Much further progress needs to be made in this new field (of GP). For example, consider the possibility of using GP techniques to "grow" electronic neuro circuits in special machines called "Darwin Machines", an idea discussed above in the section on future research possibilities. A Darwin Machine is merely an example of the kind of thing being proposed here which would both self-assemble and self-test.

## 11.6 Ongoing Work

This section contains two subsections,

a) Evolvable Hardware & Darwin Machines
b) Evolving Artificial Nervous Systems

Both subsections can be classified as ongoing work. They contain ideas only, due to the fact that neither of them has as yet been implemented and tested. They therefore deserve to be placed in this chapter on future work.

### 11.6.1 Evolvable Hardware & Darwin Machines

As stated several times throughout this book, the author dreams of the possibility of building machines which are capable of evolution, called "Darwin Machines" (see the Postscript). As a result of several brain storming sessions with some American colleagues (Dr. Kenneth Hintz and Dr. Richard Auletta) from the Electrical Engineering Department of George Mason University in Virginia, the author now realizes that hardware devices are on the market today, which use "software configurable hardware" principles, that the author believes could be adapted and then applied to building Darwin Machines. This could probably all be done within a few years.

This subsection suggests there are at least two approaches to be taken. The first approach uses "software configurable hardware" chips, e.g. FPGAs (Field Programmable Gate Arrays), HDPLDs (High Density Programmable Logic Devices), or possibly a future generation of chips based on the ideas that FPGAs etc embody.

The second approach uses a special hardware device called a "hardware accelerator" which accelerates the simulation in software of digital hardware devices containing up to several hundred thousand gates. Darwin Machines will be essential if artificial nervous systems are to be evolved for biots (i.e. biological robots) which consist of thousands of evolved neural network modules (called GenNets). The evolution time of 1000-GenNet biots will need to be reduced by several orders of magnitude if they are to be built at all. It is for this reason that Darwin Machines may prove to be a breakthrough in biotic design. When molecular scale technologies come on line in the late 1990s, the Darwin Machine approach will probably be the only way to build self assembling, self testing molecular scale devices.

#### Software Configurable Hardware

The basic idea behind a Darwin Machine, is to use a type of hardware device which accepts some bit string software instruction and uses it to configure (i.e. to "wire up") a hardware circuit and not just once, but repeatedly. The outputs of this newly configured hardware circuit are then tested with another piece of software configurable hardware to determine the quality of its performance, i.e. its fitness. Those bit string instructions which configure high fitness circuits, survive (a la GA), to reproduce in the next generation. Thus every time a new bit string instruction is input, a new circuit is

generated or configured. Modern "software configurable hardware" technologies [Broesch 1991] (e.g. using EEPROM (electrically erasable PROMs) or SRAM (static RAMS)) allow this possibility, so the idea of "evolvable hardware" can become a reality. It is a most exciting prospect, and may have a major impact not only upon biotics, the immediate concern of the author, but on the whole of electronics. As electronic and computer circuits reach complexity levels which are beyond human capacities to comprehend, the Darwin Machine may end up being the only effective circuit design tool.

FIG. 11.6.1.1 shows one possible Darwin Machine architecture. It is more centrally controlled than the alternative, more distributed architecture shown in FIG. 11.6.1.2 The architecture of FIG. 11.6.1.1 contains a single master circuit and a population of slave circuits which function in parallel. The master circuit sends each slave a "GA chromosome" (i.e. a bit string instruction which is used to configure a hardware circuit), the population fitness definition (i.e. a second bit string instruction which is used to configure the hardware circuit which measures the fitness of the outputs from the first circuit), and initial input values. Each slave takes its chromosome, configures a circuit according to the instructions contained on the chromosome, measures the circuit's fitness according to the fitness definition, and reports this value back to the master, which then calculates the next generation of chromosomes in a GA like way, to complete the cycle.

