Compiler and system techniques for SoC distributed reconfigurable accelerators

Joël Cambonie[1], Sylvain Guérin[2], Ronan Keryell[2], Loïc Lagadec[3], Bernard Pottier[3], Olivier Sentieys[4], Bernt Weber[2], and Samar Yazdani[3]

[1] STMicroelectronics/MPU
[2] ENST-Bretagne/LIT
[3] UBO/A&S
[4] IRISA/R2D2

Abstract. To answer new challenges, systems on chip need to gain flexibility and FPGAs need to gain structure. We propose a general framework for SoC architectures and software tools in which different kind of processing units are programmed at high level. We show a reconfigurable unit suitable for this framework and we draw the outline of a super-compiler able to address such an architecture.

1 Introduction

Adding some on-chip flexibility is very attractive to extend the lifetime of a submicron ASIC product and to capitalize on the engineering cost but one faces the trouble of an heterogeneous system having different units programmed with different languages. These units can be powerful processors, micro-programmable architectures, reconfigurable data-paths, or fine-grain embedded FPGAs. The global view includes an operating system that runs on a control processor, leading the execution of distributed processes. Each process has its program broken into tasks that can run on different units, based on performance requirement and resource availability. The system has a global view of task status coming from the units, communications and synchronizations being achieved on packet or transaction bases. This paper presents an open framework, associating compilers, system and low level tools.

The case of reconfigurable units is perhaps the most difficult challenge, because their usual hardware description tools are not usable in a software flow. In section 2, we will present a mixed grain reconfigurable unit associating memories, operators and fine grain resources primarily chosen for a wireless modem.

The general flow shown in figure 1 can make use of a decomposition in scalar processing and stream processing. Cooperation between the two sides of the forks can be loose or tight. The strategy of mapping on reconfigurable units uses a first layer where the concurrency is represented by cooperating processes. Code for these processes is produced in the form of a graph of operations compatible with target architectures. Then these graphs are mapped to the available hardware. Higher level tools can make their decisions based on feed-backs coming from the low level tools.

The main objective of this paper is to bring up the full figure of a software oriented reconfigurable system, including real hardware, low level tools for reconfigurable units, and compiler techniques to be developed.

In section 4 we will present a tool structure for mapping on mixed grain units at the logic synthesis level (FPGAs) or at the arithmetic level (data path units) that co-exist in the harware. Section 5 shows a system organization suitable for the concurrency for mixed grain reconfigurable units and for microprogrammable units. Section 6 will discuss code generation for this scheme from high level language down to the basic software using high performance computing techniques.

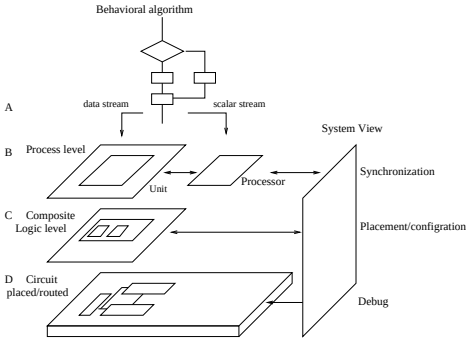


Fig. 1. Simplified flow for multi-target compilation ($A \rightarrow B$), synthesis ($B \rightarrow C$) and system activities.

2 A reference architecture

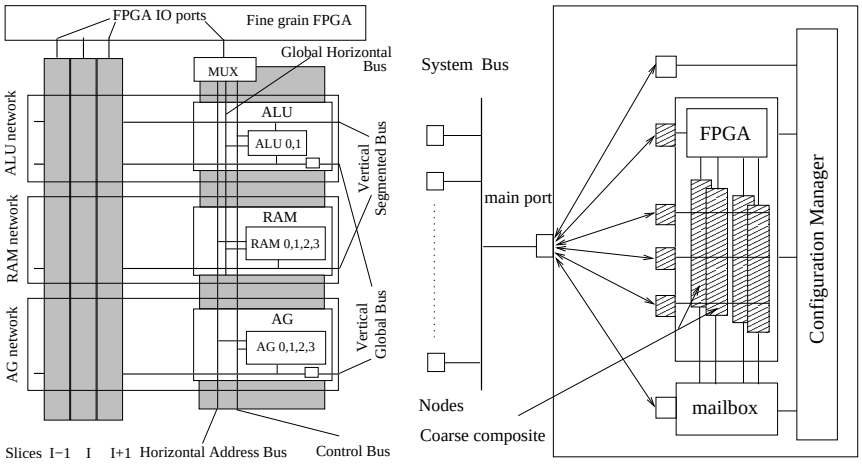
Reconfigurable architectures have received significant attention in recent years, and some directions have emerged that combine desirable features such as low power and flexibility [10] for portable signal processing applications. One idea is that, from an energy perspective, coarse grain arithmetic functions with programmable interconnects outperform instruction based processors where the storing in cache and decoding of the wide instructions account for a significant energy overhead. On the contrary, FPGA is less efficient on context switching, and so the area efficiency is bad, since more functions must be mapped at the bit level on the matrix, in order to meet real time constraints.

