Published in *Proceedings of the NASA/DoD Conference on Evolvable Hardware 2003*, Chicago, IL, USA, 9-11 july 2003, which should be cited to refer to this work.

DOI: 10.1109/EH.2003.1217666

Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot

Daniel Roggen, Stéphane Hofmann, Yann Thoma* and Dario Floreano
Autonomous Systems Laboratory, Institute of Systems Engineering
*Logic Systems Laboratory, Institute of Computing and Multimedia Systems
EPFL, Lausanne, Switzerland
http://asl.epfl.ch
*http://lslwww.epfl.ch

E-mail: name.surname@epfl.ch

Abstract

A cellular hardware implementation of a spiking neural network with run-time reconfigurable connectivity is presented. It is implemented on a compact custom FPGA board which provides a powerful reconfigurable hardware platform for hardware and software design. Complementing the system, a CPU synthesized on the FPGA takes care of interfacing the network with the external world. The FPGA board and the hardware network are demonstrated in the form of a controller embedded on the Khepera robot for a task of obstacle avoidance. Finally, future implementations on new multi-cellular hardware are discussed.

1. Introduction

Spiking neurons depart from traditional connectionist models in the sense that the information is transmitted by the mean of pulses (or spikes) [9], rather than by firing rates. This may allow spiking neurons to have richer dynamics and to exploit the temporal domain to encode or retrieve information in the exchanged spikes. Spiking neurons have been used as controllers in evolutionary robotics, e.g. to perform vision-based obstacle avoidance [7] and phototaxis [4]. Hardware implementations of spiking networks may bring benefits such as very low power consumption, for example in neuromorphic sensors [13, 17], or very fast update speed [12, 22].

Here a cellular digital hardware implementation of a spiking neural network with run-time reconfigurable connectivity is presented. It is composed of a regular 2D array of cells where each cell acts as a spiking neuron. Cells have a configuration input which allows to change the type and the connectivity of the neuron at run-time. The system is validated in a task of navigation for the Khepera robot [15].

The custom FPGA board (FPGA module) provides a compact and powerful reconfigurable hardware platform. Embedded hardware and software design is possible (soft-core processor) and can be used for evolvable hardware or evolutionary robotics research. It can be used as a standalone device or on robots such as the Khepera.

Section 2 describes the FPGA module. Section 3 explains the principles of the cellular spiking neural network with reconfigurable connectivity. Section 4 describes the implementation on the FPGA and the experiment with the Khepera robot. Results are discussed in section 5 before concluding in section 6.

2. FPGA Module

FPGAs are very flexible for implementing hardware systems. Any digital system can be implemented inside, provided that the resources are sufficient (fig. 1). A compact FPGA module has been developed, based on the APEX20K200E FPGA (200'000 gates) from Altera. It can be used as a standalone device, or plugged on mobile robots. In particular it is fully mechanically and electrically compatible with the Khepera miniature robot [15] which is a robot which accepts extension modules. This FPGA module provides a powerful reconfigurable hardware platform for evolvable hardware or evolutionary robotics experiments.

There exists another FPGA module compatible with the Khepera, developed by Applied AI Systems Inc. [2]. It is based on the discontinued XC6216 FPGA from Xilinx [32]. Although that FPGA is a very suitable device for unconstrained evolution [28], it does not suit our application because of the few resources available (24'000 gates).

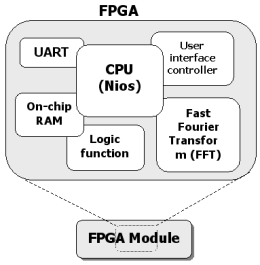

Figure 1. Any digital system can be implemented on an FPGA (given enough resources): e.g. CPU, dedicated hardware units (FFT unit, logic functions), serial communication controller, on-chip RAM.

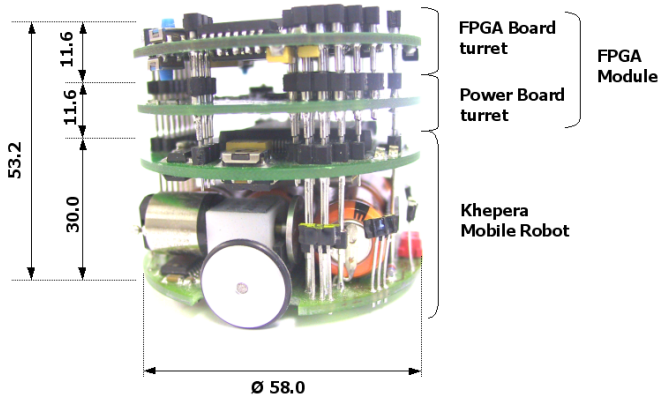

Figure 2. The FPGA module mounted on top of the Khepera robot. The module is composed of two PCBs, one for the FPGA and another for the power supply. The module can also be used standalone or on other robots.

2.1 Description and Architecture

The FPGA module contains the FPGA together with SRAM and Flash memory and several connectors to interface with a PC and user extensions (table 1 lists the features of the module). When used on the Khepera robot, the FPGA module extends considerably the capabilities of the robot. However, the module can also be used standalone or on another robot. In the latter case, the I/O pins previously used to communicate with the Khepera become user I/O.

