A VLSI Implementation of an Analog Neural Network suited for Genetic Algorithms

Johannes Schemmel[1], Karlheinz Meier[1], and Felix Schürmann[1]

Universität Heidelberg, Kirchhoff Institut für Physik, Schröderstr. 90, 69120 Heidelberg, Germany,
schemmel@asic.uni-heidelberg.de,
WWW home page: http://www.kip.uni-heidelberg.de/vision.html

Abstract. The usefulness of an artificial analog neural network is closely bound to its trainability. This paper introduces a new analog neural network architecture using weights determined by a genetic algorithm. The first VLSI implementation presented in this paper achieves 200 giga connections per second with 4096 synapses on less than 1 mm^2 silicon area. Since the training can be done at the full speed of the network, several hundred individuals per second can be tested by the genetic algorithm. This makes it feasible to tackle problems that require large multi-layered networks.

1 Introduction

Artificial neural networks are generally accepted as a good solution for problems like pattern matching etc. Despite being well suited for a parallel implementation they are mostly run as numerical simulations on ordinary workstations. One reason for this are the difficulties determining the weights for the synapses in a network based on analog circuits. The most successful training algorithm is the back-propagation algorithm. It is based on an iteration that calculates correction values from the output error of the network. A prerequisite for this algorithm is the knowledge of the first derivative of the neuron transfer function. While this is easy to accomplish for digital implementations, i.e. ordinary microprocessors and special hardware, it makes analog implementations difficult. The reason for that is that due to device variations, the neurons' transfer functions, and with them their first derivatives, vary from neuron to neuron and from chip to chip. What makes things worse is that they also change with temperature. While it is possible to build analog circuitry that compensates all these effects, this likely results in circuits much larger and slower than their uncompensated counterparts. To be successful while under a highly competitive pressure from the digital world, analog neural networks should not try to transfer digital concepts to the analog world. Instead they should rely on device physics as much as possible to allow an exploitation of the massive parallelism possible in modern VLSI technologies. Neural networks are well suited for this kind of analog implementation since the compensation of the unavoidable device fluctuations can be incorporated in the weights.

A major problem that still has to be solved is the training. A large number of the analog neural network concepts that can be found in the literature use floating gate technologies like EEPROM of flash memories to store the analog weights (see [1] for example). At a first glance this seems to be an optimal solution: it consumes only a small area making therefore very compact synapses possible (down to only one transistor [2]), the analog resolution can be more than 8 bit and the data retention time exceeds 10 years (at 5 bit resolution) [3]. The drawback is the programming time and the limited lifetime of the floating gate structure if it is frequently reprogrammed. Therefore such a device needs predetermined weights, but to calculate the weights an exact knowledge of the network transfer function is necessary. To break this vicious circle the weight storage must have a short write time. This would allow a genetic algorithm to come into play. By evaluation of a high number of test configurations the weights could be determined using the real chip. This could also compensate a major part of the device fluctuations, since the fitness data includes the errors caused by these aberrations.

This paper describes an analog neural network architecture optimized for genetic algorithms. The synapses are small, $10 \times 10\ \mu\text{m}^2$, and fast. The measured network frequency exceeds 50 MHz, resulting in more than 200 giga connections per second for the complete array of 4096 synapses. For building larger networks it should be possible to combine multiple smaller networks, either on the same die or in different chips. This is achieved by confining the analog operation to the synapses and the neuron inputs. The network inputs and neuron outputs are digitally coded. The synapse operation is thereby reduced from a multiplication to an addition. This makes the small synapse size possible and allows the full device mismatch compensation, because each synapse adds either zero or its individual weight that can include any necessary corrections. Analog signals between the different analog network layers are represented by arbitrary multi-bit connections.

The network presented in this paper is optimized for real-time data streams in the range of 1 to 100 MHz and widths of up to 64 bits. We plan to use it for data transmission applications like high speed DSL[1], image processing based on digital edge data produced from camera images by an analog preprocessing chip [4] and for the fitness evaluation of a field programmable transistor array [5] also developed in our group.

