The CAM-Brain Machine (CBM) : Real Time Evolution and Update of a 75 Million Neuron FPGA-Based Artificial Brain

Hugo de GARIS[1], Michael KORKIN[2]

[1] Evolutionary Systems Dept., ATR - Human Information Processing Research
Laboratories, 2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
degaris@hip.atr.co.jp, http://www.hip.atr.co.jp/~degaris
Tel: +81-774-95-1079, Fax: +81-774-95-1008
[2] Genobyte, Inc., 1503 Spruce Street, Suite 3, Boulder CO 80302, USA
korkin@genobyte.com, http://www.genobyte.com
Tel: +1-303-545-6790, Fax: +1-303-449-1671

Abstract. This article introduces ATR's "CAM-Brain Machine" (CBM), an FPGA based piece of hardware which implements a genetic algorithm (GA) to evolve a cellular automata (CA) based neural network circuit module, of approximately 1,000 neurons, in about a second, i.e. a complete run of a GA, with 10,000s of circuit growths and performance evaluations. Up to 65,000 of these modules, each of which is evolved with a humanly specified function, can be downloaded into a large RAM space, and interconnected according to humanly specified artificial brain architectures. This RAM, containing an artificial brain with up to 75 million neurons, is then updated by the CBM at a rate of 130 billion CA cells per second. Such speeds should enable real time control of robots and hopefully the birth of a new research field that we call "brain building". The first such artificial brain, to be built by ATR starting in 1999, will be used to control the behaviors of a life sized robot kitten called "Robokoneko".

1 Introduction

This article introduces ATR's "CAM-Brain Machine" (CBM) [11], a Xilinx XC6264 FPGA [19] based piece of hardware that is used to evolve 3D cellular automata based neural network [15] circuit modules at electronic speeds, that is in about a second per module. 65,000 of these modules can then be assembled into a large RAM space according to humanly specified artificial brain architectures. This RAM is updated by the CBM fast enough (130 billion CA cell updates/sec) for real time control of robots. ATR's CBM should be built and delivered by the third quarter of 1999.

The CBM is the essential tool in ATR's "Artificial Brain (CAM-Brain) Project" [2, 4], which at the time of writing (Summer 1999), has been running for 6.5 years. Although the focus of this article is on the functional principles and design of the CBM, a certain background needs to be provided so that the motivation for its construction is understood.

The basic (and rather ambitious) aim of the CAM-Brain Project as first stated in 1993 was to build an artificial brain containing a billion artificial neurons by the year 2001. The actual figure in 1999 will be maximum 75 million, but the billion figure is still reachable if we really want. The ATR Brain Builder team is hoping that the CBM will revolutionize the field of neural networks (by creating neural systems with tens of millions of artificial neurons, rather than just the conventional tens to hundreds), and will create a new research field called "Brain Building". The CBM will make practical the creation of artificial brains, which are defined to be assemblages of tens of thousands (and higher magnitudes) of evolved neural net modules into humanly defined artificial brain architectures. An artificial brain will consist of a large RAM memory space, into which individual CA modules are downloaded once they have been evolved. The CA cells in this RAM will be updated by the CBM fast enough for real time control of a robot kitten "Robokoneko" (Japanese for "robot kitten").

Since the neural net model used to fit into state-of-the-art evolvable electronics has to be simple, the signaling states of the neural net were chosen to be 1 bit binary. We label this model "CoDi-1Bit" [8] (CoDi = Collect & Distribute). This article will summarize the principles of this 1 bit neural signaling model, since the CBM is an electronic implementation of it. We realize that limiting ourselves to only 1 bit per neural signal (to fit into the Xilinx XC6264 chips), is rather severe (although nature uses a 1 bit signal scheme with its evoked potentials, i.e. the spikes in the axons), so it is possible that future versions of the CBM may use multibit neural signaling to obtain higher "evolvability" of neural module functionality.

The remainder of this article is structured as follows. Section 2 gives an explanation of the "CoDi-1Bit" neural net model that is implemented by the CAM-Brain Machine (CBM). Section 3 discusses briefly the representation that our team has chosen to interpret the 1 bit signals which are input to and output from the CoDi modules (we call this representation "SIIC" = Spike Interval Information Coding). This representation is important because the CBM measures the "fitness" (i.e. the performance measure of the evolving circuit) using analog output values obtained by convolving the binary outputs of the module with a digitized convolution function. Section 4 shows how analog time-dependent signals can be converted into spike trains (bit strings of 0s and 1s) to be input into CoDi modules using the so-called "HSA" (Hough Spiker Algorithm). The SIIC (spiketrain to analog signal conversion) and the HSA (analog signal to spiketrain conversion) allow users (evolutionary engineers) to think entirely in analog terms when specifying input signals and target (desired) output signals, which is much easier than thinking in terms of spike intervals (the number of 0s between the 1s). This analog thinking for evolutionary engineers simplifies the evolution of modules, and overcomes the limitation to some extent of the 1 bit binary signaling of the CoDi modules (and hence the CBM). Section 5, the heart of this article, provides a detailed summary of CBM design and functionality, using the ideas already discussed in the earlier sections. Since an artificial brain without a body (such as a robot) seems rather pointless, section 6 introduces early work on the behavioral repertoire and mechanical design of the kitten robot "Robokoneko" that our artificial brain will control. Section 7 presents a (software simulated) sample of what evolved CoDi modules will be able to do, once the CBM is complete and delivered. Our Brain Builder team will then be evolving thousands of such modules. Section 8 discusses ideas for interesting future modules and multi-module systems to be evolved. Section 9 talks about some related work, and Section 10 concludes.

2 The CoDi-1Bit Neural Network Model

The CBM implements the so called "CoDi" (i.e. Collect and Distribute) [8] cellular automata based neural network model. It is a simplified form of an earlier model developed at ATR (Kyoto, Japan) in the summer of 1996, with two goals in mind. One was to make neural network functioning much simpler and more compact compared to the original ATR model, so as to achieve considerably faster evolution runs on the CAM-8 (Cellular Automata Machine), a dedicated hardware tool developed at Massachusetts Institute of Technology in 1989.

In order to evolve one neural module, a population of 30-100 modules is run through a genetic algorithm [9] for 200-600 generations, resulting in up to 60,000 different module evaluations. Each module evaluation consists of - firstly, growing a new set of axonic and dendritic trees, guided by the module's chromosome (which provide the growth instructions for the trees). These trees interconnect several hundred neurons in the 3D cellular automata space of 13,824 cells (24×24×24). Evaluation is continued by sending spiketrains to the module through its efferent axons (external connections) to evaluate its performance (fitness) by looking at the outgoing spiketrains. This typically requires up to 1000 update cycles for all the cells in the module.

On the MIT CAM-8 machine, it takes up to 69 minutes to go through 829 billion cell updates needed to evolve a single neural module, as described above. A simple "insect-like" artificial brain has hundreds of thousands of neurons arranged into ten thousand modules. It would take 500 days (running 24 hours a day) to finish the computations.

Another limitation was apparent in the full brain simulation mode, involving thousands of modules interconnected together. For a 10,000-module brain, the CAM-8 is capable of updating every module at the rate of one update cycle 1.4 times a second. However, for real time control of a robotic device, an update rate of 50-100 cycles per module, 10-20 times a second is needed. So, the second goal was to have a model which would be portable into electronic hardware to eventually design a machine capable of accelerating both brain evolution and brain simulation by a factor of 500 compared to CAM-8.

The CoDi model operates as a 3D cellular automata (CA). Each cell is a cube which has six neighbor cells, one for each of its faces. By loading a different phenotype code into a cell, it can be reconfigured to function as a neuron, an axon, or a dendrite. A neuron is a brain cell. An axon is the branching of a neuron which carries a neural signal away from the neuron to other neurons. A dendrite is the branching of the neuron which carries a neural signal towards the neuron from other neurons. Neurons are configurable on a coarser grid, namely one per block of 2×2×3 CA cells. Cells are interconnected with bidirectional 1-bit buses and assembled into 3D modules of 13,824 cells (24×24×24).

Modules are further interconnected with 188 1-bit connections to function together as an artificial brain. Each module can receive signals from up to 188 other modules and send its output signals to up to 64,640 modules. These intermodular connections are virtual and implemented as a cross-reference list in a module interconnection memory (see below).

In a neuron cell, five (of its six) connections are dendritic inputs, and one is an axonic output. A 4-bit accumulator sums incoming signals and fires an output signal when a threshold is exceeded. Each of the inputs can perform an inhibitory or an excitatory function (depending on the neuron's chromosome) and either adds to or subtracts from the accumulator. The neuron cell's output can be oriented in 6 different ways in the 3D space. A dendrite cell also has five inputs and one output, to collect signals from other cells. The incoming signals are passed to the output with an 5-bit XOR function. An axon cell is the opposite of a dendrite. It has 1 input and 5 outputs, and distributes signals to its neighbors. The "Collect and Distribute" mechanism of this neural model is reflected in its name "CoDi". Blank cells perform no function in an evolved neural network. They are used to grow new sets of dendritic and axonic trees during the evolution mode.

Before the growth begins, the module space consists of blank cells. Each cell is seeded with a 6-bit chromosome. The chromosome will guide the local direction of the dendritic and axonic tree growth. Six bits serve as a mask to encode different growth instructions, such as grow straight, turn left, split into three branches, block growth, T- split up and down etc. Before the growth phase starts, some cells are seeded as neurons under genetic control. As the growth starts, each neuron continuously sends growth signals to the surrounding blank cells, alternating between "grow dendrite" (sent in the direction of future dendritic inputs) and "grow axon" (sent towards the future axonic output). A blank cell which receives a growth signal becomes a dendrite cell, or an axon cell, and further propagates the growth signal, being continuously sent by the root neuron, to other blank cells. The direction of the propagation is guided by the 6-bit growth instruction, described above. This mechanism grows a complex 3D system of branching dendritic and axonic trees, with each tree having one neuron cell associated with it. The trees can conduct signals between the neurons to perform complex spatio-temporal functions. The end-product of the growth phase is a phenotype bitstring which encodes the type and spatial orientation of each cell.