Some tough problems of space optimization had to be solved to fit the system on a PCB of 58mm of diameter (size of the Khepera robot). The result is a module composed of two stacked PCBs (fig. 2):

- The FPGA board turret (digital part) includes the major components such as the FPGA, the configuration controller (EPM7064), the memories, the oscillator, the serial transceiver, etc. A supplementary clock pin is placed next to the existing Khepera bus pins to allow for stacked FPGA modules to share the same clock.

- The Power board turret (analog part) comprises the voltage regulators that generates the 1.8V and the 3.3V required by the digital components. The supplementary power pins (1.8V, 3.3V, GND) are placed next to the existing Khepera bus pins to be used by the FPGA board turret or by other user turrets.

• APEX20K200E-2X FPGA with 200'000 gates, 106'496 RAM bits (8320 logic elements)
• 1 MByte Flash memory (512Kx16)
• 256 KBytes SRAM (two 64Kx16 chips)
• 2 user and 2 system push-buttons, 3 user LEDs, configuration switches
• 26 3.3V user I/O, 2 5V-compatible user I/O (e.g. TTL serial line)
• 26 5V-compatible I/O for the Khepera bus or user I/O
• Stackable modules (multi-FPGA system sharing a single clock)
• RS232 connector with transceiver
• JTAG connector for Altera ByteBlasterMV and MasterBlasters programmers
• Compatible with Excalibur board software development Kit
• Supply voltage: 4.5V to 25V
• Power board generates 1.8V and 3.3V (1.4A each voltage)
• On-board logic for configuring the FPGA from Flash
• Self-reconfiguration may be triggered by the FPGA
• Oscillator (33MHz) and zero skew clock distribution
• Power-on reset circuitry
• Clock and power pins available for user modules

Table 1. Features of the FPGA module

The architecture of the FPGA module is similar to that of the Excalibur development board [1], modified to perform as a fully Khepera-compatible extension module. The module is also compatible with the Excalibur development tools. The latter allow to design SoC (System on a Chip), mixing hardware and software. Those tools provide a synthesizeable 16 and 32 bit CPU (called Nios) and several peripherals are readily available (e.g. UART or SPI communication controllers, timers, parallel I/O). The main parts of the FPGA module architecture are shown in fig. 3 and are described below.

Memories and memory bus. The Flash memory contains the configuration of the FPGA, used on power-up or reset. It can also store user data or CPU-executable files. Two SRAM chips can be used for program and data memory. A memory bus interfaces the SRAMs and Flash memories to the APEX FPGA and to the configuration controller. The FPGA has full access to the memories. The configuration controller only accesses the address bus and control signals.

Configuration controller. The FPGA is SRAM-based and its configuration is volatile, i.e. its configuration needs

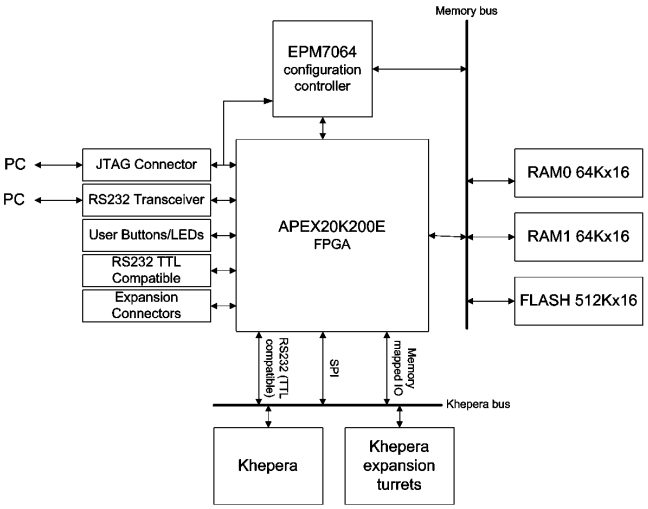

Figure 3. FPGA module architecture: in addition to the FPGA there are RAM and Flash memories and connectors for interfacing.

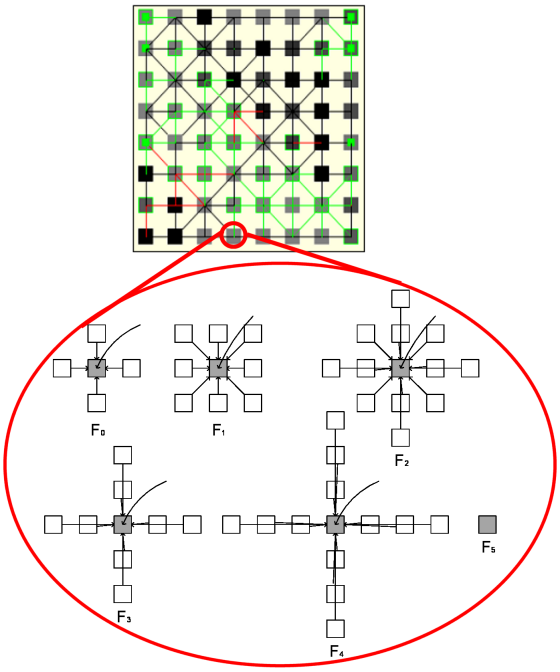

Figure 4. Top: cellular structure of the network. Each box is a cell (neuron). Bottom: input connectivity pattern allowed for each neuron. The connectivity pattern indicates from which neighbours (outlined boxes) the current neuron (gray box) receives spikes. In addition, each neuron has an extra input from the oustide world. In total there are 6 connectivity patterns. The bottom right connectivity pattern is an unconnected cell. The complete set of functionalities is doubled because each neuron can be excitatory or inhibitory, giving a total of 12 different functionalities.

to be downloaded after each power-up or reset. The configuration controller takes care of the download procedure by transferring the configuration file from the Flash to the FPGA. The configuration controller is a non-volatile device and is programmed only once. In normal use this device is never reconfigured.

