

ARCHITECT-R: A System for Reconfigurable Robots Design

R. A. Gonçalves^{*}, P.A. Moraes^{*}, J. M. P. Cardoso⁺, D. F. Wolf[‡], M. M. Fernandes,
R. A. F. Romero^{*}, E. Marques^{*}

^{*}Instituto de Ciências
Matemáticas e de Computação
- Universidade de São Paulo
Av. Trabalhador São-carlense,
400 13560-970 - São Carlos -
SP - Brasil

{richard, priscila, rafrance,
emarques}@icmc.usp.br

⁺Faculty of Sciences and
Technology
University of Algarve
Campus de Gambelas
8000-117 Faro - Portugal

jmpc@acm.org

[‡]Robotics Research Laboratory
Computer Science Department
University of Southern California
Los angeles, California
denis@robotics.usc.edu

ABSTRACT

An increasing interest in the design of mobile robots has been observed in recent years, which is mainly motivated by technological advances that may allow their application to consumer markets, in addition to industrial areas.

Although sophisticated techniques have been developed, choosing the appropriate hardware-software partitioning and programming robot functions are still very complex tasks.

Current approaches often involve the design and implementation of hardwired solutions, with the associated problems of a long development cycle and inflexibility.

In this paper we present a framework called ARCHITECT-R, which aims to design and program specialized hardware for robots based on FPGAs. We also present the first results obtained using this framework.

Categories and Subject Descriptors

C.3 [Computer Systems Organization]: Special-Purpose and Application-Based Systems – *Real-time and embedded systems*.

C.5.4 [Computer System Implementation]: VLSI Systems.

D.3.4 [Programming Languages]: Processors – *compilers*.

General Terms

Algorithms, Performance, Design, Languages.

Keywords

FPGAs, System-Level Synthesis, Design Methods, Reconfigurable

Permission to make digital or hard copies of all or part this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2003, Melbourne, Florida, USA © 2003 ACM 1-58113-624-2/03/03...\$5.00

Robots.

1. INTRODUCTION

Mobile robots have been the central focus of many research works in recent years. Those works have allowed important advances in areas such as algorithms for machine learning and probabilistic models, which can be used to deal with the uncertainty associated with the environment which the robot will interact with, and the data received via sensors.

Although progress has been made, the task of programming a robot is still very complex and time consuming. In order to tackle this problem, some new languages have been proposed, such as those presented in [2][12][18]. We believe that the most promising one is CES (C for Embedded Systems) [1]. To achieve its goal CES combines procedural and object orientation concepts with special structures able to deal more efficiently with probability and machine learning. By doing so, it is possible to integrate programming tasks that used to be done independently from each other.

Another common issue in the design of robots is the computational complexity of some critical algorithms. Those can have real-time constraints preventing the use of general purpose hardware components, thus requiring specialized hardware. The use of ASICs (Application Specific Integrated Circuits) for this purpose implies a lack of flexibility and a long development cycle, which are unacceptable restrictions for several application domains. An alternative to ASICs consists of a hardware model based on FPGAs (Field Programmable Gate Arrays) [14]. In this case the hardware functionality can be changed according to the task to be executed, which can be particularly useful for a robot working on a changing environment (*e.g.*, indoor/outdoor).

However, designing and programming such a specialized hardware can still be very complex, which has motivated us to propose ARCHITECT-R [9], a framework for the design of reconfigurable robots using the CES language. The aim of ARCHITECT-R is to allow applications developed in CES to be translated into hardware/software components (*i.e.*, code to be executed in a microprocessor and optimized hardware structures, both to be implemented in reconfigurable hardware).

In the next sections we present an overview on the main modules and functionality of ARCHITECT-R, followed by early experimental results obtained using the framework. We conclude the paper by presenting some related work and conclusions.