Thus there are two main phases - neural net growth and neural net signaling. In the CoDi-1Bit model, the signal states contain only 1 bit. With an 8 bit signal for example (as was the case in the old CAM-Brain Project model) one simply looks at the signal state to see the signal value. With 1 bit signaling, one needs to choose an interpretation of the signals, e.g. frequency based (count the number of spikes (1s) in a given time), or interpret the spacing between the spikes as containing information etc. These interpretation issues will be taken up in the next section.

3 The Spike Interval Information Coding Representation, "SIIC"

3.1 Choosing a Representation for the CoDi-1Bit Signaling

The constraints imposed by state-of-the-art programmable (evolvable) FPGAs in 1998 were such that the CA based model (the CoDi model) had to be very simple in order to be implementable within those constraints. Consequently, the signaling states in the model were made to contain only 1 bit of information (as happens in nature's "binary" spike trains). The problem then arose as to interpretation. How were we to assign meaning to the binary pulse streams (i.e. the clocked sequences of 0s and 1s which are a neural net module's inputs and outputs? We tried various ideas such as a frequency based interpretation, i.e. count the number of pulses (i.e. 1s) in a given time window (of N clock cycles). But this was thought to be too slow. In an artificial brain with tens of thousands of modules which may be vertically nested to a depth of 20 or more (where the outputs of a module in layer n get fed into a module in layer $n + 1$, where n may be as large as 20 or 30) then the cumulative delays may end up in a total response time of the robot kitten being too slow (e.g. if you wave your finger in front of its eye, it might react many seconds later). We wanted a representation that would deliver an integer or real valued number at each clock tick, the ultimate in speed. The first such representation we looked at we called "unary". If N neurons on an output surface are firing at a given clock tick, then the firing pattern represented the integer N, independently of where the outputs were coming from. We found this representation to be too stochastic, too jerky. Ultimately we chose a representation which convolves the binary pulse string with the convolution function shown in Fig. 1. We call this representation "SIIC" (Spike Interval Information Coding) which was inspired by [14].

This representation delivers a real valued output at each clock tick, thus converting a binary pulse string into an analog time dependent signal. Our team has already published several papers on the results of this convolution representation work [12]. Fig. 2 shows the result of deconvolving an arbitrary analog curve (that is, converting an analog signal into a spike train (binary string) as explained in section 4), and then convolving it back (i.e. converting a spike train into an analog signal) to the original analog curve. The smooth curve is the original curve, and the spikey curve is the result of the two conversions. The percentage errors obtained between the original curve and the result of the two conversions were only about 2%, so we thought these two conversions were very useful. Of course, it is one thing to have accurate conversions from analog signals to spike trains and vice versa. It is another that a CoDi-1Bit neural net module can evolve a spike train that when convolved can produce a desired analog output. Fig. 3 shows just such an example (of a target 3 period sine curve) which evolved quite successfully, showing that the basic idea is sound. (The solid curve is the target curve, and the dashed curve is the evolved and convolved result. The actual spikes (i.e. the 1s in the binary string output from the CoDi module) are shown beneath the curves). Fig. 4 shows two outputs of a "halver" circuit which was evolved to take a constant analog input (e.g. 600 or 400) and to output half its value (300 or 200). This case is a good example of how an evolutionary engineer can think entirely in analog terms when evolving modules. The analog input is automatically converted to a spike train, which enters the neural net module, and the spike train output of the module get automatically converted to an analog signal whose values are compared with a target curve to evaluate the fitness (performance) of the evolving circuit. Further examples of evolved modules (although using only binary I/O), are to be found in section 7.

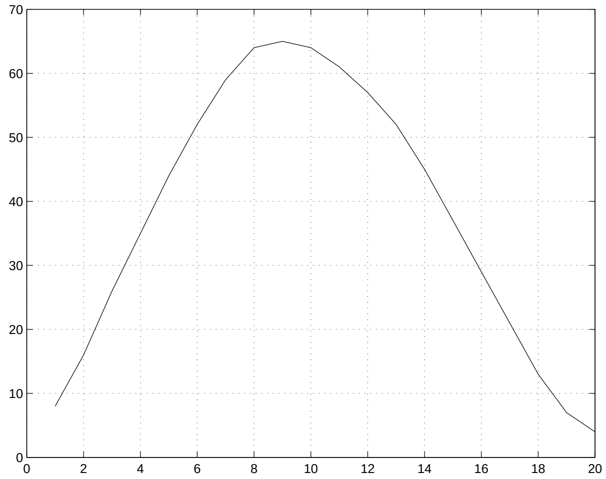

Fig. 1. The convolution function used in the "SIIC" representation

3.2 The SIIC Convolution Algorithm

The convolution algorithm we use takes the output spiketrain (a bit string of 0s and 1s), and runs the pulses (the 1s) by the convolution function shown in the simplified example below. The output at any given time t is defined as the sum of those samples of the convolution filter that have a 1 in the corresponding spiketrain positions. The example below should clarify what is meant by this.

Simplified Example Convolve the spiketrain 1101001 (where the left most bit is the earliest, the right most bit, the latest) using the convolution filter values

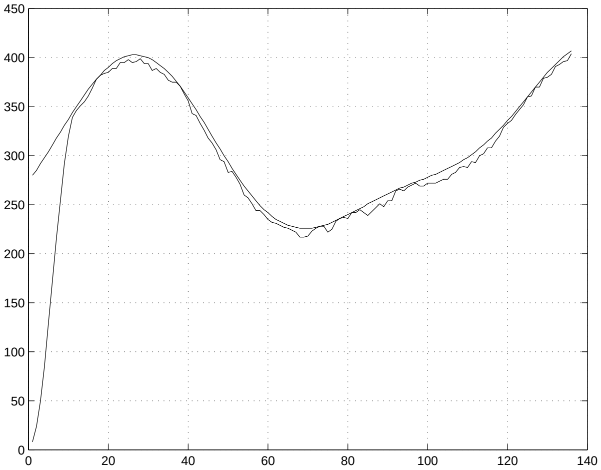

Fig. 2. An analog (smooth) curve and its deconvolved/convolved approximation (jerky) curve.

{ 1 4 9 5 -2 }. The spiketrain in this diagram moves from left to right across the convolution filter. Alternatively, one can view the convolution filter (window) moving across the spiketrain. The number to the right of the colon shows the value of the convolution sum at each time *t*.

```
time-shifted spike train :  1 0 0 1 0 1 1 ---> (moves left to right)
convolution filter :   1  4  9  5 -2

  1  0  0  1  0  1  1
                       0  0  0  0  0  : 0    t = -1

     1  0  0  1  0  1  1
                       1  0  0  0  0  : 1    t = 0

        1  0  0  1  0  1  1
                       1  4  0  0  0  : 5    t = 1

           1  0  0  1  0  1  1
                       0  4  9  0  0  : 13   t = 2

              1  0  0  1  0  1  1
```

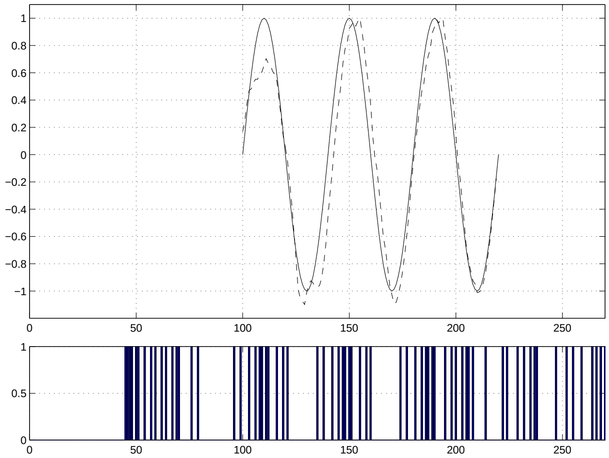

Fig. 3. A 3 period sine curve resulting from convolution of an evolved CoDi-1Bit. The lower figure shows the actual spikes that generated the waveform.

```
           1   0   9   5   0   : 15     t = 3

   1   0   0   1   0   1   1
           0   4   0   5  -2   : 7      t = 4

       1   0   0   1   0   1
           0   0   9   0  -2   : 7      t = 5

           1   0   0   1   0
           1   0   0   5   0   : 6      t = 6

               1   0   0   1
           0   4   0   0  -2   : 2      t = 7

                   1   0   0
           0   0   9   0   0   : 9      t = 8

                       1   0
           0   0   0   5   0   : 5      t = 9
```

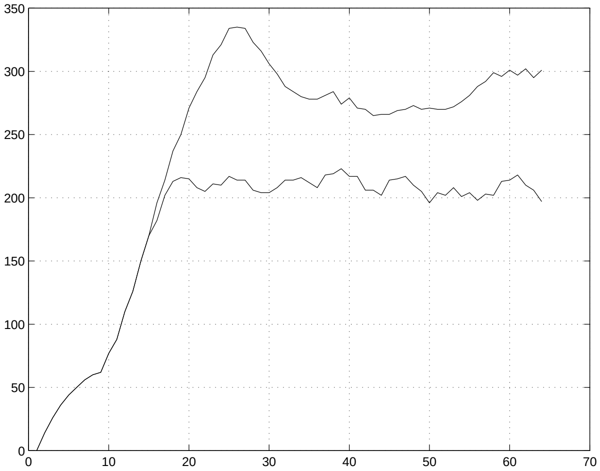

Fig. 4. Outputs of a halver circuit (with inputs 600 and 400) using fully analog I/O.

```
                1
0   0   0   0 -2   : -2    t = 10
```

Hence, the time-dependent output of the convolution filter takes the values (0, 1, 5, 13, 15, 7, 7, 6, 2, 9, 5, -2). This is a time varying analog signal, which is the desired result.

4 The "Hough Spiker Algorithm" (HSA) for Deconvolution

Section 3 above explained the use of the SIIC (Spike Interval Information Coding) Representation which provides an efficient transformation of a spike train (string of bits) into a (clocked) time varying "analog" signal. We need this interpretation in order to interpret the spike train output from the CoDi modules to evaluate their fitness values (by comparing the actual converted analog output waveforms with user specified target waveforms). However, we also need the inverse process, namely, an algorithm which takes as input, a clocked (digitized, binary numbered) time varying "analog" signal, and outputs a spike train. This conversion is needed as an interface between the motors/sensors of the robot bodies (e.g. a kitten robot) that the artificial brain controls, and the brain's CoDi modules. However, it is also very useful to users, i.e. evolutionary engineers to be able to think entirely in terms of analog signals (at both the inputs and outputs) rather than in abstract, visually unintelligible spiketrains. This will make their task of evolving many CoDi modules much easier. We therefore present next an algorithm which is the opposite of the SIIC, namely one which takes as input, a time varying analog signal, and outputs a spike train, which if later is convolved with the SIIC convolution filter, should result in the original analog signal.