User interfaces. The module provides two user push-buttons, three user LEDs and two system push-buttons (CPU-clear and system-reset). Configuration switches allow the user to select configuration options of the module (JTAG target device, clock source for multi-module systems, self-reconfiguration enable, main/alternate FPGA configuration selection). In addition, several extension connectors are available, providing 26 3.3V I/O pins, 2 5V-compatible I/O pins, power (GND, 1.8V, 3.3V, 5V) and clock pins.

Programming interfaces. A JTAG and a serial interface with an RS232 transceiver is available through the same connector. The serial interface allows for communication with the FPGA and downloading user programs in the CPU memory. The JTAG interface is used to download a design in the configuration controller or the FPGA.

Khepera bus. All the digital I/O pins of the Khepera expansion bus are connected to the FPGA and hence several interfacing modes are possible between the FPGA and the Khepera or the Khepera extension modules. Memory mapped I/O is the simplest form of interfacing. However it is also possible to use serial communication (RS232) or the K-Net protocol developed by K-Team [14]. When used as a standalone module, 26 supplementary 5-V compatible pins are available for general purpose I/O.

3. Cellular spiking network with reconfigurable connectivity

In this project we are interested in exploring the process of growth and differentiation in multi-cellular organisms. Therefore the chip is organized in a number of cells (detailed in section 3.1) that can express a certain functionality resulting from a simple morphogenetic process (described in section 3.3).

The multi-cellular organism has the structure of a regular 2D array of cells. Each cell implements one functionality, which is selected from a set of predefined functionalities (something akin to skin, muscle, neuron cells, etc. in living

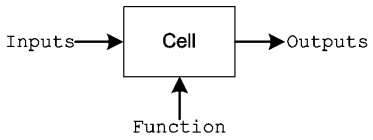

Figure 5. The cell has `inputs` (spikes coming from neighbouring cells), `outputs` (spike sent to neighbouring cells) and a `function` input indicating what is the function of the cell (connectivity and sign of the neuron). In total 12 functions can be selected with this input: 6 connectivities times two sign types.

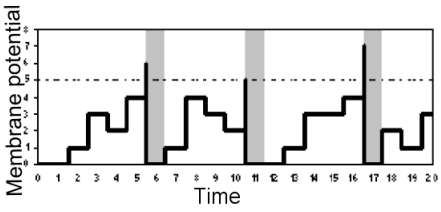

Figure 6. Effect of incoming spikes on the membrane potential. Each time a spike is received the membrane potential is increased. When a threshold is reached the neuron emits a spike (gray column) and the potential is reset to 0.

organisms). In the present case the cellular system implements a spiking neural network. Hence the functionality of each cell is a spiking neuron. However there exist several kinds of spiking neuron. Each kind is considered as a different functionality that a cell can assume. The different functionalities are given by a combination of connectivity pattern (one of 6 possible connectivity patterns) and type of neuron: excitatory or inhibitory (spike "sign"). The connectivity pattern indicates, for each neuron, from which connected neighbouring neurons it will receive spikes. In total, each cell can assume one of 12 different functionalities. This set of functionalities is called a *family of functions*. Figure 4 (top) shows an array of 8 by 8 cells, where each cell is a spiking neuron. Figure 4 (bottom) shows the 6 connectivity patterns that are available.

3.1 Cells

Cells have inputs and outputs to exchange data, as well as a configuration input (fig. 5). Here cells act as spiking neurons (although one could imagine different functionalities, such as filters, etc.) Inputs are used to receive incoming spikes from neighbouring neurons, and outputs are used to send outgoing spikes. The configuration input (`function`) is used to select the functionality of the cell inside the family of functions.

The cellular implementation has two characteristics. First, all cells are architecturally identical, although they may be functionally different. Second, all the necessary logic for the behaviour of a cell resides within it (common blocks among several cells cannot be shared).

To allow run-time reconfiguration a condition must be met: cells must be *totipotent*. In other words cells must implement all possible functionalities (combinations of connectivity pattern and neuron type). The effective cell functionality changes at run-time according to the `function` input.

Totipotent cells must be able to connect according to any of the predefined connectivity patterns. This is done by implementing all the connectivity patterns for the cell at design-time. At run-time the `function` input is used to select which pattern to effectively use. This approach includes some overhead as more routing resources and logic are used on the FPGA than what will be effectively used for a given configuration at run-time.

In addition each totipotent cell must be able to implement excitatory or inhibitory neurons. The neuron type has no effect on the computation happening within it, but has effect on the postsynaptic neurons. Neurons must hence know the type of the presynaptic neurons they are connected to. In the current implementation each neuron has two outputs: one along which the spikes are sent, and another indicating the type of the neuron. If routing ressources are critical alternatives exist: it is possible to use a serial implementation where neurons first transmit their sign, then the spikes.

The functionality of each cell comes from a memory storing the configuration of the whole network. However, this is not the FPGA configuration itself (i.e. the bitstream generated by the synthesis tools and downloaded to it). Similarities exist between cells described here and SBlocks of the virtual FPGA [11] or the Virtual Reconfigurable Circuits [23], which are also configurable blocks or systems implemented over an FPGA to allow abstraction from the FPGA architecture.