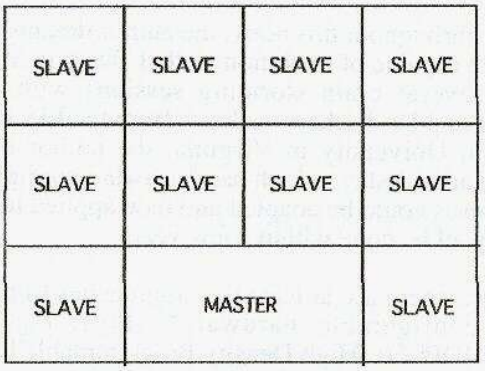


FIG. 11.6.1.1 A Centrally Controlled DARWIN MACHINE Architecture

Each slave therefore consists essentially of two circuits, one which evolves, and the other which does not, but is used simply to measure the fitness of the circuit which does evolve. However, this raises a new problem. The whole point of a Darwin Machine, besides accelerating the evolution time, is to enable the evolution of complex circuits, in typical GP fashion. That is, the circuits which evolve, may be too complex for human comprehension. However, the fitness measuring circuit must (at some level) be humanly comprehensible, because it has to be humanly specifiable. At some point in the whole procedure, a human genetic programmer must specify the fitness definition, and provide the corresponding bitstring instruction which configures the fitness measuring circuit.

One can imagine however that the fitness measuring circuit is itself evolved using a simpler fitness definition. Thus a whole chain of fitness circuit evolutions becomes possible. At the end of the chain is a humanly comprehensible fitness definition and corresponding configuration instruction.

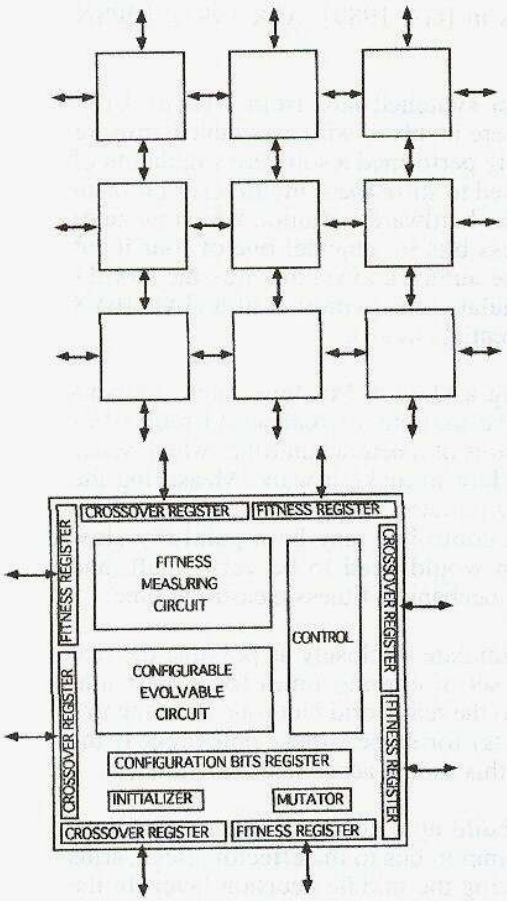


FIG. 11.6.1.2 A Distributed DARWIN MACHINE Architecture & CHIP

FIG. 11.6.1.2 shows a more distributed Darwin Machine architecture, where there is effectively no master circuit. Each chip or cell (as shown in more detail on the RHS of the figure) is an element in a two dimensional grid. The chip design is merely suggestive and makes absolutely no pretence at being realistic. Broadly speaking, it would generate its own initial configuration bitstring (i.e. chromosome), which then configures the evolvable circuit. For the author, this evolvable circuit would be a hardware implementation of a GenNet or circuit of GenNets. However, since the idea of evolvable hardware is quite general, the evolvable circuit could be virtually anything. It is this generality which makes the Darwin Machine concept so potentially interesting for the electronics industry.