A brief description of the algorithm used to generate a spiketrain from a time varying analog signal is now presented. It is called the "Hough Spiker Algorithm" (HSA) and can be viewed as the inverse of the convolution algorithm described above in section 3.

To give an intuitive feel for this deconvolution algorithm, consider a spiketrain consisting of a single pulse (all 0s with one 1). When this pulse passes through the convolution function window, it adds each value of the convolution function to the output in turn.

A single pulse: (100000... → $t = +\infty$) will be convolved with the convolution function expressed as a function of time. At $t = 0$ its value will be the first value of the convolution filter, at $t = 1$ its value will be the second value of the convolution filter, etc. Just as a particular spiketrain is a series of spikes with time delays between them, so too the convolved spiketrain will be the sum of the convolution filters, with (possibly) time delays between them. At each clock tick when there is a spike, add the convolution filter to the output. If there is no spike, just shift the time offset and repeat.

The same example.

```
spike train        1 1 0 1 0 0 1
convolution filter 1 4 9 5 -2


    t -> 0  1  2  3  4  5  6  7  8  9 10
out:
 1       1  4  9  5 -2
 1          1  4  9  5 -2
 0             0  0  0  0  0
 1                1  4  9  5 -2
 0                   0  0  0  0  0
 0                      0  0  0  0  0
 1                         1  4  9  5 -2
         -------------------------------
         1  5 13 15  7  7  6  2  9  5 -2
```

In the HSA deconvolution algorithm, we take advantage of this summation, and in effect do the reverse, a kind of progressive subtraction of the convolution function. If at a given clock tick, the values of the convolution function are less than the analog values at the corresponding positions, then subtract the convolution function values from the analog values. The justification for this is that for the analog values to be greater than the convolution values, implies that to generate the analog signal values at that clock tick, the CoDi module must have fired at that moment, and this firing contributed the set of convolution values to the analog output. Once one has determined that at that clock tick, there should be a spike, one subtracts the convolution function's values, so that a similar process can be undertaken at the next clock tick. For example, to deconvolve the convolved output (using the same value of the convolution function as in the simple example of the previous section.

```
         1  5 13 15  7  7  6  2  9  5 -2
compare: 1  4  9  5 -2   conv.vals<analog sig vals, so spike: 1
         0  1  4 10  9  7  6  2  9  5 -2 subtract (time++)
compare:    1  4  9  5 -2                less, so spike: 11
         0  0  0  1  4  9  6  2  9  5 -2 subtract (time++)
compare:       1  4  9  5 -2             not less, so no spike: 110
         0  0  0  1  4  9  6  2  9  5 -2 (time++)
compare:          1  4  9  5 -2          less, so spike: 1101
         0  0  0  0  0  0  1  4  9  5 -2 subtract (time++)
compare:             1  4  9  5 -2       not less: 11010
         0  0  0  0  0  0  1  4  9  5 -2 (time++)
compare:                1  4  9  5 -2    not less: 110100
         0  0  0  0  0  0  1  4  9  5 -2 (time++)
compare:                   1  4  9  5 -2 less, so spike: 1101001
         0  0  0  0  0  0  0  0  0  0  0 subtract (time++)
```

It is assumed that spiking will irreversibly raise the value of the convolved output. If the convolution filter value at a given clock tick is less than that of the target waveform, spiking will bring the two values closer together. If the waveform value is still too low after a spike has occurred, a near future spike will bring the two closer together.

Fig. 5 shows an example of an HSA spiketrain output. It is the spike train corresponding to Fig. 2 in fact. The original input analog signal is the solid line in Fig. 2. The spiketrain resulting from each analog input is sent into the SIIC convolver (shown in Fig. 1). The resulting analog output (the jerky curve) should be very close to the original solid line as Fig. 2 shows it to be. The HSA seems to work well when the values of the waveforms are large and do not take values close to zero, and do not change too quickly relative to the time width of the convolution filter window. It may be possible to simply add a constant value to incoming analog signals before spiking them and to ensure that the analog signal does not change too rapidly.

```
( time ---> )
111100010001101111110100010111110110100010101110100100010011010100
100010101010100101001010110001101010011001101011010101011101110101101
```

Fig.5. The spiketrain output of Fig. 2, as generated by the Hough Spiker Algorithm (HSA).

Note however, that the HSA deconvolution algorithm was only discovered fairly recently, so the neural net module evolution that is discussed in section 7 below, does not use it. The I/Os to these modules as specified by the evolutionary engineer were in binary, not analog.

5 The CAM-Brain Machine (CBM)

5.1 CBM Overview

The CAM-Brain Machine (CAM stands for Cellular Automata Machine) is a research tool for the creation of artificial brains. An original set of ideas for the CAM-Brain project was developed by Dr. Hugo de Garis at the Evolutionary Systems Department of ATR HIP (Kyoto, Japan), and is currently being implemented as a dedicated research tool by Genobyte, Inc. (Boulder, Colorado). Genobyte is licensed by ATR International and Japan's Key Technologies Center to manufacture and sell CBMs to third parties.

An artificial brain, supported by the CBM, consists of up to 64,640 neural modules, each module populated with up to 1,152 neurons, a total of 74.5 million neurons. Within each neural module, neurons are densely interconnected with branching dendritic and axonic trees in a three-dimensional space, forming an arbitrarily complex interconnection topology. A neural module can receive afferent axons from up to 188 other modules of the brain, with each axon being capable of multiple branching in three dimensions, forming hundreds of connections with dendritic branches inside the module. Each module sends efferent axon branches to up to 64,640 other modules.

A critical part of the CBM approach is that the detailed dendritic/axonal tree structure of the neural modules is not "manually designed" or "engineered" to perform a specific brain function, but rather evolved directly in hardware, using genetic algorithms, in the spirit of the growing research field of evolvable hardware [16, 10, 12, 17].

Genetic algorithms operate on a population of chromosomes, which represent neural networks of different topologies and functionalities. Better performers for a particular function are selected and further reproduced using chromosome recombination and mutation. After hundreds of generations, this approach produces very complex neural networks with a desired functionality. The evolutionary approach can create a complex functionality without any a priori knowledge about how to achieve it, as long as the desired input/output function is known.

5.2 CBM Architecture

We begin the description of the CBM with a brief overview, followed by several paragraphs giving a somewhat greater level of detail. These paragraphs also attempt to justify to some extent the architectural decisions we made. Note that we have compromised here between a need for corporate secrecy (Genobyte, Michael Korkin's company [7], has a licensing agreement with ATR to build and sell CBMs, hopefully free from imitators for several years) and academic openness, so the description below is somewhat lacking in critical details.

In the CBM we have implemented what is called "function-level" evolvable hardware, as opposed to "gate-level" evolvable hardware, which directly operates on a sea of Boolean gates. Our functions take the form of cellular automata cells, which are manually designed and configured in Xilinx XC6264 FPGA chips. (Note that Xilinx removed the XC6200 family of chips from the market. We managed to salvage the few remaining XC6264 chips from Xilinx, enough to build approximately 8 CAM-Brain Machines (CBM) in the next few years.) Each of these cellular automata cells contains a 6-bit register and some additional logic, which allows it to exchange signals with its neighboring cells. The contents of the register is the subject of evolution. So, instead of using FPGA configuration memory space to instantiate different circuits, our design utilizes our own "configuration" space made up of multiple 6-bit registers in CA cells, which are pre-loaded into the FPGAs. In fact, the CBM design uses three different cell functions for three different phases of operation (i.e. growth, signaling, and genetic), so we reconfigure the entire FPGA chips multiple times in the process of cycling through the CBM phases. A high reconfiguration speed and direct access to the user-level registers in the XC6264 chips allow us to achieve high overall throughput.

The following provides further details of our CBM implementation.

The CBM architecture is designed around the architectural features of Xilinx's XC6264 FPGA chips. These SRAM-based FPGAs allow rapid reconfiguration logic at the rate of 60 Mbytes/s. A full CBM array of 72 FPGAs forms a cellular automata cubic space of $24\times 24\times 24$ cells. Each FPGA holds a subspace of 8x6x4 CA cells, a total of 192. These FPGAs are further interconnected to provide a continuous, uninterrupted space. Each FPGA has 208 bidirectional connections with its neighboring FPGAs in a three-dimensional logical space. Each FPGA is located on a separate PCB, which also carries a tightly coupled 16Mbyte DRAM SIMM and control logic CPLD. Interconnections are made via a large backplane panel carrying all 72 FPGA module PCBs. The cellular space is wrapped around all three axes of the CA cube, forming a toroidal cube. All 72 FPGA functions are accomplished in parallel for the complete array under central control, while each FPGA has its own data to work with in its own 16 Mbytes memory space. Thus, the CBM architecture is of the SIMD (single instruction multiple data) type.

The FPGA array is time shared between multiple neural modules during an evolution run, or during brain run mode, by rapid instantiation of each module for a period of 12 microseconds, during which time the CA space is clocked 96 times at 9.47 MHz. At the end of this period, the status of the cells is saved in the 16 Mbytes of DRAM, while the next module configuration is uploaded into the CA space from the DRAM. The resultant cellular update rate in the CBM's array of 72 FPGAs is on the order of 114 billion cells/second.

Each CA cell contains function logic and control registers which determine its operation. A cell typically occupies a rectangular FPGA subspace of 64 fine-grain function units, and a control register typically contains 7 to 35 bits. Cell registers can be written or read through a 32-bit FPGA data interface in the same manner as the FPGA configuration space is accessed, which is a distinctive feature of the XC6264. Cells are interconnected inside the FPGA with their neighboring cells using internal routing resources. Those cells which form the external surface of the CA subspace connect to cells inside the neighboring FPGAs in the array, a total of 208 connections. All inter-chip connections in the CBM have an open-drain configuration with external pull-ups to protect them from potential damage resulting from certain configuration patterns in the connected CA cells belonging to different FPGAs.