2 Realization of the Neural Network

2.1 Principle of operation

Figure 1 shows a symbolic representation of a recurrent neural network. Each input neuron (small circle) is linked to each output neuron (large circle) by a synapse (arrow). The output neurons are fed back into the network by a second set of input neurons. The input neurons serve only as amplifiers, while the

[1] digital subscriber line

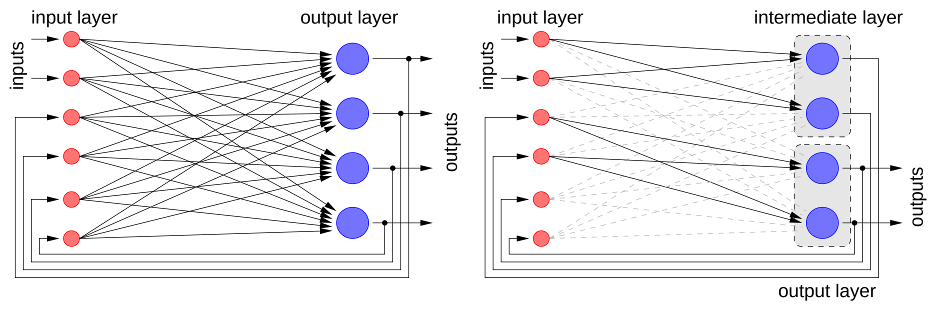


Fig. 1. Left: A recurrent neural network. Right: The same network configured as a two layer network.

processing is done at the output neurons. This architecture allows virtual multi-layer networks by choosing the appropriate weights. On the right of Figure 1 an example is shown for two layers. Synapse weights set to zero are depicted as dashed arrows. A recurrent network trained by a genetic algorithm has usually no fixed number of layers. Of course, the algorithm can be restricted to a certain number of layers, as in Figure 1, but usually it seems to be an advantage to let the genetic algorithm choose the best number of layers. Also, there is no strict boundary between the virtual layers. Each neuron receives input signals from all layers. To avoid wasting synapses if not all the feedback pathways are used, the presented network shares input neurons between external inputs and feedback outputs.

Figure 2 shows the operation principle of a single neuron. The synaptic

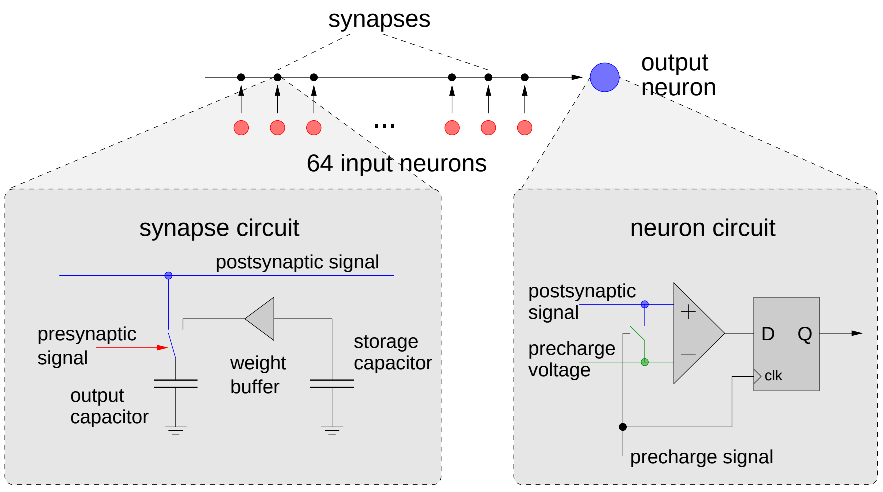


Fig. 2. Operation principle of the neuron.

weights are stored as charge on a capacitor (storage capacitor). The neuron operation is separated in two phases, *precharge* and *evaluate*. In the precharge phase all the switches in the synapses are set towards the buffer and the precharge signal in the neuron is active. In each synapse the output capacitor is charged via the weight buffer to the same voltage as the storage capacitor. The neuron consists of a comparator and a storage latch. The precharge signal closes a switch between both inputs of the comparator. This precharges the post-synaptic signal to a reference voltage that constitutes the zero level of the network.

In the evaluate phase the sum of all the synapses is compared to this precharge voltage. If the synapse signal exceeds it, the neuron fires. This neuron state is stored in the flip-flop at the moment when the phase changes from evaluate to precharge. In the evaluate phase the synapse switch connects the output capacitor with the post-synaptic signal if the pre-synaptic signal is active. The pre-synaptic signals are generated by the input neurons depending on the network input and feedback information.

This cycle can be repeated a fixed number of times to restrict the network to a maximum layer number and limit the processing time for an input pattern. The network can also run continuously while the input data changes from cycle to cycle. This is useful for signal processing applications.