2. ARCHITECT-R OVERVIEW

Since ARCHITECT-R needs to solve complex design tasks it has been sub-divided into smaller components in order to make its development feasible. Figure 1 shows its sub-systems (components):

- Front-end/Back-end Compilers;
- Legus System;
- Softcore processors;
- Special purpose RPUs (Reconfigurable Processing Units);
- System simulator.

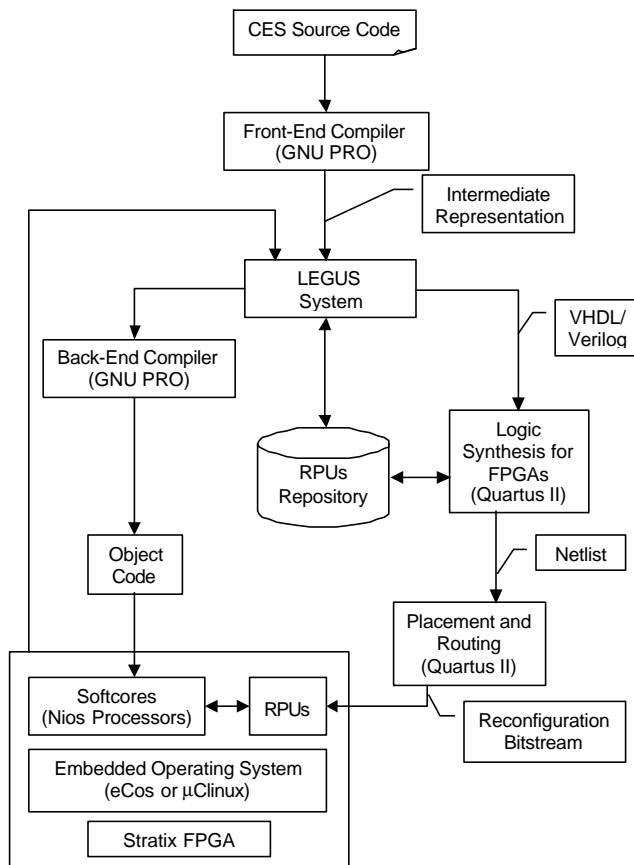


Figure 1. Architect-R Functionality.

The compilation tasks are performed by a front-end compiler and a back-end compiler. The front-end compiler takes responsibility for translating a robot application source code written in CES into an intermediate representation format that is best suited to be used by the Legus System (see Figure 2) and the back-end compiler. During this stage, a few machine independent optimizations [11] and an initial partition of the software and the hardware parts of the robotics system are made. The Legus System is a special artificial intelligence agent responsible for defining the hardware/software

partitioning, the architecture of each processor, and the number and type of RPUs that will be used for a given task.

The hardware/software partitioning is an interactive and iterative process: the Legus System receives an initial partition (specified by the user and identified by the front-end compiler). The performance is then evaluated by using the simulator. Successive changes in the partitioning and simulator invocations are realized until optimal (or adequate) performance is achieved.

The hardware generated by the Legus System is based on softcores and RPUs. A softcore is a processor core described in a hardware description language. The software image to be executed in the softcore might include invocations to RPU functionalities. The RPUs are stored in a library of functional units (RPUs repository), and can be organized in a way that best suits a given application, like clusters of functional units and register files, for instance. The RPUs might be manually designed hardware units or architectures, to implement specific computational structures, obtained using an architectural synthesis tool.

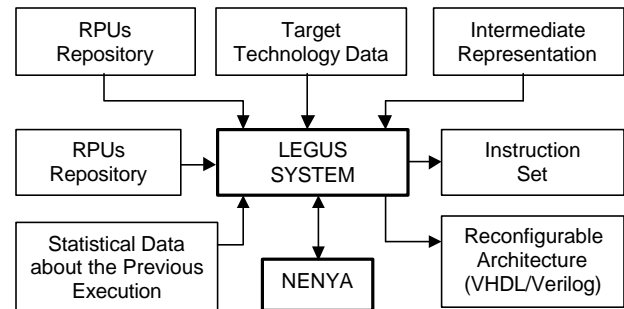


Figure 2. Legus System Overview.

