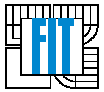
The Genetic and Evolutionary Computation Conference

Evolution of digital circuits

Lukáš Sekanina
Brno University of Technology
Faculty of Information Technology
Brno, Czech Republic

sekanina@fit.vutbr.cz
http://www.fit.vutbr.cz/~sekanina



Scope

- The tutorial covers basic techniques for evolutionary design of digital circuits and shows on several case studies how evolutionary computing can produce results that are competitive with conventional methods.
- What is not covered
  - evolution of analog circuits, antennas, MEMS and other hardware.
  - adaptive hardware

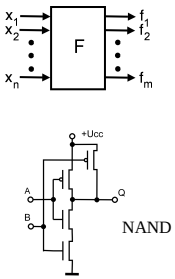
Agenda

- Digital Circuits
  - Basics of digital design and testing
  - Reconfigurable devices
- Evolutionary Circuit Design and Evolvable Hardware
  - Principles
  - Cartesian Genetic Programming
  - Scalability problems
- Case Studies
  - Logic synthesis
  - Image filter design
  - Benchmark circuits with predefined testability
- FPGAs for Circuit Evolution
- Conclusions
- References

Digital Circuits - Combinational

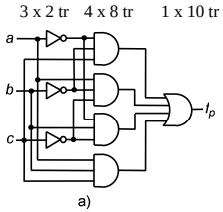
- The outputs depend only on current inputs.
- Multi-output Boolean function F: $\{0,1\}^n \rightarrow \{0,1\}^m$
- Representation:
  - truth table, logic expressions, Binary Decision Diagram, AND-Invert graph etc.
- Logic circuits are composed of logic gates, e.g.

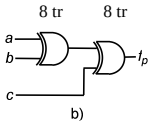
- Logic synthesis and minimization
  - start with a normal form – disjunctive normal form (DNF), conjunctive normal form (CNF), …
  - apply axioms and theorems of Boolean algebra to simplify expressions, i.e. reduce the number of gates (or area, delay, interconnect, …)
    - e.g. combining theorem (xy + xy' = x), De Morgan etc.
  - Methods and tools: Karnaugh map, Quine-McCluskey, Espresso, ABC, SIS, …



The 3-XOR Example

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





DNF: f = a'b'c + a'bc' + ab'c' + abc

f:

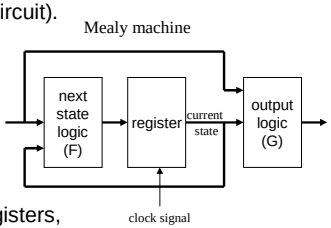
	a=0	a=1
b=0	0	1
b=1	1	0

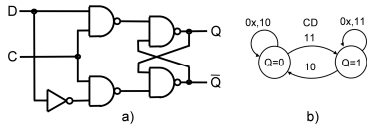
c

- a) The optimal solution consists of 48 transistors in disjunctive normal form (AND, OR, NOT).
- b) The optimal solution consists of 16 transistors when XOR gates are available.
- Not all solutions are achievable by a particular minimization method!

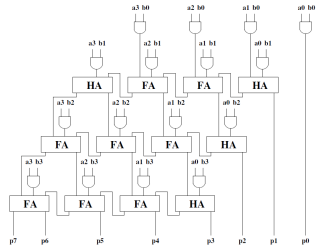
Digital Circuits - Sequential

- The outputs depend not only on the current inputs but also on the past sequences of inputs (represented in the state of a circuit).
- Mealy machine
  - Next state = F (current state, input)
  - Output = G (current state, input)
- Moore machine
  - Next state = F (current state, input)
  - Output = G (current state)
- Building blocks: latches, flip-flops, registers, counters
  - Example: D-latch

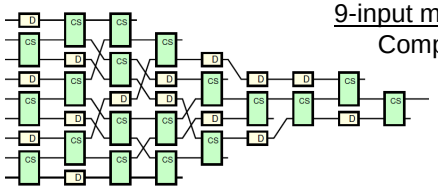




Examples of digital circuits

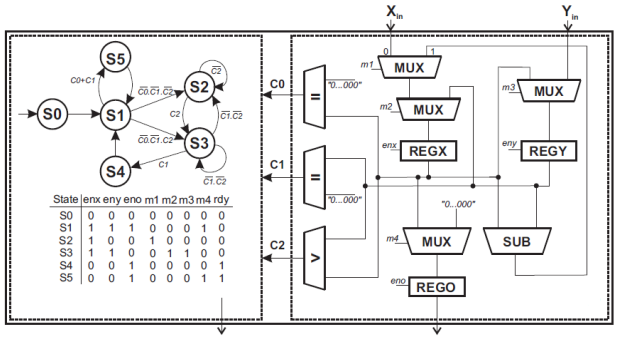


4b x 4b parallel multiplier
Full Adder (FA)
Half Adder (HA)



9-input median pipelined circuit
Compare & Swap & D (CS)
D flip-flop (D)

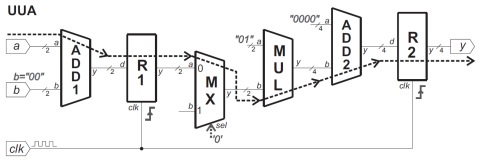
Datapath and Controller



MUX (multiplexer)
REG (register)
=, > (comparators)
SUB (subtractor)

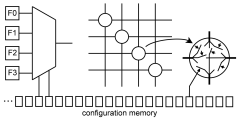
Diagnostics and Testing

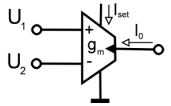
- Fault → Error → Failure
  - Fault – physical defect
  - Error - incorrect behavior caused by a fault
  - Failure - inability of the system to perform its specified service
- Fault models: stuck at 1, stuck at 0, bridging, delay…
  - Single vs multiple, permanent vs transient
- ATPG – Automatic Test Pattern Generator
  - Input patterns required to check a device for faults are automatically generated by a program. Device's response is compared with the expected response.
  - The goal is to maximize a given measure (fault coverage) and minimize the cost of testing.
- Testability analysis
  - Controllability
  - Observability



Reconfigurable Devices

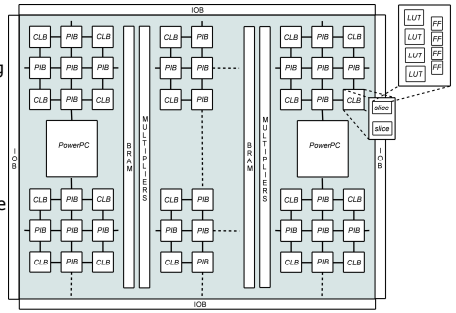
- Functionality of hardware is defined by a configuration bit stream.
- Examples:
  - Programmable Logic Device (PLD)
  - Field programmable gate array (FPGA)
  - Field programmable transistor array (FPTA)
  - Field programmable analog array (FPAA)
    - Operational Transconductance Amplifiers (OTA)
    - Switched capacitors
  - Reconfigurable multiprocessors (e.g. PicoChip)
  - Reconfigurable antenna array
  - Reconfigurable optics (e.g. deformable mirrors)
  - Reconfigurable molecular array (e.g. NanoCell)





Field Programmable Gate Arrays (FPGA)

- Xilinx FPGA consists of
  - array of configurable logic blocks (CLB)
  - configurable interconnecting system
  - configurable I/O ports
- Integrated hard cores
  - BRAMs, multipliers, processors, DSP, …
- Reconfiguration
  - Full – all FPGA resources are reconfigured
  - Dynamic partial reconfiguration – a part of FPGA is reconfigured while remaining circuits work unchanged
- ICAP - Internal Configuration Access Point
  - frame – configuration unit (1312-bit column)
- 80-90% area of FPGA not accessible to users



Xilinx Virtex 5, 65 nm
6-input LUTs (delay 0.9 ns)
The FX200T FPGA contains 122,880 6-LUTs.
www.xilinx.com

Examples of optimization problems in digital design

- Logic minimization - finding coverage with the minimum cost
- BDD optimization w.r.t various criteria
- High-level synthesis – finding modules satisfying design timing constraints while minimizing the total design cost (area)
- Partitioning, mapping, routing, floorplanning in physical design
- Test vector reordering to reduce power consumption
- Test scheduling optimization
- and many others …

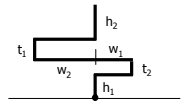
Evolutionary optimization has been utilized intensively.