3.2 Spiking neuron model

The spiking neuron model is a minimalistic model which allows compact implementations and which has been previously implemented on a microcontroller with only 60 bytes of RAM [8]. It is a discrete-time, integrate-and-fire model with leakage and a refractory period. Each neuron has weighted inputs (+2 or -2 depending on whether the presynaptic neuron is excitatory or inhibitory) from connected

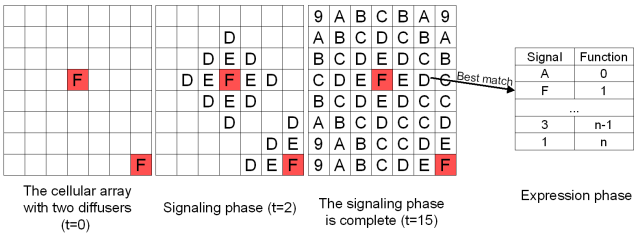

Figure 7. The three arrays on the left are snapshots of the signaling phase with one type of signal and two diffusers (gray cells) at the start of the signaling phase (left), after two time steps (middle) and when the signaling is complete (right). The number inside the cells is the intensity of the signal in hexadecimal. The expression table used in the expression phase is shown on the right. In this example the signal D matches the second entry of the table with signal F (smallest Hamming distance), thus expressing function F_1.

neurons. Each neuron has one more external input, e.g. to connect from a sensor, with fixed weight of +10. The neuron integrates the incoming spikes in the membrane potential, according to the weights of the connections. Once the membrane potential reaches a threshold (here set to 4), the neuron fires (emits a spike), resets its membrane potential to 0 and enters a refractory period where it does not integrate incoming spikes for one time step. After the integration phase and if the neuron has not fired, leakage is applied by decrementing membrane potential by 1. If the potential is below zero then it is reset to zero to limit its dynamic range. The main characteristics compared to other models is that few computations are needed to update the neuron state at each network step (e.g. no multiplications or exponentials). Figure 6 shows the effect of incoming spikes on the membrane potential.

3.3 Evolution

Finding the functions to put in each cell of the system can be difficult and hence evolution is used. Direct codings [33] are not very flexible in cases where new cells are added to the system (no bits are available to give a function to that cell), or when the system reorganizes, because the number of elements in the system must be known in advance and cannot change throughout the life of the system. Also direct codings may pose some problems of scalability.

Therefore we have developed an alternative genetic coding, called *morphogenetic system*, which has been designed specifically for cellular circuits. It assigns a functionality to each cell of the circuit from a set of predefined functionalities. The process works in two phases: first a *signaling phase* then an *expression phase* (fig. 7). The signaling phase relies on the ability of the cellular circuit to exchange signals among adjacent cells to implement a diffusion process. The second phase, expression, finds the functionality to be expressed in each cell by matching the signal intensities in each cell with a corresponding functionality stored in an expression table. The genetic code or chromosome contains the position of diffusing cells (diffusers) and the contents of the expression table, which are both evolved using a genetic algorithm. For more details see [21]. The morphogenetic system is remotely inspired by the mechanisms of gene expression and cell differentiation of living organism [3], notably by the fact that concentrations of proteins and inter-cellular chemical signaling regulate the functionality of cells. Related work include the use of development mechanisms coupled with genetic algorithms, e.g. [5, 10]. Also biologically plausible development models have been proposed which may help evolve more complex systems [16].

The morphogenetic system is implemented in software in the Nios CPU within the FPGA. The CPU sets the `function` input of each cell according to the chromosome.

4 Hardware implementation

An array of 64 neurons has been implemented on the FPGA module. As an example application, it was used in a task of obstacle avoidance on the Khepera robot. For this reason we not only describe the network but also the necessary interfaces to communicate with the robot.

4.1 Description

The implementation is a mixed hardware and software design. The hardware contains mainly the spiking network itself. The software (executed in the Nios CPU) takes care of interfacing the network with the outer world, i.e. the robot, the user and the I/Os. The architecture shown in figure 8 includes the Nios CPU, a neural network layer, a configuration layer and a network I/O interface. The function of each part is described below.

4.1.1 CPU

A 16-bit Nios CPU synthesized on the FPGA interfaces between the network and the rest of the system. This includes communication with the robot (sending motor commands, reading sensors) through the TTL-compatible serial communication line of the Khepera robot. The CPU also communicates with the host PC by sending monitoring infor-

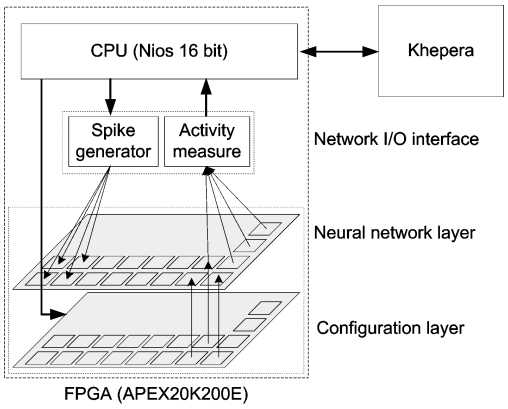

Figure 8. System architecture. The neural network layer contains the cellular spiking network. The configuration layer holds the configuration of the network layer. The network I/O interface allows to write spiketrains and read the activity of neurons. The CPU handles the communication with the Khepera.

