# Evolvable Hardware as Non-linear Predictor for Image Compression

Lukáš Sekanina

Department of Computer Science and Engineering
The Faculty of Electrical Engineering and Computer Science, Brno University of Technology

**Abstract. Evolvable hardware (EHW) is a new technology, which was discovered at intersection of artificial intelligence and the circuit design. Unique hardware architecture is searched for each task. The circuit connection is subject of evolution and the function of such circuit can adapt to dynamic changing environment. Genetic algorithm is used to simulate evolution – its chromosomes encode potential solutions (configuration bits of used reconfigurable chip). Simply, we can speak about hardware evolution. A concept for modelling of EHW and EHW-based applications is explained. Models of reconfigurable circuit and evolution are used in the application – lossy image compression – where EHW works as non-linear predictor for block of data. The new approach to compression quality control is described too. The results are compared with JPEG algorithm.**

**Key words: genetic algorithm, evolvable hardware, image compression, modelling, prediction, reconfigurable circuit.**

## Introduction: Evolvable hardware

Engineers have been inspired by certain natural processes, which opened new domains as artificial neural networks and evolutionary algorithms. We need in our applications such qualifications as evolution, adaptation and fault tolerance that have difficult implementation using traditional methodologies, but they are the main sign of living beings. Evolvable hardware is looking for its inspiration at level of phylogeny – the temporal evolution of the genetic program within individuals and species [Sanc97].

Evolvable hardware consists of two main components: a reconfigurable circuit and an evolutionary mechanism. A connection of reconfigurable circuit is stored in SRAM and can be quickly changed during execution. The FPGA (Xilinx) or special chips are usually used. Genetic algorithm works with population of individuals – every as a bit string of architecture configuration. The goal is to find the connection (function) which solves the task. The quality of chromosome (connection) is defined by the fitness function. Fitness calculation depends on the task – a set of training data is often constructed. The initial population is generated randomly and the next populations are created using by reproduction, crossover and mutations. Generally – EHW-based applications should lead to fast execution (HW implementation) with excellent adaptation in changing environments (genetic algorithm). Evolvable hardware can be used in two ways [Yao97]:

- *Evolutionary circuit design* – user only defines training data and the evolution is stopped when desirable function is reached. Everything can be simulated in software and the result configuration is downloaded into chip only. This approach is sometimes called as *extrinsic* evolvable hardware.
- *Intrinsic evolvable hardware* – every chromosome is downloaded into chip and is evaluated there. All operations (including GA) should be implemented in HW. On-line adaptive systems work with changing training data (it reflexes changing environment) and the evolution can not be stopped because the environment can change (concept of the open evolution).

Only the concept with open evolution and the total hardware implementation should be called as evolvable hardware [Yao97, Sanch97]. The first one is better to call as the circuit design with usage of evolution. Modelling and simulation are the key points in EHW-based application design from our point of view. They enable to decide when it is useful to design some application with this technology. Genetic algorithm doesn't find totally correct solution in all cases. Some applications can be very sensitive for a bad (or worse) solution and it is unacceptable for them. Simulation can discover these problems.

