# Slate of the Art:
# An Evolving FPGA-Based Board for Handwritten-Digit Recognition

Ron Levy, Stefano Lepri, Eduardo Sanchez, Gilles Ritter, and Moshe Sipper
Logic Systems Laboratory, Swiss Federal Institute of Technology,
IN-Ecublens, CH-1015 Lausanne, Switzerland.
E-mail: {name.surname}@epfl.ch, Web: lslwww.epfl.ch.

## Abstract

*We describe a completely autonomous evolutionary hardware system, which is able to recognize handwritten decimal digits. The system can adapt itself to different users on the fly. A working prototype has been implemented on an FPGA using VHDL, following the positive results of a C-based simulation.*

## 1 Introduction

Since its inception, research in the field of evolvable hardware has been in search of a "killer application" involving online, real-world evolution [1]. For such an application, the genetic operators (selection, crossover, mutation) and the calculation of fitness must take place in an autonomous way within the hardware, without recourse to external elements (software or other hardware).

Autonomous robotics was quickly identified as a desirable target field. Serious limitations, however were quickly identified: the high cost of robots, which eliminates the possibility of working with a truly large population, and the unsuitability of robots for short learning time spans.

In this paper, we describe a new online evolutionary hardware system capable of autonomously recognizing characters. In this system, the entire genetic population is physically stored in two memories and no software simulation is used to perform the different genetic operations. Due to the difficulties encountered implementing a recognition system entirely in hardware, we have limited ourselves to the recognition of decimal digits as a first stage.

In Section 2 the problem of recognizing digits with the chosen algorithm is discussed. In Section 3 the genetic algorithm used for the evolution of the system and the genome structure is described. The general architecture and the required hardware needed for the implementation of the system are presented in Section 4. The results obtained and future developments are discussed in Section 5. (NB: the project described herein is still work in progress)

## 2 Recognition of decimal digits

### 2.1 Introduction

The objective of the work described below is to obtain a hardware system capable of recognizing hand-written decimal digits, able to operate on its own, and capable of evolving to improve its own performance. The hand-written digits are inputted into the system through a touch screen with a resolution of 16x16 pixels. The result obtained is one of ten possible solutions.

Although limited to the ten decimal digits, the solution search space is enormous: a combination of an input of 256 variables (the touch screen) and the ten output variables, the solution matrix has $2^{256} \approx 10^{77}$ lines.

There exist at present various solutions to the problem of recognizing characters, including commercially available products [2, 3, 4]. Nevertheless, to the best of our knowledge, most of these are implemented in software. Furthermore, several pre-processing phases are required on the input character, in order to compensate for the differences in size, pitch, and placement imposed by the different users: the character is placed at the center of the display, it is straightened up and its size is normalized. Due to the problems such pre-processing would present if performed by hardware alone, only the centering of the character has been chosen for this project. Our constraints render recognition of characters more difficult than what is possible with currently available software.

Draft

0-7695-0762-X/00 $10.00 © 2000 IEEE

### 2.2 The recognition algorithm

On account of the choice of limited character pre-processing as described in the previous section, an algorithm that was not sensitive to pitch and size had to be developed. After examining several possibilities, we chose an algorithm based on the projected shadow of the character on four axes. This solution yielded the best compromise between hardware complexity and performance.

The principle utilized is fairly simple. It consists of recognizing the character not by its 16x16 matrix but by a combination of four vectors, each representing the shadow as projected on a given axis (horizontal, vertical, or one of two diagonals) (*see Figure 1*). Each of the 16x16 matrix lines, be it horizontal, vertical, or diagonal, yields the corresponding value of the shadow on the given axis, which is the number of active pixels on that line. For a realistic shadow, the weighted distance of the pixels to the axis should usually be taken into account, but in order to simplify the hardware, this weighting was not used. This did not result in degradation of system performance.

The four vectors representing the unknown digit are then compared to a set of reference vectors chosen for each of the ten digits. The key to this project consists in choosing these reference vectors through the use of a genetic algorithm, which takes into account the quirks of the user's handwriting.