From an other standpoint, future generation of ASIC have to face the challenge of increasing computing power at low cost, which obviously is incompatible with a fine grain flexibility. This is why a promising direction for *soft* ASIC design is heterogeneous architectures, combining application specific IPs selected by profiling the set of targeted applications, together with some quantity of configurable fabric, flexibly interconnected, with task synchronization being performed by a micro controller.

The previous SoC architecture of a programmable baseband processor for an OFDM modem combined dedicated IP, fine grain FPGA, micro controller and a crossbar bus interconnect on a single die [13]. In order to increase the area efficiency and computing power of the FPGA, a mixed grain configurable fabric has been developed, encapsulated into a system wrapper that permits multithreaded communication with the system bus. The configurable fabric has been described at the RTL level, validated by simulation of configurations generated from various functions used in the 802.11a OFDM modem. It has been synthe-

sized in $130nm$ ST Microelectronics technology, and is integrated on a test chip. Die size is 14 mm^2 for the composite coarse plus fine grain fabric.

This section focus on the configurable fabric architecture description, its internal structure and how it interacts with the rest of the system.



(a) Mixed grain structure (b) System Wrapper

Fig. 2. Two architecture views.

2.1 Configurable Fabric

This device is composed of 4 different types of resources organized as separate interconnected networks, called coarse networks in the following. These are: fine grain LUT network, ALU network, RAM network, and address generator (AG) network.

- The fine grain LUT network provides bit level computing resource. This is a standard embedded FPGA with 350 available bi-directional ports and 32×32 LUT.
- the ALU network is an organization of 16 rows with 2 tiles per row. Each tile has two 8 bits registered ALU, one 16 bits registered multiplier, one segmented bus for horizontal local interconnect, one segmented bus for vertical local interconnect, plus global horizontal and global vertical bus for long range inter tile interconnect, and communication with the other networks. A global horizontal control bus provides timing signal to the registers.
- For each row in the ALU network, there is a correspondent static RAM row with a dual port of 256×8 bits capacity appearing as shared memory onto the processor address space. Global interconnect busses allow inter and intra network connection in both directions, and combination of elementary RAM banks to produce 256×m words of 8×p words storage. Four 8 bits busses for

address propagation are available per row. One global control bus is available on each row. The designed instance has 16 lines$\times$4 tiles in the RAM network.
- Each row has also address generators connected to its RAM by a local segmented bus that propagate control signals.

Basic resources are statically configured to perform elementary operations. For each resource, data are selected from or connected to the interconnect by multiplexers. The segmented bus is configurable through a set of switches to perform local interconnections. The set of ALU, RAM and AG coarse networks constitute the set of coarse grain resources, while the FPGA is the network of fine grain resources.

Coarse networks are made of vertically stacked rows with horizontal connections on their left and right sides, so that by simple abutment of the three networks, the horizontal global signals can propagate. The horizontal bus width is 117 bits for one row. The three abutted coarse networks are now called composite coarse network.

The composite coarse has 16$\times$117 bi-directional data and control wires available for connection with the fine grain. In order to have a regular connection density through the rows between fine grain and coarse composite network, configurable multiplexers are added to select horizontal busses from coarse composite network to be connected to the 350 available fine grain ports (see figure 2(a)).

The close coupling of the fine grain with the coarse grain allows efficient mapping of mixed control and data processing. All bit level state machines and data processing being efficiently mapped on the fine grain, while word level arithmetic is mapped on the coarse composite. The vertical global lines that propagate through each network are selectively connected to the data ports of the fabric.