A specific architecture to implement a given computing structure can be generated using the GALADRIEL and NENYA framework [4][5]. The framework targets reconfigurable architectures based on reconfigurable logic and one or more memories. It accepts a substantial subset of programming constructions (*e.g.*, loops, conditionals, arrays), enabling the compilation of representative algorithms. NENYA is an architectural synthesis tool especially developed to target reconfigurable computing platforms. NENYA extracts hardware images from the sequential description to be executed by the reconfigurable logic. For designs requiring more reconfigurable hardware resources than the physically available, the approach integrates temporal partitioning techniques in the compilation process. The compiler exposes several degrees of parallelism (operation, basic block, and functional), all of them fundamental for obtaining an efficient reconfigurable hardware implementation. The compiler generates, for each temporal partition, a control unit and data-path modules (including memory interfaces) aimed at a standalone reconfigurable hardware platform or a software/hardware approach

Each hardware image generated by NENYA for a given configuration is described in VHDL, which defines the units used from the library set, the interconnection between them, and the control of shared resources. These descriptions are passed to a VHDL-based logic synthesis tool (*e.g.*, Synopsys Design Compiler), which has access to a library of functional units based on the FPGA used. The synthesis tool translates each hardware description into a list of primitive cells supported by the FPGA technology library in a

format accepted by the placement and routing tools of the FPGA family. More details and results regarding GALADRIEL and NENYA can be found in [5].

By using the right architectural organization and compiler techniques to exploit ILP (Instruction-Level Parallelism) it is possible to achieve high levels of performance [6][7]. The output of the ARCHITECT-R framework is a combination of an embedded processor, RPU, and peripherals. This set aims to create the control system for a mobile robot that is highly integrated at system-level.

The high-level system integration characterizes the System-On-Chip (SOC) concept. More specifically, we are dealing with a System-On-A-Programmable-Chip (SOPC), since the target technology consists of programmable logic.

Altera's Excalibur System [1] with the Nios Soft Core Embedded Processor has been chosen as the softcore of the SOPC system. ARCHITECT-R modules are being developed to manage different softcores and/or more than one instance of a certain softcore. Other softcores will be considered when the use of a certain instruction set architecture might improve system performance or might lead to cost reductions.

Runtime reconfiguration of the SOPC is allowed due to specific needs of mobile robot applications. Reconfigurable resources might process different processing elements during the execution, while the robot adapts itself to the surrounding environment. Those reconfiguration opportunities have been studied in [15][17]. The Legus System is able to exploit such reconfiguration capabilities. The approach needs modular hardware units. Those modular units are then allocated at the exact time they are needed. To achieve this goal, a softcore processor and some specialized reconfiguration hardware are used. This way the robot stays operational, even during the reconfiguration steps.

3. RECONFIGURABLE PROCESSING UNITS FOR ARCHITECT-R

To improve the system performance and to assist the mapping phase, a number of CES language functions have been directly implemented in hardware. Those functional units are viewed as building blocks by the Legus system. The functional units include:

1. A set of Floating-Point Processing Units (FPUs) used to assist the mapping phase when a floating- to fixed-point data type conversion is not desired;
2. A Neural Network specially trained for indoors navigation;
3. A Digital PID (Proportional, Integral and Derivative) Controller for the purpose of DC motor control used on Robotics;
4. Genetic Algorithms tasks;
5. Topological Map;
6. Probabilistic functions.

In this paper we present three RPUs: a Neural Network, a digital PID controller, and sub-modules of a genetic algorithm.

3.1 Neural Network

The interface between men and robots is a widely open research area. Most robots do not have a friendly interface. A successful gesture based interface for human-robot interaction has been

developed by Waldherr et al. [19] in Carnegie Mellon University (CMU). That interface is based on dynamical gestures that are recognized and interpreted by the robot. There is a video camera on the robot and an on-board computer processes the captured images. The camera generates 30 fps (frames per second) (320×240 pixels). The tracking system process 10 fps and the gesture recognition system interface can process up to 4 fps (10×10 pixels). The gesture recognition system is based on an MLP (Multi Layer Perceptron) artificial neural network. In this case, the neural network software implementation limits the overall performance of the entire system. A research has been developed to improve the performance of the system. A hardware model of the neural network has been built with the FPGA technology [21].