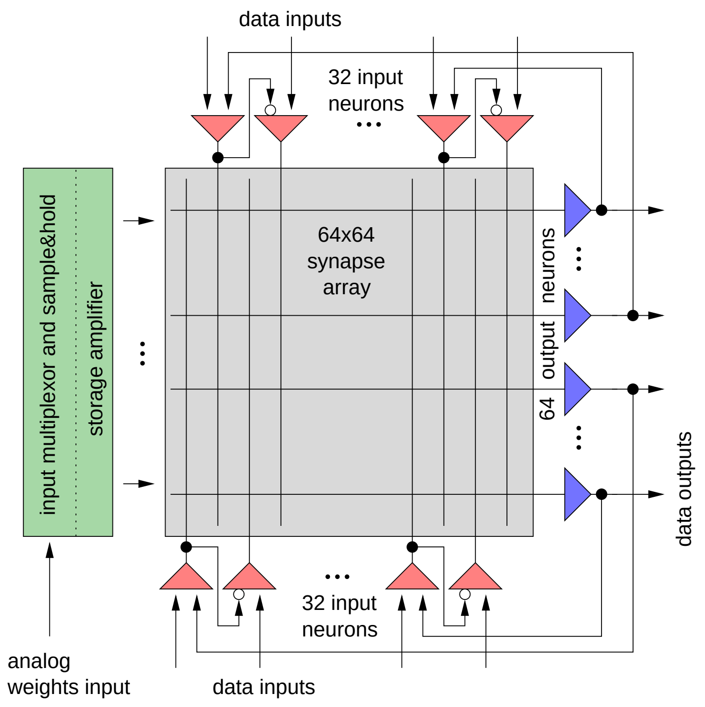


Fig. 3. Block diagram of the neural network prototype.

Figure 3 shows a block diagram of the developed neural network prototype. The central element is an array of 64×64 synapses. The post-synaptic lines of the 64 output neurons run horizontally through the synapses, while the pre-synaptic signals are fed into the array from the input neurons located below and above. Each input neuron can be configured to accept either external data or data from the network itself for input. This internal data comes alternately from two sources. The odd input neurons receive a feedback signal from an output neuron while the even ones get the inverted output from its odd neighbor. If the even neuron is switched to its odd counterpart, they together form a *differential* input since the binary signal is converted into a pair of alternately active inputs. This is useful for two purposes: if binary coded data is used the number of active input neurons stays always the same, independently of the input data. The second reason is linked to the way the post-synaptic voltage is calculated:

$$V_{postsyn} = \frac{\sum_{i=1}^{64} I_i Q_i}{\sum_{i=1}^{64} I_i C_i} \tag{1}$$

Q_i is the charge stored on the synapse output capacitor C_i. I_i is the pre-synaptic signal. As a binary value it is either zero or one. The neuron fires if $V_{postsyn} > V_{precharge}$. Not only the numerator, but also the denominator depends on all the input signals. This has the drawback that if one input signal changes, the other weights' influence on $V_{postsyn}$ changes also. Even though it can be shown that in the simplified model of Eq. 1 the network response stays the same, the performance of the real network may suffer. The differential input mode avoids this effect by activating always one input neuron per data input. The capacitance switched onto the post-synaptic signal line becomes independent of the data. Therefore the denominator of Eq. 1 stays the same for any changes of a differential input. The disadvantage is the reduced number of independent inputs since each differential input combines an odd with an even input neuron.

2.2 Implementation of the network circuits

A micro photograph of the fabricated chip can be seen in Figure 4. The technology used is a 0.35 μm CMOS process with one poly and three metal layers. The die size is determined by the IO pads necessary to communicate with the test system. The synapse array itself occupies less than 0.5 mm^2. It operates from a single 3.3 volt supply and consumes about 50 mW of electrical power. Figure 5 shows the circuit diagram of the synapse circuit. Both capacitors are implemented with MOS-transistors. The weight buffer is realized as a source follower built from the devices M1 and M2. The offset and the gain of this source follower vary with the bias voltage as well as the temperature. Therefore an operational amplifier outside of the synapse array corrects the weight input voltage until the output voltage of the source follower equals the desired weight voltage which is fed back via M7. The charge injection error caused by M6 depends on the factory induced mismatch that can be compensated by the weight value. M3 is closed in the precharge phase of the network to charge the output capacitor

