TECHNICAL REPORT NO. 494

# Using Cyclic Genetic Algorithms to Reconfigure Hardware Controllers for Robots

by

Steven D. Johnson
Gary B. Parker
Ingo Cyliax
David Braun

October 1997

*An abbreviated ("poster") version of this report was presented at the 1997 ACM/SIGDA Fifth International Symposium on Field Programmable Gate Arrays (FPGA'97), February 9–11, 1997, at Monterey, California.*

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
BLOOMINGTON, INDIANA 47405-4101

# Using Cyclic Genetic Algorithms to Reconfigure Hardware Controllers for Robots

Steven D. Johnson, Gary B. Parker, Ingo Cyliax, David Braun
Laboratory for Robotics and Analog VLSI
Indiana University Computer Science Department

ABSTRACT: Control automata for a small hexapod robot are generated by a *cyclic genetic algorithm*. From these automata a Xilinx net list is synthesized through a series of translations steps. The resulting implementation is downloaded over the communication network of an experimental robot colony. This reconfiguration of the hexapod's "nervous system" is part of a general environment for experimentation with multi-agent robotics. The process described in this article supports investigations of dynamic adaptive control.
KEYWORDS: genetic algorithm, robot, colony robotics, FPGA, adaptive control.

## 1 Introduction

This paper describes an automated process by which the control modules—realized as Xilinx XC030PC44 devices—for multiple robot agents are synthesized and down-loaded over a local communication network. The process involves a form of genetic algorithm generating locomotion control in very simple hexapod agents. Frequent redesign of agent control is part of the evolving design process. In addition, this infrastructure will be used dynamically for adaptive and hierarchical control, multi-agent robotics experimentation, and investigation of novel control technologies.

Designing a small, very inexpensive robot is central to this work. This goal imposes numerous engineering challenges, not the least of which is to discover how control distributes between individual agents and the governing command/control system. This boundary is unlikely ever to be fixed, and certainly not during early design phases. The table based FPGA configuration has proven flexible for experimenting with the on-board control we have on our simple agents. In particular, they provide high numbers of control signals and high degrees of parallelism at the expense of a moderately complicated compilation process.

Two versions of a six legged robot are driven by either nitinol wire or hobby servos. Variants of the controller contains submodules for locomotion, twelve actuator drivers, network communication, and reinitialization. During design phases, a basic downloading capability is needed to efficiently explore configurations of locomotion control. In the work described here, gait optimizations are computed off-line, with fitness evaluation through simulation. Our first experimental task is to calibrate the simulation model by measuring actual locomotion.

However, the system is destined for more dynamic use. The agents' limited local control capabilities must be reconfigured for tactical goals such as "walk straight ahead" or "find a power source." Equally important, gait control must adapt to degradation and, possibly, failure of individual actuators. Thus, reconfiguration of on-board electronics is a fundamental function for the governing system.

The input to the re-synthesis process is an automaton description generated by a cyclic genetic algorithm, as described in Section 2. Section 3 describes the host robot, which receives the re-synthesis output, as well as the system infrastructure in which re-synthesis takes place. In Section 4 we explain the re-synthesis steps,

In related research, Thompson uses FPGAs in a genetic programming process [8]. A genetic algorithm mutates a configuration bit stream for an XC6200 until it implements an oscillator for a particular bit rate. This approach can be viewed as generating a program, where our GAs are generating a dedicated design. In prior work on robot control, Thompson uses a fixed-hardware sequencer to emulate a net list evolved by software [7]. The hardware is evolved more directly, whereas the process we describe produces a finite-state machine implementation from which a net list is synthesized. Although Thompson's controllers are simpler—2 motors versus 12 actuators—it they implement a reactive function for obstacle avoidance; the controllers we generate do not have sensor feedback.

## 2 CGAs for hexapod gait optimization

Genetic Algorithms (GAs) are based on the laws of natural selection and survival of the fittest [4]. The basic algorithm consists of three operators: selection, crossover, and mutation, that transform a randomly generated population into a near-optimal one. The individuals of the population are possible solutions to a problem, usually represented as fixed-length strings of bits called *chromosomes*. The *genes* of the chromosome are substrings that characterize properties, or *traits*, of the solution. The survivability of the individual depends on how a fitness measure compares with the other individuals in the population.

