An FPGA Dynamically Reconfigurable Framework for Modular Robotics

Andres Upegui, Rico Moeckel, Elmar Dittrich, Auke Ijspeert, Eduardo Sanchez
Ecole Polytechnique Fédérale de Lausanne – EPFL
Logic Systems Laboratory - LSL
(andres.upegui, Rico.Moeckel, auke.ijspeert, eduardo.sanchez)@epfl.ch, elmar.dittrich@web.de

Abstract

Dynamic Reconfiguration has always constituted a challenge for embedded systems designers. Nowadays, technological developments make possible to do it on Xilinx FPGAs, but setting up a dynamically reconfigurable system remains a painful and complicated task. In this paper we propose a framework for performing it in an easy way, for a specific application: Modular Robotics. We propose an architecture containing a Microblaze processor and a reconfigurable module. The module is defined in VHDL and synthesized by the user; then we provide the scripts for easily generating the corresponding configuration bitstreams for a dynamic partial reconfigurable controller for our Modular Robot. The proposed framework is easily extendable to other applications.

1 Introduction

1.1 Self-Reconfigurable Machines

A Self-Reconfigurable Machine is a machine that has the possibility to modify its own hardware configuration. This feature provides an enhanced flexibility that intends to reduce product and computational cost – defining computational cost C in terms of power consumption P and execution time T by the equation $C = aP + (1-a)T$ where a is a trade-off term for giving more importance to P or T, given the application. These reductions would be mainly achieved by two facts:

- **Reusability**: The same hardware substrate allows any number of functionalities without increasing chip area, just by reconfiguring the hardware.
- **Power Consumption:** For low power applications, specialized low power circuits can be loaded when required. Additionally, the reusability should avoid consumption in unused circuits.

Our platform intends to provide the possibility of self-configuring a system implemented on an FPGA, taking advantage of the Dynamic Partial Reconfiguration (DPR) property from Xilinx FPGAs. The proposed reconfiguration is module based, providing the possibility of self-reconfiguring a full peripheral (or several of them), a full processor, the full system, or just some components of a peripheral. For our purposes a peripheral can be a neural network, a fuzzy controller, or a coupled oscillators set. Several algorithms implementations on FPGAs have been documented as being faster than on processors, while for power consumption just some case studies have demonstrated that FPGAs can be less energy demanding [1].

The reconfiguration is driven by an embedded soft-processor as shown in figure 1. This processor loads partial bitstreams from a bitstream repository via wireless communication, and stores them on an on-board memory; previous works on wireless sensor networks platforms [2, 3] provide an appropriate low power framework. This processor partially reconfigures the system looking for minimizing the computational cost, given a set of operations to be performed for a given task. Operations should be executable by hardware or software as described in [4]; the choice should be done according to the expected computation cost. Future developments on FPGAs technologies should allow a pipelined reconfiguration as described in [5], thanks to reductions in reconfiguration latencies, or storage of backplane bitstreams.

The applicability of Self-Reconfigurable Machines extends to diverse fields such as Wearable Computing [1], Wireless Sensor Networks [6], and Modular Robotics as described in this paper. The first board prototype is described in section 3. This board provides the requirements needed to implement our Self-Reconfigurable Machines. It supports DPR as well as the implementation of a soft-processor, taking into account a set of constraints specified for these types of designs. It provides also a Bluetooth interface for allowing access to the bitstream repository

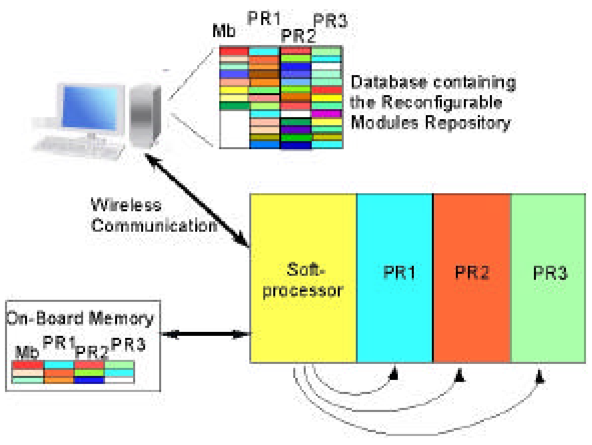

Figure 1 Self-Reconfigurable Machine schema.