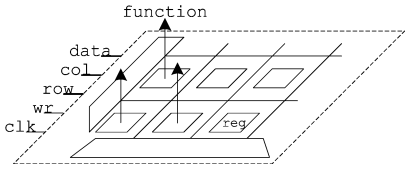

Figure 9. The configuration layer holds the functionality of each neuron. It is implemented as a writeable array of registers. The contents of each register are sent to the `function` input of the corresponding cell.

mations. The CPU can run and stop the spiking network, for example to monitor its activity step by step or to match its speed with that of the rest of the system. The CPU also interprets the chromosome by running the signaling and expression phases of the morphogenetic system. It then configures the network accordingly by the mean of the configuration layer.

4.1.2 Configuration layer

To clarify the architecture, the configuration layer is separate from the network layer. The configuration layer (see fig. 9) holds the information controlling the functionality of each cell (neuron in this case) in the network layer. The configuration layer is implemented as an array of registers, with one register per cell. The contents of the registers are connected to the `function` input of the cell. The `row` and `col` inputs allow to select a specific register by means of decoders which can then be written by the CPU using the `wr` and `data` inputs

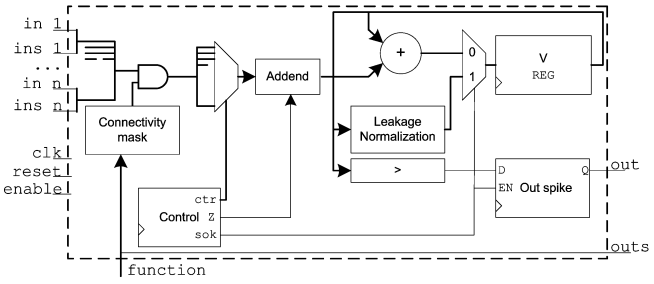

Figure 10. Cell architecture implementing a spiking neuron. The main parts are a register holding the membrane potential value, a connectivity mask block, an addend block, a leakage and normalization block and a control unit.

4.1.3 Neural network layer

The neural network layer is composed of an array of totipotent cells which implement spiking neurons (outlined boxes in the network layer in fig. 8 are cells). The cells are connected at design-time to a set of 25 neighbouring cells so that they can express all connectivity patterns described previously (a 26th input is used as the external input to the neuron).

Figure 10 shows the architecture within the cell to implement a spiking neuron. Control inputs (`clock`, `enable`, `reset`) allow to run, pause and reset the network. The inputs from connected neurons are `in1`..`in`n, which convey the incoming spikes, and `ins1`..`ins`n which indicate the type of connected neurons (spike "sign"). The `function` input tells the neuron which connectivity pattern to use and what is the type of the neuron. The two outputs are `out`, along which output spike is sent, and `outs`, which is the type of the current neuron, as defined by `function`.

The spiking neuron contains a 7-bit register (V) holding the membrane potential. The register size is defined by the number of inputs and by their maximum weights (25 inputs with a weight of +/-2 and one external input with a weight of +10). The inputs are time-multiplexed. The number of clocks for one network step is equal to $n+1$ (where n is the total number of inputs, $n = 26$ here).

The *control* unit is in charge of generating the necessary sequence to multiplex the inputs in clock cycles 1 to n, to update the register with the contribution of the inputs (`sok=0`). At clock cycle $n + 1$ (`sok=1`) the output is updated (emission of a spike if the register is above a

given threshold) and the register is loaded with the content of the *leakage and normalization* block. That block decrements the value of the register if it is not above the threshold (leakage) or resets the register to 0 if its value is above the threshold or below 0.

The generation of the proper run-time connectivity is done by the *connectivity mask* unit. It generates a connectivity mask, according to the `function` input, implemented as a look-up table. The connectivity mask is then ANDed with the inputs to give the effective input spikes. Those are, together with the signs, multiplexed and sent to the *addend* block. This block generates the value to be added to the register, by means of multiplexers. Signal `Z` of the *control* unit indicates whether the addend block handles input 1 (external input) which has a fixed weight of 10, or inputs 2 to n which have a weight of +2 or -2, depending on the sign.

4.1.4 Network I/O interface

The network I/O interface is composed of two types of units (fig. 8) which allow the network to run with minimal software intervention. The first unit, the *spike generator*, is a frequency-programmable generator of spikes which is used to feed sensory informations to the network (e.g. spike frequency proportional to the activation of a sensor). The second unit, the *activity measure*, measures the average activity of the connected neuron over a given time window. This allows to actuate effectors proportionally to the firing rate of a neuron (the network is updated several times before reading the activity measurement unit).

4.2 Implementation results

The behaviour of the whole system has been tested by generating random networks, applying random inputs, and comparing the spike trains of each neuron to a software implementation.

After place and route, timing analysis gives a theoretical maximum frequency of about 42MHz. A network update requires 27 clock cycles. At the operating frequency of the FPGA module of 33MHz this gives 1.2 millions updates/second.

Table 2 summarizes the number of logic elements (LE) and the longest register to register delay for the complete system and its main blocks. Data for the main blocks are obtained by standalone compilation of the blocks. Synthesis optimizations may reduce resource use when blocks are part of the complete system. This is the case for the neuron: it uses 109 LE standalone but when part of a network of 64 neurons the resources are of about 90 LE per neuron. Clearly the network and configuration layer is taking most of the space (5939 LE of the 8222 LE of the whole system) and is where optimizations should be seeked. Also there is a significant amount of delay introduced when the neurons are interconnected (the network and configuration layer has a 75% longer register to register delay than a single neuron). As this may be a performance bottleneck, further study is needed to identify what causes that delay and whether it can be decreased.