*Cyclic genetic algorithms* (CGAs) adapt basic GAs to develop cycles of sequential instructions [5]. The genes of the chromosome are tasks that are to be completed in a set amount of time. A typical chromosome consists of an entry segment (for transition from rest to activity) an iterative segment (for cyclic activity), and an exit segment (for transition back to a rest state).

Chromosomes contain global inhibitors and coordinators. For example, in our gait controllers, an inhibitor prevents tandem legs from doing the same thing at the same time; a coordinator works on a single leg, ensuring it is down whenever it moves backward. These constraints are learned by the CGA during training; no advanced knowledge of how they should be applied is given.

The genetic operators used in CGAs are standard. Selection is based stochastically on fitness which is, in our case, determined by the distance traveled by a robot using that individual's controller. Crossover and mutation take place either between the genes or internal to the genes of the individual. A non-standard additional operator is the gene-by-gene evaluator that determines the fitness of each gene sequentially and eliminates those that show marked decrease in performance from their predecessors.

Figure 1 depicts the chromosome generated by a CGA for forward motion on a level surface. We draw it as a transition system—actually it is just a sequence of 16-bit strings—whose states show the status of the 12 actuators, one horizontal and one vertical for each of the six legs. The number in the center is the duration of that state. For the Stiquito robot (described in the next section) the full leg thrust takes 10 time units. The CGA discovered that thrusting only 9 units made the movement smoother because the time to lift then drop the leg takes 9 in the current thrust cycle and would just hit the ground before the alternate thrust cycle. It sacrifices using its full throw to get better efficiency in the cycle. The CGA eliminated entry and exit segments as unnecessary for this gait.

## 3 The colony framework and robot hosts

*MARE (MultiAgent Robotic Environment)* is an infrastructure under development for experimenting with multi-agent robotics. It is a tool set for

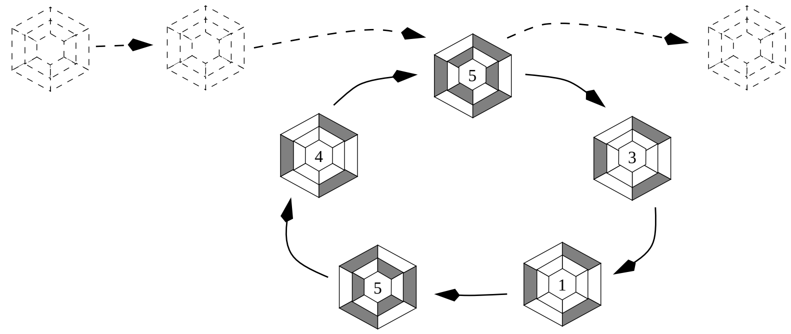


Figure 1: CGA generated chromosome for the Stiquito Hexapod

exploring the coordinated use of many small robotic agents of very limited capability to perform larger functions. It is also an environment for investigating novel control technologies, collective behavior, and neuroscience, as described elsewhere [1]. Currently, MARE serves as a laboratory for refining individual robot agents, where the principal design goal is to develop inexpensive, relatively simple, and easily reproducible robots. There are three levels in the system's structure, as depicted in Figure 2.

The *user level* is an application domain for experiment design and execution. It monitors the global status of the colony, schedules tasks, and provides virtual capabilities not physically present on the agents, such as positional awareness. The user level also contains a simulation environment, providing, among other data, a visualization of aggregate behavior. However, fitness evaluation for the CGA was not done using this simulator, but with a special purpose model.

The *colony level* is the command and control layer. It implements communication and positioning subsystems and serves as the interface between the agents themselves and the user level. It hides the low level characteristics a particular agent from the user's view, when that is desired. However, it also provides access to those details when the work requires it.

The *agent level* consists of the on-board electronics of each agent. For the two agents described below components at this level include a Xilinx 3030, an EEPROM, and a Xilinx communications pod. Embedded in the Xilinx chip are functional modules for communication and control (fig. 4).