Each CA cell's internal control registers are implemented as dual pipeline registers. The first stage is used to upload new bitstrings into all 192 cells in an FPGA through the 32-bit data interface, while the second stage holds the current cell configuration of the functioning cellular automata space. The first stage register's contents can be loaded into the second stage register for all cells in parallel using a global signal. This accomplishes complete CA space reconfiguration in a matter of nanoseconds as well as simultaneous execution of the CA states with a background reconfiguration for the next neural module instantiation. Thus, the hardware core of the CBM is continuously utilized without any considerable idle time.

For each of the three operational phases of the CBM, during every generation of a genetic algorithm (growth phase, signal phase, genetic phase), the full array of the 72 FPGAs is rapidly reconfigured with a completely different set of CA cell functions. In the growth phase, the CA cells perform a network growth algorithm, while their control registers are uploaded with the neural module's chromosomes. The result of the growth phase is the neural module phenotype to be saved at the end of the growth phase. The phenotype is further used to configure the signal phase cells during the signal phase. In the genetic phase, the function of the cells is to create an offspring chromosome from two parent chromosomes using crossover and mutation masks.

Reconfiguration is accomplished by loading the configuration data from the DRAM SIMM via the 32-bit FPGA data interface. Complete FPGA reconfiguration takes less than one millisecond. All 72 FPGAs are reconfigured in parallel. An alternative to reconfiguring an FPGA for each operational phase would have been implementing more complex CA cells capable of functioning in all phases. This would have resulted in a significantly smaller cellular space fittable into the FPGA. The rapid reconfiguration capability of the XC6264 provided a solution which allows a large number of cells with a high functional diversity, in exchange for a small additional operation time. This additional time is less than 3 seconds per 1000 generations of evolution.

In addition to the main FPGA array, the CBM utilizes four XC6264 FPGAs for spiketrain buffer logic and for a fitness evaluation unit. The fitness evaluation unit holds eight separate 24-tap convolution filters for output / target spiketrain deviation computation during the evolution runs.

The CBM consists of the following six major blocks:

1. Cellular Automata Module
2. Genotype/Phenotype Memory
3. Fitness Evaluation Unit

4. Genetic Algorithm Unit
5. Module Interconnection Memory
6. External Interface

Each of these blocks is discussed in detail below, followed by some further architectural points in section 5.3. A summary of CBM capacities can be found in table 5.3.

Cellular Automata Module The cellular automata module is the hardware core of the CBM. It is intended to accelerate the speed of brain evolution through a highly parallel execution of cellular state updates. The CA module consists of an array of identical hardware logic circuits or cells arranged as a 3D structure of 24×24×24 cells (a total of 13,824 cells). Cells forming the top layer of the module are recurrently connected with the cells in the bottom layer. A similar recurrent connection is made between the cells on the north and south, east and west vertical surfaces. Thus a fully recurrent toroidal cube is formed. This feature allows a higher axonic and dendritic growth capacity by effectively doubling each of the three dimensions of the cellular space.

The CBM hardware core is time-shared between multiple modules forming a brain during brain simulation. Only one module is instantiated at a time. The FPGA firmware design is a dual-buffered structure, which allows simultaneous configuration of the next module while the current module is being run (i.e. signals are propagated through the dendrites and axons between neurons). Thus, the FPGA core is run continuously without any idle time between modules for reconfiguration.

The surfaces of the cube have external connections to provide signal input from other modules. Each surface has a matrix of 64 signals, which is repeated on the opposite surface due to wrap around connections. Thus, a total of 192 different connections is available. Four connections, i.e. one on each of the surfaces, and one at one of the 8 corner cells of the cube, are used as output points. Due to wrap around, any corner cell has 3 wrap-around faces, so it is within two cells maximum of any other corner cell, including the opposite corner, and at the same time equidistant from the three other outputs. The fourth output is equivalent to the center of the cube, so the set of all 4 outputs looks nice and symmetric.

The CA module is implemented with Xilinx FPGA devices XC6264. These devices are fully and partially reconfigurable, feature a new co-processor architecture with data and address bus access in addition to user inputs and outputs, and allow the reading and writing of any of the internal flip-flops through the data bus. An XC6264 FPGA contains 16,384 logic function cells [19], each cell featuring a flip-flop and Boolean logic capacity, capable of toggling at a 220 MHz rate. Logic cells are interconnected with neighbors at several hierarchical levels, providing identical propagation delay for any length of connection. This feature is very well suited for a 3D CA space configuration. Additionally, clock routing is optimized for equal propagation time, and power distribution is implemented in a redundant manner.

To implement the CA module, a 3D block of identical logic cells is configured inside each XC6264 device, with CoDi specified 1-bit signal buses interconnecting the cells. Given the FPGA internal routing capabilities and the logic capacity needed to implement each cell, the optimal arrangement for a XC6264 is 4×6×8 (192 cells). This elementary block of cells requires 208 external connections to form a larger 3D block by interconnecting with six neighbor FPGAs on the south, north, east, west, top, and bottom sides in a virtual 3D space. A total of 72 FPGAs, arranged as a 6×4×3 array are used to implement a 24×24×24 cellular cube.

The CBM implements interconnections between 72 FPGAs, each placed on a small individual printed circuit board, in the form of one large backplane board, carrying all 72 FPGA daughter boards.

The CBM clock rate for cellular update is selected between 8.25 MHz, 9.42 MHz, and 11 MHz. At this rate all 13,824 cells are updated simultaneously, which results in the update rate of 114 to 130 billion cells/s. This rate exceeds the CAM-8 update rate by a factor of 570 to 650 times.

Genotype and Phenotype Memory Each of the 72 FPGA daughter boards includes 16 Mbytes of EDO DRAM to be used for storing the genotypes and phenotypes of the neural modules, a total of 1,180 Mbytes. The genotype is the set of genes in a cell and the phenotype is the final product of the genotype, the body and behavior that the genotype builds/generates. There are two modes of CBM operation, namely evolution mode and run mode. The evolution mode involves the growth phase and signaling phase. During the growth phase, memory is used to store the chromosome bitstrings of the evolving population of modules (module genotypes). For a module of 13,824 cells there are over 91 Kbits of genotype memory needed. For each module the genotype memory also stores information concerning the locations and orientations of the neurons inside the module, and their synaptic masks.

During the run mode, memory is used as a phenotype memory for the evolved modules. The phenotype data describes the grown axonic and dendritic trees and their respective neurons for each module. The phenotype data is loaded into the CA module to configure it according to the evolved function. The genotype/phenotype memory is used to store and rapidly reconfigure (reload) the FPGA hardware CA module. Reconfiguration can be performed in parallel with running the module, due to a dual pipelined phenotype/genotype register provided in each cell. This guarantees the continuous running of the FPGA array at full speed with no interruptions for reloading in either evolution or run modes. The phenotype/genotype memory can support up to 64,640 interconnected neural modules at a time. An additional memory will be based in the main memory of the host computer (Pentium-Pro 300 MHz) connected to the CBM through a PCI bus, capable of transferring data at 132 Mbytes/s.

Fitness Evaluation Unit Signaling in the CBM is accomplished with 1-bit spiketrains, a sequence of ones separated by intervals of zeros, similar to those of biological neural networks. Information, representing external stimuli, as well as internal waveforms, is encoded in spiketrains using a so-called “Spike Interval Information Coding (SIIC)”. This method of coding is implemented by nature in animal neural networks, and is very efficient in terms of information capacity per spike. Conversion from spiketrains into “analog” waveforms representing external stimuli, or internal signaling, is accomplished by convolving the spiketrain with a special multi-tap linear filter.

When a module is being evolved, it must be evaluated in terms of it’s fitness for a targeted task. During the signaling phase, each module receives up to 188 different spiketrains, and produces up to four different output spiketrains, which are compared with a target array of spiketrains in order to guide the evolutionary process. This comparison gives a measure of performance, or fitness, of the module.

Fitness evaluation is supported by a hardware unit which consists of an input spiketrain buffer, a target spiketrain buffer, and a fitness evaluator. During each clock cycle an input vector is read from its stack and fed into the module’s inputs. At the same time, a target vector is read from its buffer to be compared with the current module outputs by the evaluator. The fitness evaluator performs a convolution of the spiketrains with the convolution filter, and computes the sum of the waveform’s absolute deviations for the duration of the signaling phase. At the end of the signaling phase, a final measure of the module’s fitness is instantly available.

Genetic Algorithm Unit To evolve a module, a population of modules is evaluated by computing every module's fitness measure, as described above. A subset of the best modules are then selected for further reproduction. In each generation of modules, the best are mated and mutated to produce a set of offspring modules to become the next generation. Mating and mutation is performed by the CBM hardware core at high speed, configured for the genetic phase. During this phase, each cell's firmware implements crossover and mutation masks, two parent registers and an offspring register. Thus, each offspring chromosome is generated in nanoseconds, directly in hardware. Crossover is performed in parallel in hardware by all of a module's 14K CA cells. One crossover act takes about 100 ns for two parent chromosomes, each of which is 91Kbit long, using a 91Kbit crossover mask and a 91Kbit mutation mask. The selection algorithm is performed by the host computer in software, using access to the CBM via a PCI interface.

Module Interconnection Memory In order to support the run mode of operation, which requires a large number of evolved modules to function as one artificial brain, a module interconnection memory is provided. Each module can receive inputs from up to 188 other modules. A list of these source modules referenced to each module is stored in a CBM cross-reference memory (3 Mbytes) by the host computer. This list is compiled by CBM software using a module interconnection netlist in EDIF format. This netlist reflects the module interconnections as designed by the user, using off-the-shelf schematic capture tools.

The length of module interconnections is 96 cells (clock cycles). For each of the 64,640 modules, a Signal Memory stores up to three 96-bit long output spiketrains.

During the run mode, at the time each module of a brain is configured in the CA hardware core (by loading its phenotype), a signal input buffer is also loaded with up to 188 spiketrains according to the netlist in the module interconnection memory. The spiketrains are the signals saved from the previous instantiation and signaling of the 188 sourcing modules. At the same time, the three output spiketrains of the currently instantiated module are saved back to the Signal Memory. This repetitive cycling through all the modules which form the brain, results in a repetitive saving and retrieving of the spiketrains to/from the Signal Memory. It provides the signaling between modules according to the brain interconnection structure reflected in the schematics, designed by the user.