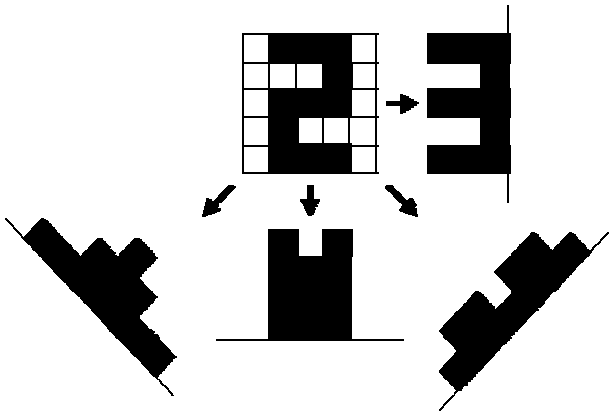

**Figure 1: The projection of a digit on 4 axes, shown for a 5x5 matrix.**

## 3 The genetic algorithm

The genetic algorithm used, described in this section, is essentially a simple genetic algorithm [5, 6].

### 3.1 The organization of the genome

The totality of all reference vectors can be thought of as a genome, which evolves from a randomly generated population.

The genome was divided into ten parts, or genes, one for each decimal. Each gene is made up of four vectors, each corresponding to one axis of projection. The size of the vectors is not uniform: both the vertical and horizontal vectors contain 16 values, one for each line of the 16x16 matrix, while the diagonal axis contain 31 values, corresponding to all the diagonals of the matrix. Each value can vary between 0 and 16, according to the number of active pixels per line. Thus at least 5 bits are needed to encode each value.

The size of the genome is thus:

10*(5*(2*16+2*31)) = 4700 bits

In order to simplify the hardware implementation of the algorithm, the number of required bits was reduced by making use of the following simplifications:

1. Elimination of 6 diagonals in each corner, which results in having only 19 values per diagonal, instead of the original 31. Performance is not affected because of the centering of each inputted character.
2. Limiting the maximum number of active pixels per line to 7, which results in the reduction in the coding of each value from 5 to 3 bits.

As a result the size of the genome is reduced to:

10*(3*(2*16+2*19)) = 2100 bits

Since 2100 is not a multiple of 32, a 2240 bit genome was chosen which could easily be stored in and retrieved from a 32-bit wide memory.

### 3.2 Genetic operations

In order to shorten the time needed for character recognition, the initial population is not completely random. The first generation genomes are projection vectors generated from a reference database, which are subsequently used to establish a fitness criterion. This reference database contains 10 samples per decimal digit, representing a user's handwriting.

0-7695-0762-X/00 $10.00 © 2000 IEEE

In order to create a next-generation population, a selection procedure is applied. It requires several rounds, each consists of randomly picking two genomes that are subsequently compared according to their fitness. The one with the best fitness will have the highest probability of being selected. This selected genome will be crossed over with the selected genome of the following round, thereby giving birth to two more evolved genomes. This procedure is repeated until a new population of genomes has been created. A bit mutation can take place, with a very low probability of occurrence (Pr{m} = 3.5e-4). The newly evolved population replaces the old one and its degree of fitness is again determined (*see Figure 2*). With each new generation the best genome (with highest fitness) serves as a reference vector for the character-recognition process.

The process of evolution continues indefinitely and occurs in parallel with the character-recognition process.

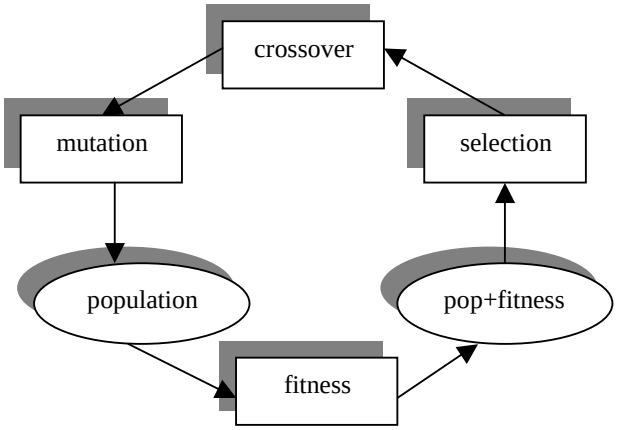


**Figure 2: The genetic algorithm.**

### 3.3 The learning curve

The evolutionary process is dependent on the way the fitness is determined. In the described system, the fitness is a value between 0 and 200, 200 being the best possible fitness.