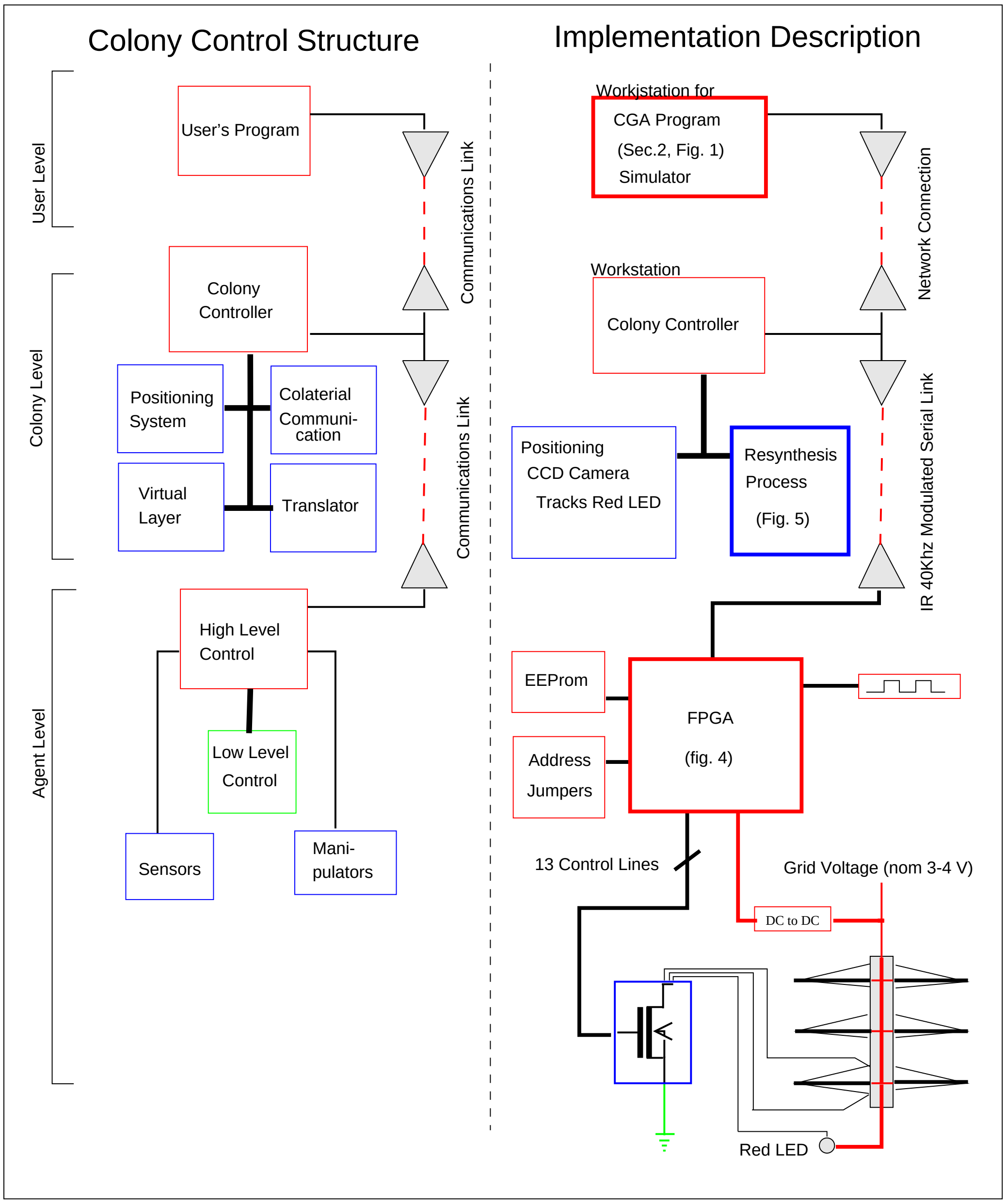


Figure 2: Structure of the colony

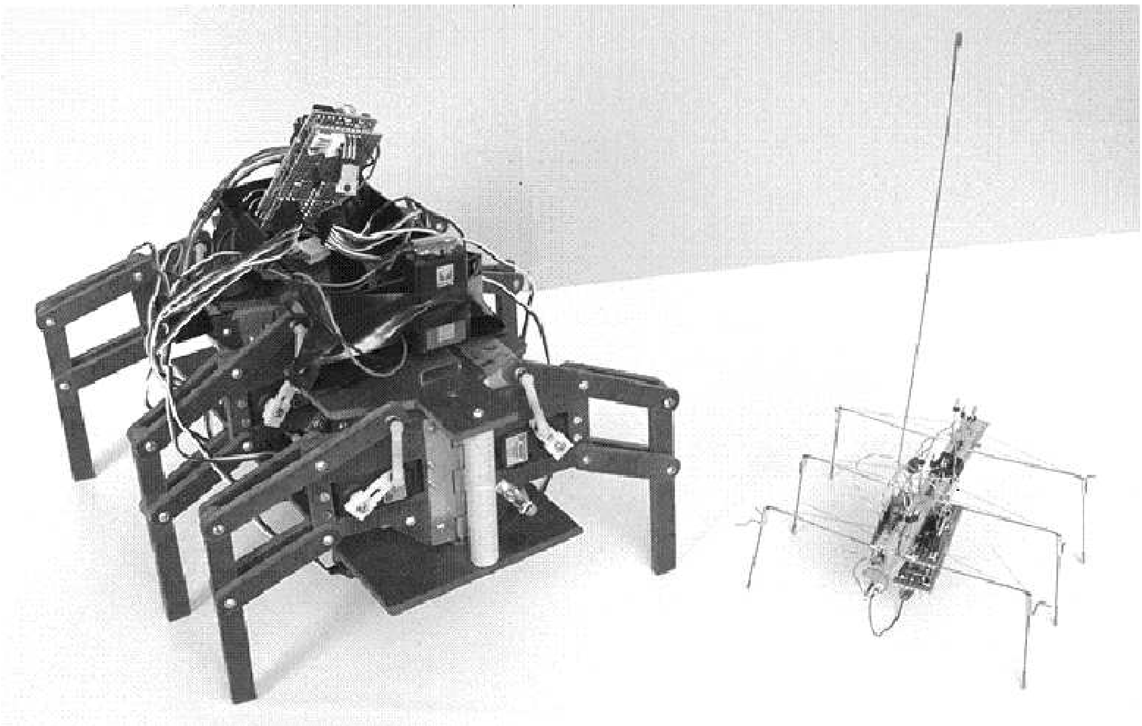

Figure 3: Servobot (left) and Stiquitto II (right)

### 3.1 The agents

*Stiquito II* (TM) is a nitinol actuated robot developed for the low-cost study of hexapod robots in a colony setting [2]. The robot is constructed from spring steel wire, aluminum tubing and polystyrene plastic. The wire is bent to form the legs for a plastic body. The aluminum tubing is used to attach the nitinol 'muscles' to the body and legs. Stiquito's mechanical structure imposes significant constraints on the on-board controller module's size (two inches by six inches) and weight (40–60 grams).

Standard digital signals are used to gate a FET driving the nitinol actuators (Fig. 2). Heat resulting from current flow causes the nitinol wires to contract, moving the legs. When cool, the spring-wire legs return to a neutral position, stretching the nitinol as they do. Within the control module, the individual leg drivers use a variable duty cycle to moderate power to the actuators.

Since the Stiquito II has a very limited payload capacity, we use a grid system to supply power. The system is similar to the way bumper cars at an amusement park are powered: the robots have contact brushes connecting them to a copper plate that serves as the floor of the colony and a metal screen is suspended above the floor.

The control module needs I/O ports to drive at least 12 actuators, various sensors, and a communications interface. In addition to flexibility, I/O pin capacity and cost were significant factors in the choice of FPGA technology for the module.

The *Servobot* prototype uses hobby servo actuators in place of nitinol. Dimensions of Servobot are roughly four times those Stiquito II. It and can carry much more weight, including its own power supply. It is also considerably more rugged and forgiving, making it a good host for preliminary experimentation. By intention, the control architecture is similar to that of Stiquito II, the difference being in the leg driver interfaces.