In a maximum brain with 64,640 modules, the CBM update rate is such that each cell propagates approximately 288 bit-long spiketrains per second. A 288 bit-long spiketrain can carry on the order of 72 bytes of signal information, using the SIIC coding method. Each neuron receives up to 5 spiketrains, so there are up to 188 million spiketrains being processed by neurons in the brain. Thus the maximum information processing rate by all neurons in the brain is of the order of 13.5 Gbytes/s.

Additional spiketrain processing in multiple dendritic branches can be estimated by assuming 50% of the total cellular space to be occupied by dendrite cells, each cell on average having 2.5 branches out of 5 possible. Informational throughput of dendrite cells is then of the order of 40.8 Gbyte/s.

External Interface The CBM architecture can receive and send spiketrains not only from/to the Signal Memory, but also from/to the external CBM interface. Any module can receive up to 188 incoming spiketrains and send up to 4 spiketrains to an external device, such as a robot, a speech processing system, etc. In a brain with 16,384 modules, the information rate, as measured at the external interface is up to 4.5 Kbytes/s per each module, or up to 74 Mbyte/s overall. In a smaller brain with less number of modules, the external information rate is higher, for example, a brain with 4,000 modules provides quadruple the external information rate for each module (18 Kbyte/s).

5.3 Further CBM Architectural Points

The CBM core is implemented as a large 12-layer backplane with 72 FPGA module boards plugged in. Each FPGA module board contains one Xilinx XC6264 BG560 FPGA, one Xilinx XC95216 BG352 CPLD, and a 16 Mbyte EDO DRAM module. (Each of the 72 FPGAs has a tightly coupled unshared 16Mbyte EDO DRAM that it is connected via the FastMap interface to the FPGA to provide the fastest possible speed for FPGA reconfiguration, as well as loading and saving neural module configurations in signal and growth phase.) Each FPGA contains 16K reconfigurable function units. Memory is used under CPLD control to load and save FPGA configurations to accomplish time sharing of the fast FPGA hardware. The datapath between memory and an FPGA is 32-bits wide and provides a data transfer rate of 66 Mbyte/s. An FPGA is thermally coupled with a temperature sensor circuit which is pre-programmed to shut-off the main clock when a temperature limit is exceeded.

The backplane serves primarily as a means to interconnect all 72 FPGAs. Each FPGA has 208 bi-directional connections to six other FPGAs arranged as a three-dimensional array of 6 by 3 by 4 FPGAs. In addition, the backplane's opposite side hosts several other boards used for overall sequencing and control of the system, implementing an SIMD (Single Instruction Multiple Data) architecture. Overall, there are 7.2 million reconfigurable gates in the CBM. To accomplish this connectivity, a High Density Metric connector system is used with press-fit contacts, providing over 30,000 connections.

The CBM is connected as a PCI target to a Pentium II computer which initializes the system and performs some background auxiliary control.

Although the CBM has been developed primarily to implement a specific neural network model based on cellular automata, its architecture is quite universal and very flexible. In fact, the CBM can be used for a large variety of applications which benefit from a high speed and fast reconfigurability of its hardware. Hardware-based implementations of a variety of algorithms have been shown to exceed the computational speed of high-cost super computers, as is the case with the CAM-Brain algorithm. The maximum computational power of the CBM is estimated to be equivalent to ten thousand Pentium II 400 MHz computers in the CAM-Brain algorithm implementation. Since this figure of 10,000 may be surprising to some readers, a quick justification is given. From the Xilinx data books, one can deduce that 72 Xilinx XC6264 chips contain 1.2 million FPGA functional units with 6 bit inputs and 6 bit outputs, operating at 11 MHz. Assume this is N times the bit processing rate of a Pentium II 400 MHz. Hence, in terms of bit processing rates, we have 1.2 million $\times$ 12 $\times$11 million $\approx N \times$ 400 million $\times$ 32 (bit word). N is roughly 10,000.

In particular, one application supported by the CBM architecture is gate-level and function-level evolvable hardware, which is based on applying a genetic algorithm to evolve complex digital circuits for a specific task. With 7.2 million gates, the resulting circuit complexity is likely to exceed human ability to design, debug, or even understand the dynamics of such a circuit. The CAM-Brain algorithm itself is an example of function-level evolvable hardware, where a basic unit of evolution is a function of a cellular automata cell, implemented as a specific (non-evolvable) logic circuit. This circuit can implement a number of different functions selectable by loading a chromosome bit string into the cell's genotype register which switches the cell to perform a specific function.

A summary of the CBM technical specifications can be found in Table 1.

6 “Robokoneko”, the Kitten Robot

An artificial brain with nothing to control is rather useless, so we chose a controllable object that we thought would attract a lot of media attention, i.e. a

Table 1. Summary of CBM Technical Specifications

Cellular Automata Update Rate (max.)	130 billion cells/s
Cellular Automata Update Rate (min.)	114 billion cells/s
Number of Supported Cellular Automata Cells (max.)	843 million
Number of Supported Neurons (max., per module)	1,152
Number of Supported Neurons (max., per brain)	74,465,244
Number of Supported Neural Modules	64,640
Data Flow Rate, Neuronal Level (max.)	13.5 Gbytes/s
Data Flow Rate, Dendrite Level (estimated average)	40.8 Gbytes/s
Data Flow Rate, Intermodular Level (max.)	74 Mbytes/s
Number of FPGAs	72
Number of FPGA Reconfigurable Function Units	1,179,648
Phenotype/Genotype Memory	1.18 Gbytes
Chromosome Length	91,008 bits
Power Consumption	1.5 KWatt (5 V, 300 A)

cute life-size robot kitten that we call "Robokoneko". We did this partly for political and strategic reasons. Brain building is still very much in the "proof of concept" phase, so we want to show the world something that is controlled by an artificial brain, that would not require a PhD to understand what it is doing. If the kitten robot can perform lots of interesting behaviors, this will be obvious to anyone simply by observation. The more media attention the kitten robot gets, the more likely our brain building work will be funded beyond 2001 (the end of our current research project).

Fig. 6 shows the mechanical design our team has chosen for the kitten robot. Its total length is about 25 cms, hence roughly life size. Its torso has two components, joined with 2 degrees of freedom (DoF) articulation. The back legs have 1 DoF at the ankle and the knee, and 2 DoF at the hip. All 4 feet are spring loaded between the heel and toe pad. The front legs have 1 DoF at the knee, and 2 DoF at the hip. With one mechanical motor per DoF, that makes 14 motors for the legs. 2 motors are required for the connection between the back and front torso, 3 for the neck, 1 to open and close the mouth, 2 for the tail, 1 for camera zooming, giving a total of 23 motors.

In order to evolve modules which can control the motions of the robot kitten, we thought it would be a good idea to feed back the state of each motor (i.e. a spiketrain generated from the pulse width modulation PWM output value of

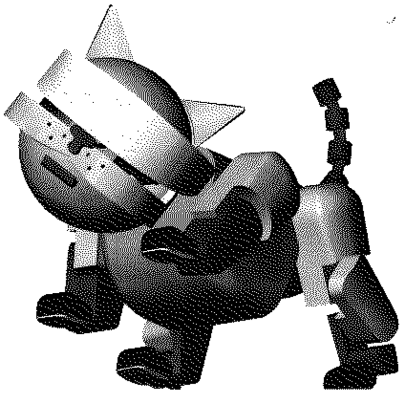

Fig. 6. "Robokoneko", the life-sized kitten robot to be controlled by our artificial brain

the motor) into the controlling module. Since each module can have up to 188 inputs, feeding in these 23 motor state values will be no problem. We may install acceleromotors and/or gyroscopes which may add another 6 or more inputs to each motion control module. It can thus be seen that the mechanical design of the kitten robot has implications on the design of the CBM modules. There need to be sufficient numbers of inputs for example.

The motion control modules will not be evolved directly using the mechanical robot kitten. This would be hopelessly slow. Mechanical fitness measurement is impractical for our purposes. Instead we will soon be simulating the kitten's motions using an elaborate commercial simulation software package called "Working Model - 3D". This software will allow output from an evolving module to control the simulated motors of the simulated kitten. This software simulation approach negates to some extent the philosophy of the CAM-Brain Machine and the CAM-

Brain Project, i.e. the need for hardware evolution speeds. This compromise was felt to be a necessary evil. In practice, the proportion of modules concerned with motion control will be very small compared to the total. Potentially, we have 64K modules to play with. Probably most of them will be concerned with pattern recognition, vision, audition, etc. and decision making. Designing the kitten robot artificial brain remains the greatest research challenge of the CAM-Brain Project and will occupy us through 1999, and beyond.

Work on the evolution of the motions of the robot kitten has already begun and a journal article on this work has been submitted for publication [5]. The strategy employed was as follows. The evolution of the kitten's behaviors will not occur at electronic speeds. The kitten's motions were simulated with Working Model 3D (WM3D) software, which is very elaborate, incorporating gravity, moments of inertia, frictions, etc. This software is used by major companies to simulate their new designs before fabrication. Our team wrote a software interface to WM3D which allows a genetic algorithm (GA) to be performed on the simulated motions. The user can specify a "fitness" (performance criterion) definition which is then used to evolve the desired motion. In practice, this evolution was very slow, taking several days per motion. We were thus motivated to find ways to accelerate this process. We took several options. One was to hand code "ball park" motion vectors (of the form of an angular acceleration per motor per clock tick, for all motors, for a given interval of clockticks) for a given desired motion. This hand coded ballpark solution served as an initial population in the genetic algorithm, and the GA was then used to "fine tune" the motion which was often jerky at first. Another option is to use a cluster of work stations with one GA chromosome (motion vector) per machine, resulting in an Nfold speedup with N machines. Once a motion vector is evolved under simulation it becomes the target vector that the CBM uses to evolve a corresponding module giving the same time dependent output. This evolved motion control module (actually one module per motor for all motors, in a set we call a "cluster") is then downloaded into the RAM of the artificial brain. (At least that is the plan. We have not tested this yet.) Evolving motions may be slow, but it is a "once off" affair, and although a major compromise to the "evolution of neural net modules at electronic speeds" philosophy, it is seen as being unavoidable, but not a crippling handicap.