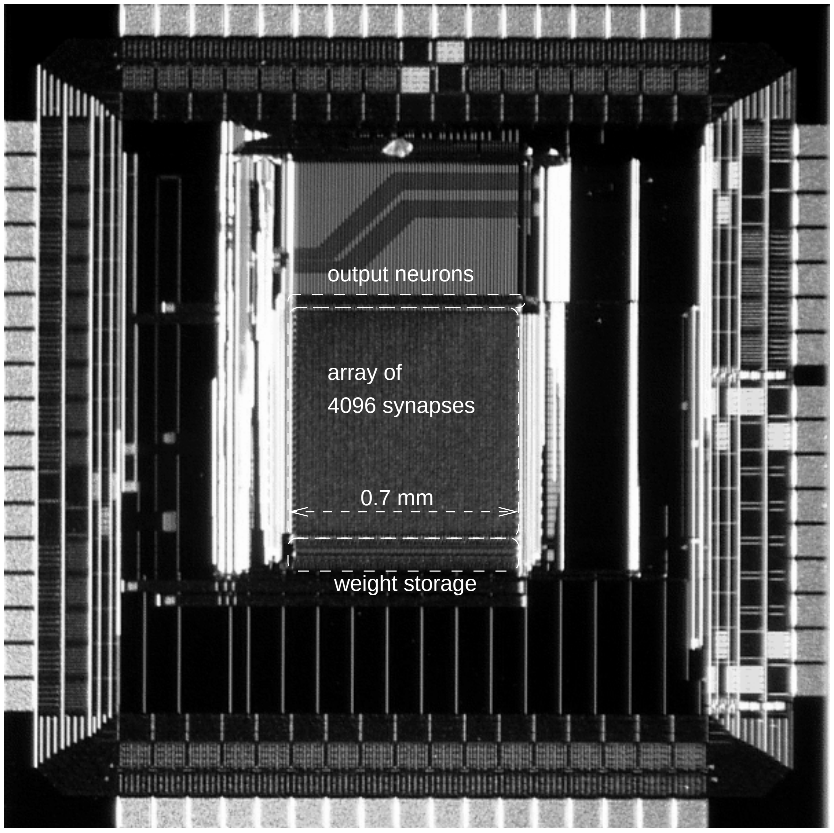


Fig. 4. Micro photograph of the neural network chip.

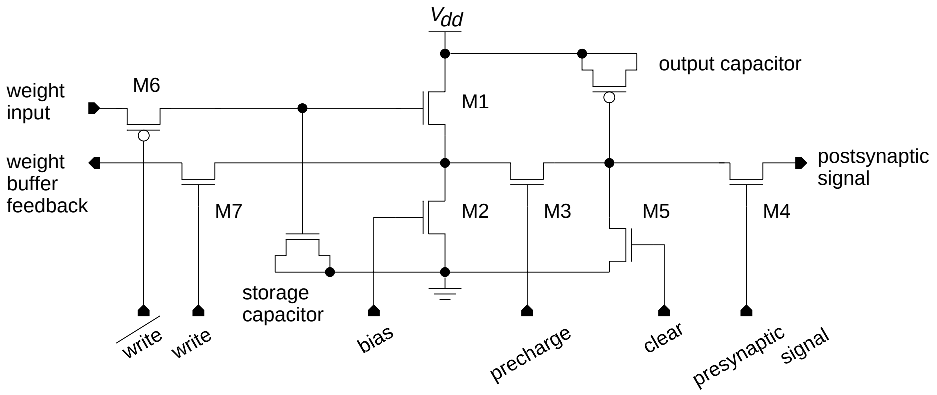


Fig. 5. Circuit diagram of the synapse.

to the weight voltage. M5 speeds up this process by fully charging the capacitor first. Since the output current of the source follower is much larger for a current flowing out of M1 instead of into M2 it discharges the capacitor faster than it is able to charge it. The total time to charge the output capacitor to the desired voltage decreases therefore by this combination of discharging and charging. In the evaluate phase M4 is enabled by the pre-synaptic signal of the input neuron connected to the synapse. Charge sharing between all the enabled synapses of every output neuron takes place on the post-synaptic lines. In Figure 6 a part of the layout drawing of the synapse array is shown. Most of the area is used up by the two capacitances. The values for the storage and output capacitances are about 60 and 100 fF respectively. The charge on the storage capacitors must be periodically refreshed due to the transistor leakage currents. In the training phase this happens automatically when the weights are updated, otherwise the refresh takes up about 2 % of the network capacity.