2.2 System Wrapper

The configurable fabric is embedded in a SOC. It supports on-the-fly reconfiguration, and multi-threaded applications.

It is encapsulated into a system-wrapper to permit communication with the system, as shown in figure 2(b). The main I/O port is connected to the system crossbar bus, steering incoming data from bus towards one of the 6 internal input ports: 4 bi-directional data ports are connected to the fabric, 1 to the configuration port (write only), and the last to the mailbox port.

The main output port takes data from the 5 internal output ports among which four are the data ports and a mailbox port. Priority is assigned on a fixed order basis.

Data exchange between fabric and system are performed in 2 different modes: (1) receiving and sending data to the other peripherals of the system is achieved through the data ports connected to the crossbar bus, (2) data exchange with the processor are executed through the multi-bank dual ported shared memory embedded into the fabric.

The mailbox ensures synchronization of the tasks running on the fabric with the rest of the system. It decodes messages coming through the bus from other peripherals and produces control signals for the fabric, and on occurrence of conditions in the processing from the fabric, it generates message to be sent on the bus to the other peripherals.

For instance, configuration commands are sent through the mailbox to the configuration manager, which returns acknowledgment on completion. This is also a way of interrupting the software to change context on events occurring in the data-path.

3 Application of an 802.11a Receiver

The 802.11a is an OFDM based WLAN standard, where payload data are preceded by a preamble for receiver synchronization. The block diagram in figure 4 illustrates the different steps of the receiving algorithm. Some are advantageously performed by dedicated coprocessors (FFT, Viterbi, Demapping) connected to the main bus. The grey boxes in figure 4 show the functions mapped on the configurable fabric (preamble detection, carrier offset frequency estimate, channel estimate, channel equalizer, deinterleaver and descrambler). We describe in the following how the different phases of the receiver algorithm are mapped on the fabric, and the kind of interactions occurring between fabric and other processors.

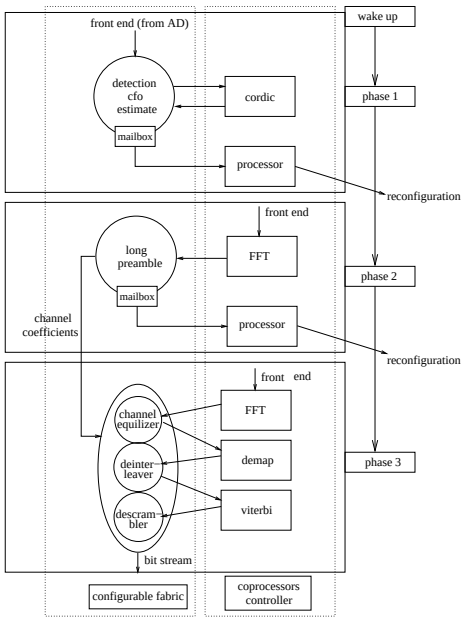


Fig. 3. 802.11a baseband receiver: Simplified diagram with the 3 main configuration phases triggered by the microprocessor during frame receiving. First phase corresponds to OFDM synchronization scheme which is in fact a two-step process namely: frame detection and carrier frequency offset estimation and correction.

In a first phase, where the receiver is waking up, a preamble detection algorithm is mapped on the fabric. This function jointly performs detection and CFO estimate in real time. A call to a Cordic rotation (rotor) in the process is implemented on a dedicated coprocessor. Once a detection criterion is met, a message is sent via the mailbox to the processor that reads data produced by the fabric from the shared memory. Those data will be used to generate data dependent configuration for the next phase. Processor will then order reconfiguration of the data-path for the following step: channel estimate using long preambles. Again when this step is done, a partial reconfiguration is

performed to support the payload data processing, of which channel equalizer, de-interleaving and de-scrambling are mapped on the mixed grain fabric. Figure 3 shows the different processes composing the receiver, and how the fabric interacts with other coprocessors and controller via shared memory, mailbox and system bus along the phases.

Practical mapping of the OFDM preamble detection and CFO estimation is shown in figure 3 in the form of pipeline of medium grain processes communicating by shared memories. A simple implementation is achieved by connecting memories to operative parts. The control of this pipeline needs a control process that feed status information to the global control of the system. The stages are loops implementing correlation computations.