1.2 Modular Robotics

A modular robot can be defined as a robotic system consisting of a set of discrete components (which we will call units) that can be assembled in different ways to obtain diverse shapes such as snake, quadruped, biped, or hand-like robots. Each one of these units should be autonomous, and should have the possibility "to do" something independently from other units (i.e. to move, to sense, to compute …).

Self-reconfigurable modular robots arise as a great engineering challenge. These systems are composed of homogeneous or heterogeneous components and have the ability of self- assembling or disassembling multi-unit structures, configuring their shapes without human intervention. This feature allows also the robot to self-repair in case of a unit failure by replacing it. [7, 8]

This special field of robotics holds many interesting challenges in fields as diverse as mechatronics, MEMS (Micro Electro-Mechanical Systems), smart actuators, distributed control, autonomous strategies, learning algorithms, ad-hoc networks, nanotechnologies, bio-inspired systems, etc.

One can foresee that a system with such degree of mechanical flexibility can largely benefit from the logic flexibility offered by self-reconfigurable machines. Different types of controllers and functionalities could be tested for different robot shapes, and it can be interesting for exploring locomotion controllers [9], bio-inspired architectures [10], learning algorithms, evolutive controllers [11, 12], etc.

In our case, we are particularly interested in the adaptive control of movement and locomotion in the multi-unit structures. The units described in this article will be used to implement adaptive control of locomotion based on the biological concept of central pattern generators (CPGs) [9]. CPGs are neural networks capable of producing coordinated oscillatory signals without any oscillatory inputs. The CPGs for swimming and walking can be simulated using neural network models or coupled oscillator models [13]. CPGs are an interesting concept for modular robotics because of their distributed nature (they are made of multiple coupled oscillatory networks) and because of their robustness against perturbations and lesions. In our implementation, movements of each robotic unit will be controlled by one or several nonlinear oscillators which will synchronize with their neighbors through coupling connections implemented with the Bluetooth communication protocol.

In this paper we present a framework for performing DPR for a modular robot controller. However, discussions about hardware implementations of any specific controller are beyond the scope of this paper. Anyway, related work is described in [10, 12]. In section 2 we describe our robot mechanical characteristics, section 3 describes the electronics, section 4 gives an introduction to DPR, in section 5 we propose our framework including the system architecture and the bitstreams generation, and finally section 6 concludes.

2 YaMoR, our Modular Robot

One of the essential features of YaMoR (Yet another Modular Robot) [14] is its simple and low cost mechanical design (figure 2). Each unit has a one-degree of freedom rotational actuator. The body of the element can be described as a block with a round shaped side. For connecting and transmission of the torque, a pivotable u-shaped aluminium profile is connected with the actuator.

The support structure of YaMoR is composed of two layers of printed circuit board in a sandwich-like design. The motor is positioned perpendicularly between the two layers and fixed directly to the lower layer. The actuator is a standard rc-servo, an off-the-shelf motor with integrated position controller. To operate the units we use two Li-Ion batteries, which theoretically allow the unit to work up to 45 min depending on the load.

YaMoR units are simple and robust, but require manual reconfiguration due to passive connectors. The problem with connectors in Modular Robotics is that they should be reliable and must withstand high forces and on the other hand they should be compact and easy to open or attach. For our first experiments we used a combination of strong Velcro fasteners and screws to support the structure at heavy load.