The leg drivers for the Servobot produce a pulse width modulated signal driving the hobby servos, which expect a pulse repetition rate of 20ms. A pulse width of 1.0 to 2.0 ms is linearly mapped to a angular motion of $-45°$ to $45°$. The angular motion of the servos translates to nonlinear motion in the legs, allowing the servo to supply more torque at the extents of the servo rotation, hence maximal force at the extremes of the leg motion. The advantage is that when the leg is down, the weight of the robot is on the servo shaft and not producing a torque against the servo.

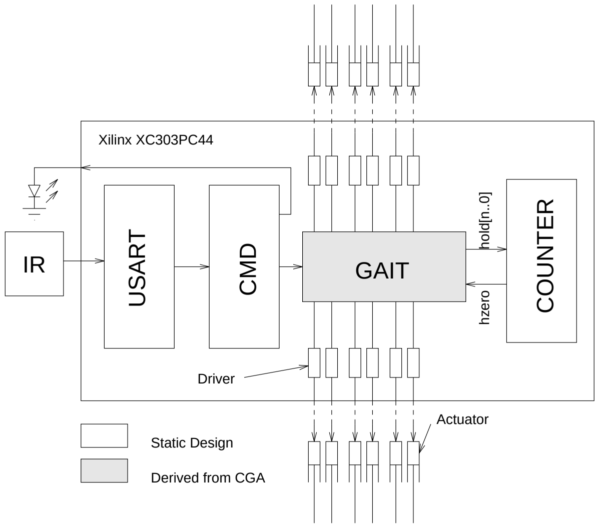


Figure 4: Top-level design of the FPGA controller module

## 4 CGA to robot controller design flow

Figure 4 shows the sub-architecture of the controller as implemented on the FPGA. In addition to gait control, the principal submodules include clock distribution and pre-scaling, actuator drivers, I/O buffers, and a USART to interface with an infra-red receiver.[1] The redesign process described in this paper affects only the gait control. We refer to the other unmodified components as the *static architecture* of the controller.

The chromosomes generated by the CGA are compiled into a hardware description which is merged with the static architecture. The aggregated design is synthesized using Xilinx mapping software. The resulting implementation is down-loaded over the communication channel into the controller hardware. Figure 5 shows this design flow and names the primary functions, as described next.

The chromosome evolved from the CGA has a series of genes (nodes), each containing the leg activation state and a repetition count. The chromosome is interpreted as a state transition system with an initial sequence followed by a cycle, as depicted in Figure 1. As suggested by the figure, the chromosome

[1]For the experimentation described in this submission, a direct wire tether is used for communication with the agent. This is to be replaced by an IR based wireless network for colony communication. If completed in time, this work will be described in the final article.

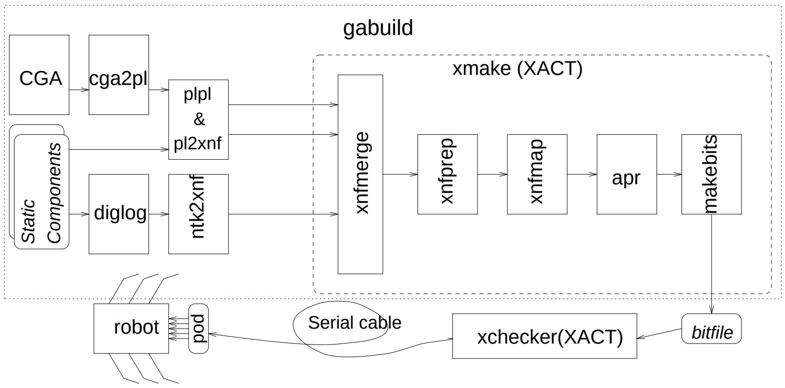


Figure 5: Overview of design flow

is implemented as a finite state machine, control-state numbers are allocated sequentially. Transition into a state pre-loads a count-down timer with the duration of the state. When the counter reaches zero, a transition is enabled.

Activation is implemented by a digital signal presented to the actuator driver. For both Stiquito and Servobot, there are two activation states from the gait generator: forward (down) motion and return-to-rest. Based on this signal, the agent-specific driver performs the signal conditioning for the given actuator, as described in Section 3.