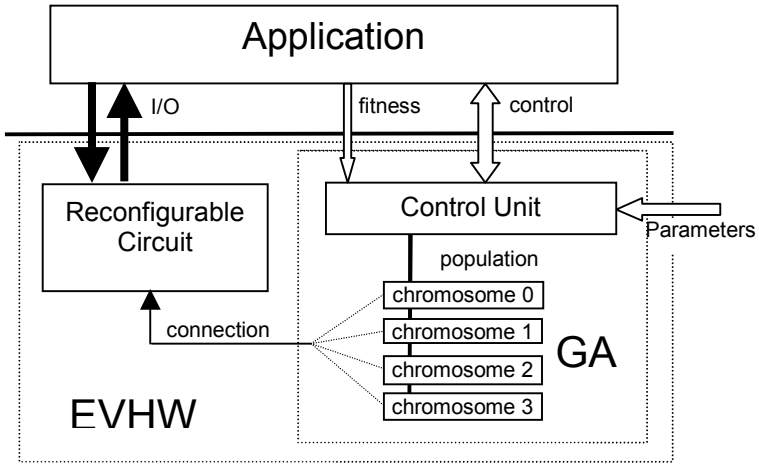


Figure 1. Concept of EHW modelling

## Modelling of evolvable hardware

The concept for modelling of evolvable hardware is shown at Figure 1. An application uses inputs and outputs of the reconfigurable circuit and furthermore it must define quality of connection (fitness of chromosome). C++ library EVHW contains an implementation of genetic algorithm, which works with general type of chromosome (abstract class of the object model). Special chromosomes (inherited classes) are associated with different kinds of circuit simulators and encode their circuit connections. Fitness of the chromosome is calculated in steps: (1) load connection, (2) set inputs, (3) simulation, (4) store result values, (5) repeat 2-4 for all inputs in training data set, (6) calculate fitness using stored results.

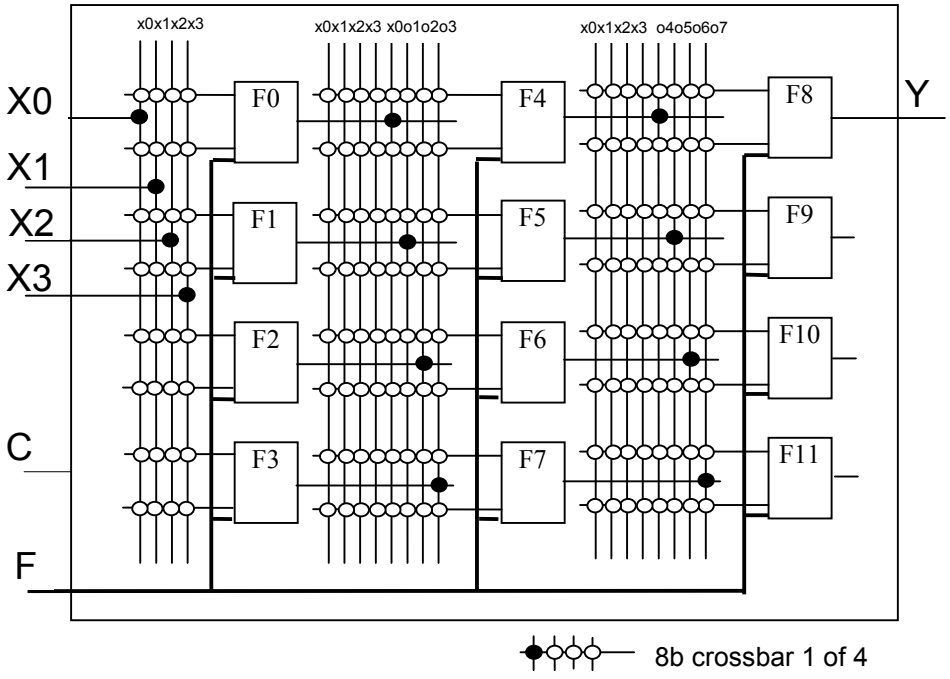


Figure 2. Designed model of reconfigurable circuit

Figure 2 shows architecture of a reconfigurable circuit, which is similar to the model in [Mura96]. This architecture is called as *evolvable hardware at function level*. It is the typical kind of constrained evolution (the result function is evident from the circuit structure) – [Thom99] investigates unconstrained evolution, where the result function can depend on some physical properties of given chip. The model consists of the function blocks (FB). Every FB can implement one of the functions such an adder, a subtracter, a multiplier, a sine generator, xor and so on. Chromosome genes given to the FB determine selection of the function. Data width is 8 bits and the lowest 8 bits are used when the overflow occurs. Chromosome is defined as the set of used FBs (genes) in the connection. Nine bits (three bits for function selection, three bits for operand 1 connection and three bits for operand 2 connection) encode the FB. The FB input can be connected to one of the circuit inputs or to one of the FB outputs in the previous column.