Because of the genderless characteristic of the used Velcro we can interconnect any surface onto another. Interesting properties show up when connecting the alu-levers of two units. In this way we can obtain configurations similar to M-TRAN [7]. The first experiments showed usability for achieving different types of locomotion like crawling, hopping and snake- or inchworm-like locomotion. (See movies in [14])

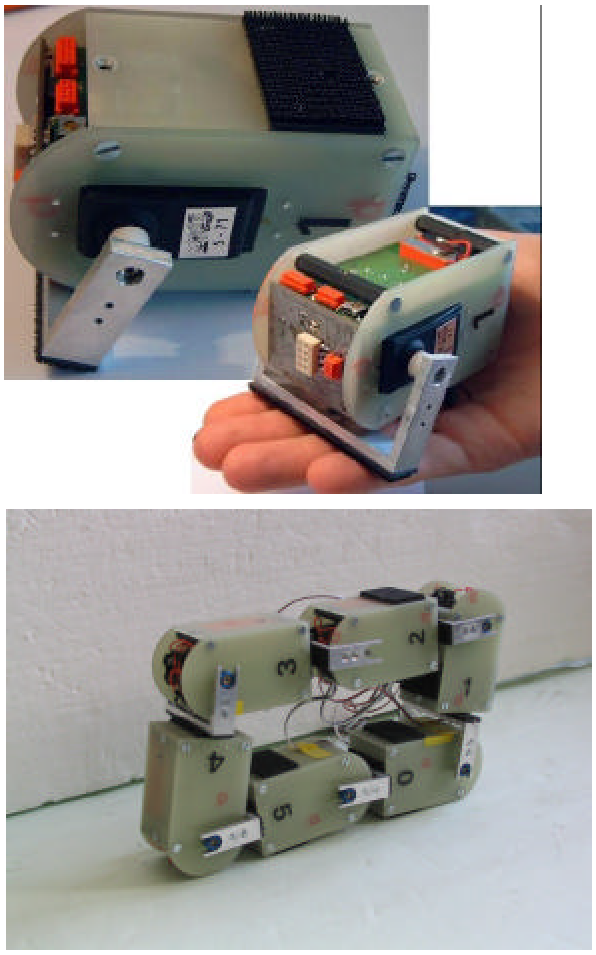

Figure 2. YaMoR robot: A single unit and a rolling track configuration.

With the help of some passive connection element (like in the interesting configurations of Polybot [8]) one can easily build up some pedal structures (like spiders or reptiles). Last but not least, with a minimum of six YaMoR units one can configure a rolling track, which turns its construction wheel-like on the ground (figure 2).

3 The Hardware platform

YaMoR includes two separated control boards: one board containing a Bluetooth-ARM System and one carrying a Spartan-3 FPGA. Furthermore there is a service board containing power supply and battery management inside each robot unit. This architecture with distributed electronic components gives a flexible solution where the pins of the microcontroller and the FPGA can be connected together as necessary or one of the boards can be left out or be replaced if useful.

3.1 FPGA board

The main electronic component of this board is a Spartan-3 XC3S400 FPGA with 400.000 gates meeting most requirements for reconfigurable hardware. The FPGA board supports two different reconfiguration modes: Slave Serial and Boundary Scan (JTAG). It supports partial reconfiguration and Microblaze implementations. The FPGA board also contains a 4 MBit high speed SRAM directly connected to the Spartan-3. Because for modular partial reconfiguration inside a Xilinx FPGA one may divide the device in columns, we respected the constraints for connecting the SRAM to the pins of the Spartan-3 so that a MicroBlaze can take advantage of the SRAM while using the partial reconfiguration feature (See board in figure 3).

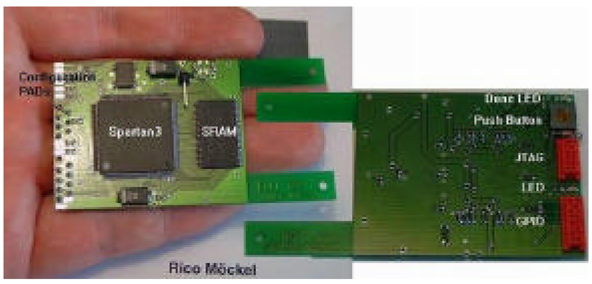

Figure 3. FPGA Board

The FPGA is directly driven by a 50 MHz oscillator. However, the Digital Clock Manager included in the Spartan-3 FPGA allows modifying the internal clock frequency. General purpose Input-Output pins distributed around the FPGA are accessible through micromatch connectors on the PCB for debugging purposes. A push button allows the implementation of a reset or test input.