Once the evolvable circuit is configured, its outputs are sent to the fitness measuring circuit which measures the evolvable circuit's fitness. Various algorithms could be implemented. For example, if the parental fitness value is higher than the child's, then the child's mutation(s) can be undone, and a new mutation is tried. Occasionally, for example, after a considerable number of mutations, portions of chromosomes could be exchanged between neighboring cells.

A lot more research needs to be undertaken on parallel implementations of GAs, so that improved Darwin Machine architectures can be conceived. The literature on

parallel GAs is still rather small (e.g. see papers in [GA 1989], [GA 1991], [PPSN 1991], [PPSN 1992]).

At the beginning of 1993, when the author switched labs from ETL to ATR, several people in the author's former ETL section were involved with evolvable hardware research. For example, Higuchi et al had successfully performed a software simulation of the evolution of a GAL 16V8 chip [Lattice 1992] used to solve the 6-multiplexor problem [Higuchi et al 1992]. What was evolved (actually the hardware evolution was simulated) was a combinatorial circuit which took two address bits to channel one of four input channels to the single output channel. As far as the author knows, this was the world's first evolved (simulated) circuit. Higuchi also simulated the evolution of a GAL 16V8 chip which functioned as a 3 bit counter (i.e. a sequential circuit).

However, even if one succeeds in building a Darwin Machine, there remains another problem, and that is how does one evolve motions of real world biots? For example, imagine that one wishes to evolve the circuits of a neural controller which sends instructions to the legs of a real world (physical) biot to make it walk. Measuring the fitness of such a creature takes a long time, namely minutes, rather than milliseconds as in a computer simulation. Thus evolving such a controller may be a painfully slow process. The size of the chromosome population would need to be very small, and techniques would need to be devised to shorten the mechanical fitness measuring time.

One way round this problem might be to simulate as closely as possible the real world biot's mechanics and evolve a "ball park" set of chromosomes for a particular motion. These chromosomes could then be ported to the real world biot (one at a time to a single biot, or in parallel to a set of identical biots) for some fitness polishing. If the simulation is reasonably "real world accurate" then this should accelerate the evolution.

Using these techniques, one can probably build up a library of motion GenNets, which could be stored in ROMs and linked to a common bus to the effectors (legs, arms etc). The real challenge in building biots, is evolving the middle decision layer. In the human brain, only a small minority of neurons are concerned with detector input and motion output. Most neurons are concerned with internal chores, such as building associations between multisensor inputs, or for storing memories, or for constructing speech, etc etc. The evolution of these internal circuits would be ideal for Darwin Machines, because it would not require the real world mechanical speeds to measure the fitnesses of motions. One could then evolve increasingly complex neural circuits to perform increasingly complex functions, but at computer hardware speeds.

## FPGAs, HDPLDs, and Hardware Accelerators

This few paragraphs present the above ideas in a little more detail. This subsection is not a presentation of completed work. Research projects based on the evolvable hardware and Darwin Machine concepts are presently underway at both ETL and at the author's present lab at ATR. (See the Postscript).

One of the aims of this subsection is to persuade more GA, ALife, and Neural Network people to get into electronics, with the intention of filling in the details, which at the present time are lacking here. It is hoped that the basic notion of "evolvable hardware", or "software configurable hardware", and hence the possibility of building genuine Darwin Machines within a year or so, will be sufficiently inspiring to motivate readers to start talking with colleagues in electronics about how to build Darwin Machines in itty-gritty detail.

There appear to be two main approaches to Darwin Machine design. One can either evolve the hardware directly, using technologies which underlie such devices as FPGAs or HDPLDs (whose characteristics will be described shortly), or one can use

special hardware accelerators, which are special hardware boxes designed to accelerate the software simulation of hardware designs containing up to several hundred thousand logic gates. To gather information on FPGAs, one might try contacting Xilinx Corporation (San Jose, CA). For HDPLDs (High Density Programmable Logic Devices), try Lattice Corporation (Hillsboro, Oregon), and for hardware accelerators, try Zycad Corporation (Menlo Park, CA). For a general overview of such devices, see [Broesch 1991].