In order to evaluate fitness, the genome is tested against the reference database. An actual recognition takes place using the genome to be evaluated, but instead of trying to recognize an input digit, the reference characters are being recognized. For each recognized character, the genome is awarded 1 point. Since the reference database contains 10 samples of each decimal digit, a maximum of 100 points can be obtained in this way. In order to eliminate "hesitation" in the choice of recognition, an extra point is awarded if the distance between the best two possible choices for a given digit is large. Thus an extra 100 points can be obtained, allowing a maximum obtainable fitness of 200. The algorithm used to calculate fitness is delineated in *Figure 3*.

Because the evaluation of the fitness depends so heavily on the reference database, it is possible to change the database even when the system is running. This also enables the system to adapt itself to the handwriting of a new user.

```
fitness(i) = 0, i = 0 :99  {initialize table of 100
characters containing the   fitness for each genome}
for (each genome G = 0 :99) do
  for (each referenceCharacter C = 0 :99) do
    min1 = 224  {initialize smallest distance}
    min2 = 224  {initialize second smallest distance}
     for (each gene g = 0 :9) do
      distance = referenceCharacter(C) – gene(g)
        if (distance < min1) then
          minTemp = min1
          min1 = distance
          recognisedCharacter = g
          if (minTemp < min2) then
            min2 = minTemp
          end if
        else
          if (distance < min2) then
            min2 = distance
          end if
        end if
      end for
      if (recognisedCharacter = ⌊C/10⌋) then
        fitness(G) = fitness(G) + 1
        if ((min1 – min2) >= 10) then
          fitness(G) = fitness(G) + 1
        end if
      end if
    end for
end for
```

**Figure 3: Pseudo-code of fitness computation.**

0-7695-0762-X/00 $10.00 © 2000 IEEE

## 4 Hardware implementation

### 4.1 Introduction

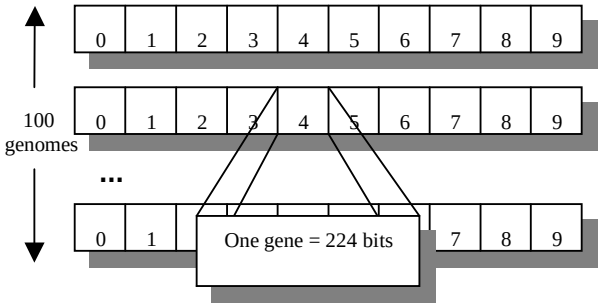


**Figure 4: A population of genomes.**

The total population consists of 100 genomes of 2240 bits each (*see Figure 4*), and is stored in two memories, one for the current population, the second for the newly evolved one. In order to obtain online evolution, it is necessary to design a processor that is capable of performing the above-described genetic operations, producing the best genome needed to recognize the input digit. In order to be completely autonomous, the system also needs to perform the following operations: data input through a touch screen, projection and centering of the unknown digit, and digit recognition. The use of an FPGA circuit greatly simplifies the implementation of the required tasks. The only external elements to the FPGA are three memories (one for the reference database, two for the populations) and the touch screen (*see Figure 5*).