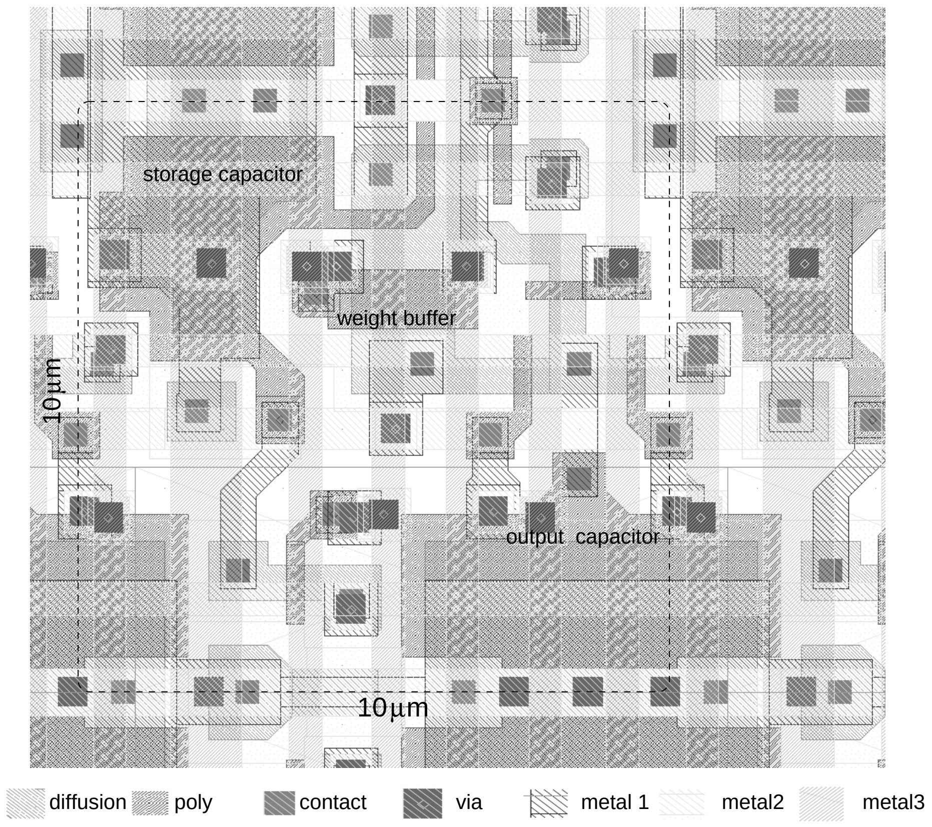


Fig. 6. Layout drawing of the synapse array showing one synapse.

Figure 7 shows the circuit diagram of the neuron circuit. It is based on a

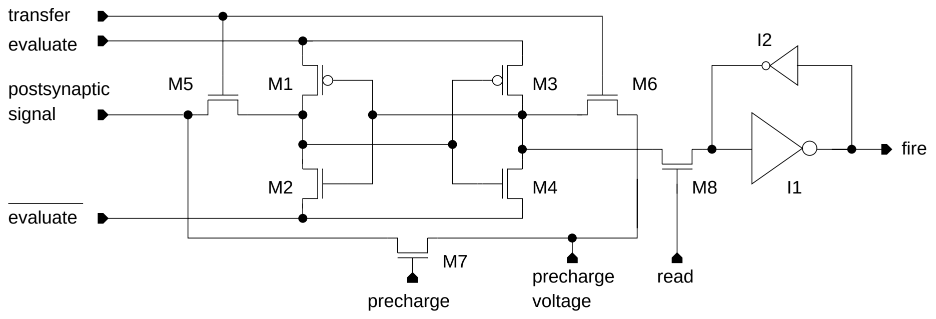


Fig. 7. Circuit diagram of the neuron.

sense amplifier built from the devices M1 to M4. In the precharge phase it is disabled (*evaluate* and $\overline{evaluate}$ are set to the precharge voltage). The activated *precharge* and *transfer* signals restore the post-synaptic input signal and the internal nodes of the sense amplifier to the precharge voltage. At the beginning of the evaluate phase the *precharge* signal is deactivated while *transfer* stays on. The potential on the post-synaptic input changes now by the activated synapses. Transistor M5 transfers it onto the gates of M3 and M4. The small differential voltage between the gates of M1/M2 and M3/M4 is amplified by disabling *transfer* and activating *evaluate*/$\overline{evaluate}$. At the same moment the synapses switch back to the precharge phase. The sense amplifier restores the signal in about 1 ns to the full supply voltage. With the *read* signal the result is stored in the output latch formed by the inverters I1 and I2. The output of I1 is fed back to the input neurons. The output neuron forms a master/slave flip-flop with the sense amplifier as the master and the output latch as the slave. This results in a discrete-time operation of the network. Together with the feedback the network acts as a kind of mixed-signal state machine. The neurons are the state flip-flops while the synapse array represents the logic that determines the next state. The simulated maximum clock frequency of the network is 100 MHz.