If one uses a hardware accelerator, it might be possible to evolve at a higher level of abstraction than at individual gate level, as would be the case with FPGAs and HDPLDs. There are arguments both ways. FPGA and HDPLD chips are rather cheap now, around several tens of dollars, whereas a hardware accelerator costs tens of thousands of dollars. What one might gain in terms of flexibility of evolutionary level with a hardware accelerator, might be more than offset by its price, compared to FPGA or HDPLD gate level evolution.

We present now a more detailed description of one of the software configurable hardware devices, namely FPGAs (Field Programmable Gate Arrays). A similar story could also be given for HDPLDs (High Density Programmable Logic Devices). This description of FPGAs is provided to give the reader unfamiliar with such devices, a feel for what software configurable hardware can do. However, as will be shown shortly, these devices in their present form, are probably unsuitable to serve as a basis for evolvable hardware.

FIG. 11.6.1.3 shows a fairly typical FPGA chip architecture. It consists of three basic element types :- CLBs (Configurable Logic Blocks), IOBs (Input Output Blocks), and Interconnects. Quoting from Xilinx's technical literature [Xilinx 1991], "Like a microprocessor, the ... device is a program-driven logic device. The functions of the ... configurable logic blocks and I/O blocks, and their interconnections, are controlled by a configuration program stored in an on-chip memory. The configuration program is loaded automatically from an external memory on power-up or on command, or is programmed by a microprocessor as a part of system initialization". "Since ... FPGAs can be re-programmed an unlimited number of times, they can be used in innovative designs where hardware is changed dynamically ...".

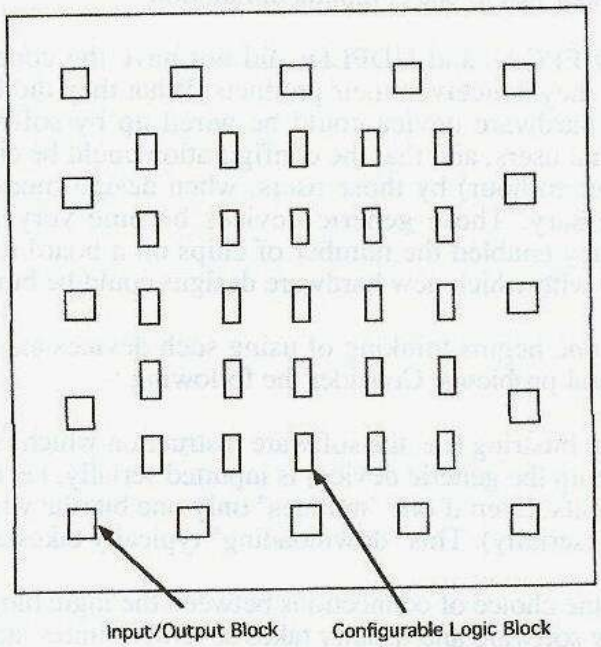

FIG. 11.6.1.3 FPGA ARCHITECTURE

The CLBs (Configurable Logic Blocks) consist essentially of "programmable combinatorial logic and storage registers. The combinatorial logic section of the block is capable of implementing any Boolean function of its input variables ... " (usually between 4 and 9 input lines). The I/O Blocks can be programmed to connect CLBs to the input/output pins of the FPGA chip. The interconnects (which run vertically and horizontally between the columns and rows of the CLBs) are used to connect the CLBs to other CLBs or to IOBs. The number of CLBs in a (Xilinx) FPGA ranges from 64 up to 900. The number of IOBs ranges from 58 to 240. The number of equivalent gates ranges from 1200 to 20,000. In the Xilinx 4000 series, the internal RAM configuration memory can range in length from 2000 to 28,800 bits. (Note, it is this configuration memory, which may serve as the chromosome for the circuit evolution).

