliability. 2016, 65(4), p. 1871-1883

Energy-efficient implementation of ANNs



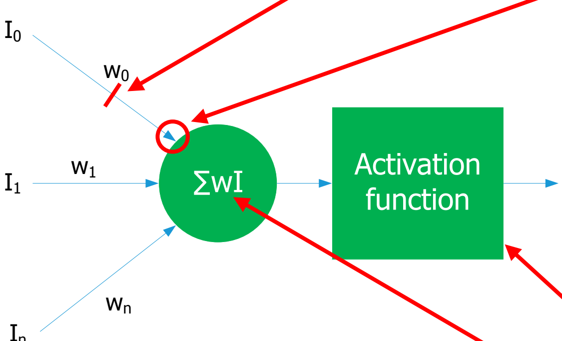


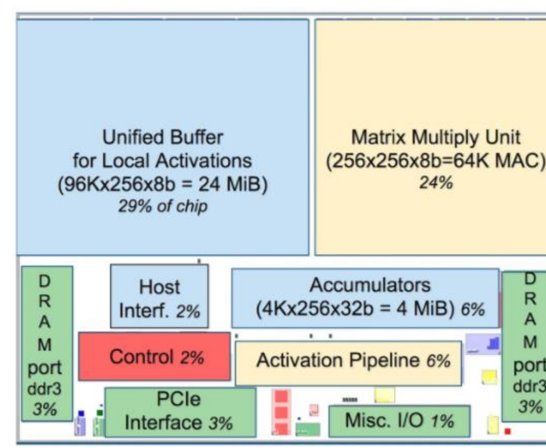
[Judd et.al. WAPCO'16]

Approximations proposed:

- Pruning – weights and neurons
- Data compression (weights)
- Memory – approximate cells and Load/Store

- Datapath
  - Reducing data bit-width
  - Multiplication (~45% of total power)
    - Multiplierless multiplication
    - Weights: {-1, 1}
  - Activation function
  - Sum function





Google TPU: 24% for MAC

Energy-efficient implementation of ANNs

MNIST dataset classification: 32x32 – 100 – 10 MLP network (classification accuracy 94.16% with accurate implementation). We introduced an approximate multiplier by adding a jitter function $\Delta(a,b)$, resulting in a 5.2% error for multiplication.

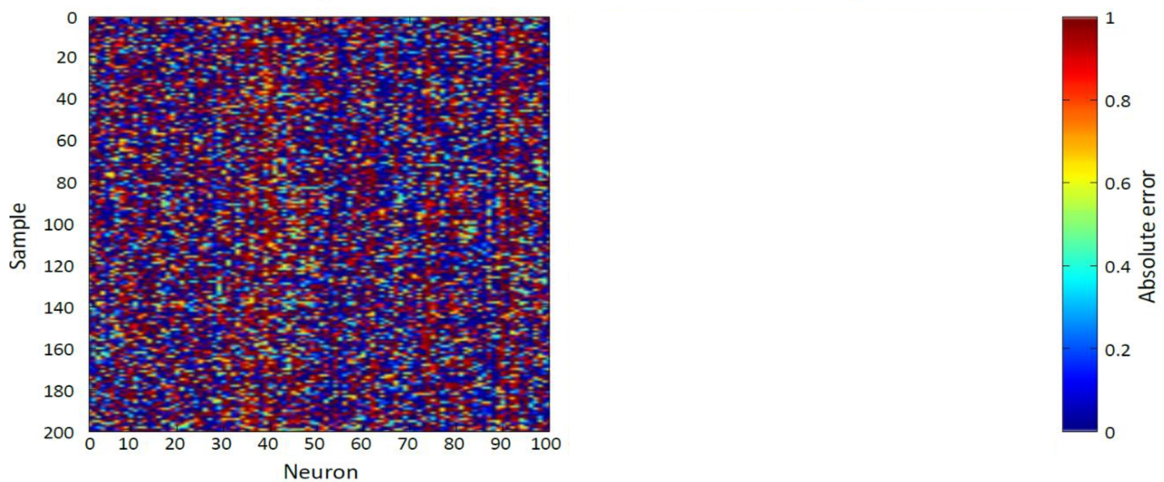
Scenario A:

- Multiplication
  $$m(a,b) = a \cdot b + \Delta(a,b)$$
- Classification accuracy :
  10.77%

Scenario B:

- 80% of multiplications are by 0
- Multiplication
  $$m'(a,b) = \begin{cases} 0 & if\ a = 0 \vee b = 0 \\ a \cdot b + \Delta(a,b) & otherwise \end{cases}$$
- Classification accuracy : 94.20%

Output error of neurons in the hidden layer



(a)

Mrazek, Sarwar, Sekanina, Vasicek, Roy: "Design of power-efficient approximate multipliers for approximate artificial neural networks," ICCAD 2016

CGP in approx. multiplier design for ANNs

Accurate multiplier – initial circuit (6)

- CSAM RCA, CSAM RCA, RCAM, WTM CLA, WTM CSA, WTM RCA

Allowed errors: $\varepsilon \in \{0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%\}$

CGP parameters

- $n_i \in \{14{,}22\}$; $n_o \in \{14{,}22\}; n_r = 1; 250 < n_c < 780$
- Functions: {NOT, AND, NAND, OR, NOR, XOR, XNOR}
- Error constraints:
  1. $\forall a, b: |m(a, b) - a * b| \leq \varepsilon \cdot 2^{n_o}$
  2. $\forall a: m(a, 0) = m(0, a) = 0$
- Fitness function:

$$C(m) = \begin{cases} -GatesCount(m) & if\ constraints\ (1)\ and\ (2)\ met, \\ -\infty & otherwise \end{cases}$$

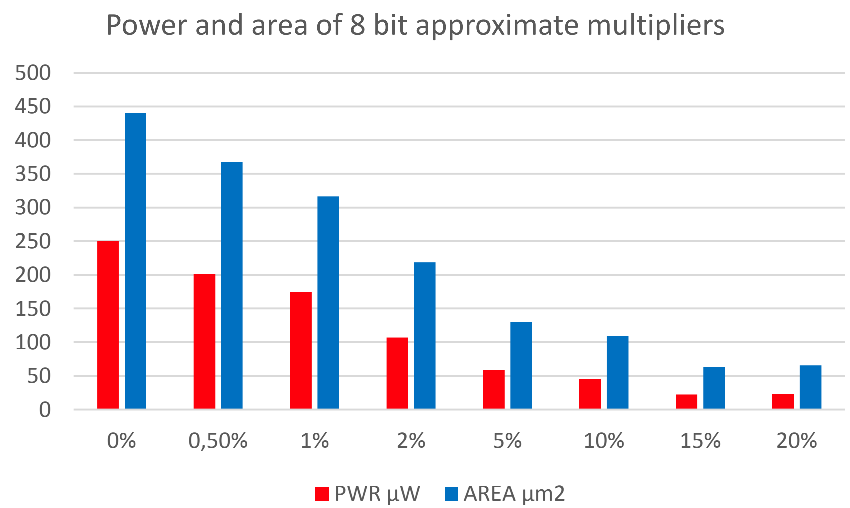
Mrazek, Sarwar, Sekanina, Vasicek, Roy: "Design of power-efficient approximate multipliers for approximate artificial neural networks," ICCAD 2016

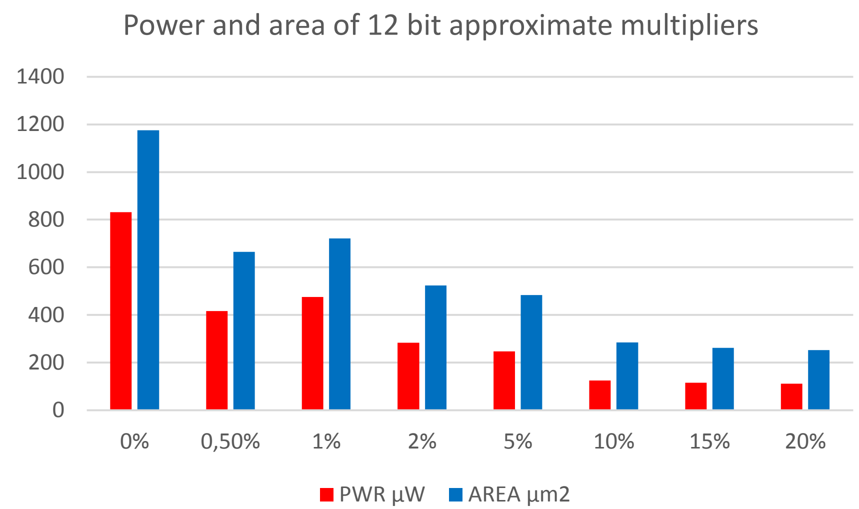
CGP in approx. multiplier design for ANNs

- In total, 852 approximate 7-bit and 11-bit multipliers were evolved by CGP.
- Multipliers were sign-extended using one's complement.
- The 8-bit and 12-bit multipliers were applied in NNs.
- The NNs were retrained with approximate multiplication operation using the backpropagation algorithm.
- Approximate multipliers showing the best trade off between power and accuracy in NN were selected (for different error targets).

