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

Copyright is held by the author(s).

GECCO’11,
July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

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) 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!

\[
\text{DNF: } f = a'b'c + a'bc' + ab'c' + abc
\]

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

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 ...

---

**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?**

- **Extrinsic evolution**
  - 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 (AIIST)
  - Adaptive cache mappings (U. of Paderborn)
  - ...
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

- 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

- 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:
**CGP: Search algorithm**

1. Randomly generate $1+\lambda$ 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 $\lambda$ offspring of the parent individual.
   - Create a new population using the parent and its $\lambda$ offspring.
   - Evaluate the population.

![Population and Selection Diagram]

**Gate-level evolution of multipliers**

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

The number of 2-input gates and CGP setting

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Best conv.</th>
<th>Best CGP</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>2b x 2b</td>
<td>7</td>
<td>7</td>
<td>1 x 7</td>
</tr>
<tr>
<td>3b x 2b</td>
<td>17</td>
<td>13</td>
<td>1 x 17</td>
</tr>
<tr>
<td>3b x 3b</td>
<td>30</td>
<td>23</td>
<td>1 x 35</td>
</tr>
<tr>
<td>4b x 3b</td>
<td>47</td>
<td>37</td>
<td>1 x 56</td>
</tr>
<tr>
<td>4b x 4b</td>
<td>64</td>
<td>57</td>
<td>1 x 67</td>
</tr>
</tbody>
</table>

ES(1+4), h=3, [x AND y, x OR y, (not x) AND y]

**Scalability limit**

**The best evolved 4x4 multipliers**

[Gajda, Sekanina: ICES2010]

56 gates with the gate set
- $G = \{\text{and, or, not, NAND, NOR, XOR, XNOR, id, 0, 1}\}$
- delay = 18
- seed: VJM, EH2000 (67 gates)
- 400 transistors

[Vassilev, Job, Miller: EH2000]

57 gates with $(\text{and, xor, not}(x) \text{ and } y)$, 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: $\lambda = 14$, $h = 7$, max. generation = 100M, array 1 x ‘gates from ABC’

# of gates

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Best Conv.</th>
<th>CGP</th>
<th>Reduction [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LGSynth91 benchmarks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b M/S circuit</td>
<td>309 gates</td>
<td>1050 s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2b M/S circuit</td>
<td>250 gates</td>
<td>211 s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8b M/S circuit</td>
<td>305 gates</td>
<td>5.8 h</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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

**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 $\Rightarrow$ long chromosome $\Rightarrow$ large search space $\Rightarrow$ a search algorithm is inefficient
- Experience: Max. chromosome size $\sim$ 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 $\sim$10 inputs in case of multipliers and $\sim$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

• 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 C_1 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.
  \[(\neg y \vee \neg x) \land (y \vee x)\]
- 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)
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 \( g = \text{OP}(a, b) \)
  - Hence, a CNF formula \( \varphi(y, a, b) \) = 1 iff the predicate \( y = \text{OP}(a, b) \) holds true.

\[
\begin{array}{|c|c|}
\hline
\text{Gate} & \text{Corresponding CNF representation} \\
\hline
y = \text{AND}(a, b) & \{(y \land 1) \lor (y \land \neg a) \lor (y \land \neg b)\} \\
y = \text{OR}(a, b) & \{(y \land a) \lor (y \land b)\} \\
y = \text{NAND}(a, b) & \{(y \land \neg a) \lor (y \land \neg b)\} \\
y = \text{XOR}(a, b) & \{(y \land a) \lor (y \land \neg b)\} \\
\hline
\end{array}
\]

Example:

\[
x = ab \\
(x \Rightarrow ab)(ab \Rightarrow x) \\
(\neg x) = (\neg a) \land (\neg b)
\]

Various optimizations can be applied to reduce the decision time.