3.2 Bluetooth board

The second control board inside the YaMoR unit includes a Bluetooth-ARM System on Chip (SoC) driven by a 12 MHz clock. The ARM containing the embedded Bluetooth stack also allows running user defined code e.g. for reconfiguring the FPGA via Bluetooth or for doing the partial reconfiguration of the FPGA. Communication between FPGA and ARM can be established through the flexible connection system and an UART. It is possible to send commands from a Microblaze implemented on the Spartan-3 to the ARM to ask the ARM to perform a reconfiguration of the FPGA. The necessary bit file for the FPGA configuration and the program code for the SoC are stored inside a 16 MBit Flash memory on the microcontroller board.

3.3 Power supply board

The third electronic board is a service board containing the power supply for the other electronics: Three high efficiency step-down converters produce the different voltages necessary to drive the motor and the electronics of each unit. Furthermore, the power supply board contains the components used for battery management like protection from discharging the batteries too much and a battery charger.

4 Dynamic Partial Reconfiguration

FPGAs are programmable logic devices that permit, by software reconfiguration, the implementation of digital systems. They provide an array of logic cells that can be configured to perform a given function by means of a configuration bitstream. This bitstream contains the configuration information for all the internal components. Some FPGAs allow performing partial reconfiguration, where a reduced bitstream reconfigures only a given subset of internal components. Dynamic Partial Reconfiguration (DPR) is done while the device is active: certain areas of the device can be reconfigured while other areas remain operational and unaffected by the reprogramming [15]. For the Xilinx's FPGA families Virtex, Virtex-E, Virtex-II, Virtex-II Pro (applicable also for Spartan-II and Spartan-IIE) there are two documented flows to perform DPR: Module Based and Difference Based.
With the Difference Based flow the designer must manually edit low-level changes. Using the FPGA Editor the designer can change the configuration of several kinds of components such as: look-up-table equations, internal RAM contents, I/O standards, multiplexers, flip-flop initialization and reset values. After editing the changes, a partial bitstream is generated, containing only the differences between the *before* and the *after* designs. For complex designs, the Difference Based flow results inaccurate due to the low-level edition in the bitstream generation.
The Module Based flow allows the designer to split the whole system into modules. For each module, the designer generates a configuration bitstream starting from an HDL description and going through the synthesis, mapping, placement, and routing procedures, independently of other modules. Placement and timing constraints are set separately for each module and for the whole system. Some of these modules may be reconfigurable and others fixed (see figure 4). A complete initial bitstream must be generated, and then, partial bitstreams are generated for each reconfigurable module.
Hardwired Bus Macros must be included. These macros guarantee that each time partial reconfiguration is performed the routing channels between modules remain unchanged, avoiding contentions inside the FPGA and keeping correct inter-module connections. On the same way, the Modular Based flow impose some placement constraints: (1) the size and the position of a module cannot be changed, (2) input-output blocks (IOBs) are exclusively accessible by contiguous modules, (3) reconfigurable modules can communicate only with neighbor modules through bus macros (See Figure 4), and (4) no global signals are allowed (e.g., global reset), with the exception of clocks that use a different bitstream and routing channels [15].