Mrazek, Sarwar, Sekanina, Vasicek, Roy: "Design of power-efficient approximate multipliers for approximate artificial neural networks," ICCAD 2016

Evolved approximate multipliers for ANNs





Results of synthesis of sign-extended multipliers with Synopsys DC

45 nm technology

Timing:

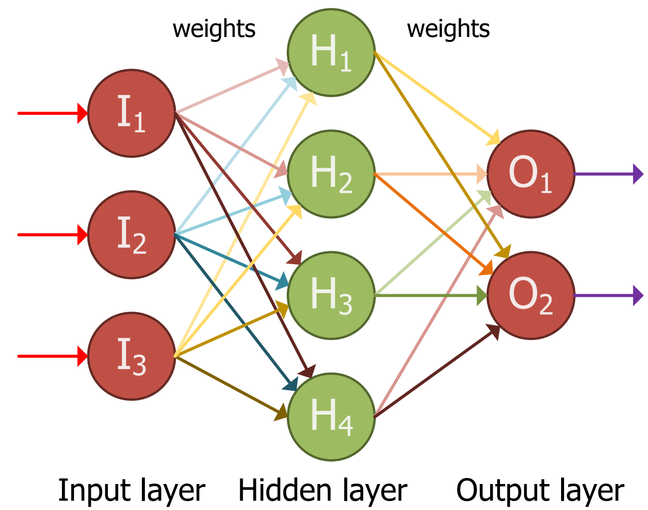
- 8-bit multipliers: 2.5 GHz
- 12-bit multipliers: 2 GHz

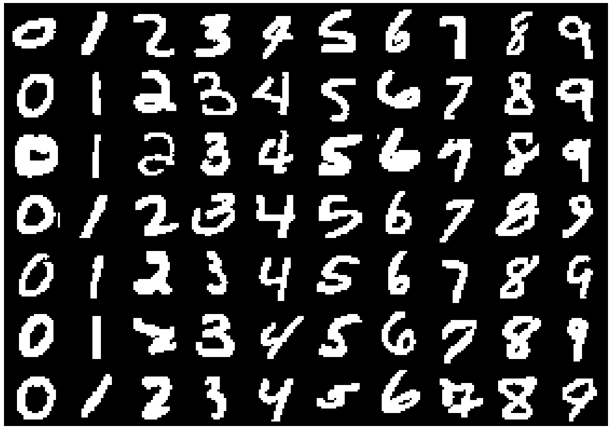
Accurate multiplier was implemented in Verilog using standard * arithmetic operator

Mrazek, Sarwar, Sekanina, Vasicek, Roy: "Design of power-efficient approximate multipliers for approximate artificial neural networks," ICCAD 2016

Energy-efficient implementation of ANNs: MLP

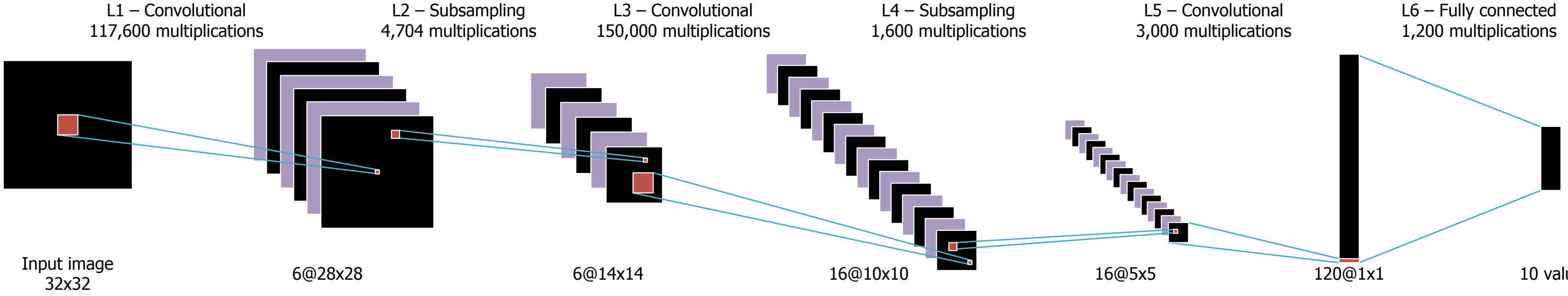
- Handwritten number dataset (dataset used for benchmarking)
- Fully connected MLP network
- 28x28 inputs, 300 hidden neurons, 10 outputs
- 60k training images
- 10k testing images
- More than 238k multiplications for approximation
- Initial classification accuracy:
  - 8b: 97.67%
  - 12b: 97.70%



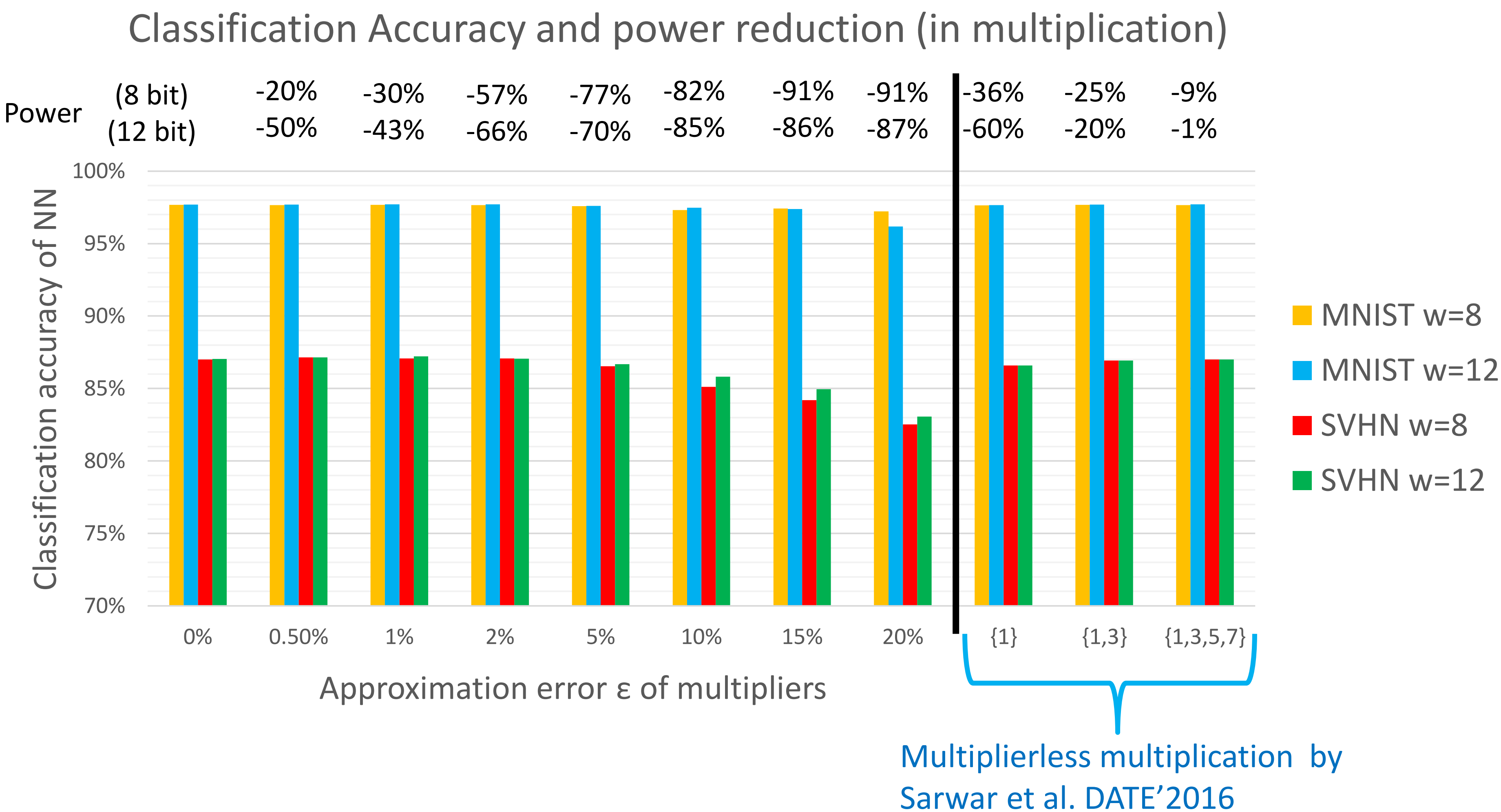


Energy-efficient implementation of ANNs: LeNet

- Complex real-world problem
- Convolutional LeNet NN
- 278,104 multiplications in 6 layers
- 73k training images
- 26k testing images
- Approximation introduced in L1,L3,L5 and L6 layers
- Initial classification accuracy:
  - 8b: 86.85%
  - 12b: 86.90%