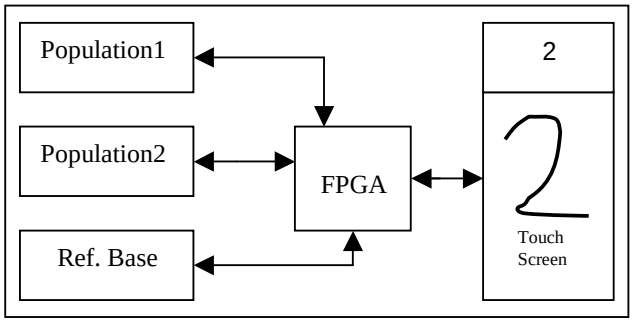


**Figure 5: Schema of the complete onboard system.**

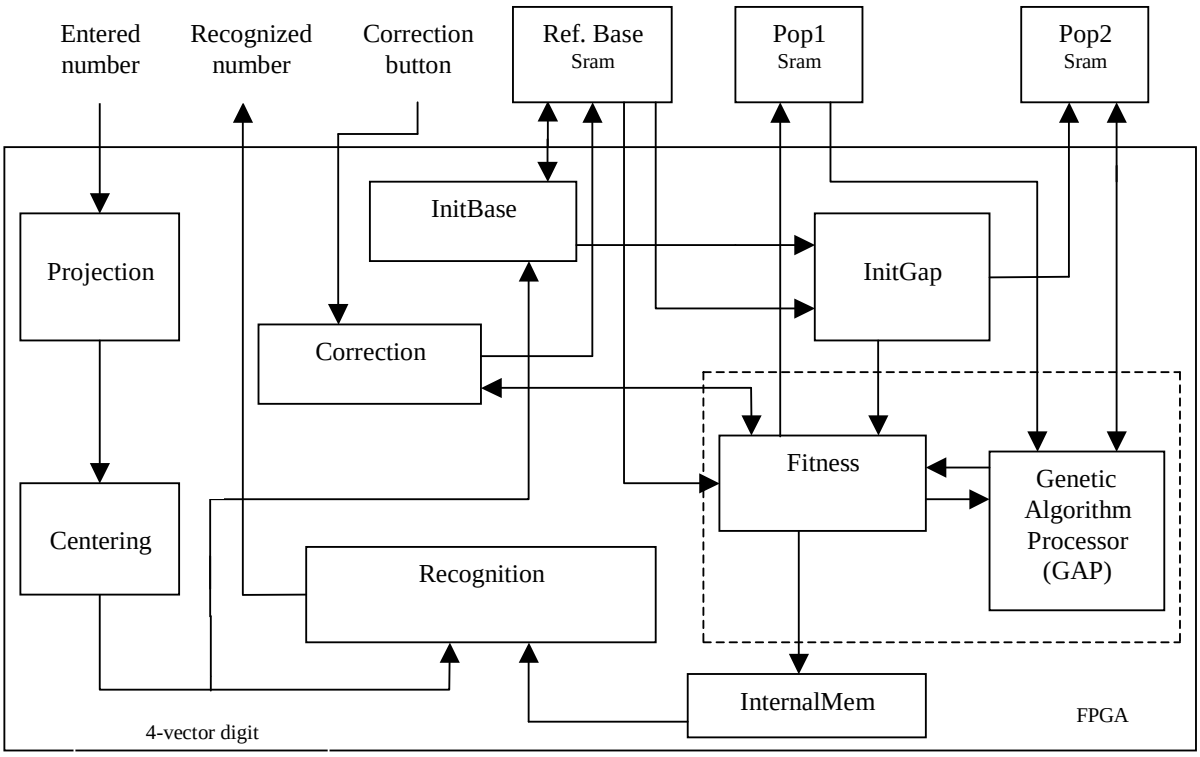


**Figure 6: The system architecture.**

0-7695-0762-X/00 $10.00 © 2000 IEEE

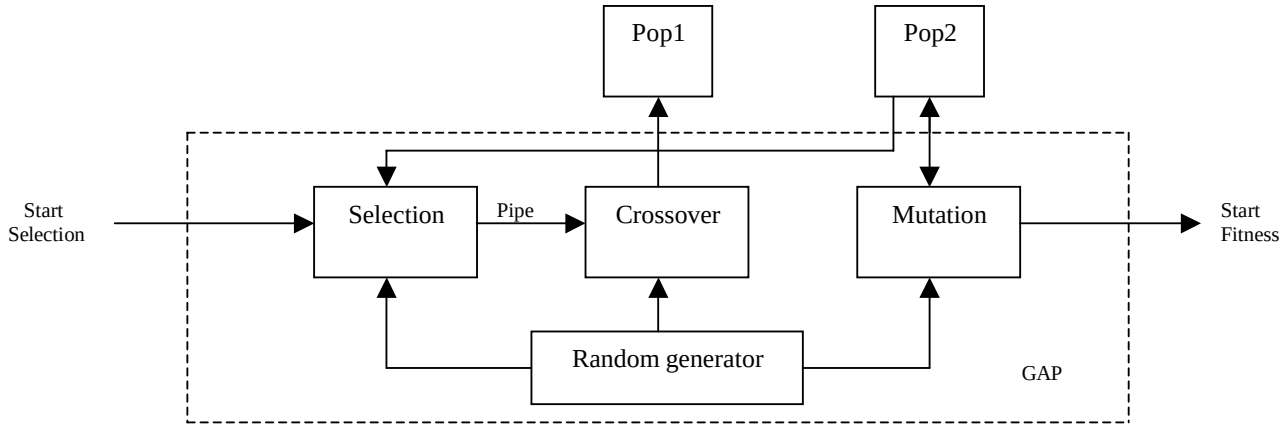


**Figure 7: The Genetic Algorithm Processor.**

### 4.2 FPGA circuits

An FPGA, or field-programmable gate array [7], is an array of logic cells placed in an infrastructure of interconnections, which can be programmed at three different levels: (1) the function of the logic cells, (2) the interconnection between cells, and (3) the inputs and outputs. All three levels are configured via a string of bits that is loaded from an external source. FPGAs are highly versatile devices that offer the user a wide range of design choices.

For testing purposes, the system was implemented on a RC1000-PP PCI bus plug-in card made by Embedded Solutions Limited (www.embeddedsol.com). It has a fairly powerful FPGA, type XCV1000 BG560 speed grade –4 of the Virtex® family made by Xilinx (www.xilinx.com). This particular circuit can be configured with a system equivalent to one million logic gates. There are also four memory banks of 2 Mbytes each that can be accessed by both the host PC and the FPGA.

### 4.3 General architecture of the system

For the time being, the host PC performs the input of the character to be recognized and the character that is recognized is displayed by the LEDs on the RC1000-PP card. All other functions are performed directly by the FPGA. Figures 6 and 7 present schematics of the global architecture of the system.

**The memories:**

- The reference database is used to store 10 samples of each decimal digit. It uses one of the on board 32-bit wide 2Mbyte SRAM memory banks.
- The two population memories represent the present (Pop1) and future population (Pop2); each uses a 32-bit wide SRAM memory bank.