Perhaps some clarification may be useful at this point. One may wonder that once the GA-using-simulation has evolved the needed motion vector, what is gained by then evolving a neural net that produces this same motion vector, which is, in effect, merely using the CBM to convert from the motion vector to the neural net. This is done for several reasons. One is to take advantage of the generalization properties of neural nets. Another is to have modules that can then be run efficiently on the CBM. Thus it may appear that the CBM is being used only as a "motion vector to neural net converter", and as a runtime environment for the set of 10,000s of modules. It appears as though the FPGA is not being used for the real learning. This is true for the motion control modules, but is not true for the vast majority of modules, such as the aural and visual pattern detector modules, the logic control modules, etc. We estimate that the motion control modules will constitute less than a few percent of the total. Since the non motion control modules will be evolved directly in the CBM, the claim that the (evolutionary) learning does not occur in the CBM is not true in most cases.

We do not know yet how well we can get modules to interact together. This question remains unexplored. Without a CBM's speed, the evolution and updating of many interconnected modules remains impractical. No one has tried to build an artificial brain before on the scale attempted in this project. Of course, we can define an "artificial brain" to be whatever we like (in the case of the CAM-Brain project, the definition is "an assemblage of evolved CA based neural net modules"). With the CBM evolving a module in about 1 second, and with hundreds of evolutionary engineers helping out, maybe it is realistic to build an artificial brain with 10,000s of modules in a few years. We don't know. That is the research challenge.

7 A Sampler of CoDi-1Bit Evolved Neural Net Modules

Since the whole point of using the CBM is to attain a high evolution speed, it is useful if the representation chosen to interpret the 1 bit signals which enter and leave the CoDi modules can be unique, otherwise several representations would need to be implemented in the electronics. (For the CBM to be efficient, i.e. to evolve CoDi modules in about 1 second, fitness measurements need to be performed at electronic speeds, which implies that the representation chosen for the signals be implemented directly in the hardware). We chose the SIIC to be our unique representation. However, as mentioned at the bottom of section 5, most of the evolutionary experiments presented here were already undertaken before the SIIC representation was chosen. Since the results of these earlier experiments are interesting in their own right, we report on them here. They show to what extent that CoDi modules are evolvable and the power of their functionality. The evolution of SIIC-representation-based and HSA-based modules will be the subject of work in the very near future, given that both algorithms are now ready. So is the CBM multi-module simulation code, so progress should be rather rapid in the coming months prior to the delivery of the CBM itself. Once the CBM is delivered, multi-module systems should be built as fast as we can dream them up. The bottleneck in building large scale multi-module systems will become human creativity lag, not module evolution lag (as was the case with software evolution speeds in the "pre-CBM era".) We now provide a sample of evolved CoDi neural net modules, their specified functionalities, and their actual performances, to give a feel for what they can do.

7.1 Multiple Timer Module

Since a 100% fitness score does not test the limits of evolvability of a module, a more demanding output function was tried. The target output (similar to the above pattern) and the actual evolved output (placed immediately under the target pattern for comparison) were as follows:

Target
000000000000000000000000000000 11111111111111111111
Evolved
000000000000000000000000000000 00011111111111111111
Target ctd.
000000000000000000000000 1111111111111111 00000000000000000000
Evolved ctd.
100000000000000000000000 0111111111111111 10000000000000000000

The fitness definition was as follows. If a 0 appeared in the first (0) block, score 12 points. If a 1 appeared in the second (1) block, score 7 points. If a 0 appeared in the third block (0), score 3 points. If a 1 appeared in the fourth block (1), score 2 points. If a 0 appeared in the fifth block (0), score 1 point. Hence a perfect score would be 30×12 + 20×7 + 24×3 + 16×2 + 20×1 = 624. These weightings were chosen so as to encourage the earlier outputs to be correct before the later outputs. Population size was 30. No crossover. This result converged after 100 generations with a fitness value of 0.957.

It is interesting to note that these good results were evolving in 100 generations, and yet the chromosome length is very large. The standard CBM chromosome length is of the order of 90K bits. One might think that such a long chromosome would be very slow in evolving, but this was not the case. One possible explanation for this is that there may be so many possible solutions, that (any reasonable) one is quickly found.

7.2 Pattern Detector Module

With a slight modification of the code used to evolve the above module, a pattern detector module was evolved which was capable of distinguishing between two square wave inputs, of 111000111000... and 11111000001111100000... In this case, no switch input was used. Two experiments were run. In the first, the input was the 6 clocktick cycle square wave input, applied at the fixed input point $(8, 8, 0)$. In the second experiment, the circuit was regrown with the same chromosome and the 10 clocktick cycle square wave input was applied to the same fixed input point. The fitness definition was the same as above. Results are shown below. Over 90 clockticks, the first output had 48 more 1s than the second output.

```
Square wave input 111000111000...
Output
00000000000000000000000010011011101111111111111111111111111111111
1111111111111111111111111
Square wave input 11111000001111100000...
Output
00000000000000000000000000001000100010001000100010001000100010001
0000000000010001000100010
```

Since the CoDi modules seem capable of evolving such detectors, it may be possible to evolve modules which are capable of detecting a specific phoneme analog input (e.g. the spike train (bit string) which when convolved with a particular convolution function gives the time dependent analog signal). In a manner similar to the above, one could input the signal in the first experiment, and a random signal in the second, in a multi-test experiment, and evolve the phoneme detector. Maybe one could evolve a set of detectors, one for each phoneme. By using the SIIC and HSA digital/analog conversions, this kind of thing may become quite practical.

For the evolution of the pattern detector module and the module in the next section, the population size was 15. Fitness values reached saturation after about 200 generations or less. After the evolution, observation of the occupancy of the CA cells showed that most of them were filled (more than 95%), with roughly equal numbers of axons as dendrites, which is to be expected given that the initial growth instruction at a neuron's face could be an axon or dendrite with a 50/50 chance depending on whether a bit was set to 1 or 0 in the module's chromosome. Since it took an overnight run to evolve a single module, usually only one run was undertaken. Once the CBM is functional, more robust statistics can be undertaken by averaging over many runs.

7.3 Hubel-Wiesel Line Motion Detector Module

The results of the following experiment were significant for the CAM-Brain Project as a whole, we felt. It involved the evolution of a Hubel-Wiesel type line motion detector. Hubel and Wiesel won a Nobel prize for discovering that certain neural cells in the visual region of the cat's brain fired strongly when lines of light at particular orientations and speeds were shone onto a screen that the cats were observing. These cells (neurons) were detecting the motion of lines at a particular orientation. The evolution of this "Hubel-Wiesel" module used the same fitness definition and a similar methodology as in the above case. In the first experiment, a square 12×12 neuron input grid was used. At the first clock tick, the top horizontal 12 neurons were made to fire - at the second clock tick, the second horizontal row of 12 neurons was made to fire, etc, for 12 clock ticks, then the cycle was repeated. This input firing pattern simulated the motion of a line of light moving horizontally down the visual field on the retina of a cat. In the second experiment, 12 randomly positioned input neurons were fired at each clock tick. These 12 positions were randomly generated for each clock tick. This second input firing pattern simulated input noise, to be contrasted with the line motion input. Output results are shown below.

Line Motion Input Case

Output

```
00000000000000000000000010011010011011111111111111111111111111111
1111111111111111111111111
```

Random Input Case

Output

```
00000000000000000000000000001000000000000000010001010101010101010
1010101010101010101010101
```

There were 35 more 1 bit outputs in the first case than the second. Since the inputs to the second case are positioned randomly, the same neural net module will generate a different fitness value depending on the input. Nevertheless the evolution still improved over time, developing a fairly robust net giving fitness values corresponding to over 30 1-bit differences (between the two experiments) in most cases (e.g. the top 5 fitness chromosomes were saved for each generation and not crossed over or mutated. The fitness values (1-bit difference count) of these top 5 were 31, 34, 35, 30, 29 after several hundred generations). Thus it was possible to evolve a Hubel-Wiesel Line Motion Detector. Actually, the statement that it was possible to evolve a Hubel-Wiesel line motion detector may be disputed here. For example, the target function could be stated as: "Output 1 if and only if at least one of the horizontal rows of inputs contains all 1's." This can be done with a multi-input AND gate (one neuron perhaps) for each row, followed by an OR (one dendrite). A more interesting conclusion might be that Hubel-Wiesel type systems can be more naturally achieved because of the crucial element of spatial layout in the CAM-Brain, as distinct from purely topological networks. Of course, we have no idea how the circuit does what it does. This is the great strength of "evolutionary engineering". Evolved circuits can at times achieve performance levels beyond what human engineers can achieve with traditional top-down design techniques, i.e. attain superior engineering performance levels, but the price is that one loses scientific understanding, due to the overwhelming structural and dynamical complexity of these CoDi circuits. The reason why many evolutionary engineers feel that evolutionary engineering can potentially evolve a level of functionality superior to what human beings can design, is due to the so-called "complexity independence" of the genetic algorithm. This notion of complexity independence is informal. It means that the GA does not care about the inherent structural and dynamic complexity of the system it evolves. All that matters to a GA is that its (scalar) fitness values keep increasing. Hence a GA can evolve an extremely complex system (in structure and/or dynamics) which may surpass the complexity level limit that a human brain can comprehend. That extra complexity can provide an extra level of functionality. This feature is thought to be one of the great advantages of evolutionary engineering. Thus "evolutionary engineering" can sometimes provide a superior form of engineering. In practice, once evolutionary engineers can generate tens of thousands, even millions of modules, only a few die-hard analysts will want to know how an individual module functions. For the most part, no one will care how a particular module amongst millions actually does what it does.

8 Ideas for Interesting Future CoDi Modules to be Evolved

8.1 Multi-Test Modules

The CBM hardware automatically performs a fitness measurement on the assumption that the 1Bit signals which leave the evolving module into the fitness measurement circuit are interpreted with the SIIC approach, i.e. the hardware actually implements the SIIC convolution algorithm. We have implemented the

CBM having a single very general fitness measurement methodology, to simplify the electronics. Hence evolutionary engineers using the CBM will need to specify the functions of the modules they want to evolve using the SIIC methodology. However, there is a problem with this unified approach, namely how to give the same circuit several tests, i.e. several sets of different inputs in a single run. For example, imagine one aims to evolve a module which detects a time dependent input pattern P. One inputs the pattern P for T_p clocks. One wants the module to respond strongly when the pattern in detected, and weakly if any other pattern is presented. Hence the same circuit needs to be tested for several pattern inputs, i.e. P and others. The pattern P is called the positive case, while the others are called the negative cases. (It is also possible that there may be several positive cases (P_i)). One does not want a module which responds well to any pattern. It must discriminate.