Energy-efficient implementation of ANNs: Summary



Mrazek, Sarwar, Sekanina, Vasicek, Roy: "Design of power-efficient approximate multipliers for approximate artificial neural networks," ICCAD 2016

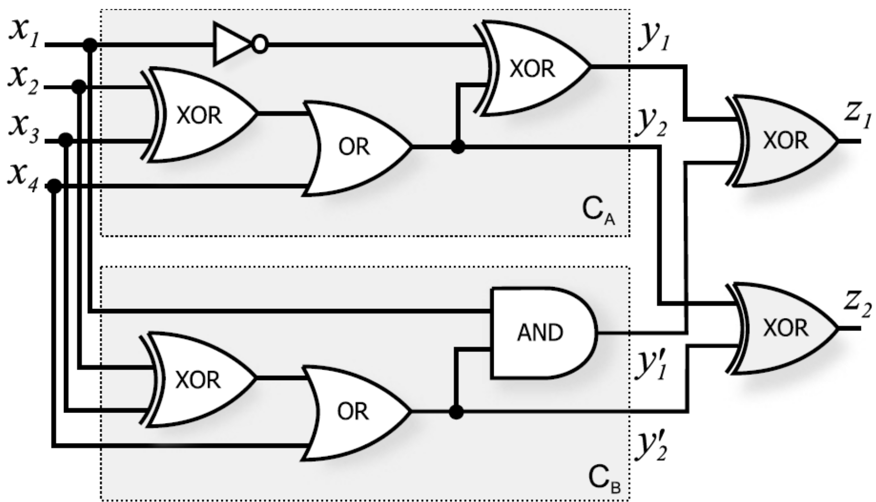
Circuit approximation with CGP and BDD

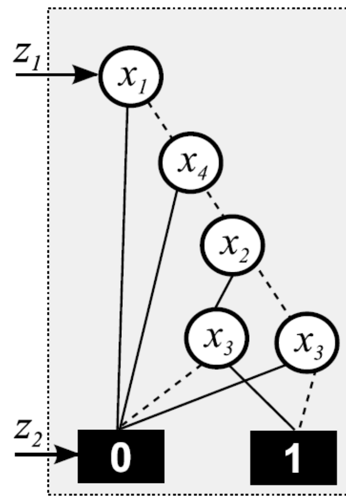
- Three criteria
  - relative area, delay and error
  - Error is the average Hamming distance (10 target error values E_i = 0.1 … 0.9 %)
- CGP parameters
  - Rows = 1; Columns = # of gates in the original circuit
  - 5 mut./chromosome, $\lambda = 5$, 30 min/run, 10 independent runs
  - Function set (relative area): and (1.333), or (1.333), xor (2.0), nand (1.0), nor (1.0), xnor (2.0), buf (1.333), inv (0.667)
- Two stages:
  - Find a circuit showing E_i, but a small (< 5%) imperfection tolerated
  - weight fitness (error / area / delay): $(w_e; w_a; w_d) = (0.12; 0.5; 0.38)$

    (but the error still kept under 5% of E_i)
- 16 benchmark circuits

Vasicek Z., Sekanina L.: Evolutionary Design of Complex Approximate Combinational Circuits. Gen. Prog. and Evol. Mach. 17(2), 2016

Hamming distance using BDDs





SatCount (*f*) – gives the number of input assignments for which *f* is '1'.

$$SatCount(z_1) = 2$$
$$SatCount(z_2) = 0$$

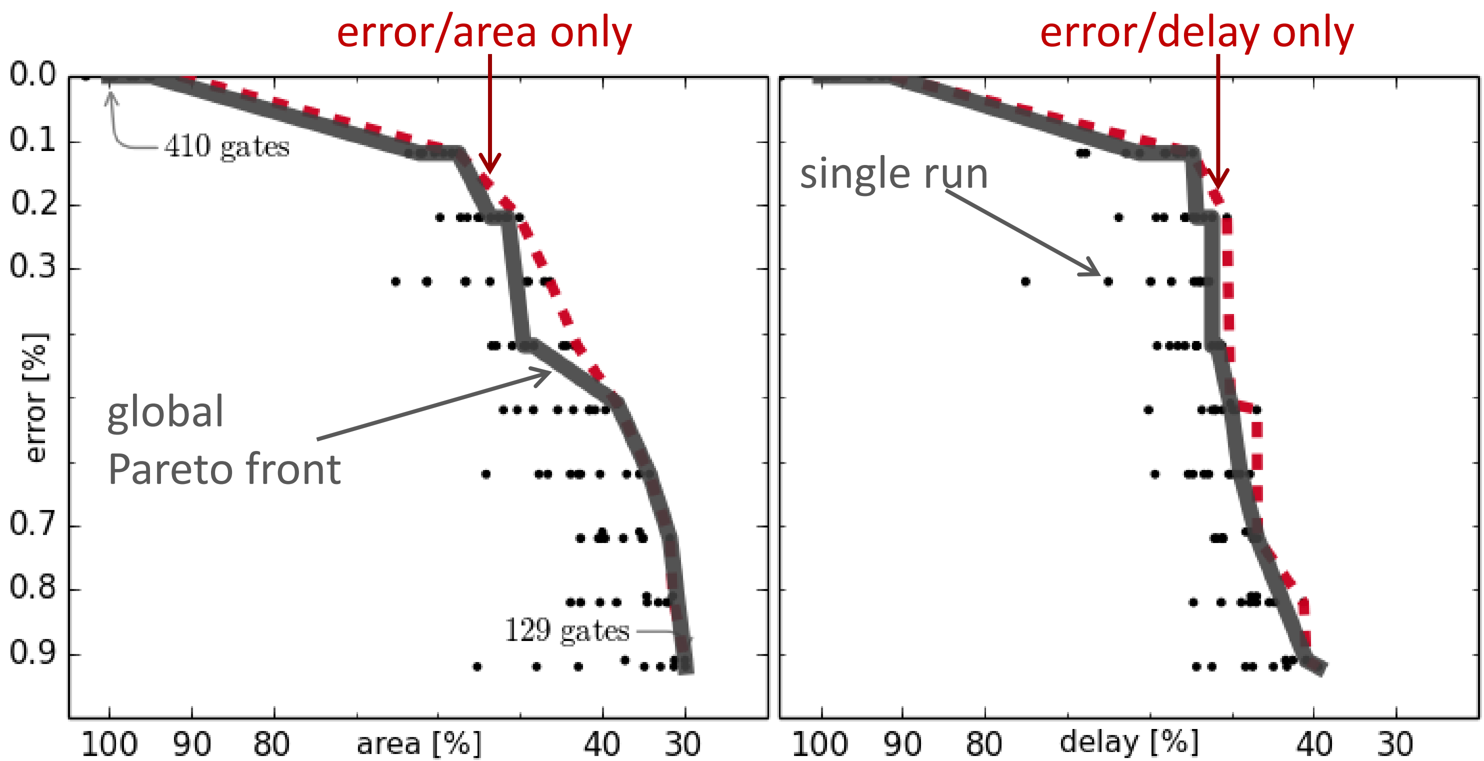
x_1	x_2	x_3	x_4	# combinations
0	0	0	0	1
0	1	1	0	1

- Create ROBDD for the parent circuit C_A, the offspring circuit C_B and the XOR gates.
- Average Hamming distance:

$$e_{HD} = \frac{1}{2^{inputs}} \sum_{i=1}^{outputs} SatCount(z_i)$$

Vasicek Z., Sekanina L.: Evolutionary Design of Complex Approximate Combinational Circuits. Gen. Prog. and Evol. Mach. 17(2), 2016