---

SAT solver in the fitness function (3)

\[
(x_2 + x_3)(x_2 + x_5) \\
(x_2 + x_3)(x_2 + x_5)
\]

Results for LGSynth93 benchmarks

<table>
<thead>
<tr>
<th>circuit</th>
<th>PT</th>
<th>PO</th>
<th>SIS</th>
<th>ABC</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>CGP</th>
<th>impr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>apex1</td>
<td>45</td>
<td>45</td>
<td>1394</td>
<td>1862</td>
<td>1439</td>
<td>1272</td>
<td>1368</td>
<td>847</td>
<td>33.4%</td>
</tr>
<tr>
<td>apex2</td>
<td>39</td>
<td>3</td>
<td>151</td>
<td>225</td>
<td>219</td>
<td>199</td>
<td>299</td>
<td>90</td>
<td>40.4%</td>
</tr>
<tr>
<td>apex3</td>
<td>54</td>
<td>50</td>
<td>1405</td>
<td>1737</td>
<td>1494</td>
<td>1332</td>
<td>1515</td>
<td>1038</td>
<td>22.3%</td>
</tr>
<tr>
<td>apex5</td>
<td>117</td>
<td>88</td>
<td>751</td>
<td>768</td>
<td>728</td>
<td>699</td>
<td>921</td>
<td>613</td>
<td>-0.7%</td>
</tr>
<tr>
<td>cmlnc</td>
<td>23</td>
<td>2</td>
<td>67</td>
<td>61</td>
<td>67</td>
<td>49</td>
<td>90</td>
<td>32</td>
<td>34.7%</td>
</tr>
<tr>
<td>cws</td>
<td>24</td>
<td>109</td>
<td>1128</td>
<td>1109</td>
<td>1150</td>
<td>973</td>
<td>967</td>
<td>565</td>
<td>39.5%</td>
</tr>
<tr>
<td>dske2</td>
<td>22</td>
<td>29</td>
<td>406</td>
<td>356</td>
<td>417</td>
<td>305</td>
<td>357</td>
<td>260</td>
<td>27.9%</td>
</tr>
<tr>
<td>etb4</td>
<td>65</td>
<td>65</td>
<td>192</td>
<td>384</td>
<td>183</td>
<td>191</td>
<td>255</td>
<td>129</td>
<td>29.5%</td>
</tr>
<tr>
<td>exsp</td>
<td>128</td>
<td>28</td>
<td>488</td>
<td>523</td>
<td>468</td>
<td>467</td>
<td>555</td>
<td>349</td>
<td>25.3%</td>
</tr>
<tr>
<td>minc2</td>
<td>25</td>
<td>18</td>
<td>111</td>
<td>121</td>
<td>94</td>
<td>89</td>
<td>108</td>
<td>71</td>
<td>20.2%</td>
</tr>
<tr>
<td>rcc2</td>
<td>25</td>
<td>8</td>
<td>95</td>
<td>113</td>
<td>88</td>
<td>83</td>
<td>109</td>
<td>78</td>
<td>6.6%</td>
</tr>
</tbody>
</table>

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**

- More time $\Rightarrow$ better results
- A promising optimization method for hard-to-synthesize circuits.

**Functional-level evolution of image filters**

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**

- Input 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 $\Rightarrow$ a training set is employed

Image size: $K \times K$ pixels ($K=128$)

Fitness value: Mean Absolute Error (MAE)

\[
\text{fitness value} = \sum_{i,j} |i, j| - w(i, j)|
\]
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

Example: Shots + Edges

Evolved edge detector resistant against the salt-and-pepper

Example: Burst noise

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

Selected different implementations constitute a bank of filters.

Filter 1
Filter 2
Filter 3
Filter 4

Pre-processing

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

Comparison of various filters

Mean PSNR for 25 test images

The single filter and 3-bank filter are evolved filters.