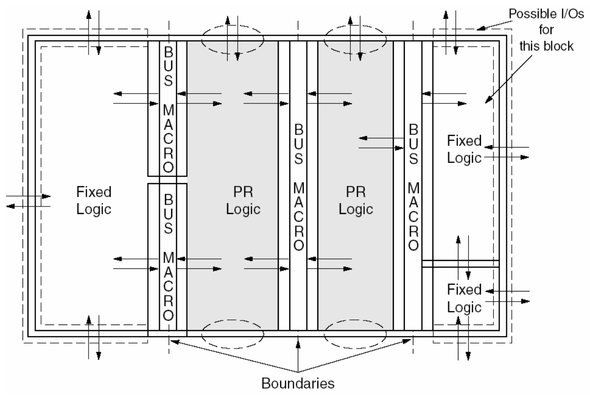

Figure 4 Design Layout with Two Reconfigurable Modules. (From [15])

5 Reconfigurable Controllers

The process for implementing partially reconfigurable designs in Xilinx devices is a task that remains painful and complicated for FPGA designers (even for experts!!); our goal is to provide the framework necessary to profit from the advantages that DPR offers for Self-reconfigurable Machines. We present here a basic architecture, and the scripts needed for allowing a non-expert designer, with just some knowledge on VHDL, to implement his own reconfigurable controller for YaMoR.
The approach consists on proposing an initial structure, application-dependant. In this case the system will not offer the maximum flexibility, but just the flexibility that should be useful for a given application and for a given board. In this way we provide user-friendliness under the cost of losing unneeded flexibility. It is clear that the concept of "unneeded flexibility" remains very subjective and nobody can determine what kind of structure is the most appropriated for a given application.

Our modular robot control unit disposes of a hardware-software platform. The user can describe the whole controller in software, to be run on a soft-processor, getting rid of all the hardware stuff. Or he can also describe his own hardware peripherals, having the possibility to replace them in a dynamical way, for reducing power consumption or execution time as discussed in section 1.1.

5.1 System Architecture

Our basic architecture contains two modules (see figure 5). The first one is a fixed module that mainly contains a Microblaze processor from Xilinx [16], featuring a RISC architecture with Harvard-style, separate 32-bit instruction and data busses running at full speed to execute programs and access data from both on-chip and external memory. The second one is a reconfigurable module, allowing the implementation of the user defined logic: the robot controller.

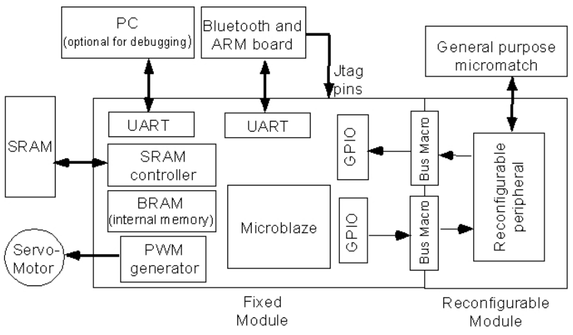

Figure 5. Reconfigurable controller.

Fixed module: The fixed module contains a Microblaze processor including some peripherals: 2 UART ports – one for communicating with the Bluetooth chip, and the second one for monitoring from a PC –, a PWM generator for controlling the servomotors, two 32 bits general purpose input-output (GPIO) for interfacing with the reconfigurable module, and the necessary peripherals for memory managing: external SRAM controller and internal block RAM (BRAM) data and instructions controllers.

Reconfigurable Module: The reconfigurable module is connected to the Microblaze GPIOs through a Bus Macro, making the module content a peripheral. This module can also access four external pads, connected to an external micromatch, allowing a direct interface of the reconfigurable module from outside the FPGA for debugging purposes. This module allows reconfiguring a peripheral while keeping the processor core. Different kinds of hardware controllers are interesting for us, namely: non-linear oscillators, neural networks, and fuzzy logic.

5.2 DPR on Spartan-3 devices

DPR is supported for Virtex families (E, II, II-pro, IV), however, even if the Spartan families (E, II, II-E, 3) can be partially reconfigured the dynamic feature is not supported – i.e. the FPGA can be partially reconfigured, but the unaffected logic is disabled. This limitation on Spartan-3 FPGAs does not allow reconfiguring modules directly by the soft-processor contained in the FPGA as described in 1.1. Instead of this, the partial reconfiguration, as well as the initial configuration, is done by the ARM microcontroller. Future robot versions using FPGAs that support dynamic reconfiguration should allow doing it directly.