The neural network implemented by Waldherr et al. has 260 neurons (100×100×60). The hardware model corresponding to a MLP model has been designed using the Altera Excalibur (Quartus Design Tool and an APEX20k200E FPGA device) [1]. To generate hardware models from software algorithm, some simple logic and arithmetic blocks such as: multipliers, adders, divisors, and logic gates have been used. Some modifications have been done to make easier the implementation. The floating-pointing numbers have been modified to integer numbers. The transfer function used in the neurons (sigmoid function) has been converted to a linear piecewise function (faster and easier execution).

The advantage of this implementation is in the way by which the MLP neurons are assembled in the hardware model. Each layer of the neural network is represented by only one FPGA neuron. This FPGA neuron works with three distinct memories: input values, weights, and results. Each neuron of MLP model receives: an input value set, a weight value set, and a bias.

The FPGA neuron processes the operations using the input set and its corresponding weight set for each neuron of the MLP model. In this way, an entire layer can be processed by just one FPGA neuron. The computation of the overall layers of the neural network is performed by a sequential processing task. The FPGA neuron works as follows: the input and weight signals are received by the FPGA neuron sequentially. After each multiply operation, the accumulator receives the result. When all inputs are processed, the result of the accumulator pass through the transfer function, and the final results are available. To have a correct synchronization among these operations, a number of address counters have been used. These counters calculate the correct addresses of each memory used in the process. The output memory of a layer is used as an input to the following layer and so on.

The advantage of the use of few neurons to process the entire neural network (only one neuron in this implementation) is the capacity of fitting large neural networks into a single FPGA device. Of course a non-sequential implementation is faster, but it stills not possible to do a completely parallel implementation of a neural network with thousands of neurons in just one chip due to its lack of capacity. A sequential hardware processing is an interesting alternative because the hardware implementation can perform the operations much faster than the software implementation and large neural networks can be built in just one FPGA device. An important advantage of this implementation is that it can be adapted to different MLP topologies through the use of different size memories and different synchronization circuits.

The original system (software) can process up to 4 fps. The FPGA based system has a performance 781 times faster (it can process 3,125 fps). The hardware implementation of the FPGA neuron occupies 10% of an APEX20k200 chip.

A number of tasks, such as training the neural network, are considerably difficult to be implemented directly in hardware. Most hardware implementations do not allow on-chip learning. To explore the software flexibility and hardware performance, the Nios processor has been used [22]. The Nios Processor performs the memory and neural networks hardware interfaces, the control of the neural network topology, the synchronization control of the FPGA neuron and allows on-chip learning in software. The back-propagation algorithm has been implemented (using C) to allow the neural network learning.

The hardware/software implementation reached a performance 15 times faster than the original system, even with a 40 MHz clock (the original system operates at 200 MHz). Table 1 shows the performance for different implementations.

Table 1. Performance Comparisons

Implementation	Hardware Used	Clock Frequency	Processing Time	Frames per Second
Software	Pentium MMX	200MHz	250 ms	4
Hardware	FPGA APEX 20k200E	50MHz	0.32 ms	3,125
Hardware + Software	FPGA APEX 20k200E	40MHz	16.1 ms	62

Being initially designed as a stand-alone block to be used with the robot camera, the integration of this hardware with the rest of the robot would have to be a handcrafted job. To surpass this problem, the neural network hardware is adapted to the RPU format accepted by ARCHITECT-R. The hardware is translated into a macro cell and its properties (such as space occupancy, time constraints, inputs and outputs, etc.) are stored in a description file. Using this scheme, the neural network hardware can be easily integrated in the overall design by the Legus system.