**Inside the FPGA:**

- The internal memory stores the genome used as a reference for the unknown character to be recognized. This 32-bit wide memory is implemented using the dedicated blocks of on-chip, true dual-read/write port synchronous RAM with 4096 memory cells, which the Virtex FPGA provides.

- The Init database module initializes the reference database memory. The user has to manually enter a decimal digit, which is then projected, centered and stored in vector form in the database memory. This is repeated 100 times in total.

- The Init GAP module works only once to initialize the Pop2 memory so the GAP (Genetic Algorithm Processor) can start the genetic operations. This module randomly copies vectors from the reference database to the Pop2 memory

0-7695-0762-X/00 $10.00 © 2000 IEEE

in order to create an initial population of 100 genomes.

- The fitness module calculates the fitness for each genome stored in the Pop2 memory. This module uses the algorithm described in Section 3.3 and copies the genomes with their fitness from Pop2 to Pop1. Once these tasks are completed for one generation, the fitness module replaces the genome in the internal memory with a better one. The module must thus communicate with four different memories: the two population memories, the reference database memory, and the internal memory. The difference between the reference vectors from the database memory and the genes from the Pop2 memory is calculated 16 bits at a time using 16 xor gates and an adder.

- The GAP (Genetic Algorithm Processor) can be further divided into the following modules (*see Figure 7*):

  - The 36-bit pseudo random number generator generates a new random number each clock cycle. This method eliminates the need to request a new random number when needed. The random numbers are used at various stages in the genetic process.

  - The selection module selects a genome using the method described in Section 3.2. It then reads the selected genome from the Pop1 memory and transfers it to the crossover module. It is connected to the crossover module through a 32-bit wide pipeline.