As explained in section 4, Modular Partial Reconfiguration requires Bus Macros. However, Xilinx does not provide them for Spartan-3 FPGAs. That is the reason why we designed our own Bus Macros. Bus Macros are usually implemented with internal 3-state buffers, with the goal of guaranteeing a fixed connectivity among modules for every reconfigurable module. Instead of 3-state buffers (not available in Spartan-3) we used slices' LUTs. We provide two bus macros: one for signals going from left to right, and another one for the inverse. Specifically, for the Spartan-3 XC3S400 it allows a maximum bus macro width of 132 bits.

Additionally, the Modular Based flow on the Spartan-3 family is currently not documented and supported by Xilinx. The bitgen tool (bitstream generator) does not allow generating partial bitstreams for Modular Based designs, but just for Difference Based designs. For dealing with this problem we use the Difference Based bitstream generation for emulating the Modular Based one by executing the following steps: (1) Assembling of a complete design for each possible configuration of the system – i.e. a full system including fixed and reconfigurable module. (2) Generation of partial bitstreams containing the difference between the initial system and each one of the remaining configurations and vice versa. (3) Configuration of the FPGA with the initial bitstream. (4) For loading a module, we download the partial bitstream containing the difference between the initial and the second system. (5) Before downloading a new partial bitstream containing a third system, unlike in regular Module Based flow, we must come back to the initial system, since directly downloading it may result in internal contentions. If the number of possible systems is not very large it would be possible to generate the partial bitstreams for switching from any system to any other one, for avoiding to come back to the initial configuration.

Given that the proposed reconfiguration is Difference Based, it would be possible to get rid of all the Modular Based flow, and it would still be correct. However, doing that would dramatically increase the size of the partial bitstream, since the modular flow ensures that the difference bitstream will just contain the reconfigurable module, keeping unchanged the microprocessor module.

5.3 Bitstream Generation

As stated before, the bitstream generation uses to be complicated. Given the huge complexity of FPGAs configuration bitstreams, Xilinx tools are still not very well debugged and for each reconfigurable design you have to deal with lots of incomprehensible error messages. A given system working properly on a given FPGA can generate errors when changing the FPGA or when modifying the system. Usually, these errors can be solved by "finding alternative paths": dealing with placements constraints, with tools options, or manually placing and routing components.

Given that we propose a base architecture for a specific application and hardware platform and that we have already solved several problems, it will save a lot of time to the user if he has not to solve them again. We provide a set of scripts that deals with the problems. However, it is clear that for complex reconfigurable modules new problems should appear.

Some of the proposed scripts are:

create_project n: creates the required directory structure, for a number *n* of reconfigurable modules, and copies the required initial files – user constraint file, bus macros, and netlists files for top level and the fixed module.

run_rec_module i: runs the ngdbuild, map, par, and pimcreate for the module *i*. *i* can be "*system*" (standing for the fixed module) or "*1","2"... " n"* (index number of the reconfigurable module).

assemble_complete i: runs the final assembly phase for the module *i*, generating a complete bitstream containing the processor and the module *i*.

assemble_partial i k: runs the final assembly phase for the module *i*, generating a partial bitstream containing the module *i* considering the module *k* as the initial system.

run_all: after creating a project and copying the modules' netlists, this script calls the scripts *run_initial*, *run_rec_module*, *assemble_complete*, and *assemble_partial*, for generating a complete bitstream with the processor module and the reconfigurable module 0, as well as the partial bitstreams for remaining modules.

It must be noticed that these scripts should be reusable for other designs and applications, and can be easily modified when new problems are found and solved.

6 Conclusions and future work

In this paper we mainly present our motivations and our initial platform and methodology proposals. We are convinced that an application such as Modular Robotics where mechanical flexibility is maybe the main goal can benefit a lot from the flexibility of reconfigurable controllers. Dynamically modifying controllers for different types of robot shapes (dynamically modified too) can largely enhance the capabilities of these robots.