3.2 Digital PID Controller

Digital PID controllers are frequently used on DC motor control robot units. Two hardware versions of a digital PID controller have been designed.

The first hardware version has been implemented on an ALTERA FPGA EPF10K30RC208-3 from the FLEX 10K family. A sequential architecture was used for this implementation. The design uses 1,044 logic cells of the FPGA (60% of the chip).

The second hardware version, using a parallel architecture, has been implemented on the EPF10K40RC208-3 from the same FLEX 10K family. The design requires 1,723 logic cells (74% of the chip).

For the first architecture (sequential), the maximum clock frequency achieved is 33MHz, and for the second architecture (parallel) the maximum clock frequency achieved is 50 MHz.

The achieved hardware performance and the flexibility of the FPGAs show that the chosen methodology is adequate for this application.

3.3 Genetic Algorithms

Genetic Algorithms (GA) [13] are biological inspired algorithms commonly used in optimization problems. The main concepts involved in GAs are the selection of the individuals of a population and the use of genetic operators (*e.g.*, crossover and mutation) for the creation of a new population of individuals.

In robotics, GAs have been used in robot navigation. Since they require high computational efforts, it is important to improve its performance, especially to make feasible its use in real-time systems. We have been developing RPUs implementing some GA operations. At the moment, the repository of RPUs includes implementations of Crossover and Mutation GA operations [10]. Crossover and Mutation are two operations realized repeatedly in the GA step responsible for the creation of a new population. The Crossover operation uses a probability to cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents. The Mutation operation uses a probability to mutate a new offspring at each locus (position in chromosome).

The blocks related to the Mutation and Crossover operations have been designed and implemented in a Stratix EP1F10F780C6 FPGA. The implementation realizes a 32-bit Mutation in 7.2 ns and a 32-bit Crossover in 7.6 ns, each occupying 32 Logic Elements.

4. RELATED WORK

Several efforts have been done to compile high-level programming languages to reconfigurable computing platforms. Most of them focus important optimizations that can be performed to generate efficient hardware structures to implement a certain algorithm [20]. Others have been researching efficient schemes to explore the design space when optimizations at the language level (*e.g.*, loop unrolling) are considered [16].

Most of the efforts use generic programming languages (*e.g.*, C or Java) to describe the algorithm to be compiled. On the other way there are compilers, which integrate programming constructs to deal more efficiently with some types of applications, such as stream oriented computations (*e.g.*, Streams-C compiler [8]) and image processing algorithms (*e.g.*, Cameron project [3]).

To the best of our knowledge, our work is the first approach trying to compile computational structures in CES to reconfigurable computing platforms. The system is fully tailored to embedded systems and especially to complex mobile robotic systems. The target architecture includes core microprocessors and reconfigurable hardware units both mapped to an FPGA.

5. CONCLUSION

Robotics is a fast changing research area, with new algorithms being continually proposed. Executing them in reconfigurable hardware combines the advantages of special purpose hardware with enough flexibility to adapt should new requirements arise.

We have presented ARCHITECT-R, a framework to help the design of robots employing a high-level programming language and targeting FPGAs. The ultimate goal of ARCHITECT-R is to reduce both the effort and the time associated with the design tasks.

We have presented experimental results for a number of complex functional units implemented by reconfigurable hardware. Those units implement functionalities that might be used in a given CES description and thus they are very important to assist the mapping steps.

Work on progress focuses on the development of the Legus system modules and on the interface to the NENYA compiler.