Entity	Resource use (LE)	Longest delay
• Complete system	8222	23.3 ns
• CPU (incl. UART and I/O)	2121	17.9ns
• Network interface	163	7.83ns
• Net. and config. layer (64 cells)	5939	21.7ns
• Single neuron	109	13.3ns

Table 2. Number of logic elements and longest register to register delay after place and route for the complete system and its main blocks (compiled individually). The network interface is composed of 8 spike generators and 2 activity measurement blocks.

4.3 Robot obstacle avoidance

A spiking network was evolved as a controller for a Khepera miniature robot to do obstacle avoidance using the information coming from its infra-red sensors. The fitness functions seeks to maximize forward speed and distance from the obstacles and minimize rotation.

The system, shown in figure 11, is composed of an array of 8x8 neurons. Four sensory groups of two cells are connected to the infrared sensors via the spike generators. Two other cells are connected to the activity measurement units and are used to set the speeds of the wheels.

The robot runs with a sensory-motor period of 100ms. During that period, the network is updated 20 times. At the end of the sensory-motor period, the spike generators are programmed to reflect the new activity of the infra-red sensors for the next sensory-motor period. According to the distance to the obstacles, either 0, 1 (the ”low” activity), or both input neurons (”low” and ”high” activity) of each sensory group receive a spike train of period 2 (one spike every two time steps). Then the units in charge of measuring the activity of the output neurons are read and the speed of the wheels is updated accordingly. A minimum activity of the neuron sets the speed of the wheel to +80 mm/s. A maximum activity sets the speed to -80 mm/s. The speed of the wheels scales linearly in between. Note that the speed of the wheels is inversely proportional to the activity of the output neurons: this allows the robot to move forward when no obstacles are sensed and thus when there is potentially no activity in the network. Alternatively, to avoid introducing engineering knowledge, bias neurons (constantly firing

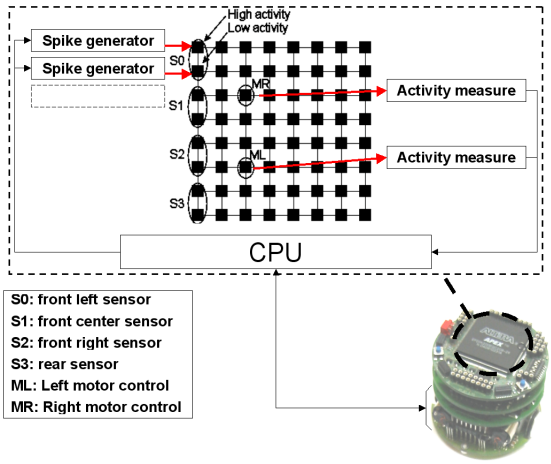

Figure 11. Application of the hardware spiking network. The objective is perform obstacle avoidance using the sensory information coming from the proximity sensors of the robot which are sent to 8 input neurons. Two output neurons control the speed of the wheels.

neurons) could be placed in the system to provide activity at all time.

Previously, evolution has been performed using a software simulation of the network (but with the real robot) to determine the parameters of the system (e.g. coding for the input and outputs). The software simulation did not use an hardware oriented language (like SystemC) but was written in C++. However, constructs which can be easily implemented in hardware were used and the software model of the neuron is the same as the hardware model. To test the hardware network, the chromosomes of some of the best individuals at the obstacle avoidance task have been downloaded to the Nios CPU, which programs the network accordingly. Experiments show that the fitness and behaviour of the robot using the hardware network is similar to that using the software network [21]. The variations depend on experimental factors (e.g. initial position of the robot).

5 Discussion

The work described in this paper is a continuation of previous research [21]. The novelty resides in the development of a compact FPGA-based module which can be used standalone or as a robot extension, and in the hardware implementation of the spiking neural network.

Other hardware implementations of spiking neurons have been proposed, in analog [19] and in digital circuits [12, 22]. Analog implementations require few transistors and can be very energy efficient. The cited digital implementations are specialized accelerator chips which focus on high speed. In this project, we are not particularly interested in speed, as the limiting factor is the speed of movement of the robot, but we are interested in exploring evolution of multi-cellular structures. Hence our system differs notably from previous architectures by its cellular nature: all the necessary logic for the behaviour of a cell resides within it, and all cells are architecturally identical.

The custom FPGA module has been demonstrated with the Khepera robot. However the module is not tied to the Khepera and can be used on other robots or in a standalone fashion.

A high network update speed is obtained: 27 clock cycles are necessary for updating a network of 64 neurons, which means 1.2 Mio updates/sec at the operating frequency of 33MHz. The number of clock cycles for a network update is independent of the number of neurons in the system. This departs from a software implementation where update time scales with the number of neurons. This network has a total of 1664 connections. In software, even if only one instruction of one clock cycle was needed to process a connection, this would result in at least 1664 clock cycles for a network update. In this case the hardware version is at least two orders of magnitude faster than the software version at identical clock speeds. High update speed can be very interesting. For example, in character recognition systems for postal addresses or in voice recognition systems, a fast throughput is needed.

In the robotic application presented here the network needs only be updated about every millisecond. Indeed it has been slowed down by disabling it during part of the time. In cases where network speed is not critical further space optimizations could be considered.