CGP with BDD in the fitness function: Example 1

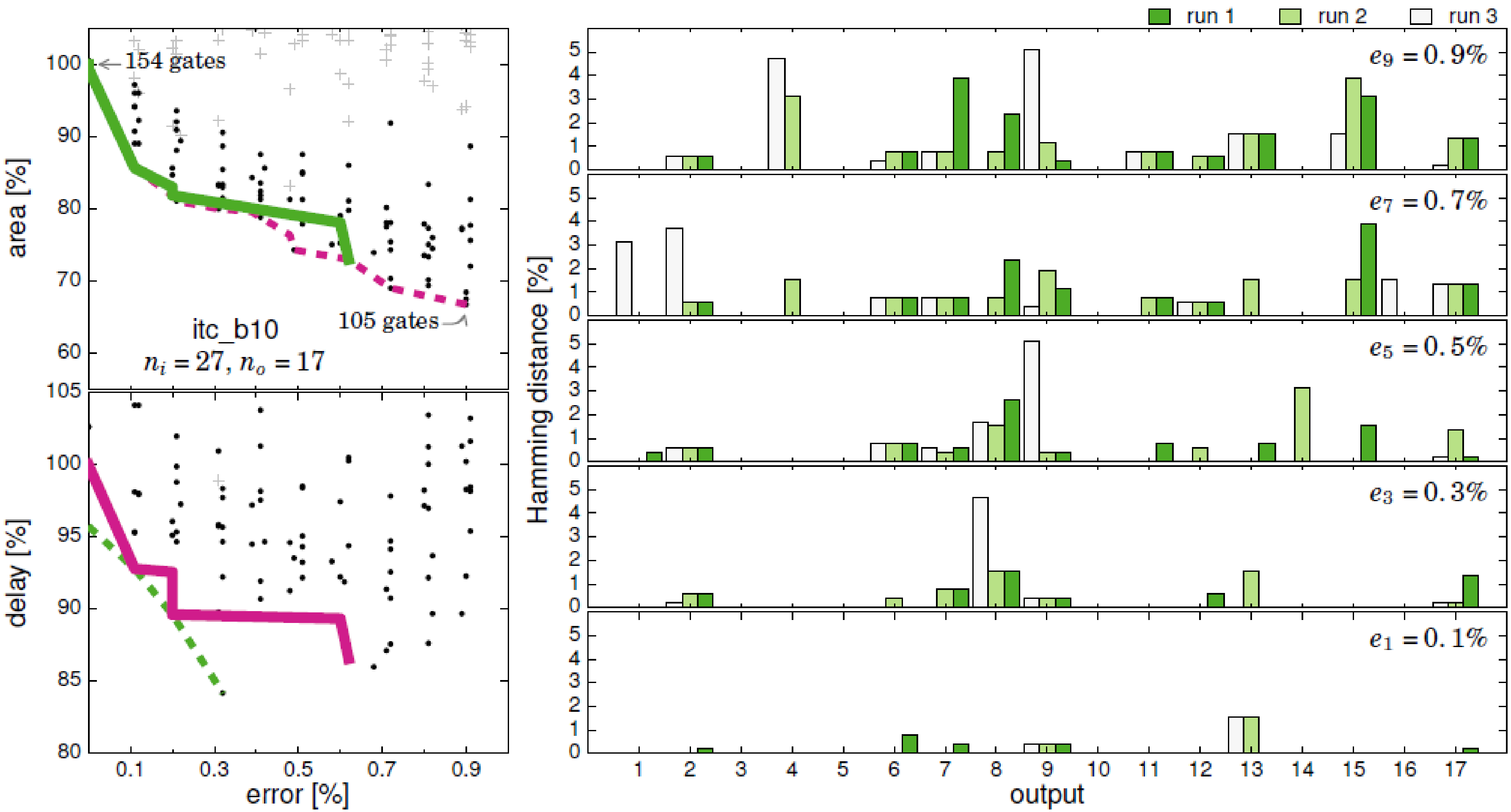


- Clmb (bus interface): 46 inputs, 33 outputs
- Original clmb: 641 gates, 19 logic levels, $|BDD| = 6966$, $|BDD_{opt}| = 627$ (SIFT in 2.3 s)
- Optimized by CGP (no error allowed):
  - Best: 410 gates, 12 logic levels -- in 29 minutes (2.9×10^6 generations)
  - Median: 442 gates, 13 logic levels

Properly optimize before doing approximations!

Vasicek Z., Sekanina L.: Evolutionary Design of Complex Approximate Combinational Circuits. Gen. Prog. and Evol. Mach. 17(2), 2016

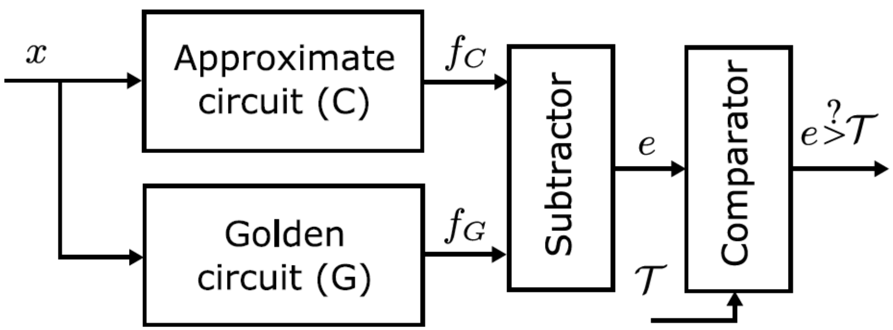
CGP with BDD in the fitness function: Example 2



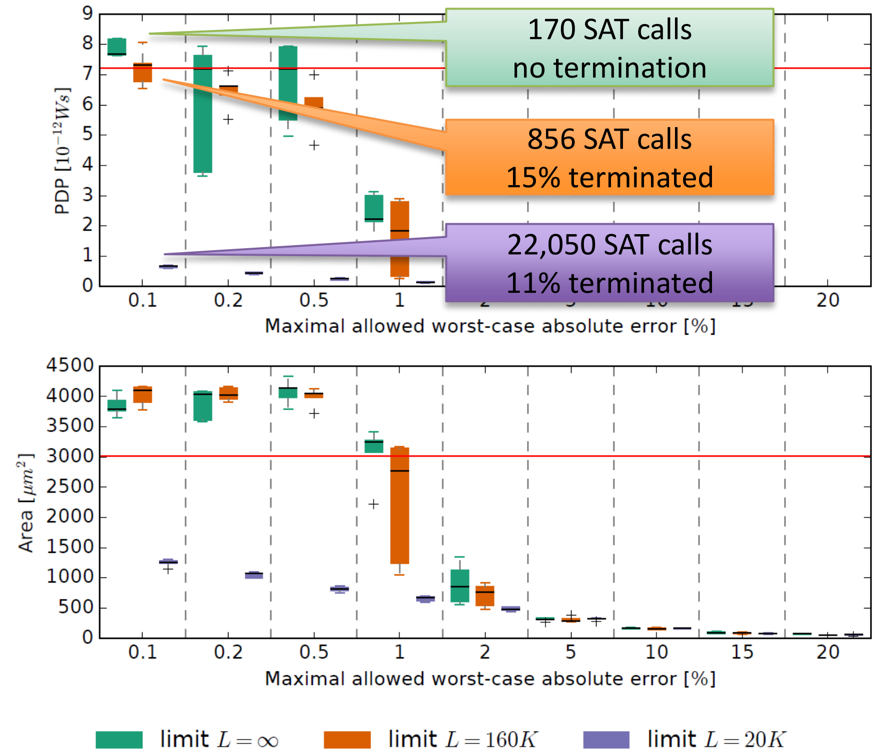
Vasicek Z., Sekanina L.: Evolutionary Design of Complex Approximate Combinational Circuits. Gen. Prog. and Evol. Mach. 17(2), 2016

Approximate circuits: CGP with SAT solver

- Worst case absolute error (WCAE) computation based on SAT solving (for adders and multipliers)
- Improved miter construction
- SAT solver terminated if no decision after spending a predefined time.
- Integrated to ABC