Genetic learning determines the FB functions and the interconnection among blocks. Only the reproduction and the mutation are used as genetic operators for this circuit. We use *roulette wheel* strategy for reproduction [Toma96]. Only one of the function blocks in each chromosome undergoes the mutation. There are two kinds of mutation: mutation of the function and mutation of the operand. Some blocks can be destroyed or a new block can appear during mutation.

## Image compression

The basic approach to image compression is described in [Sala97]. EHW can be considered as a predictive function for optimization (see Figure 3). This function is not linear and its changes depend on characteristics of the compressed block. The image is divided into number of blocks and one function has to be found for each block. Selecting one function for the whole image will produce poor results. By using EHW for finding function for different region of the image we actually implement an adaptive prediction coding for the image. The bit string of the best configuration of EHW represents block of data in the compressed file. Selection of the block size and the neighboring pixels are very important for this approach.

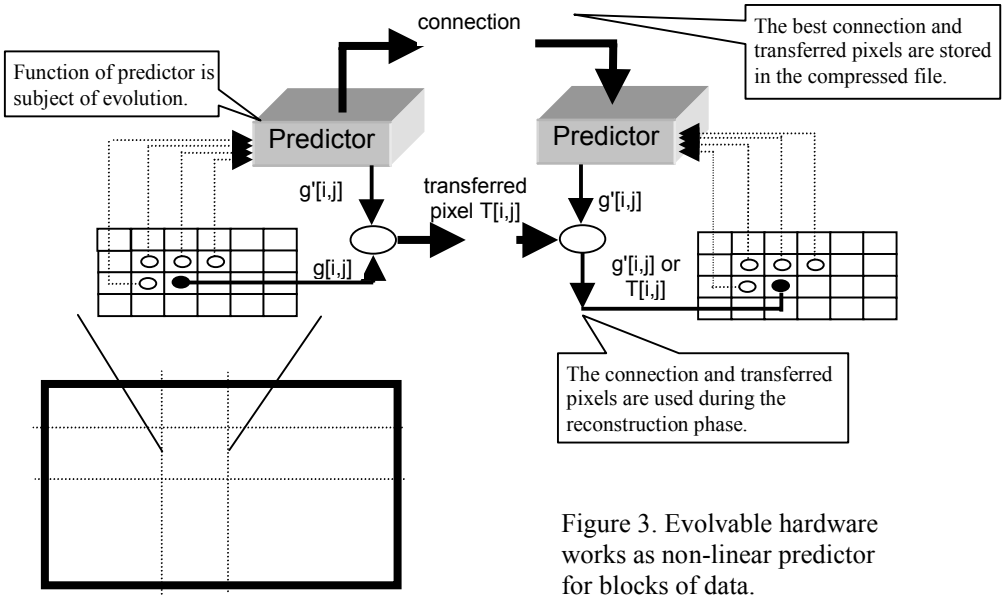


Figure 3. Evolvable hardware works as non-linear predictor for blocks of data.

The goal is 256 gray-scaled image compression. The experiments show 16 by 16 pixels of block size as the best choice. We have used prediction with four neighbor pixels according to [Sala97]. A value at position $[i, j]$ in an image $g$ is predicted as

$$g'[i,j] = f(g[i\text{-}1, j\text{-}1], g[i\text{-}1, j], g[i\text{-}1, j\text{+}1], g[i, j\text{-}1])$$

where $f$ is a function evolved by circuit at Figure 2. EHW predicts 256 values per block, which are compared with originals. The connection with the minimal average prediction error is searched by genetic algorithm. The fitness is calculated as

$$fitness = 256^2 - \sqrt{d}$$

where $256^2$ is the worst possible prediction and

$$d = \sum_{i,j} (g'[i,j] - g[i,j])^2 \qquad i = 1..16, j = 1..16.$$

This approach is described as a lossy image compression – prediction differences are not encoded. We can obtain a nice ratio of compression (about 50), but a poor visual quality of the reconstructed image.