Resources used scale linearly with the number of neurons. The neuron model is minimalistic and permits compact implementations. It is believed that its implementation is quite efficient, although further space optimization may be possible. Also, less than 64 neurons can be used for obstacle avoidance [8]. However, we have previously investigated other applications which may also be implemented in hardware and which need that amount of neurons, e.g. pattern recognition [21].

The current neuron model lacks the capacity to adapt (learn). Models with richer dynamic and learning have been described and applied to the recognition of moving patterns [6]. Optimizations and simplifications have been proposed for hardware implementations [29, 30].

Network connectivity has been evolved using software simulations of the network. Implementing the genetic algorithm and measuring the fitness in the CPU within the FPGA to have complete evolution on the robot remains to be done, but should be fairly straightforward. Also the chro-

mosome could be decoded in hardware to obtain the function of each cell. This could be done in the configuration layer by means of signaling and expression units. Alternate signaling mechanisms could take into account the environment and may render the system more robust or allow runtime relocation of sensors and actuators [26].

Although the network can change connectivity at runtime, we do not reconfigure the FPGA but a circuit which is implemented in it. Devices exist which allow fast reconfiguration (e.g. Xilinx Virtex). When used with appropriate software (e.g. Xilinx JBits) it is possible to modify partially the configuration of the FPGA at run-time. Using FPGA reconfiguration may increase the flexibility of our system. However, a miniature system such as the one described in this article has only limited memory and computational power: embedding the manufacturer's tools to use reconfiguration is likely to be impossible.

The cellular implementation of the spiking neural network is particularly suited to custom hardware using the cell as the functional block, as in Embryonic-style circuits (e.g. the BioWall [25]). Such hardware uses the cellular approach to provide a self-repairing substrate [24] by using totipotent cells which differentiate according to their coordinates in the cellular system. When faults are detected ([18] pp. 251–258), the coordinate system is altered to avoid the faulty cell. The spiking network described in this paper, which is also based on the principle of totipotent cells, would suit nicely on an Embryonic substrate. This may add the very interesting property of self-repair to the network.

6 Conclusions

A cellular spiking neural network with reconfigurable connectivity has been implemented on an FPGA. An array of 64 neurons has been implemented and demonstrated in a task of obstacle avoidance for the Khepera robot.

The cellular spiking network is very suited for a new circuit which is under development: the POEtic circuit [31]. The POEtic circuit is meant to be a platform to test several bio-inspired mechanisms. It is a multi-cellular electronic circuit which is composed of a regular 2D array of cells (i.e. functional units). It includes hardware features to implement evolution ('P' or Phylogenesis), development ('O' or Ontogenesis) and learning ('E' or Epigenesis). The POEtic circuit is reprogrammable and its functionality comes from its configuration bits or genotype, which can be easily evolved. To implement development, the complete genome of the whole system can be stored in each cell, allowing growth and self-repair. In addition, each cell has virtual input/output connections to sensors and actuators.

Furthermore the POEtic circuit has a dynamical routing mechanism through which connections betwen components can be built automatically and at *run-time* [20, 27]. The dynamical routing can be launched by the logic elements, not only by an external controller. This is a very powerful feature which does not exist on conventional FPGAs, where physical connections must be planned at *design-time*: the compilation tools place the components on the FPGA and connect them using the available resources. Afterwards, there is no possibility to change connections (some recent FPGAs allow partial reconfiguration which could be exploited to this end, but extreme care must be used to avoid short circuits).

With the dynamical routing mecanism, cells need not be connected physically at design-time. Dynamical routing relies on identifiers which mark the source and target cells and builds the necessary connections at run-time. Cells can also ask at run-time to build new connections even if not originally planned. The network described in this paper has a run-time reconfigurable connectivity up to the extent that has been planned at design-time. With dynamical routing the connectivity patterns of the spiking network could be built automatically when needed and also new connectivity patterns could be explored at run-time. This would greatly increase the richness of the connectivity compared to the current approach.

The area required by a dynamical routing mechanism on a conventional FPGA would be very large, however, the POEtic circuit will contain this dynamical routing mechanism "for free" as part of its architecture. This makes it a platform of choice to implement cellular neural networks with reconfigurable connectivity. Such implementations will be considered in the future.

7 Acknowledgments

The authors wish to thank Jesper Blynel and Gianluca Tempesti for their comments and all the members of the POEtic team. This project is funded by the Future and Emerging Technologies programme (IST-FET) of the European Community, under grant IST-2000-28027 (POETIC). The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. The Swiss participants to this project are funded by the Swiss government grant 00.0529-1.

References

[1] Altera Corporation, San Jose, CA. *Nios Embedded Processor Development Board data sheet*, March 2001 (ver. 1.1).

[2] Applied AI Systems, Inc., Ontario, Canada (http://www.aai.ca).

[3] E. Coen. *The art of genes*. Oxford University Press, New York, 1999.

[4] E. A. Di Paolo. Spike timing dependent plasticity for evolved robots. *Adaptive Behavior*, 10, 2002.

[5] P. Eggenberger. Cell interactions as a control tool of developmental processes for evolutionary robotics. In P. Maes, M. J.

Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 440–448, Cambridge, MA, 1996. MIT Press-Bradford Books.