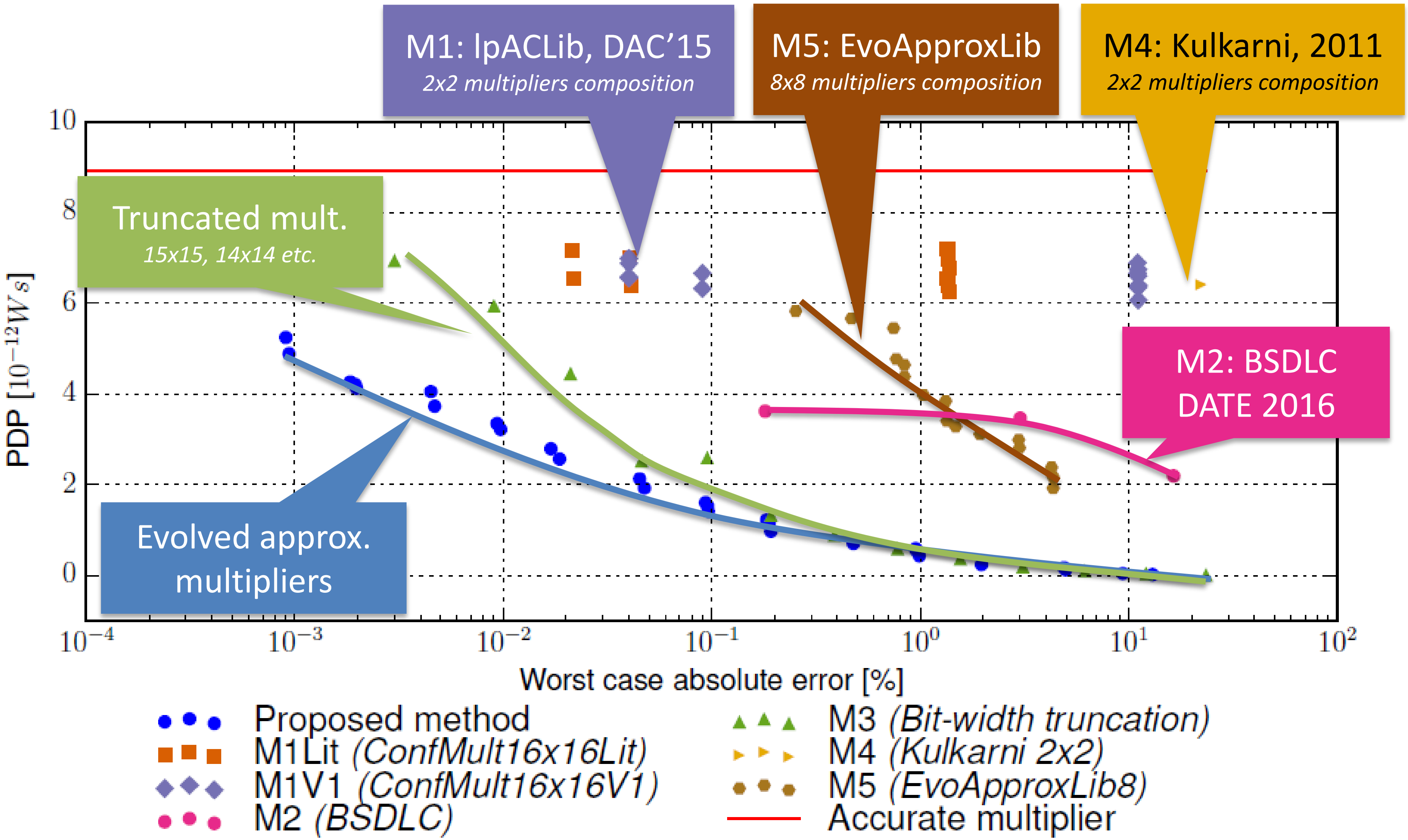
$$fitness(C) = \begin{cases} size(C) & if\ WCAE(C) < \tau \\ \infty & else \end{cases}$$



- 16-bit multipliers for 9 target WCAE
- 2 hours/1 run
- 30 circuits analyzed for each WCAE
- Synopsys Design Compiler, 45 nm
- L is the max. number of conflicts for an AIG node, L = 160 K (~120 seconds) and L = 20 K (~3 seconds).

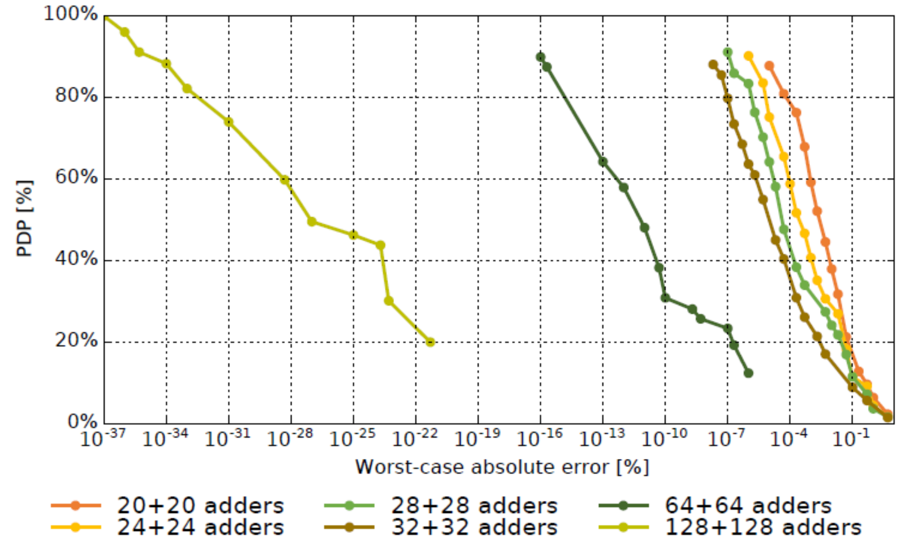
Ceska, Matyas, Mrazek, Sekanina, Vasicek, Vojnar: ICCAD 2017

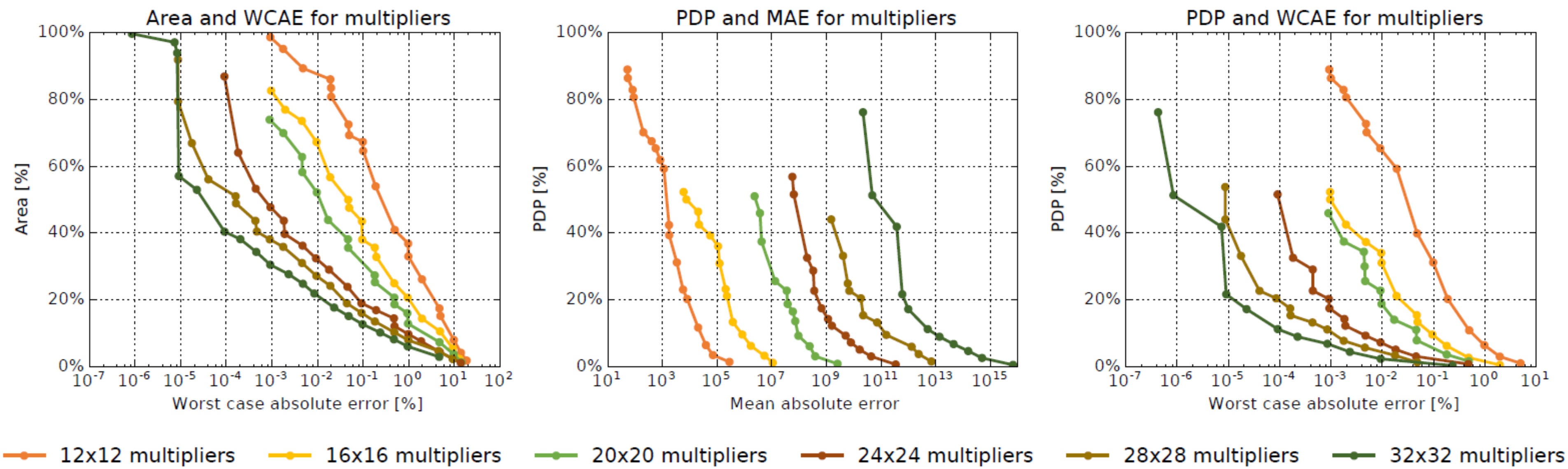
Approximate 16-bit multipliers: Comparison



Ceska, Matyas, Mrazek, Sekanina, Vasicek, Vojnar: ICCAD 2017

Approximate adders and multipliers (exact error)