But what about evolutionary design?

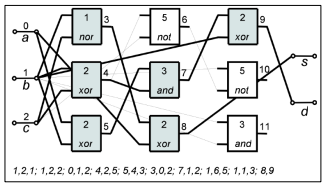
Evolutionary Algorithm

- Evolutionary algorithm (EA)
  - a robust population-oriented search algorithm
  - fitness function evaluates every candidate solution
- Evolutionary optimization
  - a search for suitable values of pre-selected parameters
  - Algorithms: GA, ES, PSO …
- Evolutionary design
  - can create a complete structure of target system (including parameters tuning)
  - Algorithms: GP, CGP …

```
set time t = 0
create initial population P(t)
evaluate P(t)
WHILE (not termination condition) DO
  t = t + 1
  P(t) = create new population using P(t-1)
  evaluate P(t)
END WHILE
```

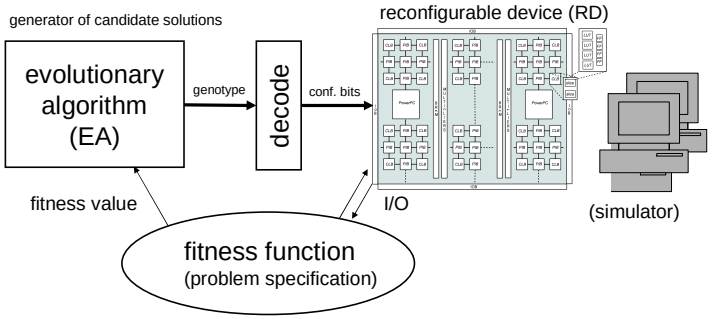


$t_1\ t_2\ h_1\ h_2\ w_1\ w_2$



Evolvable hardware

Evolutionary Algorithm + Reconfigurable Device [Higuchi et al, 1993]



- generate stimuli for RD
- obtain responses from RD
- compare the responses with target values
- calculate the fitness value

Extrinsic vs Intrinsic Evolution

- Extrinsic evolution
  - candidate circuits are evaluated using a circuit simulator
  - only the result of evolution is uploaded to a reconfigurable device
- Intrinsic evolution
  - all candidate circuits are evaluated in a physical reconfigurable device
  - could lead to solutions that exploit the reconfigurable device and external environment in a new way [Thompson, 1999]
  - could lead to solutions which are unreachable by conventional model-based design methods
    - e.g. circuit evolution in liquid crystals [Harding, 2008]

Why Evolvable Hardware?

- Allows to increase the level of design automation.
  - The design problem is transformed to the search problem!
  - Exploring "dark corners" of design spaces.
- Novel designs (unreachable by conventional techniques) can be discovered by means of EA.
  - antennas, analog circuits, digital circuits, optical lens systems, programs, protocols…
- Adaptive and self-repairing hardware can be implemented using evolvable hardware.
  - Self-Reconfigurable Analog Array (NASA JPL)
  - Adaptive image compression (AIST)
  - Adaptive cache mappings (U. of Paderborn)
  - …

Evolutionary design

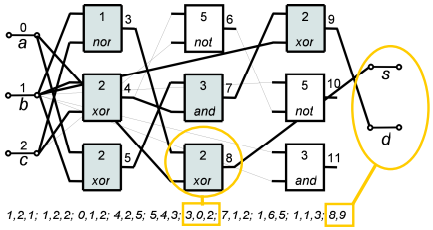
Evolvable hardware

Cartesian Genetic Programming (CGP) for Circuit Evolution *[Miller&Thompson, 2000]*

- Cartesian Genetic Programming (CGP) is a graph-based Genetic Programming (GP) method
  - GP: candidate program ~ syntactic tree (J. Koza, late 80s)
  - CGP: candidate program ~ acyclic oriented graph
- Features of CGP
  - genetic encoding is compact and simple (loosely inspired by the architecture of FPGAs)
  - mutation-based search
  - easy to implement
  - the effectiveness of CGP has been compared with many other GP methods and it is very competitive.
- Implementations
  - standard CGP, modular CGP, self-modifying CGP, multichromosome CGP
- Applications
  - digital circuit design, prime generating polynomials, robot controllers, image processing, classification, developmental neural architectures, evolutionary art, artificial life etc.

CGP: Representation

phenotype

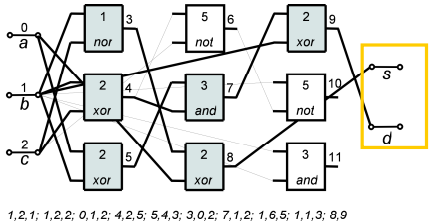


1,2,1; 1,2,2; 0,1,2; 4,2,5; 5,4,3; 3,0,2; 7,1,2; 1,6,5; 1,1,3; 8,9

genotype

- **Array of nodes:** rows = 3, columns = 3, inputs = 3, outputs = 2
- **Functions in the nodes**: {NAND (0), NOR (1), XOR (2), AND (3), OR (4), NOT (5)}

CGP: Fitness function for logic synthesis



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

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

Specification (1-bit adder), target table:

=> 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

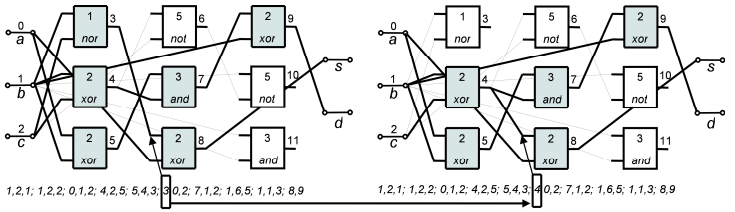
=> fitness = 10

- Fitness value: F = the number of bits correctly calculated for all possible assignments to the inputs (max. 16 in this example)
- If F reaches a maximum value then optimize the number of gates:

  F′ = F + N – U
  - where N is the number of available nodes
  - where U is the number of used nodes

CGP: Mutation

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



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

(a) mutation (b)

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

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

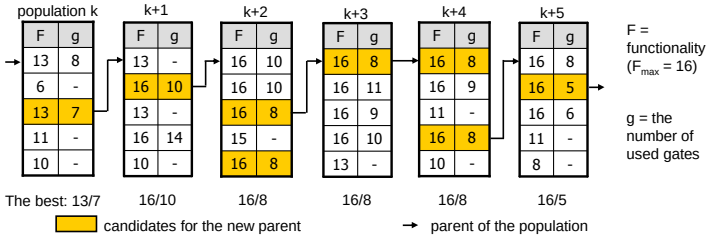
=> fitness = 10

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

CGP: Search algorithm

1. Randomly generate 1+λ individuals.
2. Evaluate the population.
3. WHILE the termination criterion is not satisfied DO
   - ❖ Select the highest scored individual – parent (see figure).
   - ❖ Use mutation to create λ offspring of the parent individual.
   - ❖ Create a new population using the parent and its λ offspring.
   - ❖ Evaluate the population.