  - The crossover module randomly chooses a crossover point for the genomes. It waits for the selection module to start sending the first genome through the pipeline. It then copies the genome to the Pop2 memory, up until the crossover point. It subsequently leaves enough space to copy the second genome and continues copying the second part of the first genome. Finally, when the second genome is sent by the selection module, it is copied to the free space skipped when copying the first genome. This process is repeated untill all of the Pop2 memory is full.

  - The mutation module randomly reads a 32-bit chunk of a genome, stored in the Pop2 memory, of which one bit is then randomly modified. The 32 bits are then written to the read-address in order to complete the mutation. This process is repeated 80 times per generation.

- The projection module calculates the four projection vectors for each input character. The handwritten character is received line by line, 16 bits at a time. Each received line will increment 4 shift-registers. There are two 16 x 3-bit incrementable shift-registers for the horizontal and vertical projection vectors and two 31 x 3-bit incrementable shift-registers for the diagonal projection vectors. The projection module accounts for the largest surface used by the FPGA: the shift-registers require a total of 282 registers, 94 adders, and 188 muxes.

- The centering module centers the four projection vectors. The centering is performed in parallel on the four vectors to improve the response time of the system. The centering is not done directly on the inputted character, but rather on the projection vectors, because of the greater ease of implementation, while the result remains exactly the same. The same shift-registers are employed by the projection module. Shift-left and shift-right operations are used to correctly center the vectors. Once centered, there are as many zeros on the left side of the vector as on the right side.

- The recognition module outputs the recognized decimal digit. It calculates the difference between the projected and centered character to be recognized, and each of the 10 genes stored in the internal memory. The gene that generated the smallest difference yields the decimal value of the input character. This value is then shown using the LEDs. As in the fitness module, the difference is calculated 16 bits at a time using 16 xor gates and an adder.

- The correction module is used to allow the system to continue to evolve after the system has been initialized. It enables the user to enter new reference characters into the database, thus replacing the older characters. In doing so, the best fitness value is reset so that a new, better-adapted genome will be stored in the internal memory. Since each newly added character will replace the oldest character of the same type, there are ten registers that point to the oldest

0-7695-0762-X/00 $10.00 © 2000 IEEE

character in the reference database memory. This creates a FIFO writing order for each digit.

## 5 Results and future developments

This project was written entirely in VHDL capable of being synthesized. The code was simulated using ModelSim by Mentor Graphics and was synthesized using Leonardo Spectrum by Exemplar. Finally, the place-and-route were performed using the Xilinx Alliance Design Manager.

In this system the following performance characteristics were obtained: only 12,28 % of the FPGA is used and the operating frequency is 30 Mhz. This translates into approximately 12 new generations per second.

The entire project was first simulated in C, using the same algorithms as the ones ultimately implemented on the FPGA. This simulation demonstrated that the system is able to correctly recognize 99% of the handwritten decimal digits of a test database.

This is still *work in progress*. We are currently in the process of developing a stand-alone board containing the same FPGA, but that will also include a touch screen.

## References


[1] “A new species of hardware”, M. Sipper and E. M. A. Ronald, IEEE Spectrum, Vol. 37, No. 3, pp. 59-64, March 2000.

[2] “A novel parallel approach to character recognition and its VLSI implementation”, H. D. Cheng and D. C. Cia, Pattern Recognition, Vol. 29, No. 1, pp. 97-119, 1996.

[3] “Historical review of OCR research and development”, S. Mori, C. Suen, K. Yamamoto, Proc. IEEE, Vol. 80, No. 7, pp. 1029-1058, July 1992.

[4] “Handwritten-digit recognition by neural networks with single-layer training”, S. Knerr, L. Personnaz, G. Dreyfus, IEEE Transactions on neural networks, Vol. 3, No. 6, Nov 1992.

[5] “The Simple Genetic Algorithm: Foundations and Theory”, M. D. Vose, The MIT Press, 1999.

[6] “Genetic Programming”, W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, Morgan Kaufmann Publishers, Inc., 1998.

[7] “Field-Programmable Gate Array Technology”, edited by S. M. Trimberger, Kluwer Academic Publishers, 1994.


0-7695-0762-X/00 $10.00 © 2000 IEEE