In the case of a hardware accelerator, its input is usually in the form of VHDL (i.e. VHSIC (i.e. very high speed integrated circuit) Hardware Description Language). The accelerator then uses this description to simulate the hardware device so described.

**Evolvable Chip Research**

The section of the lab (ETL) in which the author works, recently purchased a software configurable hardware development system to help explore the concept of evolvable hardware, and for other purposes. For those interested, it was a Lattice Corporation ispLSI 1032 system. As a result of this purchase, the author became convinced that there were too many problems with such systems for them to be immediately suitable as a means to evolve hardware. The main obstacle is speed. For the evolvable hardware concept to be useful, it needs to satisfy at least two requirements. One is that the circuits evolved be both functional and too complex for human understanding, otherwise they could be humanly designed in the traditional way. The other requirement is that the circuits be evolved quickly, i.e. a lot more quickly than simulating their evolution in software using a general purpose computer, otherwise it would be easier and cheaper just to use computer simulation.

The designers of FPGAs and HDPLDs did not have the concept of evolvable hardware in mind when they conceived their products. What they did have in mind was the idea that a generic hardware device could be wired up by software, to meet the requirements of individual users, and that the configuration could be changed easily and quickly (e.g. in less than an hour) by those users, when design modifications in their hardware became necessary. These generic devices became very popular amongst electronic engineers. They enabled the number of chips on a board to be reduced, and they increased the speed with which new hardware designs could be built and tested.

However, when one begins thinking of using such devices as a means to evolve hardware, there are several problems. Consider the following :-

a) The configuration bitstring (i.e. the software instruction which is used to configure or wire up the generic device) is inputted serially, i.e. one bit at a time, for thousands of bits. Even if one "mutates" only one bit, the whole bitstring has to be re-inputted (serially). This "downloading" typically takes about 30 seconds.

b) The routing, i.e. the choice of connections between the logic blocks and the I/O blocks is done by software and usually takes several minutes, depending upon the complexity of the circuit.

c) Generating the fuse map (i.e. mapping all the routes between logic and I/O blocks to "fusible" gates, to generate the so-called "JEDEC file") takes about a minute.

d) Most of the details as to how the circuit is wired up are company secrets, so one would be unable to know which bits in the bit string corresponded to which fusible gates.

But still, this process is far quicker than the usual time taken to change a hardware design (i.e. days to weeks), but for hardware evolution, it is unacceptable. It would be quicker to simulate in software. But, the technologies underlying these software configurable hardware devices can be used to design special evolvable chips, which could be used in special architectures to build Darwin Machines.

What is needed is an "evolvable chip" which can be sent a configuring bit string (partially) in parallel, e.g. 64 bits at a time. The chip would store each 64 bit "slice" of the configuring bit string (which may be thousands of bits long) into its "chromosome register". This (partially) parallel downloading need only be done once. The downloaded chromosome then remains in the register, to be subjected to mutation. To mutate one bit of the chromosome, the chip could be sent a "mutation address" (e.g. 16 bits), which causes a single bit in the chromosome to flip. Occasionally, portions of chromosomes could be swapped between two devices.

Further work then needs to be undertaken to design a whole system of such evolvable chips. A lot of work recently has been devoted to the parallelization of Genetic Algorithms [e.g see papers in GA 1989, GA 1991, PPSN91]. The ideas contained in these papers will very probably be needed (perhaps in modified form) to build parallelized evolvable chip systems. For example, how frequently should one crossover chromosomes (a time costly operation), or should one rely largely on mutation etc. There are many possible parallel GA algorithms.

However, the real message of this subsection is that the technologies to build such evolvable chips exist today, e.g. EEPROM and SRAM. What is needed is a research effort to design evolvable chips and systems, using these existing technologies. The author would like to make an appeal that such research be initiated in many research labs around the world.