F = functionality (F_{max} = 16)

g = the number of used gates

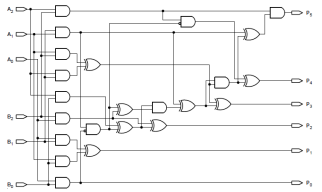
Gate-level evolution of multipliers

[Vassilev&Miller EH 2000, GENP 1(1), 2000]

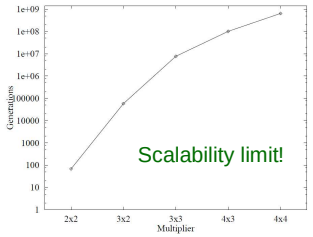
The number of 2-input gates and CGP setting

Multiplier	Best conv.	Best CGP	Array	Max. generations
2b×2b	8	7	1 × 7	10k
3b×2b	17	13	1 × 17	200k
3b×3b	30	23	1 × 35	20M
4b×3b	47	37	1 × 56	200M
4b×4b	64	57	1 × 67	700M

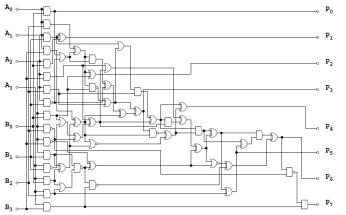
ES(1+4), h=3, $\{x\ AND\ y,\ x\ XOR\ y,\ (not\ x)\ AND\ y\}$



Evolved 3b x 3b multiplier



The best evolved 4x4 multipliers



Gajda, Sekanina: ICES2010

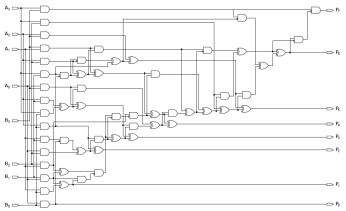
56 gates with the gate set

G = {and, or, not, nand, nor, xor, id, 0, 1}

delay = 18

seed: VJM, EH2000 (67 gates)

400 transistors



Vassilev, Job, Miller: EH2000

57 gates with {and, xor, not(x) and y}, delay=16

67 gates with the gate set G

seed: conventional solution (64 gates)

438 transistors

CGP Seeded by Conventional Designs

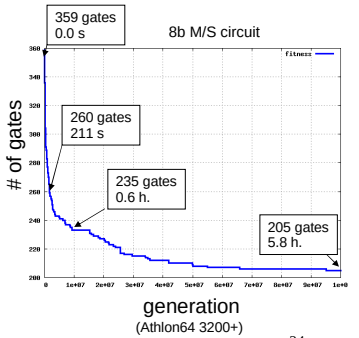
[Gajda, Sekanina: ICES2010]

- CGP used to minimize the number of gates
  - Best conv. = the best of ABC
  - CGP: λ = 14, h = 7, max. generation = 100M, array 1 x 'gates from ABC'

# of gates

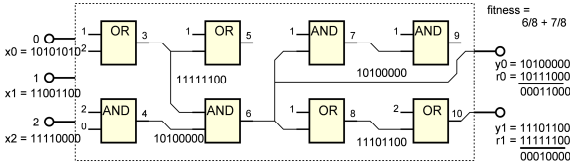
Circuit	Inputs	Outputs	Best Conv.	CGP	Reduction [%]
9symml	9	1	65	23	64,6
alu2	10	6	384	73	81,0
alu4	14	8	905	508	43,9
b1	3	4	7	4	42,9
cm138a	6	8	19	16	15,8
cm151a	12	2	24	23	4,2
cm152a	11	1	22	21	4,5
cm42a	4	10	22	17	22,7
cm82a	5	3	14	10	28,6
cm85a	11	3	38	26	31,6
decod	5	16	30	26	13,3
f51m	8	8	58	26	55,2
majority	5	1	8	8	0,0
x2	10	7	41	27	34,1
z4ml	7	4	21	15	28,6

LGSynth91 benchmarks



SW Acceleration

- Early termination of fitness evaluation
  - If a perfect functionality has been reached and the goal is to minimize some parameters (e.g. the number of gates), it is only tested whether a candidate circuit is working correctly or incorrectly; i.e. the evaluation is stopped after producing the first wrong output.
- Parallel simulation
  - utilizes bitwise operators to perform more than one evaluation of a gate in a single step



Parallel simulation: speedup is 8 wrt. a naïve simulation

The scalability problem

- EA is usually able to provide a good solution to a small problem instance; however, only unsatisfactory solutions are produced for larger problem instances.
- Solution: Use a domain knowledge in the EA!
  - representation
  - genetic operators
  - fitness function
- Recall: Evolutionary design is not suitable for all circuit design problems!

Scalability of representation

- Complex circuit ⇒ long chromosome ⇒ large search space ⇒ a search algorithm is inefficient
- Experience: Max. chromosome size ~ a few thousands of bits
- Solution: Add some domain knowledge to get shorter chromosomes
  - Incremental evolution: Divide and Conquer
    - How to make the decomposition?
  - Modular evolution
    - How to introduce modules automatically?
  - Functional-level evolution: From gates to functional units
    - How to choose functional units?
  - Development: Compress the chromosome
    - How to design the "compression" algorithm?

Scalability of fitness evaluation

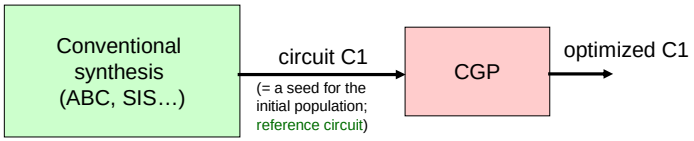
- The evaluation time grows exponentially with increasing number of circuit inputs (for combinational circuit evolution)
  - Experience: unpractical for ~10 inputs in case of multipliers and ~17 inputs in case of parity circuits
- Solution:
  - Do not insist on perfect evaluation!
    - training set for evolution and test set for validation
  - application-specific "tricks" in the fitness function

Case Studies:
How to eliminate the scalability problems?

- Logic synthesis
  - Task: Minimize the number of gates in large combinational circuits (hundreds of inputs, thousands of gates)
  - Difficulty: Standard fitness function requires exponential time for evaluation. Note that conventional methods have been developed for ~40 years.
- Image filters (Merit Award at Humies 2004)
  - Task: Design an image filter suppressing a given type of noise. Compare the quality of filtering and implementation cost with conventional solutions.
  - Difficulty: Gate-level design is not suitable for filters. How to measure the quality of filtering?
- Benchmark circuits (Silver Medal at Humies 2008)
  - Task: Design a set of synthetic benchmark circuits containing circuits with predefined testability (0-100%) and complexity (~10^6 gates) for evaluation of testability analysis methods.
  - Difficulty: Fitness calculation for a million gate circuit.

CGP for post-synthesis optimization

[Vašíček, Sekanina GENP 2011]



- A new fitness function has to be proposed to deal with complex circuits:
  - Use a **SAT solver** 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: fitness(C_i) = 0.
  - The equivalence checking can be performed for many real-world problem instances in a reasonable time.

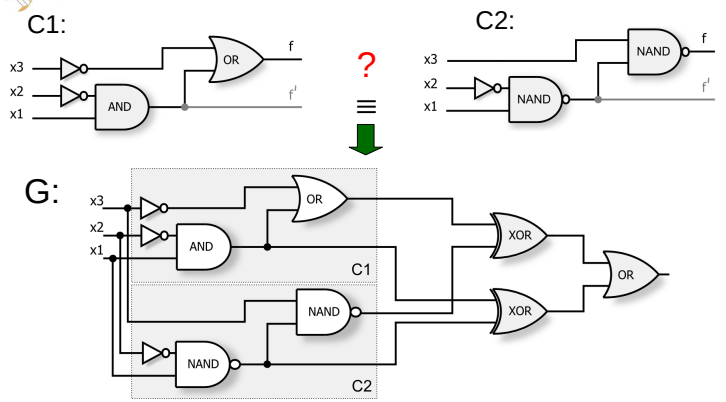
The SAT problem

- The satisfiability problem (SAT) is a decision problem, whose instance is a Boolean expression written in conjunctive normal form (CNF), i.e. as conjunction of clauses, e.g.

$$(\neg y \vee \neg x\ ) \wedge (y \vee x\ )$$

- The question is: given the expression, is there some assignment of TRUE and FALSE values to the variables that will make the entire expression true?
- The problem is NP-complete.
- SAT solvers are available that "effectively" solve the SAT problem.
  - MiniSAT, http://minisat.se/

SAT solver in the fitness function (1)



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.