# 5 Design flow

Figure 5 shows the information flow between the CGA output and agent controller. The cga2pl function translates a representation of the chromosome to an FSM described in the PLPL hardware description language [6]. PLPL is a vintage HDL offering enough features to generate state machines of medium complexity. Figure 6 shows a fragment of the resulting syntax. Omitted from the listing are signal declarations and intermediate states of the control FSM and counter. Since PLPL is a tool for designing PLDs, the FSM hardware description is mapped into sum-of-products form and optimized by the PLPL system.

Another translator, pl2xnf, takes the minimized sum of products representation into a gate level description in Xilinx's net-list format, XNF. This *gait* submodule is then merged with the components of the static architecture, *ctr4* and *pads*.

The top-level controller design, *ctr4*, was created using Diglog, a schematic-

```
... CASE (state[2:0])
    BEGIN
    0) BEGIN                                  5) BEGIN
       IF (ena * hzero) THEN                     IF (ena * hzero) THEN
         BEGIN                                     BEGIN
         state[2:0] = 1;                           state[2:0] = 0;
         hold[2:0] = 4;                            hold[2:0] = 0;
         END;                                      END;
       ELSE                             ...      ELSE
         state[2:0] = 0;                           state[2:0] = 5;
         r1v = 1; l1h = 1; r2h = 1;              END;
         l2v = 1; r3v = 1; l3h = 1;
       END;

                                                 END;


... CASE (hold[2:0])
    BEGIN
    0) BEGIN                                  4) BEGIN
       IF (/ena) THEN hold[2:0] = 0;             IF (ena) THEN
       hzero = 1;                       ...        hold[2:0] = 3;
       END;                                      ELSE
                                                   hold[2:0] = 4; hzero = 0;
                                                 END;
```

Figure 6: Fragments of the PLPL design for the gait FSM shown in Figure 1

capture editor and simulation system developed by Caltech [3]. A library of Xilinx compatible primitive gates was developed, together with a translator from Diglog's NTK net-list format to XNF. This is ntk2xnf in figure 5.

The IO buffer and pin assignments are implemented in a separate PLPL submodule, *pads*, in order to facilitate re-targeting to a different FPGA. This module is processed in the same manner as the *gait*. Although pin assignment is known to interfere with routing in large FPGAs, this has not yet become a problem in our controller.

Once the design has been merged and flattened by xnfmrg, the tools in Xilinx's XACT environment are used to map the design into a Xilinx FPGA configuration bit-stream. We are using a XC3030PC44 in our controller because of it's small (44 pin) PLCC package size and because this family of parts also has an internal crystal oscillator, which we use to drive a ceramic resonator at 480Khz to get the bit clock for the USART and the global clock used throughout the FPGA.

Once the design has been placed and routed, the bit-file is downloaded using the Xchecker serial download cable and pod provided by Xilinx [9]. This cable interfaces with a Sun Unix Workstation and a programming jumper on the robot controller PCB in Figure 5.

## 6 Summary and directions

We are working toward developing means to adapt basic functions in the presence of stress. In a working colony, each robot would have a gait tuned to both it's tactical goals and physical characteristics. As the performance changes due to damage and aging, the colony-level control might download a new gait to compensate. CGAs have been shown to be effective at producing and altering gaits in response to degraded capabilities in the robot [5]. However, we do not yet know whether this approach to performance tuning can be accomplished in real time or, instead, must be done off-line using a simulator for fitness evaluation.

A hexapod walking gait is a complex activity. We are also looking at CGAs to optimize control at finer granularities. A single leg, or an individual actuator driver might be tuned to compensate for degradation.

Two successive control states in Figure 1 have identical configurations. Actually, the underlying genes differ, but the coordinators and inhibitors make them look the same. Merging the states improves the FSM representation but does not change the efficiency of the gait; hence, there is no selection pressure to evolve. We would expect to make such implementation optimizations during the re-synthesis process.

Dynamic reconfiguration over the colony network is in development. We are looking at IR modems, RF modems, and signaling over the power grid. The approach we take is a three-step process of downloading the new configuration to the on-board EEPROM, verifying its consistency, and finally causing a re-initialization event to reload the FPGA. For the family we presently use, the total image is 2778 bytes and the dynamic portion of the configuration is roughly 40 per cent of this image. A download takes 92 seconds over our extremely inexpensive 300 baud IR channel. At 9600 baud it would be about 3 seconds, but even this throughput is marginal for colony applications.