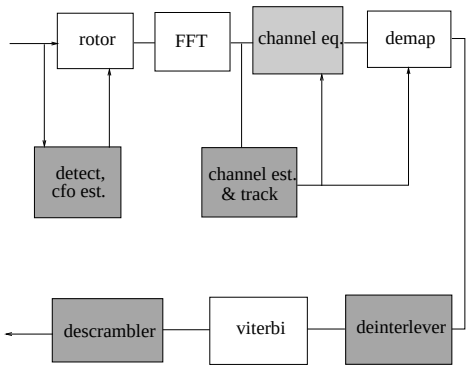


Fig. 4. Baseband Receiver

4 Physical synthesis for reconfigurable units

Physical synthesis is the translation of a software description of the application into resource allocation for a given target architecture, its internal state and constraints. In MADEO, it necessitates two inputs. The first one is the application description appearing as a hierarchical graph of operations including symbolic operations, primitives, logic based operations or memories. As MADEO tools are generic, they need to be characterized by an architecture definition to do their work: placing, routing, resource allocation, interactive or programmed editing and configuration generation. The second input is the target architecture described with grammar oriented tools conforming to an abstract model.

To define a unit model it is necessary to write or generate a program assembling and parameterizing a hierarchie of hardware elements such as wires, registers, multiplexers, functional unit, etc. Wires can be single lines, structured channels or hierarchie of channels. Tools support parametrized channel size to enable architecture prospection under place and route control. Functional units can be specified as a restricted set of permitted symbolic operations (e.g. ALUs) or any n-inputs logic function (e.g. LUTs). The model can be extended by custom parameters or embedding of custom elements.

4.1 Software integration for heterogeneous units

Physical synthesis of an application graph is based on interleaved two-steps operations implying compiler optimizations from Madeo front-end [8]) and physical

mapping. The upper layer tools are in charge of mapping to architecture functions, based on its knowledge of the target unit, and on type information provided at compile time. Detailed types are set of values used to produce optimized logic as the interval alternative for arithmetic operators.

The first stage of this compiler supports classical optimizations such as dead code and NOPs removal. Then, in the flow for addressing fine grain FPGAs, the logic is produced as PLAs representing the hierarchical graph of lookup-tables. Last, we use logic synthesis to perform technology adaptation.

Coarse grain graph elements are hardware primitives such as addition/multiplication that will be allocated on the data-path.

Madeo-BET flow integrates floor-planning, place and route, and physical characteristics computation. The default floor-planning policy is based on the TCG algorithm presented in [9], the placement algorithm relying on a simulated annealing algorithm; the global routing uses a PATH-FINDER-like algorithm on top of a maze router, performing the point-to-point routing. Custom operators can be saved as libraries for different architectures.

4.2 Interactions with higher levels

Inside Madeo, physical and logical layers interact in several ways:

1. *architecture exploration:* the back end tools enumerate the available resources in order to let the compiler parameterize its behavior,
2. *composite building:* the compiler creates some hierarchical graphs of components, called composite parts, in conformity with the unit resources: logic tables, registers, operators and memories. Composite parts ensure the migration from the high level to physical mapping as the back end tools accept them as input,
3. *feedback to the high level compiler:* the back end tools output execution measures and physical characteristics, to be evaluated by the compiler (see section 5).

As an illustration of these interactions, place and route tools return hierarchical modules that are pieces of configuration with relative position. These modules are analyzed regarding several performances criteria such as temporal or geometrical characteristics, and resource use. Other criteria such as power consumption estimation can also be added. These informations are sent back to higher level compilers, so that some optimization trade-offs could be found.

Regarding the modules produced by Madeo, several policies can be considered. One policy is to keep these modules if they can be reused in the context of the application, as for producing pipeline stages or regular design components. Another policy is to archive these modules in a database for further reuse. Madeo has now provisions to keep modules as hardware dependent libraries associated to high level object-oriented behaviors, or simple software definitions.

To conclude with physical synthesis, the proposed tools are generic, but the architectural model specializes them transparently. The effort to produce an ar-

chitecture model is relatively small. The approach strongly reduces the development time, and improves the software quality as any improvement is widespread to all architectures. As for the fabric, the same tools operate over the entire architecture. As an example, as every function unit "publishes" the functions it can implement, every node of the graph of operations the compiler produce, can be assigned a set of primitive function candidates. Once the placement is done, routing computes routes from pins to pins, without regards to the net's width or the kind of routing channels, thus fine grain and medium grains can be managed in the same framework.