Users usually want to control the quality of compression. We have investigated methods where it is possible. Four threshold values (firstly randomly generated) were appended to the chromosome – each for a quarter of the image. If the difference of the predicted and the original pixel value was greater than the relevant threshold value, then this pixel (position and value) was appended (transferred) to bits, which represent the block in the compressed file. These pixels then are not predicted during decompression. Maximum number of transferred pixels was set before compression. The fitness is calculated as

$$fitness = 256^2 - (\sqrt{d} + C.Penalty)$$

and $d$ is defined in the same way as before, but transferred pixels are not included (the difference is zero). *Penalty* is the number of transferred pixels and $C$ is a weight coefficient which depends on the penalty (small penalty determines small $C$ and bad chromosomes are debased by $C$).

The goal is to find the acceptable number (no more than maximum) of key pixels for reconstruction, according to evolved connection, which ensure high fitness. Threshold values are changed during evolution, but they are not changed by a mutation of connection (chromosomes carry these values only). They are calculated in the fitness function – lower fitness of the chromosome implies stronger changes. The evolution doesn't search the best connection only, but it searches the best connection with the best positions of transferred pixels together. The growing number of transferred pixels leads to high quality of reconstruction but to the low ratio of compression (see Figure 4). Experiments show that quality increases with number of population – so the evolution can find better position of transferred pixels.

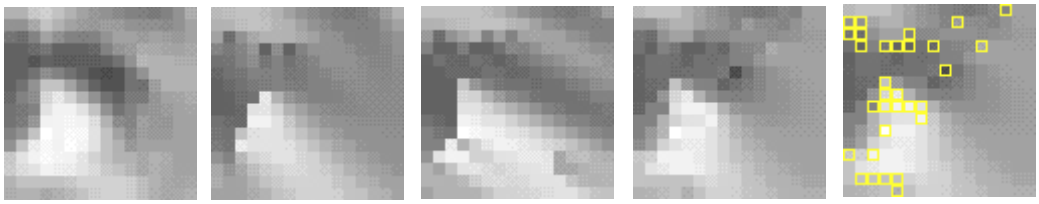


Figure 4. The number of transferred pixels and quality of reconstruction (detail of Lenna's eye)
A. an original
B. transferred pixels 8 - average error 17.2/pixel - compression 12.2
C. 16 - 16.8/pixel - 8.3
D. 31 - 8.9/pixel - 5.0
E. D with positions of transferred pixels

## Results

Designed method is simple and for demonstration only. JPEG is still two times better in ratio of compression than EHW approach (for the reconstructed images with similar visual quality, see Figure 5). This difference is mainly determined by the way of positions and values encoding of transferred pixels (several methods as Huffman encoding were tested). Better results should be obtained by e.g. "only-values" encoding with fixed positions of transferred pixels. Simulation shows very long time of compression, but extremely short time of decompression. A clock in case of hardware pipeline implementation and FBs with logical functions only can generate one prediction (arithmetic operations take more of time).

Simulations clearly show problems with the time of evolution and the reachable ratio of compression. We are going to model some applications with proposed simulation library and check possible hardware implementations. We suppose that the success of evolvable hardware will mainly be in problems of prediction and optimization. Using of evolvable hardware leads to elegant problem solutions and it is a great promise for the future experiments.