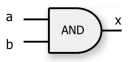
SAT solver in the fitness function (2)

- The G circuit is transformed to CNF using the Tseitin transform.
- The CNF representation captures the valid assignments between the gate inputs and outputs.
  - Consider a gate y = OP(a, b)
  - Hence, a CNF formula φ(y, a, b) = 1 iff the predicate y = OP(a, b) holds true.

Gate	Corresponding CNF representation
$y = \mathrm{NOT}(x_1)$	$(\neg y \vee \neg x_1) \wedge (y \vee x_1)$
$y = \mathrm{AND}(x_1, x_2)$	$(y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1) \wedge (\neg y \vee x_2)$
$y = \mathrm{OR}(x_1, x_2)$	$(\neg y \vee x_1 \vee x_2) \wedge (y \vee \neg x_1) \wedge (y \vee \neg x_2)$
$y = \mathrm{XOR}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1 \vee x_2) \wedge$ $(y \vee \neg x_1 \vee x_2) \wedge (y \vee x_1 \vee \neg x_2)$
$y = \mathrm{NAND}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (y \vee x_1) \wedge (y \vee x_2)$
$y = \mathrm{NOR}(x_1, x_2)$	$(y \vee x_1 \vee x_2) \wedge (\neg y \vee \neg x_1) \wedge (\neg y \vee \neg x_2)$

Various optimizations can be applied to reduce the decision time.

Example:



$x = ab$

$(x \Rightarrow ab)(ab \Rightarrow x)$

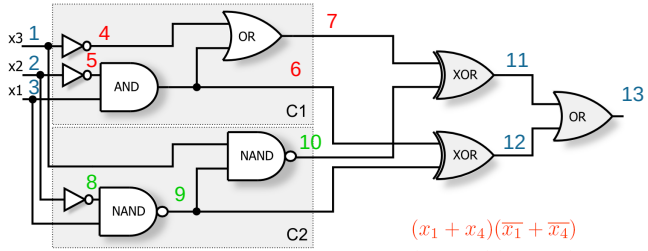
$(\overline{x} + ab)(\overline{ab} + x)$

$(\overline{x} + a)(\overline{x} + b)(\overline{a} + \overline{b} + x)$

$P \Rightarrow Q \equiv \overline{P} + Q$

$\overline{PQ} \equiv \overline{P} + \overline{Q}$

SAT solver in the fitness function (3)



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

$(x_8 + x_9)(x_3 + x_9)(\overline{x_8} + \overline{x_3} + \overline{x_9})$

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

$(\overline{x_7} + x_{10} + x_{11})(x_7 + \overline{x_{10}} + x_{11})(\overline{x_7} + \overline{x_{10}} + \overline{x_{11}})(x_7 + x_{10} + \overline{x_{11}})$

$(\overline{x_6} + x_9 + x_{12})(x_6 + \overline{x_9} + x_{12})(\overline{x_6} + \overline{x_9} + \overline{x_{12}})(x_6 + x_9 + \overline{x_{12}})$

$(x_{11} + x_{12} + \overline{x_{13}})(x_{13} + \overline{x_{11}})(x_{13} + \overline{x_{12}})$

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

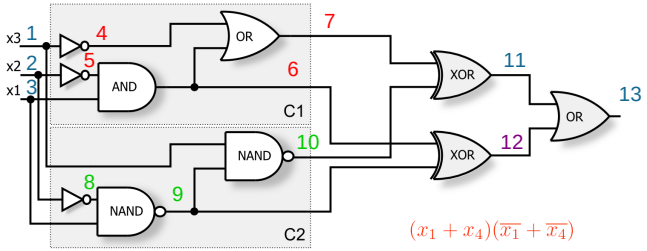
$(x_1 + x_4)(\overline{x_1} + \overline{x_4})$

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

$(x_5 + \overline{x_6})(x_3 + \overline{x_6})(\overline{x_5} + \overline{x_3} + x_6)$

$(\overline{x_4} + x_7)(\overline{x_6} + x_7)(x_4 + x_6 + \overline{x_7})$

SAT solver in the fitness function (3)



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

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

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

$x_6)$

$\overline{x_7})$

SAT solver

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

result: **SATISFIABLE / NONEQUIVALENT**

model / counter example: 0011111101011

$(x_{11} + x_{12} + \overline{x_{13}})(x_{13} + \overline{x_{11}})(x_{13} + \overline{x_{12}})$

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

Results for LGSynth93 benchmarks

[Vašíček, Sekanina DATE 2011]

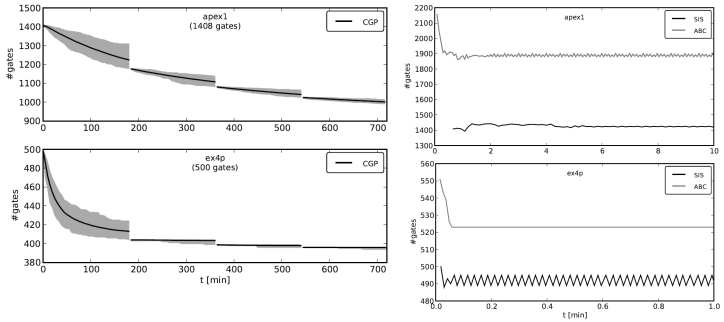
circuit	PI	PO	SIS	ABC	C1	C2	C3	CGP	impr.
apex1	45	45	1394	1862	1439	**1272**	1368	847	33,4%
apex2	39	3	**151**	225	221	195	299	90	40,4%
apex3	54	50	1405	1737	1494	**1332**	1515	1038	22,1%
apex5	117	88	751	768	728	**609**	921	613	-0,7%
cordic	23	2	67	61	67	**49**	90	32	34,7%
cps	24	109	1128	1109	1150	975	**967**	585	39,5%
duke2	22	29	406	**356**	417	366	357	260	27,0%
e64	65	65	192	384	**183**	191	255	129	29,5%
ex4p	128	28	488	523	468	**467**	555	349	25,3%
misex2	25	18	111	121	94	**89**	108	71	20,2%
vg2	25	8	95	113	88	**83**	109	78	6,0%

CGP
ES(1+1), 1 mut/chrom, seed: SIS, Gate set: {AND, OR, NOT, NAND, NOR, XOR}, 100 runs

ABC, SIS – conventional open academic synthesis tools

C1, C2, C3 – commercial synthesis tools

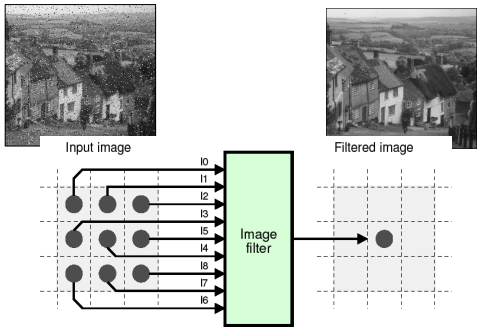
CGP for logic synthesis: Summary



- Summary
  - More time ⇒ better results
  - A promising optimization method for hard-to-synthesize circuits.