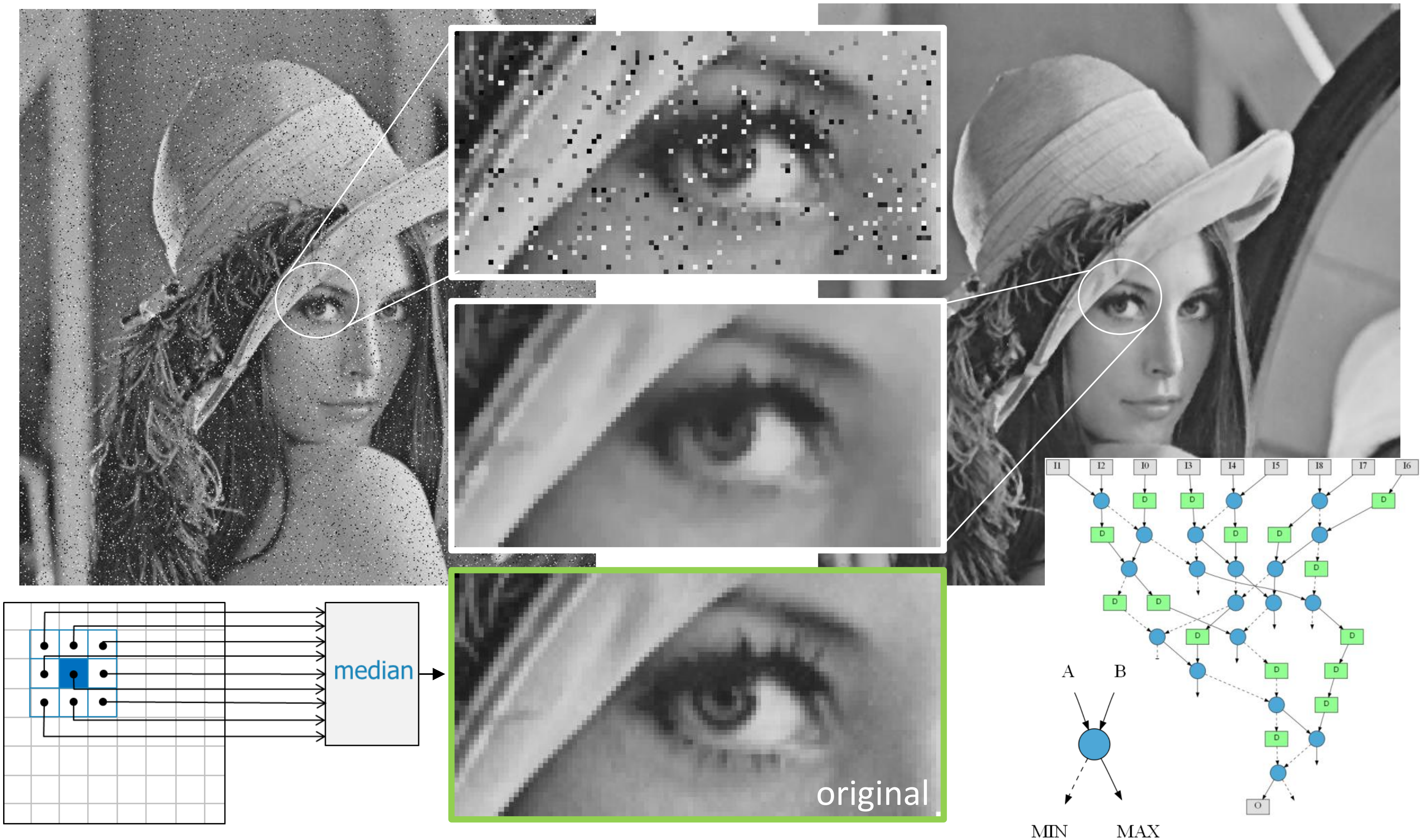


Ceska, Matyas, Mrazek, Sekanina, Vasicek, Vojnar: ICCAD 2017

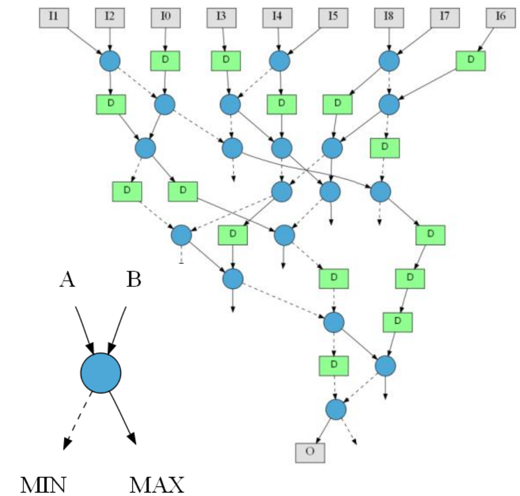
Non-linear image filters

corrupted image
(10% pixels, impulse noise)

filtered image
(9-input median filter)



Non-linear image filters: Approximation strategies



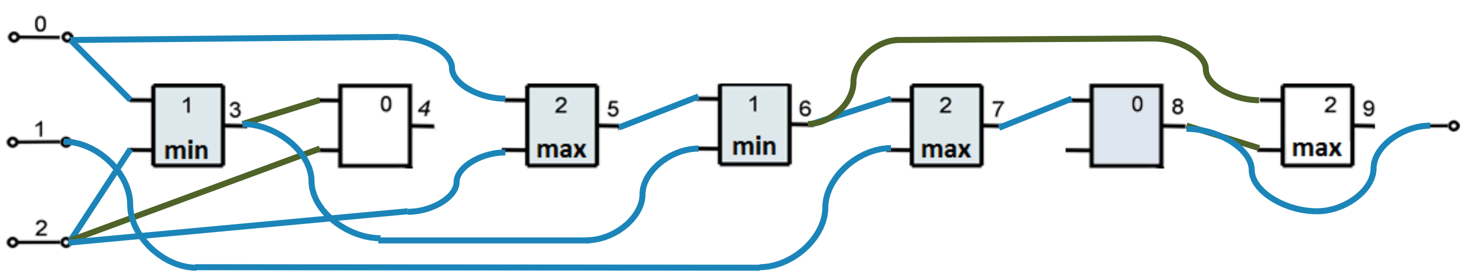
- Approximation of the comparator element
  - MONAJATI et al. Circuits, Systems, and Signal Processing, 34(10), 2015
- Approximation of the network (pruning)
  - CGP used to find a network of N comparators minimizing the error w.r.t. the original median (consisting of K comparators), but resources are limited, i.e. $N < K$.
- Evolutionary image filter design from scratch
  - CGP used to evolve an image filter showing a minimal error and cost. Filters are composed of elementary 2-input functions (min, max, +, logic functions over 8 bits).

Approximate median using CGP

- Median network (consisting of up to N operations) is represented by means of a one-dimensional array of N nodes.
- Each node can act as: identity (0), minimum (1), maximum (2) over 8 bits
- Each candidate solution is encoded using $3N + 1$ integers.
- Fitness function (single objective)

$$error = \sum_{i \in S} \left| O_{candidate}(i) - O_{reference}(i) \right|$$

- Example for a 3-input median:



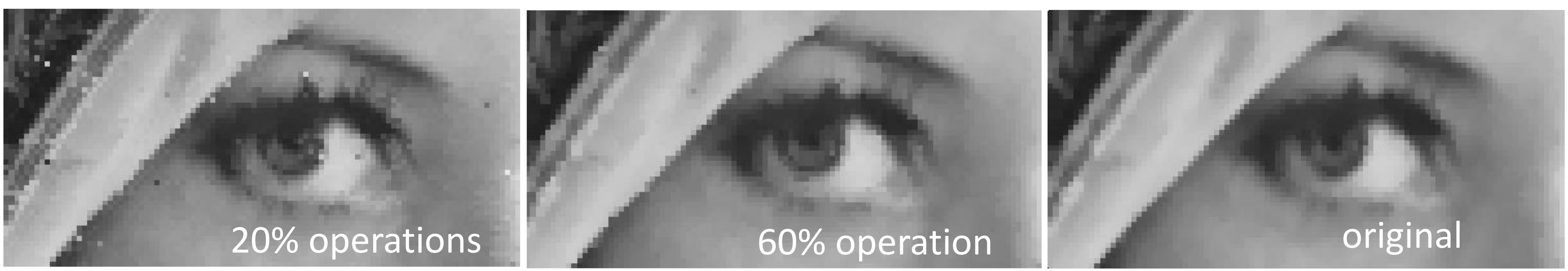
Chromosome: 0, 2, 3; 3, 2, 0; 0, 2, 2; 5, 3, 1; 6, 1, 2; 7, 0, 0; 6, 8, 2; 8