Evolution of benchmark circuits

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), 2xMULQ(8)
  - the number of circuit primary inputs and outputs
    - e.g., 4 x 8 bit input, 2 x 8 bit 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

Evolution of benchmarks: Fitness function

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

Isolated sub-circuit

undesired connection
Evolution of benchmarks: Fitness function

c) Testability analysis

![Testability Analysis Diagram]

Average controllability 81.0%
Average observability 45.0%

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

testability = 1 - 0.5 · (req.cont. - avg.cont)^2 - 0.5 · (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

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)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td># Input</td>
<td># ETL comps.</td>
</tr>
<tr>
<td>5/5</td>
<td>50</td>
</tr>
<tr>
<td>10/10</td>
<td>100</td>
</tr>
<tr>
<td>40/40</td>
<td>500</td>
</tr>
<tr>
<td>50/80</td>
<td>1,000</td>
</tr>
</tbody>
</table>

The 33% controllability and 33% observability required.

Fig. 8. Testability parameters obtained when: (a) the controllability is fixed (≈60%); or (b) the observability is fixed (50%).
Validation of proposed method (2)

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.

Evolution in FPGAs

<table>
<thead>
<tr>
<th>Authors</th>
<th>Application</th>
<th>Platform</th>
<th>EA</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thompson et al. (1999)</td>
<td>Tone discriminator</td>
<td>XC6216</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>Huisinghoven et al. (1999)</td>
<td>Oscillators</td>
<td>XC6216</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>Zhang et al. (2004)</td>
<td>Image filters</td>
<td>VRC</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>Gordon (2005)</td>
<td>Arithmetic circuits</td>
<td>Virtex CLB</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>Wainwright and Gutton (2005)</td>
<td>IR filters</td>
<td>VRC</td>
<td>DSP</td>
<td>DSP</td>
</tr>
<tr>
<td>Huisinghoven et al. (1999)</td>
<td>Tone discriminator</td>
<td>XC6216</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>Mazloom and Sekanina (2005)</td>
<td>Image filters</td>
<td>VRC</td>
<td>HW</td>
<td>HW</td>
</tr>
<tr>
<td>Sekanina and Fried (2004)</td>
<td>Logic circuits</td>
<td>VRC</td>
<td>HW</td>
<td>HW</td>
</tr>
<tr>
<td>Vasicek and Sekanina (2008)</td>
<td>CGP accelerator</td>
<td>VRC</td>
<td>PowerPC</td>
<td>HW</td>
</tr>
<tr>
<td>Salmonon et al. (2006)</td>
<td>Hash functions</td>
<td>VRC</td>
<td>HW</td>
<td>HW</td>
</tr>
<tr>
<td>Goffe (2008)</td>
<td>Face recognition</td>
<td>VRC</td>
<td>MicroBlaze</td>
<td>HW</td>
</tr>
<tr>
<td>Goffe et al. (2007)</td>
<td>Sonar spectrum class</td>
<td>VRC</td>
<td>PowerPC</td>
<td>HW</td>
</tr>
<tr>
<td>Upegui and Sanchez (2006)</td>
<td>Cellular automaton</td>
<td>Virtex CLB</td>
<td>MicroBlaze</td>
<td>HW</td>
</tr>
<tr>
<td>Vasicek et al. (2008)</td>
<td>Const. Multipliers</td>
<td>VRC</td>
<td>HW</td>
<td>HW</td>
</tr>
<tr>
<td>Cancare et al. (2010)</td>
<td>Logic circuits</td>
<td>Virtex 4 logic</td>
<td>PowerPC</td>
<td>HW</td>
</tr>
<tr>
<td>Salvador et al. (2011)</td>
<td>Image filters</td>
<td>Virtex 5 logic</td>
<td>MicroBlaze</td>
<td>HW</td>
</tr>
</tbody>
</table>

Cited from [Sekanina 2011]
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.

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.

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 μs
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)

References

Nováček, O. et al.: Handbook of Electronic Testing. ČVUT Publisher, 2009
Higuchi, T., Liu, Y., Yao, X.: Evolvable Hardware. Springer Verlag, 2006
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.: Hardware Accelerator of Cartesian Genetic Programming with Multiple Fitness Units. Computing and Informatics, 29(6), 2010, 1359-1371