Functional-level evolution of image filters

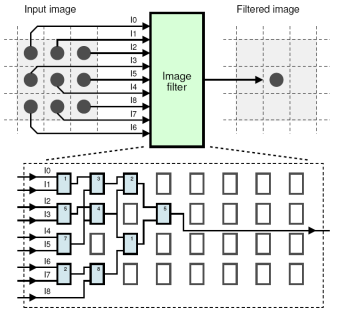
[Sekanina, EvoIASP 2002]



Can CGP design a filter which exhibits better filtering properties and lower implementation cost w.r.t. conventional solutions?

Target domain: filters suppressing shot noise, Gaussian noise, burst noise, edge detectors, …

Method: CGP at functional level



code	function	description	code	function	description
0	255	constant	8	$x \gg 1$	right shift by 1
1	x	identity	9	$x \gg 2$	right shift by 2
2	$255 - x$	inversion	A	$swap(x, y)$	swap nibbles
3	$x \vee y$	bitwise OR	B	$x + y$	+ (addition)
4	$\bar{x} \vee y$	bitwise $\bar{x}$ OR y	C	$x +^{S} y$	+ with saturation
5	$x \wedge y$	bitwise AND	D	$(x + y) \gg 1$	average
6	$\overline{x \wedge y}$	bitwise NAND	E	$max(x, y)$	maximum
7	$x \oplus y$	bitwise XOR	F	$min(x, y)$	minimum

All inputs and outputs at 8 bits!

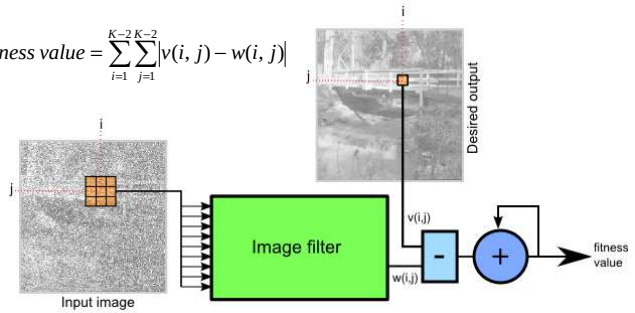
- Search method: (1+7) Evolutionary Strategy
- Population size: 8 individuals
- Mutation: max. 5%
- Array of 4 x 8 elements
- 100 runs / 30000 generations

Fitness function

Impossible to test all possible input combinations ⇒ a training set is employed

Image size: K x K pixels (K=128)
Fitness value: Mean Absolute Error (MAE)

$$fitness\ value = \sum_{i=1}^{K-2}\sum_{j=1}^{K-2}\left|v(i,j) - w(i,j)\right|$$



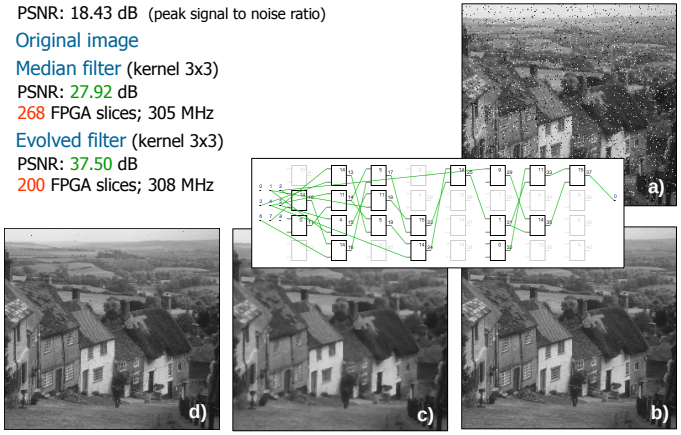
Example of evolved filter behavior

a) Image corrupted by 5% salt-and-pepper noise
PSNR: 18.43 dB (peak signal to noise ratio)

b) Original image

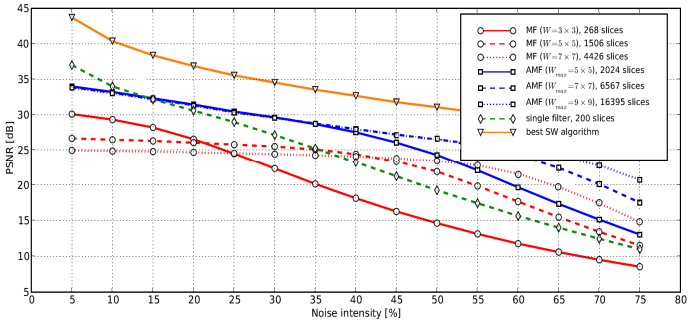
c) Median filter (kernel 3x3)
PSNR: 27.92 dB
268 FPGA slices; 305 MHz

d) Evolved filter (kernel 3x3)
PSNR: 37.50 dB
200 FPGA slices; 308 MHz



Comparison of various filters

Mean PSNR for 25 test images



MF – Median Filter
AMF – Adaptive Median Filter
The **single filter** is one of evolved filters.
Best SW - Y. Dong, S. Xu: A new directional weighted median filter for removal of random-valued impulse noise. Signal Processing Letters. vol. 14, no. 3, p. 193–196, 2007

Example: Shots + Edges

Evolved edge detector resistant against the salt-and-pepper

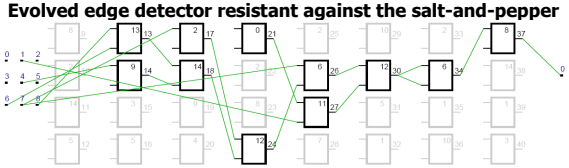
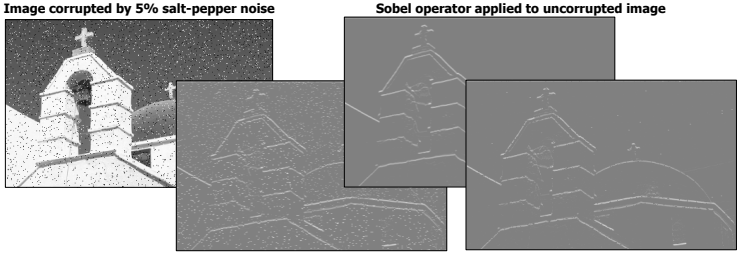


Image corrupted by 5% salt-pepper noise

Sobel operator applied to uncorrupted image

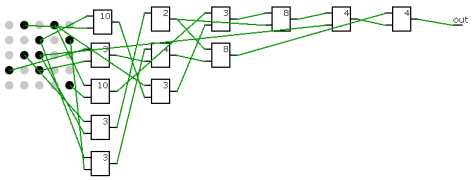


Sobel operator

The image obtained by evolved filter

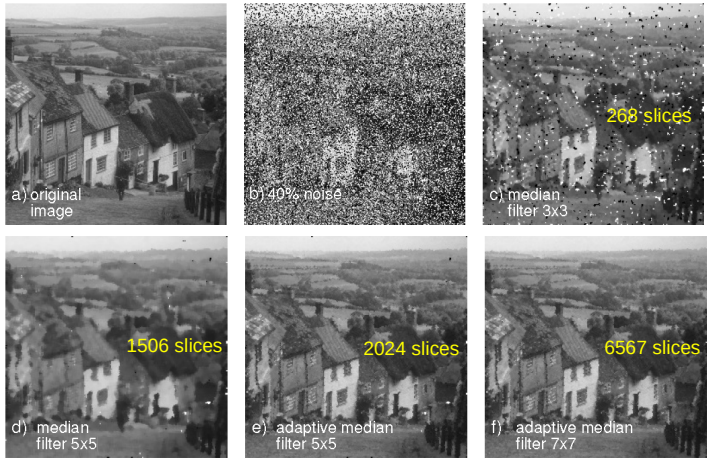
Example: Burst noise

5%, *N*(255, 30)