5 System aspects

Heterogeneous systems on chip require an assembly of complex software and CAD tools since it is necessary to address different hardware supports with transparency. The objectives to remember are to reach the best possible compromise between: (1) costs and delays for application developments, (2) energy consumption, silicon resource and computing efficiency.

The first objective implies to use efficient development methodologies, supporting very high level specifications and modular design (objects). The second objective is obtained by using a gradation of targets going from processors to reconfigurable units, and IP modules.

A view of a computer at work is an operating system controlling hardware resources and application processes. Processes can be divided into small tasks implementing concurrent or sequential computations. The system model has three components: (1) hardware resource description, (2) application processes, (3) algorithms for scheduling and allocating resources.

An advantage of the SoC approach of architecture toward unstructured FPGAs is the level of organization provided by the network, I/Os and the control processor. Reconfigurable resources are allocated as part of units, communication are simple transactions from unit to unit, or message passing. In this case the knowledge of the SoC organization is enough to allow the OS to act.

A more difficult issue is the case where reconfigurable units are shared between tasks. This situation is highly desirable to make an efficient use of the hardware when data arrives at unscheduled time or at irregular rates. Overloading means that processes can migrate from the active state to two inactive states with swaps to the local configuration memory, or to the external memory. To enable the system to operate, it is necessary to structure processes into set of tasks that will be mapped on the different units.

The flow model where data are fed to an accelerator is simple and convenient for multimedia or signal processing algorithms. In this case, computations are organized as pipelined graph of medium grain tasks. Tasks implement one step in an algorithm, act independently from peers. They embed their own control and processing parts.

The application pipeline is managed by a control task receiving status informations from the stages, and enabling the communications between them.

The control process communicates at its turn with a node management process that holds the status of the different applications, either active or inactive, communicates with the OS, and manages partial reconfiguration and swaps.

This structural organization can be described by concurrent languages (Occam, Handel-C,...), with the remark that the computation graph is not statically mapped to the computer, but dynamically processed by the OS and the skeleton thread running on the control processor.

Code for tasks are generated as physical circuit descriptions for reconfigurable units, programs for compute intensive processors or microcode for microprogrammable units. In the case of configured circuits, the physical description is a bitmap block associated to a geometry, and correspondence between high level variables and registers. At runtime, the bitmap will be merged with other task bitmaps and loaded to configuration memory. The bitmap contains the registers initial values or context values during a swap-out operation. An application mapping is given in the following section.

6 Towards general purpose compiling

During the past few years, the main concentration in the domain of embedded systems has been on high performance in a cost effective manner i.e. flexibility and power efficiency but the switch from general-purpose to multimedia workloads allows alternative architectures.

Since Reconfigurable architectures such as SCORE [5] and RaPiD [4] rely heavily on compilation processes, consistent software paradigms [14] for programming Custom Computing Machines (CCM) are becoming the need of the day. nowadays.

Vector ISA has some advantages over scalar ISA to exploit the computation pipeline [6,7] and out-of-order, speculative techniques or aggressive simultaneous multithreading [12] can be used to boost the efficiency but may have adverse effects to real-time processing. Since the computation throughput often increases the processor-memory gap, the separation of memory access and computation streams in decoupled architectures [11] is interesting since it allows out-of-order execution between the two streams.

Since many parallelism levels that were used in the high-end computers are now found in SoC, the same compilation techniques can be used to compile high level programs to control software and hardware descriptions.

If we look at compiling for distributed memory computers for example, a program is split in different parts running on various processors with their own memory. Since a processor cannot directly access data in the memory of another processor the compiler analyze the code to insert communications primitives where such accesses are needed. In the same way, to generate a program for an heterogeneous computer we need to add communication or interface code between the controller, the various hardware operators, the memories, the bus. This is why we have used super-computing compilation techniques to our control

code and hardware specification problem: global memory emulation by compilation [3] to generate memory access code or DMA initialization and HPF style compilation [2] to distribute the computation on the computational entities of the SoC.