**The Remaining Challenges**

Many challenges remain, if one wishes to use Darwin Machines to evolve electronic (neural) circuits and artificial nervous systems, e.g.

a) How are evolvable chips systems to be configured so as to behave like neurons?
b) How are fitness circuits to be designed?
c) How are GenNet modules to be combined into higher level circuits?

Research work is also currently underway in other labs to software configure analog (as distinct from digital) hardware [e.g. Ning et al 1991]. This work would obviously be useful for the evolution of the author's (analog) GenNets. In time, if the concept of the Darwin Machine proves useful, software configurable hardware devices might be designed with Darwin Machine applications specifically in mind. The author's current lab (ATR in Japan) is setting up a Darwin Machine research project. (See the Postscript). We hope to be the first in the world to simulate and build a true Darwin Machine.

But, if it is not ATR which builds the world's first Darwin Machine, it will probably be ETL, thanks to Higuchi's pioneering efforts [Higuchi et al 1992]. Higuchi was the author's colleague and room mate at ETL in 1992. Higuchi

simulated the evolution of a relatively simple software-configurable circuit, having heard the author give a section talk in the summer of 1992 on the idea of evolvable hardware. The hardware circuit whose evolution they simulated, was a GAL (Generic Array Logic) 16V8A chip, (manufactured by Lattice Corporation, Hillsboro, Oregon), which is a much simpler version of an FPGA. The (simulated) circuit was used to evolve solutions to the 6-multiplexer problem (i.e. 4 binary input data channels, 2 control bits, and 1 binary output channel, where the 2 control bits determine which one of the four binary input data channels will be passed to the output channel). See section 9.9. The fitness of the evolving (simulated) circuit, was measured using a conventional software approach. If a satisfactory solution was found, it was used as the configuration instruction bitstring (of length 268 bits) for the GAL16V8A chip. This experiment showed that the idea of evolvable hardware works (at least for the chip concerned), because there was an obvious equivalence between evolving the chip directly in hardware, and simulating its hardware evolution. The obvious difference is processing speed (i.e. chip processing speed vs. software simulation speed). However, in terms of human system-development speed, it was quicker in this case to software simulate (a process which takes a few days) than to set up a hardware breadboard system (which would take months). Presumably, with greater experience, direct hardware evolution will become practical, once the supporting system tools become available.

Before finishing this subsection, the author would like to make two remarks. The first concerns the potential importance of the concepts of evolvable hardware and Darwin Machines to the electronics industry, and the second concerns the probable link which will grow between Darwin Machines and nanotechnology.

The electronics industry is one of the biggest and richest in the world, right up there with automobiles and oil. If circuits which are evolved can become more complex and more functionally useful than humanly designed circuits, then the concept of the Darwin Machine may revolutionize an industry worth many billions of dollars world wide. This prospect ought to stimulate a lot more research into what the author hopes will prove to be a very profitable idea, namely that of "evolvable hardware". It is possible that we may see "hardware evolution" or Darwin Machine "start-up" companies within a few years.

It is likely (and it is certainly the author's intention) that a link will grow between Darwin Machines and nanotechnology. (See the Postscript). Nanotechnology [Carter et al 1988, Schneiker 1989, Drexler 1992] is molecular scale engineering, where the intention is to build nano scale robots (nanots) which pick up atoms here and put them there to build any substance, including copies of the same nanots. Both the ETL and ATR labs are heavily involved in nanotech, because it is considered a critical technology that will dominate 21st century science and economics. However, building such "Avogadro Machines" (with a trillion trillion components) with demand self assembly techniques to be practical. But the complexities will also be enormous. Therefore it will be virtually impossible to predict function from structure with these self assembled Avogadro Machines. Therefore GP techniques will be needed to build/evolve them, and hence the need for molecular scale Darwin Machines. Molecular scale self assembling manufacture has been called "embryofacture" by the author. It looks as though the Darwin Machine concept will have a critically important future.