Evolved filter

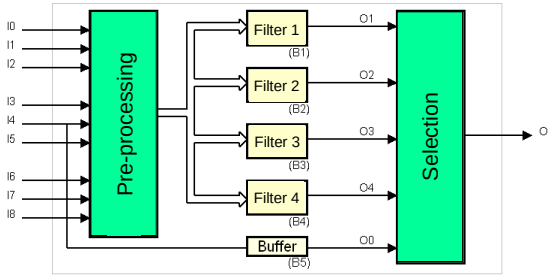
Problem: Salt&Pepper noise of high intensity



Bank of evolved filters

[Vašíček, Sekanina: FPL 2007]

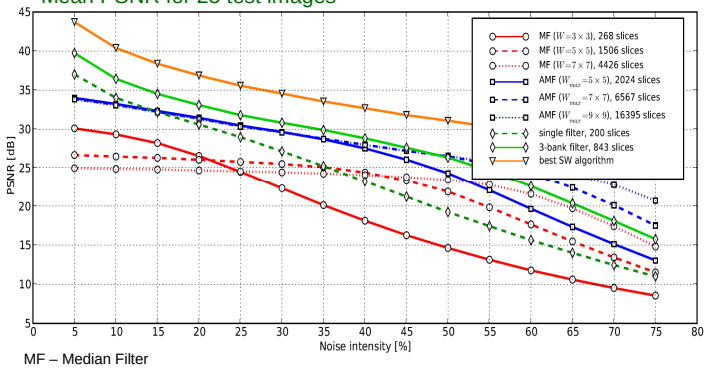
- CGP has provided many different implementations of 3x3 image filter for the 40% noise removal problem.
- Selected different implementations constitute a bank of filters.
- Pre-processing: if pixel[i] = 255 then pixel[i] := 0
- Selection: median-based selection



Bank of evolved filters 3x3 vs adaptive median filter 7x7

Comparison of various filters

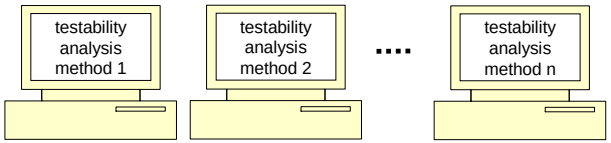
Mean PSNR for 25 test images



MF – Median Filter
AMF – Adaptive Median Filter
The **single filter** and **3-bank filter** are evolved filters
Best SW - Y. Dong, S. Xu: A new directional weighted median filter for removal of random-valued impulse noise. Signal Processing Letters. vol. 14, no. 3, p. 193–196, 2007

Evolution of benchmark circuits

[Pečenka, Sekanina, Kotásek: ACM TODAES 13(3) 2008]



Which of them is the best one (under some criteria)?

The methods can be compared using benchmark circuits.
The benchmarks should reveal weak points of the methods.

Evolution of benchmarks: Specification

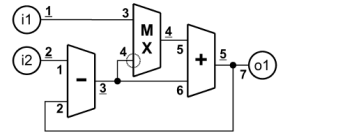
- The input (provided by user)
  - the number and type of components
    - e.g. 2xSUB(8) , 2xADD(8), 2xMUX2(8)
  - the number of circuit primary inputs and outputs
    - e.g. 4 x 8bit input, 2 x 8bit output
  - testability properties
    - average controllability (e.g. 80%)
    - average observability (e.g. 45%)

- The output
  - RTL circuit with required complexity and testability (synchronous circuits, one clock domain, no 3-state buses)
  - VHDL, Verilog, EDIF

- In the fitness function, the testability is calculated, i.e. structural properties of circuits are examined.
- Function is not evolved!

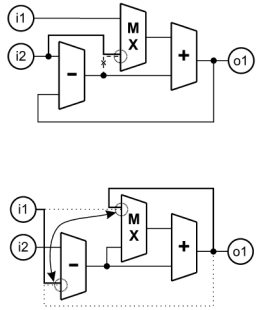
Evolution of benchmarks: EA

- Functional level evolution (thousands of components in a circuit)
- Only circuit connection is evolved
- Chromosome: string of integers



Primary outputs/ inputs of components (index)	1	2	3	4	5	6	7
Primary inputs/ outputs of components (values)	2	5	1	3	4	3	5

- Examples of mutation



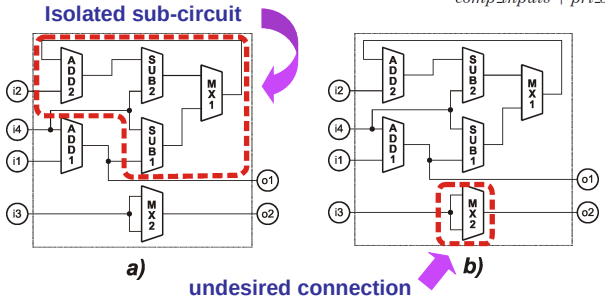
Evolution of benchmarks: Fitness function

a) Structure analysis
b) Interconnection analysis
c) Testability analysis

$$fitness = 0.25 \cdot structure + 0.25 \cdot connects + 0.5 \cdot testability$$

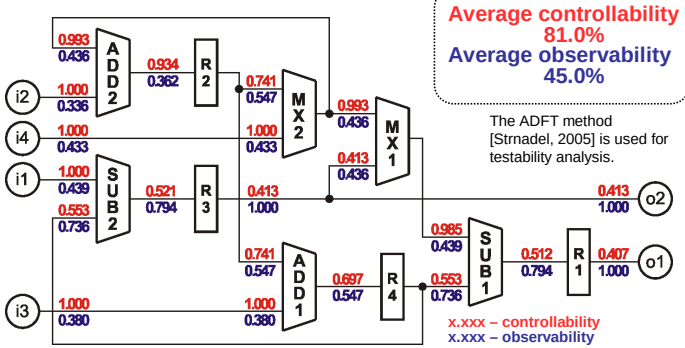
$$structure = 1 - \frac{useless\_comps}{comps\_count}$$

$$connects = 1 - \frac{shorts + direct\_connects}{comp\_inputs + pri\_outputs}$$



Evolution of benchmarks: Fitness function

c) Testability analysis



Average controllability 81.0%
Average observability 45.0%

The ADFT method [Strnadel, 2005] is used for testability analysis.

x.xxx – controllability
x.xxx – observability

$$testability = 1 - 0.5 \cdot (req\_cont. - avg\_cont)^2 - 0.5 \cdot (req\_obs. - avg\_obs.)^2$$

Evolved benchmarks

http://www.fit.vutbr.cz/~pecenka/cirgen/

- Population size: 5-20
- Mutation: 10-20%
- Replacement: 90 %

FITTest_BENCH06 synthetic benchmark circuits

Circuit	# of gates	# of FFs	# of faults	Fault coverage
a00	108 627	4 384	399 806	1,28%
a01	108 748	4 448	399 786	10,11%
a02	108 532	4 416	398 012	23,81%
a03	108 876	4 448	400 166	31,60%
a04	108 551	4 416	399 334	40,38%
a05	108 740	4 448	399 534	50,19%
a06	108 607	4 416	399 494	65,56%
a07	108 811	4 448	399 708	66,37%
a08	108 650	4 448	399 566	74,86%
a09	108 345	4 384	398 888	86,54%
a10	108 652	4 448	399 112	94,29%