6. REFERENCES

- [1] Altera Inc., <http://www.altera.com>
- [2] Banysad, O. *A Visual Programming Environment for Autonomous Robots*, Master Dissertation, Dalhousie University, 2000.
- [3] Böhm, A. P.W., Draper, B., Najjar, W., Hammes, J., Rinker, Chawathe, R., M., and Ross, C. One-Step Compilation of Image Processing Algorithms to FPGAs. in *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2001.
- [4] Cardoso, J. M. P., and Neto, H. C. Macro-Based Hardware Compilation of Java Bytecodes into a Dynamic Reconfigurable Computing System. In *Proc. of the IEEE 7th Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1999, 2-11.
- [5] Cardoso, J. M. P., Neto, H. C. Fast Hardware Compilation of Behaviors for FPGA-Based Reconfigurable Computing Systems. to appear in *IEEE Design & Test of Computers Magazine*.
- [6] Faraboschi, P., Desoli, G., Fisher, J. A. *Clustered Instruction-Level Parallel Processors*. HP Computer Systems Laboratory HPL-98-204, December 1998.
- [7] Fernandes, M.M. *A Clustered VLIW Architecture Based on Queue Register Files*. Ph.D. Thesis, University of Edinburgh, 1998.
- [8] Gokhale, M., Stone, J., Arnold, J., and Kalinowski, M. Stream-Oriented FPGA Computing in the Streams-C High Level Language. In *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, IEEE Computer Society Press, Los Alamitos, CA, USA, 2000, 49-56.
- [9] Gonçalves, R.A., Wolf, D. F., Rodrigues, M. I., Osorio, L. F., Moraes, P. A., Genuário, L. B., Teixeira, M. A., Ribeiro, A. A. L., Romero, R. A. F., Marques, E. ARCHITECT-R: A System for Reconfigurable Robots Design - An Overview and Initial Results. In *Proc. of IFIP Int'l Conf. on Very Large Scale Integration- The Global System on Chip Design & CAD Conference, (VLSI-SOC'01)*, Montpellier, France, vol. I, 2001, 60-64.
- [10] Moraes, P., *Desenvolvimento de uma RPU para Algoritmos Genéticos em Hardware Reconfigurável*. Master Dissertation Exam, USP-ICMC, 2002 in preparation (in Portuguese).
- [11] Muchinick, S. *Advanced Compiler Design & Implementation*. Morgan Kaufmann, 1997.
- [12] Nabbe, B. *A language for reconfigurable robots*, Technical Report, CMU-IRTR-98-32, 1998.
- [13] Obitko, M., Introduction to Genetic Algorithms at <http://cs.felk.cvut.cz/~xobitko/ga/>
- [14] Oldfield, J. V. *Field-programmable gate arrays: reconfigurable logic for rapid prototyping and implementation of digital systems*. John Wiley & Sons, Inc., 1995.
- [15] Ribeiro, A. *A Study of Reconfiguration Capabilities on FPGAs*. Master Dissertation, USP-ICMC, Brazil, 2002 (in portuguese).
- [16] So, Byoungro, Hall, M. W., and Diniz, P. C. A Compiler Approach to Fast Design Space Exploration in FPGA-based Systems. in *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI'02)*, ACM Press, New York, June, 2002.
- [17] Teixeira, M. A. *Reconfiguration Techniques on the APEX-Altera FPGAs*. Master Dissertation, USP-ICMC, Brazil, 2002 (in portuguese).
- [18] Thrun, S. *A framework for programming embedded systems: Initial design and results*. Technical Report CMU-CS-98-142, Carnegie Mellon, 1998.
- [19] Waldherr, S., Romero, R. A. F., Thrun, S. Gesture-Based Interface for Human-Robot Interaction. in *Journal Autonomous Robots*, vol. 9, no. 2, 2000, 151-173.
- [20] Weinhardt, M., and Luk, W. Pipeline Vectorization. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2. February 2001, 234-233.
- [21] Wolf, D., Romero, R., Marques, E. An Adjustable Size FPGA Implementation For an Artificial Neural Network. in *IC-AI'2001*, Las Vegas, USA, June, 2001.
- [22] Wolf, D., Romero, R., Marques, E. Using Embedded Processors in Hardware Models of Artificial Neural Networks. In *Proc. of the Brazilian Simposium of Intelligent Automation (SBAI'2002)*, Canela-Brazil, November, 2001.