Approximate median using CGP

Experimental setup

- (1+4)-ES, no crossover, 5 % of the chromosome mutated

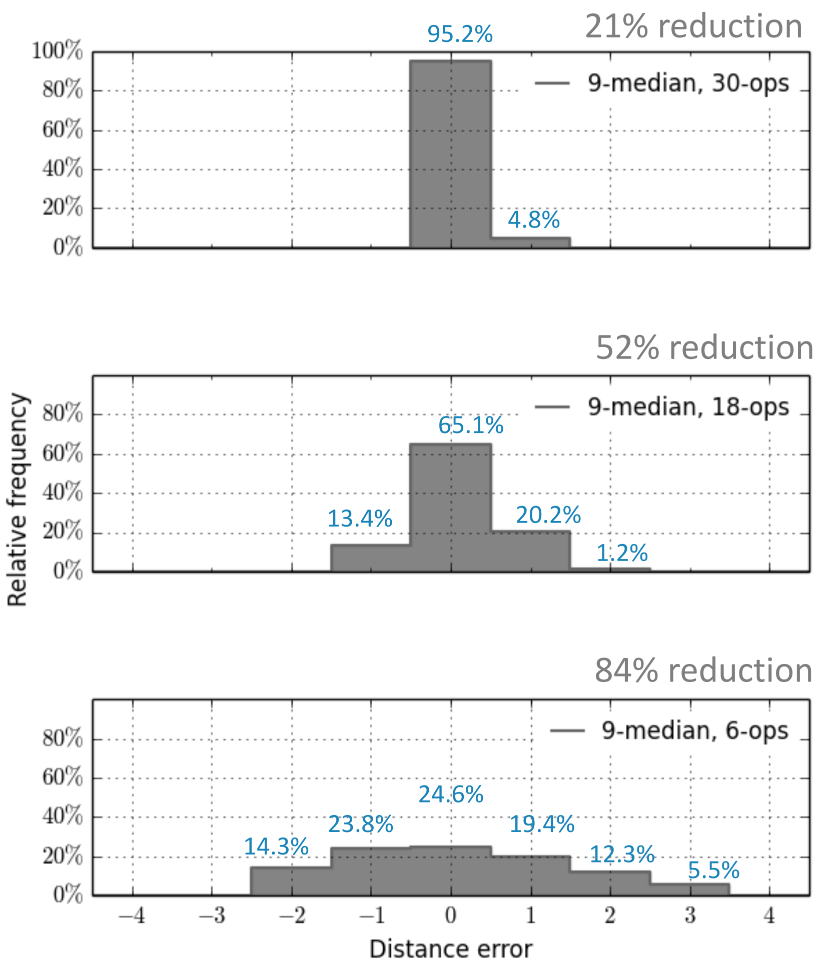
	Median-9	Median-25
Inputs	9	25
Outputs	1	1
Generations	3×10^6 (3 hours)	3×10^5 (3 hours)
Training vectors	1×10^4	1×10^5
Exact solution (K)	38 operations	220 operations
Available nodes (N)	6 – 34 operations	10 – 200 operations



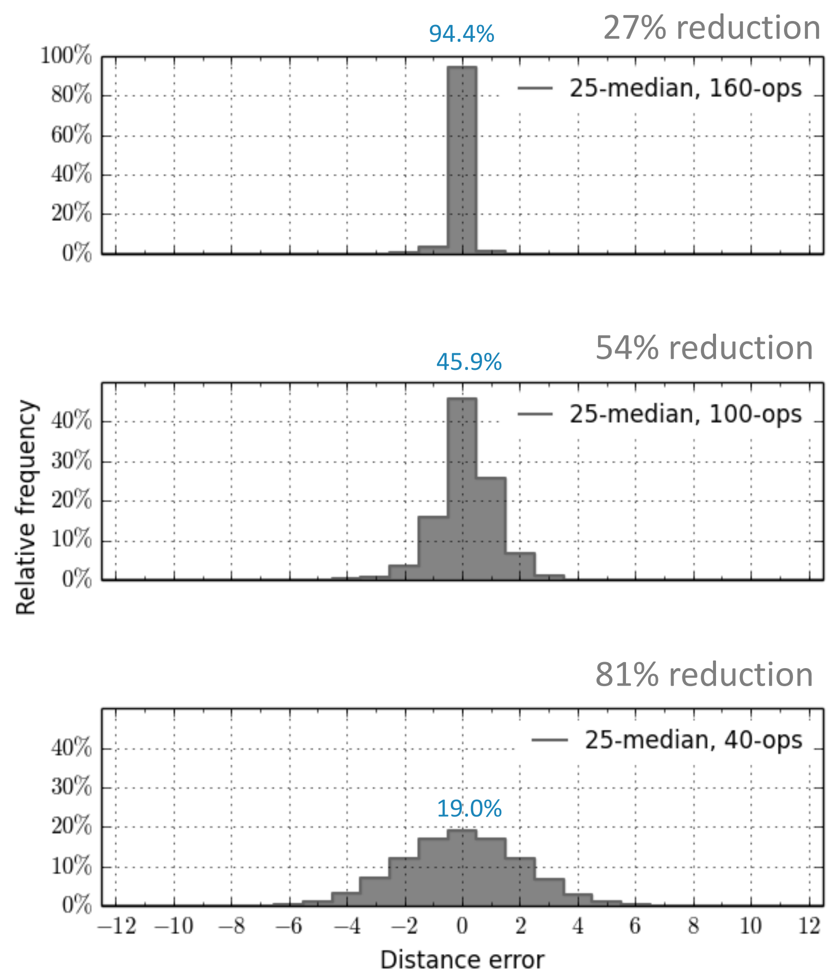
Z. Vašíček and L. Sekanina. *Evolutionary approach to approximate digital circuits design*. IEEE trans. on Evol. Comp. 19(3), 2015

Approximate median: Distance error analysis

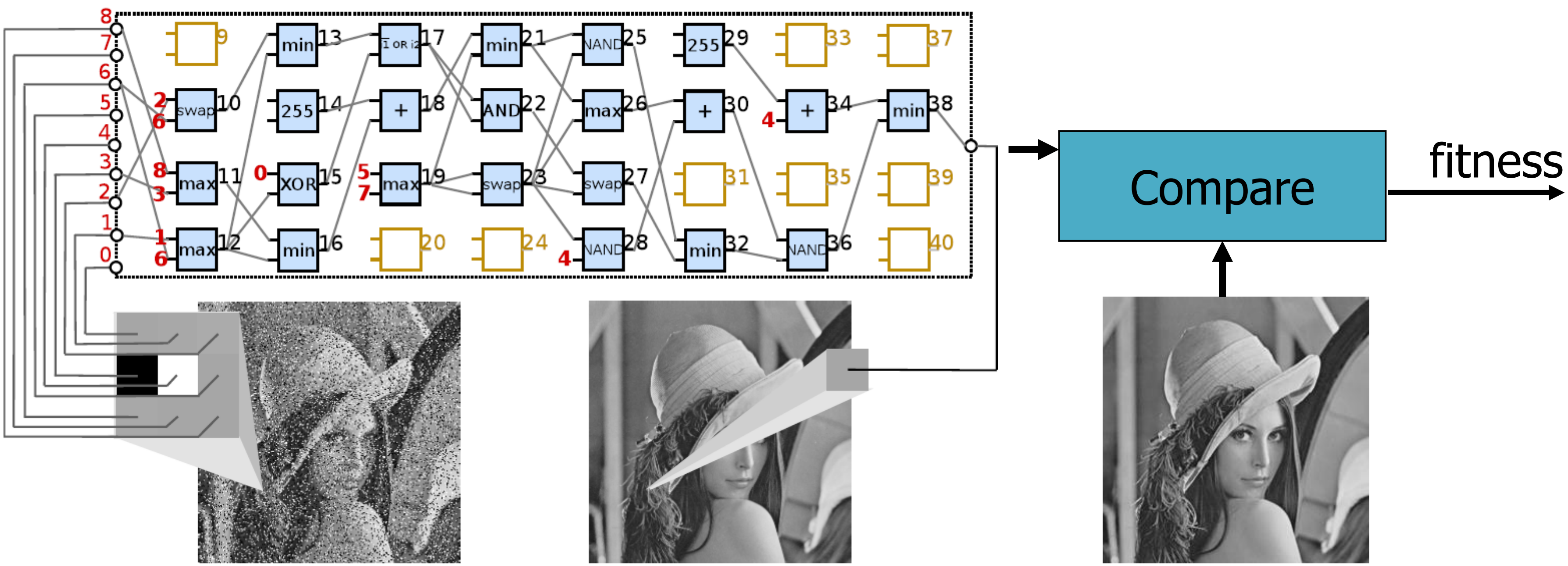
9-input median
fully-working: 38 operations