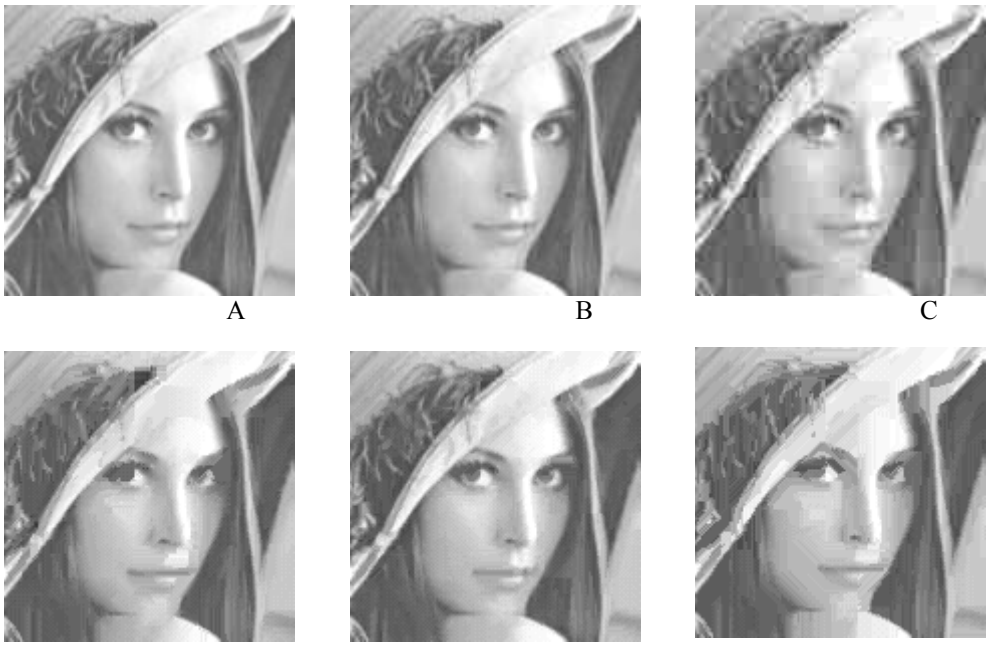


Figure 5. Results of EHW and JPEG approaches

A. Lenna - an original image
B. JPEG - compression 7.1- average error 2.82/pixel
C. JPEG 15.3 - 6,04/pixel
D. EHW 6.62 - 6.29/pixel (generations 1000, size of populace 100, enabled penalty 32, FBs 13)
E. EHW 3.94 - 4.20/pixel (generations 100, size of populace 30, enabled penalty 100, FBs 13)
F. EHW 6.46 - 7.00/pixel (generations 1000, size of populace 30, enabled penalty 32, FBs 13) (F - there were used only the logical functions in the function blocks)

# References

[Gibs98] Gibson, J., Berger, T., Lookabaugh, T., Lindbergh, D., Baker, R.: Digital Compression for Multimedia. San Francisco, Morgan Kaufmann Publishers 1998.

[Iwat97] Iwata, M., Salami, M., Higuchi, T.: Lossless Image Compression by Evolvable Hardware. 1997. URL http://www.etl.go.jp:8080/etl/kikou/paper/salami-ijcai97.ps.gz (Dec, 1998).

[Mura96] Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M., Higuchi, T.: Hardware Evolution at Function Level. 1996.
URL http://www.etl.go.jp:8080/etl/kikou/paper/murakawa-ppsn96.ps.gz (Jan, 1999).

[Sala97] Salami, M., Murakawa, M., Higuchi, T.: Data Compression Based on Evolvable Hardware. In: Evolvable Systems: From Biology to Hardware, Berlin, Springer-Verlag 1997, p. 169.

[Sanc97] Sanchez, E., Mange, D., Sipper, M., Tomassini, T., Perez, A., Stauffer, A.: Phylogeny, Ontogeny, and Epigenesis: Three Sources of Biological Inspiration for Softering Hardware. In: Evolvable Systems: From Biology to Hardware, Berlin, Springer-Verlag 1997, p. 35.

[Thom99] Thompson, A., Layzell, P.: Analysis of Unconventional Evolved Electronics. Communication of the ACM. April 1999/Vol. 42, No.4, p. 71.

[Toma96] Tomassini, M.: Evolutionary Algorithms. In: Towards Evolvable Hardware, Berlin, Springer-Verlag 1996, p. 19.

[Yao97] Yao, X., Higuchi, T.: Promises and Chalenges of Evolvable Hardware. In: Evolvable Systems: From Biology to Hardware, Berlin, Springer-Verlag 1997, p. 55.

[Yao99] Yao, X.: Following the Path of Evolvable Hardware. Communication of the ACM. April 1999/Vol. 42, No.4, p. 47.