3 Implementation of the Genetic Training Algorithm

The time needed to load the 4096 weight values into the network is about 250 μs. A single test pattern comprised of 64 input bits can be applied in about 100 ns. This makes it feasible to use iterative algorithms needing high numbers of passes. The dependency between a weight value and a neuron output could be highly nonlinear, especially if more than one network cycle is used to implement a multi-layered recurrent network. Therefore a genetic algorithm seems to be well suited to train the network. The network has also built-in hardware support for perturbation based learning [6], an iterative algorithm that needs no knowledge about the transfer function of the network.

The implemented genetic algorithm represents one weight value by one gene. To avoid close hardware dependencies the weight value is stored in a normalized way using floating point numbers between -1 for the maximum inhibitory and +1 for the maximum excitatory synapse. These numbers are converted into the voltage values needed by the analog neural network while translating the genome into the weight matrix. The genes comprising one neuron are combined to a chromosome. Up to 64 chromosomes form the genome of one individual.

The artificial evolution is always started by creating a random population. After an individual has been loaded into the weight matrix the testpatterns are applied. The fitness is calculated by comparing the output of the network with the target values. For each correct bit the fitness is increased by one. This is repeated for the whole population. After sorting the population by the fitness two genetic operators are applied: crossover and mutation. The crossover strategy is depicted in Figure 8. It shows an example for a population of 16 individuals.

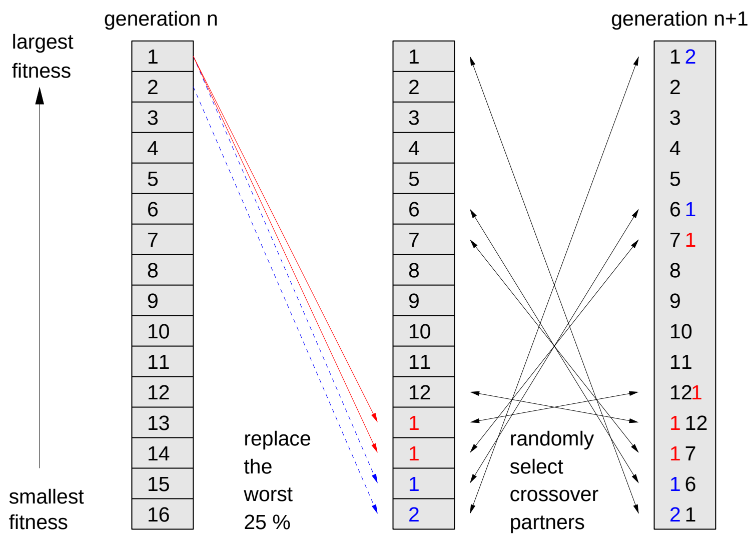


Fig. 8. Crossover pattern used in the genetic algorithm.

The worst 25% of the population are replaced in equal halves by the fittest individual (solid arrows) and the 12.5 % best ones (dashed arrows). 75% of the individuals are kept unchanged. As shown in Figure 8 the crossover partners are randomly chosen. The crossover itself is done in a way that for each pair of identical chromosomes (i.e. chromosomes coding the same output neuron) a crossover point is randomly selected. All the genes up to this point are exchanged between both chromosomes. After the crossover is done the mutation operator is applied on the new population. It alters every gene with equal probability. If a gene is subject to mutation, its old value is replaced by a randomly selected new one (again out of the range $[-1, 1]$).