We also present a technique for performing partial reconfiguration on Spartan-3 FPGAs. By using Difference Based partial reconfiguration we emulate a Module Based one. We discuss about the advantages, limitations and possible improvements for this technique.

The presented approach may increase system flexibility thanks to DPR, while keeping low memory requirements given the Bluetooth access to a bitstream repository. This wireless channel may also simplify the process of loading an initial bitstream – i.e. a complete bitstream – as well as a partial bitstream, which can be very painful when reconfiguring a set of robots.

The proposed framework remains simple and user-friendly; additionally it provides enough flexibility for the specific application. Our approach can be extended to more demanding applications by adding more reconfigurable modules, or other peripheral interfaces for connecting to modules such as IPIF instead of GPIO.

Currently we are working on the development of controllers for validating the proposed framework. Nonlinear coupled oscillators seem to be a promising approach for exploring modular robotics locomotion [9, 13], previous work on evolving neural networks using DPR [10, 11], and co-evolution of Fuzzy Systems [12] can be also of special interest for our system.

Acknowledgements

We would like to acknowledge all the people who have worked in the robot construction, mainly André Badertscher for the technical support and Alessandro Crespi for helping us to solve all kind of software and hardware problems. We acknowledge also the Swiss National Science Foundation for supporting the project Towards Smart Adaptive Artefacts and for the Young Professorship Award granted to Auke Ijspeert.

References

[1] C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, and L. Thiele, "Reconfigurable Hardware in Wearable Computing Nodes," presented at Sixth

International Symposium on Wearable Computers, Seattle, Washington, 2002.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," Acm Sigplan Notices, vol. 35, pp. 93-104, 2000.

[3] J. Beutel and O. Kasten., "A minimal Bluetooth-based computing and communication platform.," Technical Report, Computer Engineering and Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich 2001.

[4] P. Waldeck and N. Bergmann, "Dynamic Hardware-Software Partitioning on Reconfigurable System-on-Chip," presented at The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC'03), Calgary, Alberta, Canada, 2003.

[5] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "PipeRench: A reconfigurable architecture and compiler," Computer, vol. 33, pp. 70-77, 2000.

[6] A. A. Gray, C. Lee, P. Arabshahi, and J. Srinivasan, "Object-oriented reconfigurable processing for wireless networks," presented at IEEE International Conference on Communications, 2002. ICC 2002., New York, USA, 2002.

[7] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-TRAN: Self-reconfigurable modular robotic system," Ieee-Asme Transactions on Mechatronics, vol. 7, pp. 431-441, 2002.

[8] M. Yim, Y. Zhang, K. Roufas, D. Duff, and C. Eldershaw, "Connecting and disconnecting for chain self-reconfiguration with PolyBot," Ieee-Asme Transactions on Mechatronics, vol. 7, pp. 442-451, 2002.

[9] A. J. Ijspeert, "Vertebrate locomotion," in The handbook of brain theory and neural networks, M. Arbib, Ed.: MIT Press, 2003, pp. 649-654.

[10] A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," Microprocessors and Microsystems. In press, 2005.

[11] A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "A methodology for evolving spiking neural-network topologies on line using partial dynamic reconfiguration," presented at ICCI - International Conference on Computational Inteligence, Medellin, Colombia, 2003.

[12] G. Mermoud, A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "A Dynamically-Reconfigurable FPGA Platform for Evolving Fuzzy Systems," in The 8th International Work-Conference on Artificial Neural Networks (IWANN'2005), (submitted) 2005.

[13] A. J. Ijspeert, "A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander," Biological Cybernetics, vol. 84, pp. 331-348, 2001.

[14] BIRG (Biologically Inspired Robotics Group), "YaMoR modular robot Webpage." http://birg.epfl.ch/page53469.html.

[15] Xilinx Corp., "XAPP 290: Two Flows for Partial Reconfiguration: Module Based or Difference Based," www.xilinx.com, Sept, 2004.

[16] Xilinx Corp., "MicroBlaze™ Soft Processor Core," www.xilinx.com/microblaze.