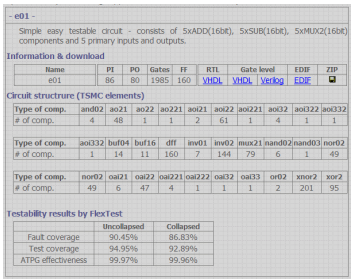
Circuit	PI	PO	# of FFs	# of gates	Fault coverage
e01	86	80	160	1,985	90.45%
e02	86	80	144	1,657	60.69%
e03	86	80	160	2,046	39.43%
e04	86	80	160	2,221	0.00%
e05	186	160	792	10,011	90.11%
e06	186	160	831	9,999	43.90%
e07	186	160	785	9,894	22.87%
e08	186	160	778	9,559	0.00%
e09	211	192	2,020	28,065	91.90%
e10	179	208	1,979	27,853	64.22%
e11	211	200	2,058	28,231	27.46%
e12	203	208	2,106	28,438	0.00%
e13	1,669	1,904	6,304	155,046	89.38%
e14	1,621	1,904	6,368	155,380	64.46%
e15	1,701	1,840	6,368	155,207	31.84%
e16	1,589	1,744	6,368	155,045	12.50%
e17	3,833	4,272	12,672	310,122	81.73%
e18	3,913	4,512	12,608	309,856	56.72%
e19	3,833	4,320	12,576	309,874	40.28%
e20	3,961	4,352	12,736	310,610	23.13%
e21	15,332	17,088	50,688	1,240,488	80.13%
e22	15,652	18,048	59,432	1,239,424	52.33%
e23	15,332	17,280	50,304	1,239,496	38.24%
e24	15,844	17,408	50,944	1,242,440	21.56%

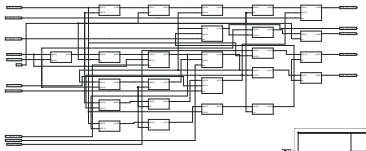
54

Examples

http://www.fit.vutbr.cz/~pecenka/cirgen/

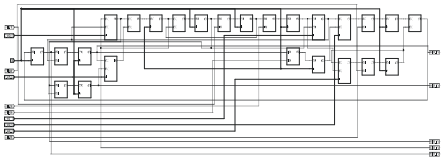
- Circuits e01-e04
  - 5 primary inputs and 5 primary outputs (16bits)
  - 5xADD(16bit), 5xSUB(16), 5xMUX2(16) and 10xREG(16)





Circuit: <e01>
Fault coverage: 90,45%

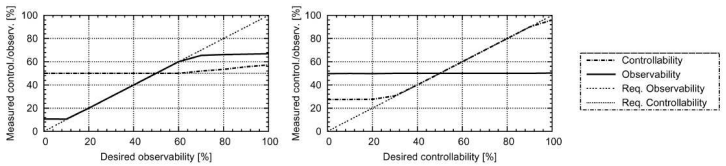
Circuit: <e04>
Fault coverage: 0,00%



Validation of proposed method (1)

Requirements		Results				
In/Out [-]	RTL comps. [-]	Controllability [%]	Observability [%]	Gates [-]	Flip-flops [-]	Time [hh:mm]
5/5	50	32.76%	32.15%	27,452	2,509	00:10
10/10	100	32.12%	30.42%	55,720	5,176	00:17
40/40	500	33.57%	30.84%	277,674	25,584	02:47
80/80	1,000	34.35%	29.92%	565,193	51,717	11:52

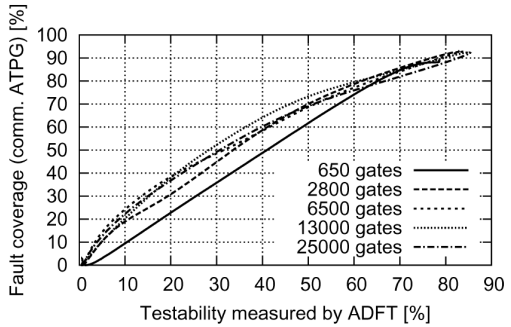
The 33% controllability and 33% observability required



(a) fixed controllability (50%) (b) fixed observability (50%)

Fig. 8. Testability parameters obtained when: (a) the controllability is fixed (=50%); or (b) the observability is fixed (50%).

Validation of proposed method (2)



Commercial ATPG vs ADFT (the testability analysis method used by EA)

Evolution of benchmarks: Summary

- Due to the low time complexity of the utilized testability analysis method, the proposed method allows the design of relatively complex circuits (millions of gates) with the required testability and complexity.
- The evolved benchmarks currently represent the most complex benchmark circuits with a known level of testability.
- Using the benchmarks it is possible to reveal problems that are hidden for classical benchmark circuits.

Case Studies: Summary

- How was the scalability problem eliminated in our case studies?
- Logic synthesis
  - representation – gate-level, standard CGP
  - genetic operators – standard CGP
  - fitness function – SAT solver, seeding by a conventional solution
- Image filters
  - representation – functional-level, standard CGP
  - genetic operators – standard CGP
  - fitness function – training image
- Benchmark circuits
  - representation – functional-level, but only interconnections evolved
  - genetic operators – mutation of interconnections
  - fitness function – testability analysis estimated in polynomial time
- EA parameters – a suitable setting improves the convergence (see corresponding articles for details)

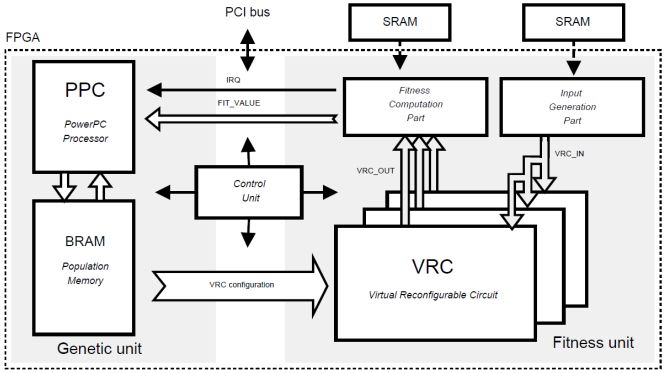
Evolution in FPGAs

Authors	Application	Platform	EA	Fitness
External Reconfiguration				
Thompson et al. (1999)	Tone discriminator	XC6216	PC	PC
Huelsbergen et al. (1999)	Oscillators	XC6216	PC	PC
Zhang et al. (2004)	Image filters	VRC	PC	PC
Gordon (2005)	Arithmetic circuits	Virtex CLB	PC	PC
Gwaltney and Gutton (2005)	IIR filters	VRC	DSP	DSP
Internal reconfiguration				
Tufte and Haddow (2000)	FIR filters	Register values	HW	HW
Martinek and Sekanina (2005)	Image filters	VRC	HW	HW
Vasicek and Sekanina (2007)	Image filters	VRC	PowerPC	HW
Sekanina and Friedl (2004)	Logic circuits	VRC	HW	HW
Vasicek and Sekanina (2008)	CGP accelerator	VRC	PowerPC	HW
Salomon et al. (2006)	Hash functions	VRC	HW	HW
Glette (2008)	Face recognition	VRC	MicroBlaze	HW
Glette et al. (2007)	Sonar spectrum class.	VRC	PowerPC	HW
Upegui and Sanchez (2006)	Cellular automaton	Virtex CLB	MicroBlaze	HW
Vasicek et al. (2008)	Const. Multipliers	VRC	HW	HW
Cancare et al. (2010)	Logic circuits	Virtex 4 logic	PowerPC	HW
Salvador et al. (2011)	Image filters	Virtex 5 logic	MicroBlaze	HW

Cited from [Sekanina 2011]

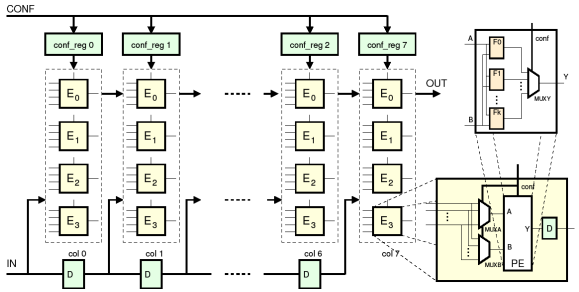
CGP accelerator in the Xilinx Virtex II Pro FPGA

[Vašíček, Sekanina: IJICA 1(1), 2007 and CAI 29(6) 2010]



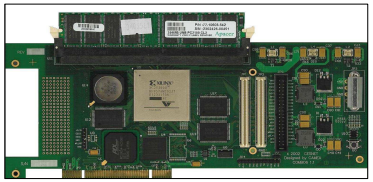
Virtual Reconfigurable Circuit

- VRC is a new MUX-based reconfigurable layer on the top of the FPGA.
- Fast pipelined reconfiguration and processing.
- Configuration register contains 384 bits for the 8x4 processing elements.