4 Experimental Results

The network and the genetic training algorithm have been tested with the setup shown in Figure 9. The population is maintained on the host computer. The

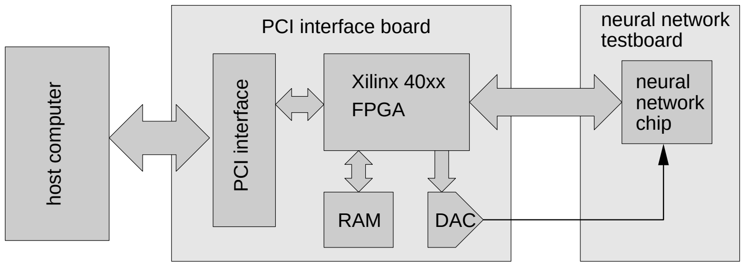
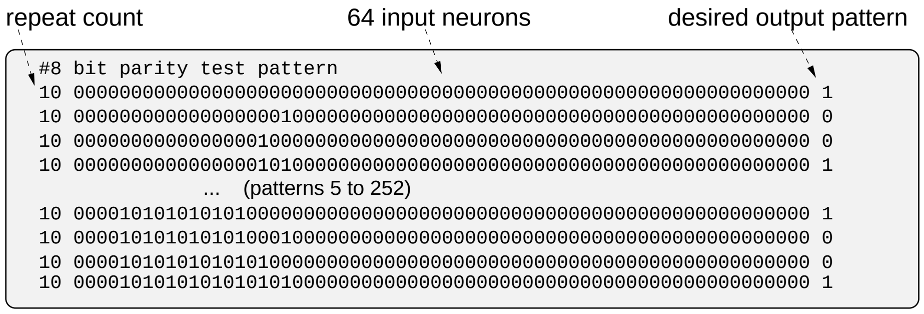


Fig. 9. Testbench used for the evaluation of the neural network.

data for each individual is sent via the FPGA to the neural network using a 16 bit digital to analog converter to generate the analog weight values from the gene data. The testpatterns and the target data are stored in the RAM on the testboard throughout the evolutionary process. They are applied to the individual after the neural network has stored its weights. The FPGA reads the results and calculates the fitness. After the last testpattern the final fitness value is read back by the host computer and the test of the next individual starts. To speed up this process the weight data for the next individual can be uploaded into the RAM while the testpatterns are applied to the current individual. Since the test board is not yet capable of the full speed of the network the number of individuals tested per second is limited to about 150 to 300, depending on the number of testpatterns used.

To test the capability of the genetic algorithm a training pattern was chosen that is especially hard to learn with traditional algorithms like back-propagation [7]: the calculation of parity. While easy to implement with exclusive-or gates, it can not be learned by a single layered neural network. Therefore it also shows the ability of the presented neural network to act as a two-layered network. Figure 10 shows the testpattern definition for an eight bit parity calculation. Since the input data is binary coded, the input neurons are configured for differential input (see Section 2.1). The number of network cycles is set to two and four output neurons are fed back into the network. This allows the genetic algorithm to use an internal layer with four neurons. For each testpattern one target bit is defined: the parity of the testpattern. Figures 11 and 12 show plots of the fitness versus the generation number for different parity experiments. At 6 bit


repeat count
64 input neurons
desired output pattern

```
#8 bit parity test pattern
10 0000000000000000000000000000000000000000000000000000000000000000 1
10 0000000000000000001000000000000000000000000000000000000000000000 0
10 0000000000000000100000000000000000000000000000000000000000000000 0
10 0000000000000000101000000000000000000000000000000000000000000000 1
             ...   (patterns 5 to 252)
10 0000101010101010000000000000000000000000000000000000000000000000 1
10 0000101010101010001000000000000000000000000000000000000000000000 0
10 0000101010101010100000000000000000000000000000000000000000000000 0
10 0000101010101010101000000000000000000000000000000000000000000000 1
```

Fig. 10. Testpattern definition for the 8 bit parity.

the network does not learn all the patterns any more. The random nature of the artificial evolution is clearly visible: the black curve approaches the same fitness as the gray one about 5000 generations earlier.

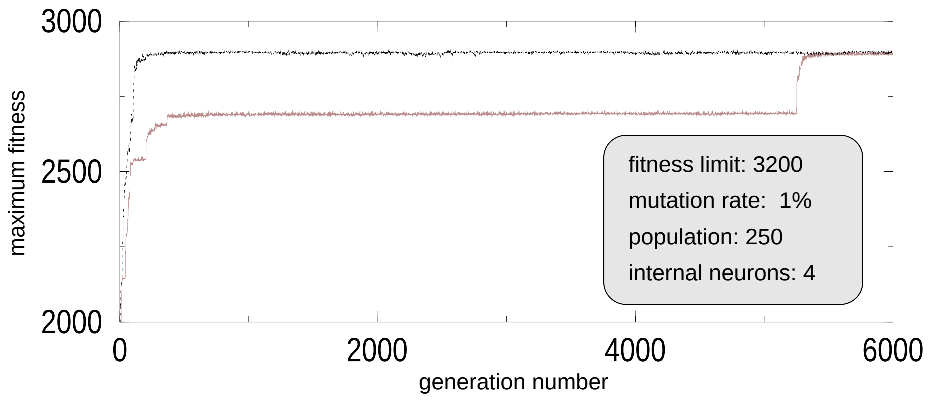


Fig. 11. Example fitness curves from 6 bit parity experiments.