How does one test all these cases (positive and negative) in a single run? By concatenating them, i.e. sandwiching them over time. For example, imagine there are 2 positive examples and 4 negative examples to be input to the same circuit. Hence there will be 6 time periods in which the patterns are presented sequentially at the input in one long run. Between each input signal presentation, the signal states in the circuit are cleared out, ready for the next signal input. This the CBM actually does. This resetting of the signal states is part of the CBM fitness measuring approach that we call "multi-test" fitness measurement. The 6 input pattern periods can be represented as "$P_1, P_2, N_1, N_2, N_3, N_4$". The periods last P_i and N_i clock ticks each. So that the total number of clock ticks for the positive periods is more or less equal to the total of the negative periods, the durations of the P_i can be lengthened. This should increase the evolvability of the positive responses, otherwise the evolution may favor the negative cases too heavily. The target output patterns one wants for these 6 periods can be represented as "high, high, low, low, low, low".

Clearing the signal states between individual inputs in multi-test runs is needed because it is possible that self sustaining reverberating loops will be set up once an initial input is switched off. Such self sustaining loops may in fact be very useful, since they can be looked upon as a form of memory, and hence may be used to make CoDi modules capable of learning, i.e. adapting to their experience. The next subsection elaborates on this idea.

The CBM evaluates each partial fitness (one for each test in the multi-test case) and then sums the partial fitnesses to get the total fitness for the circuit (the module).

8.2 Modules Which Learn

Until recently, we have always thought that the CAM-Brain Project would produce neural circuits that would be INcapable of learning, i.e. they would not modify themselves based on their run time experience. The rationale was that it would be complicated enough dealing with tens of thousands of non learning modules all interacting with each other, let alone having tens of thousands of learnable modules. Also, we saw no way of having CoDi modules which could learn. Lately however, we have begun to think that learnable CoDi modules might be evolvable. The essence of learning in a system is that some event in the past leaves some trace or memory in the system. In a CoDi module, that could take the form of reverberating internal signaling after an initiating input arrives. In some modules, once the input stops, the resulting 1Bit signals could die away, i.e. be transient. Alternatively, the reverberating signals could persist and hence constitute a form of memory. Thus CoDi modules may be evolvable which generate reverberating signals.

8.3 From Multi Module Systems to Artificial Brains

Once our group and others have gained a lot of experience in evolving single modules, the next obvious step is to start to design multi-module systems, since the ultimate goal of the CAM-Brain Project is to put many many modules together (up to 64,460 of them in the current design of the CBM) to make artificial brains. Obviously, no CAM-Brain team will try to build a 64k module brain (with maximum 75 million artificial neurons) all at once. Instead, as a first step, small multi-module systems will be built, with tens of modules. Once experience is gained in how to do this successfully, larger systems will be undertaken, e.g. with 100s of modules, then 1000s, and later 10,000s. Note that at the time of writing (Spring 1999) the authors make no pretense of knowing how to design a 64k module artificial brain. The whole point of the CAM-Brain project is to provide a tool which renders artificial brain building practical. Now that the tool exists, it is quite possible that the theory and the practice of brain building will advance rapidly. Just how to design a module artificial brain remains the major research challenge for the authors for the next few years.

Over time, artificial nervous systems should grow in complexity, until they can be called artificial brains. The robot kitten that our team is currently designing will be controlled by an artificial brain with up to 64k modules. Since this kitten robot contains a CCD TV camera, microphones for ears, touch sensors, 22 motors for the legs and body, etc, it should offer plenty of scope for brain building. This is a huge amount of work, which will need to be distributed over many CAM-Brain teams across the planet. With modern (almost cost free) internet telephone technology, coordinating such a large management effort is less expensive.

9 Related Work

This section deals with a sample of research work performed by others, which is related to the CAM-Brain Project, the CAM-Brain Machine (CBM), and the kitten robot Robokoneko that our artificial brain will control. Three aspects of our work have been chosen for comparison with comparable work by others, namely, the cellular automata machine (CAM) aspect, the runtime reconfigurable hardware aspect, and the pet robot aspect. In each of these three subsections, an initial brief summary of the related work is given, followed by a comparison with our work. The three related works we chose to discuss are -
a) Margolus and Toffoli's CAM-8 Cellular Automata Machine
b) Eldredge and Hutchings' Runtime Reconfigurable Neural Net Hardware
c) Sony's Pet Dog Robot 'Aibo'.

9.1 Margolus and Toffoli's CAM-8 Cellular Automata Machine

The Information Mechanics Group at MIT has been concerned for the past decade (until they transfered recently to Boston University) with the physics of computation, including such topics as quantum computing, crystalline (3D) computing, and the hardware acceleration of cellular automata based modeling. Margolus and Toffoli have designed 8 versions of their Cellular Automata Machine (CAM) over the years [18]. Our group purchased their 8th version CAM-8 in 1994 and used it to obtain the graphics of our evolving neural circuits, some with 10 million artificial neurons. See de Garis's home page for these images. The title of our research project, "CAM-Brain Project" was based on the idea of putting an artificial brain inside a large cellular automata space inside a Cellular Automata Machine, hence CAM-Brain. The CAM-8 is essentially a dual RAM based lookup table hardware device. A 16 bit entry address for the LUT is obtained from the state of the central cell at a given position and its 4 neighbors. If the maximum number of states is 8, i.e. 3 bits, then the 5 states in the order (center, north, east, south, west) generate a 15+1 bit string (with an extra zero). This address points to the next state of the center cell. With two such

RAM memories, the RAM-1 at time T can be used to generate the states of the CA space at time T+1, which are stored in RAM-2. At time T+1, the RAM-2 is used to generate the states of the CA space in RAM-1, overwriting the old contents. Thus the two RAMs ping-pong, updating each other. This is done at a rate of 200 million CA cells a second.

The CAM-Brain Machine (CBM) is a more specialized device, devoted to the evolution of CA based neural network circuit modules which are downloaded into a gigabyte of RAM. Once all the (maximum 64k) modules are downloaded, the CBM is used to update this space at a rate of 130 billion CA cells a second, which is some 650 times faster than the CAM-8. The CAM-8 is a general CA hardware accelerator. The CBM is exclusively for neural nets, unless one reprograms the FPGAs it contains. The CAM-8 has been used mainly in applications of CA simulated fluid flows, electromagnetic wave simulations etc. The CBM has been constructed with the specific aim of building artificial brains, although one company, Belgium's Lernout and Hauspie (L&H) has bought one for speech processing feasibility studies.

9.2 Eldredge and Hutchings' Runtime Reconfigurable Neural Net Hardware

Eldridge and Hutchings use a run time reconfigurable (FPGA) hardware system (called RRANN) to execute a backpropagation learning algorithm in a feed forward neural net [6]. They divide the learning task into three phases (feed-forward, backpropagation, and update), each with its own circuitry, which is configured consecutively into the FPGAs during run time. They achieve a greater efficiency in silicon use this way, since without the reconfiguration, far more silicon would be needed and for most of the time would not be used.

There are several similarities and contrasts which can be made between RRANN and the CBM. Both use run time reconfiguration, although in the case of RRANN, the reconfiguring occurs only twice (between the three phases) whereas the CBM is constantly reconfiguring its FPGAs into (neural net) growth mode and signaling mode, for each generation for hundreds of generations in the genetic algorithm. The CBM deals with millions of artificial neurons, whereas the RRANN deals with far fewer, the nature of the RRANN task being quite different. RRANN uses a non evolutionary approach, as distinct from the CBM. RRANN limits itself to neural nets that use feed forward signaling (characteristic of the backprop algorithm). CBM uses an evolutionary approach for which the internal complexity of the neural circuitry that is evolving is irrelevant, and hence can be much more complex in its structure and dynamics, and hopefully, because of that, more performant than feedforward networks.

9.3 Sony's Pet Dog Robot 'Aibo'

Our team thought that an artificial brain without a body for it to control would be rather useless, so we decided it would be a good idea to have it control a cute lifesized kitten robot called Robokoneko. However, after we conceived Robokoneko, SONY Corporation of Japan, unveiled its plans to make a similar robot pet, which they eventually called "Aibo" which is Japanese for "pal, mate, partner". It is about the size of a living Chiwawa dog, whose image can be seen at [13]. It has a limited number of behaviors (a dozen or so) which include, walking, turning, following and kicking a ball, getting on its feet, wagging its tail, etc. It is controlled by a few onbody microchips and costs a few thousand dollars.

The kitten robot is rather similar in concept, except that it will be controlled by an artificial brain, which will be orders of magnitude more sophisticated and performant than Aibo's microprocessors. Since the first generation artificial brain controlled by CBM-1 can contain 64k evolved neural net modules, we can afford to be ambitious. (We plan by about 2001, to have a second generation machine CBM-2, to handle a billion neuron artificial brain with a million modules, i.e. 16 times more). We can give the kitten robot hundreds of behaviors, thousands of pattern recognizers etc, and have it switch between these many behaviors depending upon its moods, its drives, its internal states (such as curiosity, hunger, boredom), its external stimuli etc. To the casual observer, the difference in the behavioral repertoire and general intelligence levels of Aibo and Robokoneko should be marked. However, Robokoneko is still a concept, whereas Aibo is already a product, due to the greater human resources SONY was able to give to its development. Robokoneko is much more of a research project, as nobody really understands yet how to build an artificial brain.

10 Conclusions

This article has provided an overview of ATR's CAM-Brain Machine (CBM) and the Artificial Brain ("CAM-Brain") Project of which the CBM is the project's fundamental tool. The CBM should be delivered to ATR in the third quarter of 1999. The CBM will update 130 billion 3D CA cells a second and evolve a CA based neural net module in about 1 second. This speed should make practical the assemblage of tens of thousands of evolved neural net modules into humanly defined artificial brain architectures, and hopefully create a new research field that we call simply "Brain Building". This article has discussed the neural net model ("CoDi-1Bit") which is implemented by the CBM. Also presented were discussions on how to convert back and forth between analog time dependent signals and spiketrains (bit strings of 0s and 1s), thus enabling users to think entirely in terms of analog input and target output signals. A sample of evolved neural network modules using the CoDi-1Bit model was given. Once the CBM is delivered and sufficient experience with it enables the construction of large neural systems, with tens of thousands of modules, an artificial brain will be designed and built to control the behavior of a robot kitten called "Robokoneko". The challenges which remain in the CAM-Brain Project are to fully test the limits of the evolvability of the CoDi-1Bit modules (using the CBM), so as to gain experience in what can be readily evolved and what cannot, and then to assemble large numbers of them to make Robokoneko's brain. The biggest challenge will probably be creating the brain's architecture, our main task for 1999, and beyond.