FPGA Accelerator: Results

- Target platform: Combo6X
- CGP
  - 4 x 8 nodes
  - training image: 128x128
  - population size: 8
- Results (FPGA at 100 Hz)
  - 1 VRC: 44 times faster than a Celeron 2.4GHz CPU
  - evaluates approx. 6k candidate filters per second
  - requires approx. 10 sec to produce a filter (~30k generations)
  - 4 VRCs: The speedup is 170.



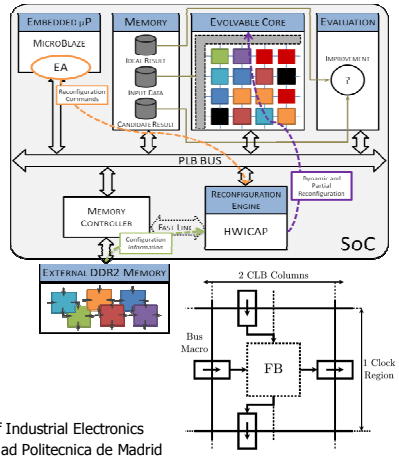
Results of synthesis for Virtex II Pro 2VP50FF1517
N_c is the number of VRCs

resource	avail.	$N_c = 1$		$N_c = 2$		$N_c = 4$	
IO blocks	852	602	70%	602	70%	602	70%
BRAM	232	16	7%	16	7%	16	7%
SLICES	23 616	4 651	20%	7 961	34%	14 582	60%
DFF	49 788	3 536	7%	4 691	9%	7 001	14%

FPGA accelerator with dynamic partial reconfiguration

[Salvador et al., AHS 2011]

- Reconfigurable systolic array
  - Dynamic partial reconfiguration at level of processing elements (PE) for image filter evolution.
  - Library of pre-synthesized PEs (40 CLBs/PE)
  - interconnection is pre-synthesized
- EA used to assign functions to PEs
  - array 6x6 PEs
  - 16 functions/PE
  - chromosome ~ 144 bits
- Virtex-5 LX110T FPGA
- PE's reconfiguration time using ICAP (250 MHz): 12 us



Centre of Industrial Electronics
Universidad Politecnica de Madrid

Conclusions: Promises of evolutionary circuit design

- CGP and its extensions seem to be very suitable for circuit evolution.
- "Innovative" solutions can be produced.
  - improving area/delay/power consumption/testability…
  - Many similar solutions can be obtained.
- Promising applications
  - Problems where it is difficult to formulate a perfect specification and a partially working solution is acceptable (e.g. filtering, classification, prediction, robot controlling)
  - Hard combinatorial/combinational problems (e.g. in logic synthesis)

Conclusions: Problems of evolutionary circuit design

- Runtime.
- (Almost) nothing is guaranteed.
- Some tricks are needed to solve the scalability problems.
- It is not easy to find a new problem where EC could successfully be applied and "beat" conventional methods.
- Sometimes considered as crazy method by "conventional designers".

References

- Drechsler, R.: Evolutionary Algorithms for VLSI CAD. Kluwer Academic Publishers, Boston 1998
- Harding S. L., Miller J. F., Rietman E. A.: Evolution in Materio: Exploiting the Physics of Materials for Computation. International Journal of Unconventional Computing, 4(2), 2008, 155-194
- Higuchi, T. et al.: Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In: SAB'92: Proc. of the 2nd International Conference on Simulated Adaptive Behaviour, MIT Press, Cambridge MA 1993, p. 417-424
- Higuchi, T. et al.: Real-world applications of analog and digital evolvable hardware. IEEE Trans. on Evolutionary Computation. 3(3), 1999, 220-235
- Gajda Z., Sekanina L.: An efficient selection strategy for digital circuit evolution. In Evolvable Systems: From Biology to Hardware, LNCS 6274. Springer Verlag, 2010, p. 13–24
- Greenwood, G., Tyrrell, A.: Introduction to Evolvable Hardware. A Practical Guide for Designing Self-Adaptive Systems. IEEE Press Series on Computational Intelligence, 2006
- Higuchi, T., Liu, Y., Yao, X.: Evolvable Hardware. Springer Verlag, 2006
- Koza, J. R. et al.: Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufmann Publishers, San Francisco CA 1999
- Koza, J. R. et al.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer Academic Publishers, 2003
- Miller J., Thomson P.: Cartesian Genetic Programming. In: Proc. of the 3rd European Conference on Genetic Programming EuroGP2000. LNCS 1802, Springer, 2000, p. 121–132
- Miller, J., Job, D., Vassilev, V.: Principles in the evolutionary design of digital circuits - Part I. Genetic Programming and Evolvable Machines. 1 (1), 2000, 8-35
- Novak, O. et al.: Handbook of Electronic Testing. CVUT Publisher, 2005
- Pecenka, T., Sekanina, L., Kotasek, Z.: Evolution of synthetic rtl benchmark circuits with predefined testability. ACM Trans. on Design Automation of Electronic Systems 13(3), 2008, 1–21

References

- Salvador R. et al.: Evolvable 2D computing matrix model for intrinsic evolution in commercial FPGAs with native reconfiguration support. In Proc. of NASA/ESA conf. on Adaptive Hardware and Systems. IEEE, 2011, in press
- Sekanina, L.: Image filter design with evolvable hardware. In: Applications of Evolutionary Computing. LNCS 2279, Springer Verlag, 2002, p. 255–266
- Sekanina, L.: Evolvable Components: From Theory to Hardware Implementations. Natural Computing Series, Springer Verlag Berlin 2004
- Sekanina L.: Evolvable Hardware. In Handbook of Natural Computing (Rozenberg G, Bäck T., Kok, J. N., Eds.) Springer Verlag, 2011 – in press
- Thompson, A., Layzell, P., Zebulum, R. S.: Explorations in design space: unconventional electronics design through artificial evolution. IEEE Trans. on Evolutionary Computation. 3(3), 1999, 167-196
- Vasicek, Z., Sekanina, L.: An evolvable hardware system in Xilinx Virtex II Pro FPGA. International Journal of Innovative Computing and Applications 1(1), 2007, 63–73
- Vasicek, Z., Sekanina, L.: An area-efficient alternative to adaptive median filtering in FPGAs. In: Proc. of 2007 Conf. on Field Programmable Logic and Applications, IEEE Computer Society, 2007, p. 216–221
- Vasicek Z., Sekanina L.: Hardware Accelerator of Cartesian Genetic Programming with Multiple Fitness Units. Computing and Informatics, 29(6), 2010, 1359-1371
- Vasicek Z., Sekanina L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genetic Programming and Evolvable Machines. Vol. 12, 2011 – in press
- Vasicek Z., Sekanina L.: A global postsynthesis optimization method for combinational circuits. In Proc. of the Design, Automation and Test in Europe, EDAA, 2011, p. 1525–1528
- Vassilev V., Job D., Miller J. F.: Towards the Automatic Design of More Efficient Digital Circuits. In Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society, 2000, p. 151–160
- Wakerly J. F.: Digital Design: principles and practices (3d edition), Prentice Hall, New Jersey, USA, 2000