[6] J. Eriksson, O. Torres, A. Mitchell, G. Tucker, K. Lindsay, D. Halliday, J. Rosenberg, J.-M. Moreno, and A. E. P. Villa. Spiking Neural Networks for Reconfigurable POEtic Tissue. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Proceedings of the Fifth International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, pages 165–173, Berlin, 2003. Springer.

[7] D. Floreano and C. Mattiussi. Evolution of spiking neural controllers for autonomous vision-based robots. In T. Gomi, editor, *Evolutionary Robotics IV*, pages 38–61. Springer-Verlag, Berlin, 2001.

[8] D. Floreano, N. Schoeni, G. Caprari, and J. Blynel. Evolutionary bits'n'spikes. In *Proceedings of Artificial Life VIII*. MIT Press, 2002.

[9] W. Gerstner and W. Kistler. *Spiking Neuron Models*. Cambridge University Press, 2002.

[10] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3:151–183, 1994.

[11] P. C. Haddow and G. Tufte. Bridging the Genotype-Phenotype Mapping for Digital FPGAs. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *The Third NASA/DoD Workshop on Evolvable Hardware*, pages 109–123, Los Alamitos, California, 2001. IEEE Computer Society Press.

[12] G. Hartmann, G. Frank, G. Schäfer, and M. Wolff. SPIKE128K-An Accelerator for Dynamic Simulation of Large Pulse-Coded Networks. In *Proceedings of MicroNeuro 97*, pages 130–139, Dresden, 1997.

[13] G. Indiveri and R. Douglas. ROBOTIC VISION: Neuromorphic Vision Sensors. *Science*, 288:1189–1190, 2000.

[14] K-TEAM S.A., Préverenges, Switzerland (http://www.k-team.com). *K-Net Communication Protocol Specifications*.

[15] K-TEAM S.A., Préverenges, Switzerland (http://www.k-team.com). *Khepera User Manual*.

[16] S. Kumar and P. J. Bentley. Biologically inspired evolutionary development. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Proceedings of the Fifth International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, pages 57–68, Berlin. Springer.

[17] M. A. Lewis, R. Etienne-Cummings, A. Cohen, and M. Hartmann. Toward Biomorphic Control using Custom aVLSI Chips. In *Proceedings of the 2000 International Conference on Robotics and Automation*, 2000.

[18] D. Mange and M. E. Tomassini. *Bio-Inspired Computing Machines*. PPUR, Lausanne, 1998.

[19] C. Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA, 1991.

[20] J.-M. Moreno Aróstegui, E. Sanchez, and J. Cabestany. An in-system routing strategy for evolvable hardware programmable platforms. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *The Third NASA/DoD Workshop on Evolvable Hardware*, pages 157–166, Los Alamitos, California, 2001. IEEE Computer Society Press.

[21] D. Roggen, D. Floreano, and C. Mattiussi. A Morphogenetic Evolutionary System: Phylogenesis of the POEtic Tissue. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Proceedings of the Fifth International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, pages 153–164, Berlin, 2003. Springer.

[22] T. Schoenauer, S. Atasoy, N. Mehrtash, and H. Klar. NeuroPipe-Chip: A Digital Neuro-Processor for Spiking Neural Networks. *IEEE Transactions on Neural Networks*, 13(1):205–213, 2002.

[23] L. Sekanina. Virtual reconfigurable circuits for real-world applications of evolvable hardware. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Proceedings of the Fifth International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, pages 186–197, Berlin, 2003. Springer.

[24] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. A self-repairing and self-healing electronic watch: The biowatch. In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, editors, *Proceedings of the Fourth International Conference on Evolvable Systems: From Biology to Hardware (ICES 2001)*, pages 112–127, Berlin, 2001.

[25] G. Tempesti, D. Mange, A. Stauffer, and C. Teuscher. The biowall: An electronic tissue for prototyping bio-inspired systems. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 221–230. IEEE Computer Society, Los Alamitos, CA, 2002.

[26] G. Tempesti, D. Roggen, E. Sanchez, Y. Thoma, R. Canham, and A. M. Tyrrell. Ontogenetic Development and Fault Tolerance in the POEtic Tissue. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Proceedings of the Fifth International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, pages 141–152, Berlin, 2003. Springer.

[27] Y. Thoma, E. Sanchez, J.-M. Moreno Arostegui, and G. Tempesti. A dynamic routing alogrithm for a bio-inspired reconfigurable circuit. Submitted to the 13th International Conference on Field Programmable Logic (FPL'03), 2003.

[28] A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In T. Higuchi, M. Iwata, and L. Weixin, editors, *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware (ICES 96)*, pages 390–405, Berlin, 1997. Springer.

[29] O. Torres, J. Eriksson, J. M. Moreno, and A. Villa. Hardware optimization and serial implementation of a novel spiking neuron model for the poetic tissue. Submitted to IPCAT'03.

[30] O. Torres, J. Eriksson, J. M. Moreno, and A. Villa. Optimization of a novel spiking neuron model for the poetic tissue. Accepted for publication at IWANN'03.

[31] A. M. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J.-M. Moreno, J. Rosenberg, and A. Villa. POEtic Tissue: An Integrated Architecture for Bio-Inspired Hardware. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Proceedings of the Fifth International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, pages 129–140, Berlin, 2003. Springer.

[32] Xilinx Inc., Xilinx Web Site (http://www.xilinx.com). *XC6200 field programmable gate arrays. Datasheet*, April 1997.

[33] X. Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 4:203–222, 1993.