We are, of course, looking at partially reconfigurable FPGAs. The more interesting engineering question is whether we should convert to a static chromosome interpreter, and thus eliminate the entire re-synthesis step. With four bits per gene to represent the next-state function, only 100 bits are needed to transmit the information content of Figure 1. This solution is inappropriate at this stage of our work, because both the mechanical and hardware architecture of the agent is still evolving. Since we will also be reconfiguring actuator drivers, communication drivers, and other submodules of the controller, we believe that resynthesis is more flexible, and ultimately has comparable throughput characteristics. Should the robot design become more stable, it may be that communication overhead can be reduced by re-programming rather than re-synthesis.

## 7 Acknowledgments

This research was supported, in part, by the National Science Foundation under grants GER93-54898 and MIP92-08745. We thank D. K. Bhatia for pointing out [8]. We also are grateful to Xilinx Corperation for their donations of software and hardware components.

# References

[1] Indiana University Analog VLSI and Robotics Laboratory. Home page `http://www.cs.indiana.edu/robotics/avlsi.robotics.html`.

[2] James M. Conrad and Jonathan W. Mills. *Stiquito: The Design and Implementation of Nitinol-Propelled Waking Robots.* IEEE Computer Society, 1996.

[3] Dave Gillespie and John Lazzaro. *Chipmunk Toolset.* Physics of Computation, Caltech.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[5] Gary B. Parker and Gregory J. E. Rawlins. Cyclic genetic algorithms for the locomotion of hexapod robots. In *Sixth International Symposium on Robotics And Manufacturing*, 1996. World Automation Congress, Montpellier, France, June 1996.

[6] Programmable Logic Division, AMD/MMI. *Programmable Logic Programming Language*, 1987.

[7] Adrian Thompson. Evolving electronic robot controllers the exploit hardware resources. In *Proceedings of the 3rd European Conference on Artifical Life (ECAL95)*, 1996. Springer 1995.

[8] Adrian Thompson. Silicon evolution. In *Proceedings of Genetic Programming 1996 (GP96)*, 1996. MIT Press 1996.

[9] Xilinx, Inc. *Hardware & Peripherals Guide*, 1994.

## A Poster materials

An HTML summary of this paper can be accessed at

`http://www.cs.indiana.edu/hmg/FPGA97/Main.html`

## B Data files

Data files supporting this report can be found at:

`ftp://ftp.cs.indiana.edu/pub/hmg/tr494.tar.Z` is a compressed archive of data sources.

`ftp://ftp.cs.indiana.edu/pub/hmg/tr494.zip` is a compressed archive of the data sources in an alternate format.

`ftp://ftp.cs.indiana.edu/pub/goo/PLD/plpl.tar.Z` is a compressed archive of PLD assembler used.

These data will be maintained for at least one year after the publication date of this technical report. If there are problems accessing the data, or other information is needed, please contact the first author.

The following is a general description of the archived data.

**Static Design (XNF netlists)**

`Data/ctr4.xnf` – module for the basic contrller architecture
`Data/pads.xnf` – IO pads for the specific FPGA implementation

**Programs**

`Data/cga2pl.c` – this program converts the CGA into a HDL description
`Data/gabuild` – script to build a CGA derived object

**CGA Data** – various gaits that have been tested

| | | |
|---|---|---|
| Data/cga-exact10 | Data/cga-exact10-5 | Data/cga-exact5 |
| Data/cga-exact5-2x | Data/cga-round10 | Data/cga-round10-5 |
| Data/cga_g2_0 | Data/cga_g2_10 | Data/cga_g2_100 |
| Data/cga_g2_1000 | Data/cga_g2_200 | Data/cga_g2_2000 |

| | | |
|---|---|---|
| Data/cga_g2_500 | Data/cga_g2_5000 | Data/cga_m_100 |
| Data/cga_m_2000 | Data/cga_m_500 | Data/cga_m_5000 |
| Data/cga_m_5000r | | |