The CBM should be fast enough for many multi-module tests to be undertaken. Multi-module systems can be evolved, assembled into the RAM, and then tested as a functional unit. Once a system has been built successfully it can be used as a component in a larger system, ad infinitum. The challenges of the CAM-Brain Project are not only conceptual in nature, but managerial as well. A back of the envelope calculation says that if an evolutionary engineer (i.e. someone who evolves a neural net module using a CBM) takes half an hour of human thinking time to dream up the fitness definition (i.e. the performance criterion) of a module, to specify the module's input signal(s), its target output signal, its input and output links with other modules, etc, then 8 evolutionary engineers would be needed to complete the design of a 64k module artificial brain within 2 years. Thus one needs to speak in terms of brain builder teams. If one wants to be a lot more ambitious and build a million module artificial brain in 2 years, then 120 evolutionary engineers are needed. Such a large team would need managers to control them. One can imagine higher level "brain architects" handing out module specifications to lower level evolutionary engineers who actually evolve them on their CBMs and report back to the brain architects with the results. The brain architects and evolutionary engineers need not be located in one place. Modern internet telephone technologies, which we use successfully on a daily basis, make globally distributed "virtual teams" practical.

If artificial brains can be made to work reasonably successfully, e.g. by making interesting robot pets, or simple household cleaner robots, etc, then a new artificial brain based computer industry will probably be created. However, this will only be possible if machines such as the CBM can deliver sufficient "evolvability" to make it happen. By evolvability is meant the degree to which some desired functionality is evolvable by a given model and implementation. As evolutionary engineers quickly learn, not all neural net modules evolve as one would wish. For example, it is quite possible that the decision to limit the CoDi model to 1 bit neural signaling (in order to implement the model in the Xilinx XC6264 chips) has limited the evolvability of the CoDi neural net modules. The first author (de Garis) evolved neural net modules (in software) with 8-10 bit neural signals for his PhD a decade ago [1], and obtained a remarkable level of evolvability, but even then there were limits. Section 7 above has provided a taste of what CoDi-1Bit modules can do. Once our team has the CBM, we will be able to broaden rapidly our experience in CoDi module evolution and hence obtain a feel for its evolvability, within the constraints of 1 bit signaling and the CA based neural nets. We will then be more able to design an artificial brain based on modules that are evolvable in practice.

As Moore's law provides more powerful evolvable chips in future years, later versions of the CBM will be able to implement more complex neural net models, with multi bit signaling, with more realistic neuron models, etc, and hence provide a greater level of evolvability, a concept fundamental to the effort of building artificial brains. As an example of evolvability, consider the attempt to evolve a module whose output should be as close as possible to some time dependent wave form. In a recent paper submitted for publication [3], we evolved a module which followed a sinusoidal curve very closely (with 2-5% error) for about 40 clockticks, and then diverged. We then changed the neural net model (in simulation) by adding more bits to the model's GA chromosome. The evolved curve then followed the target curve for about 80 clocks before divergence. This increase in the module's "MEC" ("Modular Evolvable Capacity") was due presumably to the increase in the number of bits in the chromosome, giving the module a greater potential to generate a desired behavior for longer. Any model with a finite number of bits will obviously have a limit to how extensively it can be evolved to generate some desired function. We suspect that this concept of the MEC will play a fundamental role in future evolutionary engineering and particularly in brain building. Perhaps by combing modules in some way, it may be possible to extend the MEC of the whole indefinitely. This remains a challenge for future research. We feel that future generations of the CBM, using future generations of evolvable chips, will generate a steady increase in the

MECs of the modules they evolve. Evolutionary engineers should be aware of the limitations of the evolutionary approach. They should be conscious of the concept of the MEC and the drive to increase their values (e.g. the number of clockticks during which the evolved curve follows closely the target curve before diverging).

Acknowledgements

The authors acknowledge the financial, contractual and managerial support of Katsunori Shimohara of ATR/NTT. Shimohara believed in the possibility of building an artificial brain since de Garis proposed it to ATR in 1992. The authors also acknowledge the anonymous reviewers who did a fine job in providing suggestions to improve this paper.

References

1. Hugo de Garis. *Genetic Programming: GenNets, Artificial Nervous Systems, Artifical Embryos.* PhD thesis, Brussels University, January 1992. Available at http://www.hip.atr.co.jp/~degaris.
2. Hugo de Garis. An artificial brain : ATR's cam-brain project aims to build/evolve an artificial brain with a million neural net modules inside a trillion cell cellular automata machine. *New Generation Computing Journal*, 12(2), July 1994.
3. Hugo de Garis, Andrzej Buller, Michael Korkin, Felix Gers, Norberto Eiji Nawa, and Michael Hough. ATR's artificial brain (CAM-Brain) project: A sample of what individual CAM-Brain modules can do with digital and analog I/O. Proceedings of the First NASA/DoD Workshop on Evolvable Hardware, July 19-21 1999, Pasadena, California, IEEE Computer Society, ISBN 0-7695-0256-3.
4. Hugo de Garis, Felix Gers, Michael Korkin, Arvin Agah, and Norberto Eiji Nawa. Building an artificial brain using an FPGA based 'CAM-brain machine'. *Artificial Life and Robotics Journal*, 1999. to appear.
5. Hugo de Garis, Nikolai Petroff, Michael Korkin, and Gary Fehr. Simulation and evolution of the motions of a life sized kitten robot 'robokoneko' to be controlled by a 32000 neural net module artificial brain. Submitted to publication, available on website `http://www.hip.atr.co.jp/~degaris/papers/JCG.html`.
6. J.G. Eldredge and B.L. Hutchings. Rrann: The run-time reconfiguration artificial neural network. In *Proceedings of the Custom Integrated Circuits Conference*, May 1994. Available at http://splish.ee.byu.edu/docs/cicc94.rrann.ps.
7. An Evolutionary Engineering Consultancy Company based in Boulder, Colorado, at URL `http://www.genobyte.com`.

8. Felix Gers, Hugo de Garis, and Michael Korkin. CoDi-1 Bit: A simplified cellular automata based neuron model. In *Proceedings of AE97, Artificial Evolution Conference*, October 1997.
9. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.
10. T. Higuchi, M. Iwata, and W. Liu, editors. *Evolvable Systems: from Biology to Hardware.* Springer-Verlag, 1997. Lecture Notes in Computer Science No 1259.
11. Michael Korkin, Hugo de Garis, Felix Gers, and Hitoshi Hemmi. CBM (CAM-Brain Machine): A hardware tool which evolves a neural net module in a fraction of a second and runs a million neuron artificial brain in real time. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, July 1997.
12. Michael Korkin, Norberto Eiji Nawa, and Hugo de Garis. A 'spike interval information coding' representation for ATR's CAM-brain machine (CBM). In *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES'98).* Springer-Verlag, September 1998.
13. MSNBC. Sit, aibo, sit!, msnbc, technology report. Available on website `http://www.msnbc.com/news/268649.asp`.
14. Fred Rieke, David Warland, Rob de Ruyter van Steveninck, and William Bialek. *Spikes: exploring the neural code.* MIT Press/Bradford Books, Cambridge, MA, 1997.
15. David Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the microstructure of cognition.* MIT Press/Bradford Books, Cambridge, MA, 1986.
16. Eduardo Sanchez and Marco Tomassini, editors. *Towards Evolvable Hardware: The Evolutionary Engineering Approach.* Springer-Verlag, 1996. Lecture Notes in Computer Science No 1062.
17. Adrian Thompson and Paul Layzell. Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79, April 1999.
18. T. Toffoli and N. Margolus. *Cellular Automata Machines.* MIT Press, Cambridge, MA, 1987.
19. Xilinx, Inc. *The Programmable Logic Data Book 1996*, 1996.

Figure Captions

1. Fig. 1 The convolution function used in the "SIIC" representation.
2. Fig. 2 An analog (smooth) curve and its deconvolved/convolved approximation (jerky) curve.
3. Fig. 3 A 3 period sine curve resulting from convolution of an evolved CoDi-1Bit. The lower figure shows the actual spikes that generated the waveform.
4. Fig. 4 Outputs of a halver circuit (with inputs 600 and 400) using fully analog I/O.
5. Fig. 5 The spiketrain output of Fig. 2, as generated by the Hough Spiker Algorithm (HSA).
6. Fig. 6 "Robokoneko", the life-sized kitten robot to be controlled by our artificial brain.

Table Captions

1. Table 1 Summary of CBM Technical Specifications.

BIOs

Prof. Dr. Hugo de Garis

Prof. Dr. Hugo de Garis is head of the Brain Builder Group at ATR Labs in Kyoto, Japan. He is the father of the rapidly growing research field "evolvable hardware", a concept he got off the ground in 1992. He uses evolvable hardware techniques to evolve neural network circuit modules at electronic speeds using FPGA based hardware. He is assembling 64000 of these modules in RAM to build a 75 million neuron artificial brain. He obtained his PhD in artificial nervous systems in 1991 from the University of Brussels (ULB), Belgium in 1991. From February 2000, he will be continuing his artificial brain work at Starlab, in Brussels (http://www.starlab.org).

Dr. Michael Korkin

Dr. Michael Korkin received his M.S. degree in Computer Systems Engineering from MIIT, Moscow, Russia, in 1982, and his Ph.D. degree in Digital Image Processing from MPI, Moscow, in 1988. In 1991-1997 he worked as a Senior Hardware Engineer at a medical imaging firm in Denver, Colorado, USA. He founded his company Genobyte Inc. in 1997 in Boulder, Colorado. His primary research interests are evolvable hardware, artificial brain building, and neuroscience.

This article was processed using the LaTeX macro package with LLNCS style