5 Conclusion and Outlook

This paper presents a new architecture for analog neural networks that is optimized for iterative training algorithms, especially genetic algorithms. By combining digital information exchange with analog neuron operation it is well suited for large neural network chips. Especially, the very small synapse area makes network chips with more than a million synapses possible. The mapping of input data to the network and the effective number and size of the network layers is programmable. Therefore not only the weights, but also a significant part of the architecture can be evolved. The accuracy of the network is not limited by

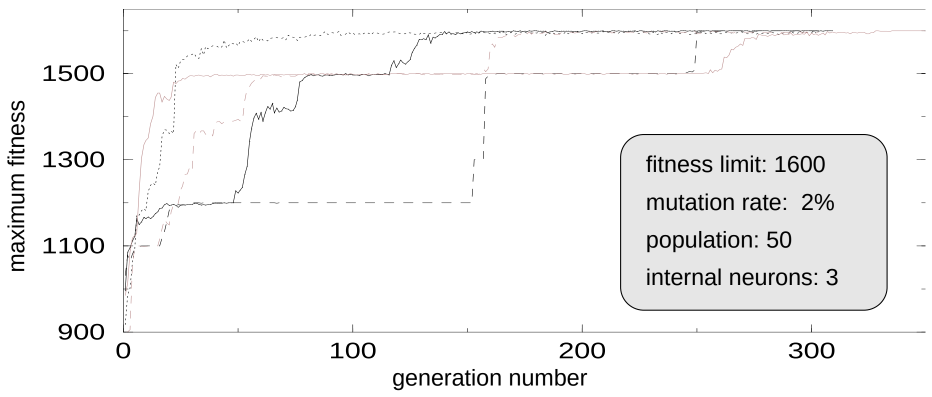


Fig. 12. Example fitness curves from 4 bit parity experiments.

the precision of a single analog synapse since arbitrary synapses can be combined. By implementing this task in the genetic algorithm, the network could automatically adapt its prediction performance to a specific data set.

The presented prototype successfully learned the parity calculation of multibit patterns. This shows that genetic algorithms are capable of training two-layered analog neural networks. At the time of this writing the test setup was limited in its analog precision. This makes it difficult to train binary patterns of 6 or more bits without errors. Also, the genetic algorithm used is a first approach to show the functionality of the system. These limitations will be hopefully overcome in the near future.

References

1. Shibata, T., Kosaka, H., Ishii, H. , Ohmi, T.: A Neuron-MOS Neural Network Using Self-Learning-Compatible Synapses Circuits. IEEE Journal of Solid-State Circuits, Vol. 30, No. 8, (August 1995) 913–922
2. Diorio, C., Hasler, P., Minch, B. , Mead, C.: A Single-Transistor Silicon Synapse. IEEE Transactions on Electron Devices, Vol. 43, No. 11, (November 1996) 1972–1980
3. Kramer, A.: Array-Based Analog Computation. IEEE Micro, (October 1996) 20–29
4. Schemmel, J., Loose, M., Meier, K.: A 66 × 66 pixels analog edge detection array with digital readout, Proceedings of the 25th European Solid-State Circuits Conference, Edition Frontinières, ISBN 2-86332-246-X, (1999) 298–301
5. Langeheine, J., Fölling, S., Meier, K., Schemmel, J.: Towards a Silicon Primordial Soup: A Fast Approach to Hardware Evolution with a VLSI Transistor Array. ICES 2000, Proceedings, Springer, ISBN 3-540-67338-5 (2000) 123-132
6. Montalvo, J., Gyurcsik R., Paulos J.,: An Analog VLSI Neural Network with On-Chip Perturbation Learning. IEEE Journal of Solid-State Circuits, Vol. 32, No. 4, (April 1997) 535–543
7. Hertz, J. Krogh, A., Palmer, R.: Introduction to the Theory of Neural Computation. Santa Fe Institute, ISBN 0-201-50395-6 (1991) 131