25-input median
fully-working: 220 operations



Evolutionary design of image filters from scratch



Input image

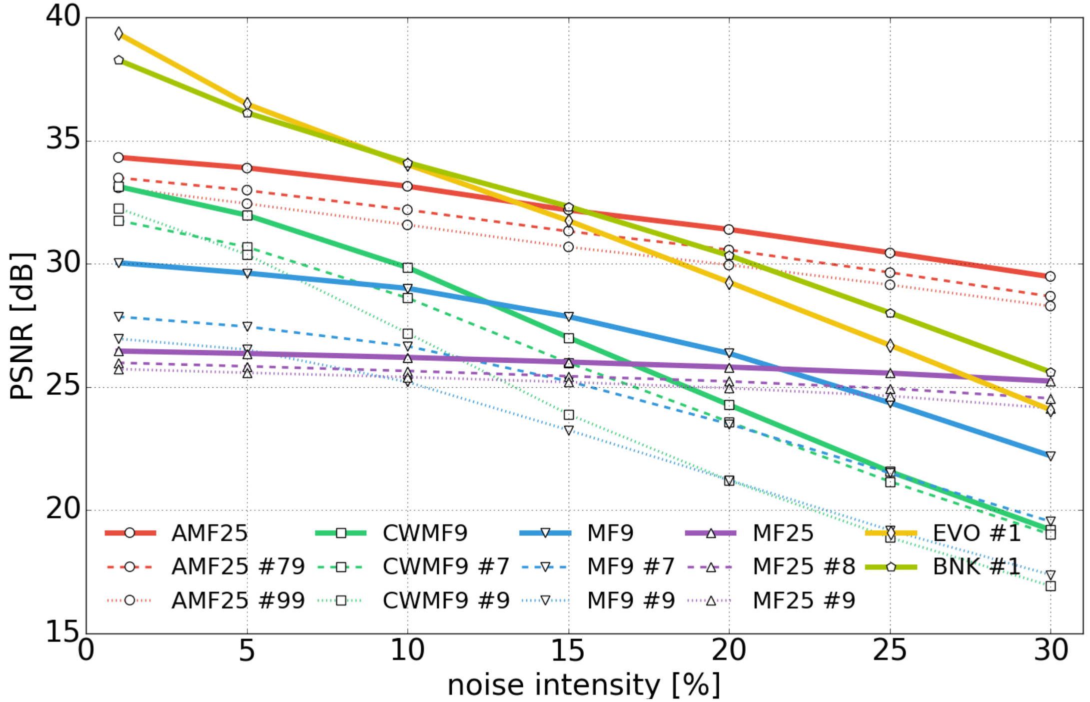
N x M pixels

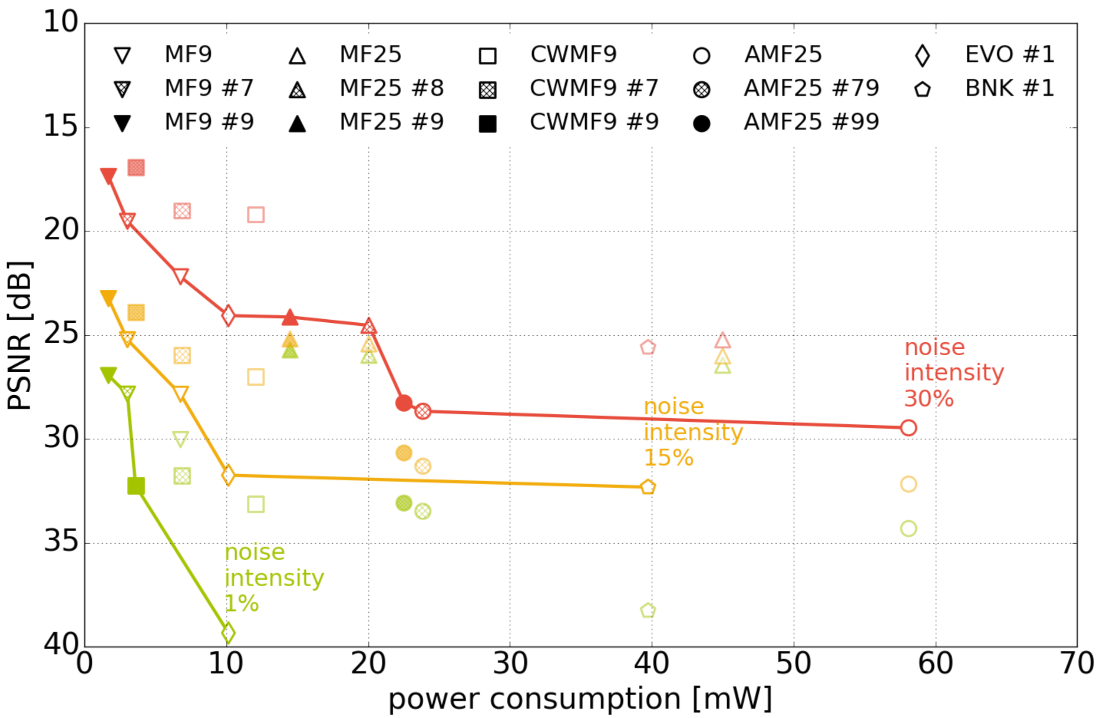
Output image - v

Golden image - w

$$fitness = \sum_{i=1}^{N} \sum_{j=1}^{M} | v(i,j) - w(i,j) |$$

Sekanina L. Image Filter Design with Evolvable Hardware. LNCS 2279, 2002





Comparison of approximate median filters and evolved filters for salt and pepper noise

ΔΔ MF	median filter
O AMF	adaptive median filter
□ CWMF	center weighted median filter
◊ EVO	evolved filter (5x5)
⌂ BNK	bank of 3 evolved filters (5x5)
9	3x3 kernel
25	5x5 kernel
# xy	approximation no. xy

PSNR – mean PSNR on 30 images

Synopsys Design compiler; 45 nm PDK

All filters are pipelined with f_{min} = 1 GHz

Sekanina, Vasicek, Mrazek: Radioengineering 26(3), 2017

Conclusions – Part II

- Design automation methods implementing functional circuit approximation
  - work at various levels (abstract, source code, RTL, gate),
  - use different strategies and heuristics to introduce the approximation (truncation, pruning, component replacement, local re-synthesis, …),
  - evaluate the quality of approximate circuits by means of simulation, probabilistic or formal methods,
  - have not been systematically compared in terms of quality.
- CGP-based methods can provide quite competitive approximate circuits
  - at different levels of abstraction (very flexible representation),
  - with formally proven quality of result (when needed),
  - because the problem can be formulated as a multi-objective one with various constraints and solved by means of a multi-objective approach,
  - but it is a computationally demanding approach.
- Properly optimize before doing approximations!

References

- See references on particular slides
- Selected tutorial and survey papers on Approximate Computing

  - J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in Proc. of the 18th IEEE European Test Symposium. IEEE, 2013, pp. 1–6
  - H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, "Neural acceleration for general-purpose approximate programs," Commun. ACM, 58(1): 105-115, 2015
  - S. Mittal, "A survey of techniques for approximate computing," ACM Computing Surveys, 48(4), 1–34, 2016.
  - Q. Xu, T. Mytkowicz, N. S. Kim. "Approximate Computing: A Survey," IEEE Design and Test, 33(1), 8-22, 2016.
  - L. Sekanina, "Introduction to Approximate Computing". IEEE International Symposium on Design and Diagnostics of Electronic Circuits, DDECS 2016
  - Z. Vasicek, "Relaxed equivalence checking: a new challenge in logic synthesis". IEEE International Symposium on Design and Diagnostics of Electronic Circuits, DDECS 2017

Acknowledgement

- EHW group at Brno University of Technology
  - Zdeněk Vašíček, Michal Bidlo, Roland Dobai
  - Michaela Šikulová, Radek Hrbáček, Vojtěch Mrázek, David Grochol, Miloš Minařík, Jakub Husa, Marek Kidoň, Michal Wiglasz and other students
- Research funding
  - IT4Innovations Centre of Excellence – National supercomputing center
  - IT4Innovations excellence in science - LQ1602
  - Advanced Methods for Evolutionary Design of Complex Digital Circuits, 2014 – 2016 (Czech Science Foundation)
  - Relaxed equivalence checking for approximate computing, 2016 – 2018 (Czech Science Foundation)
  - Brno University of Technology