To achieve such a code generation from classical higher-level numerical languages such as Fortran or C to a closer architectural description in SmallTalk for the MADEO tool of the project we have used the PIPS compiler framework [1] that is developed since 1988 as a modular compiler for high performance computing. This project began as an automatic vectorizer for vector machines but evolved later toward automatic parallelization, code optimizing, code transformation, HPF compilation and reverse engineering.

Many analyses and code transformations developed in PIPS first for high performance computing are reused here in a more hardware context. Automatic parallelization transformations using precise analysis of the code based on a linear algebra framework are used to split the code in several parts. A coarse grain parallelization is used to find different coarse grain processes that will keep the different processors and co-processors of the SoC busy with some support of the operating system run-time. A fine grain parallelization inherited from vectorization for SIMD machines is used to exploit the massive parallelism available in the configurable co-processors.

To cope with inter-procedural programs, memory accesses are modeled with polyhedral regions to express the array memory usage between different parts of the program and generate the communication operators with minimal temporary memory buffers. The programmer can specify with some pragma which parts of the code will be executed on a reconfigurable engine and the region analysis will determine the input and output data to handle between the control code or memory and the hardware accelerators.

The remaining program that has not been transformed into a hardware form becomes a control program running on the SoC processors with calls to the reconfigurable operators and data-paths. Since the reconfigurable operator programs can be pretty-printed in both MADEO format or high level C or Fortran form, the later one allows an easy functional simulation of the global program by just running it on a regular computer as with other systems such as SystemC or SpecC.

Conclusion

This project perspectives are highlighted by giving an interpretation of the heterogeneous reconfigurable SoC architecture related to known concepts in computer architecture.

Taking benefits from the reconfiguration we propose to enable a variety of programming methods for the units, including fine tuned application support for multimedia or signal processing, general purpose support with pipelined accelerators, domain specific application supports (such as cellular automata), virtual machine implementations. The program level use of these supports is envisioned

as a co-operation of two threads, one for control and memory operations, executed on the control processor, the other executed on one or several reconfigurable units.

Several of our objectives have yet been reached: portability of synthesis on reconfigurable units is no more a concern, and the architecture developed at STMicroelectronics open the way for attractive distributed reconfigurable processing. Several threads of work are currently followed for code generation, that includes architecture synthesis for signal processing, and general purpose pipelined processing under compiler control.

References

1. C. Ancourt, F. Coelho, B. Creusillet, and R. Keryell. How to add a new phase in pips: the case of dead code elimination. In *6th Workshop on Compilers for Parallel Computers (CPC'96)*, December 1996.
2. C. Ancourt, F. Coelho, F. Irigoin, and R. Keryell. A linear algebra framework for static hpf code distribution. *Scientific Programming*, 6(1):3–27, Spring 1997. Special Issue — High Performance Fortran Comes of Age.
3. C. Ancourt and François Irigoin. Automatic code distribution. In *3rd Workshop on Compilers for Parallel Computers (CPC'92)*.
4. D. C. Cronquist C. Ebeling and P. Franklin. RaPiD - reconfigurable pipelined datapath. In *FPL96*, number 1142 in LNCS, September 96.
5. E. Caspi, M. Chu, R. Huang, J. Yeh, J. Wawrzynek, and A. DeHon. Stream computations organized for reconfigurable execution. *FPL'2000, LNCS 1896*, 2000.
6. Roger Espasa, Mateo Valero, and James E. Smith. Vector architectures: Past, present and future. *International Conference on Supercomputing*, 1998.
7. C. Kozyrakis and D. Patterson. Vector vs. supersclar and VLIW architectures for embedded multimedia benchmarks. *35th Intl. Symp. on Microarchitecture*, 2002.
8. L. Lagadec, B. Pottier, and O. VillellasGuillen. An LUT-based high level synthesis framework for reconfigurable architectures. In *SAMOS'02*. M. Dekker, 2002.
9. Jai-Ming Lin and Yao-Wen Chang. TCG: A transitive closure graph-based representation for non-slicing floorplans. *38th Conf. on Design Automation*, 2001.
10. Jan Rabaey. Reconfigurable computing a solution to low power programmable DSP. *Proceedings ICASSP*, 1997.
11. J. E. Smith. Decoupled access/execute computer architectures. *In ACM Transactions on Computer Systems*, November 1984.
12. Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous Multithreading: Maximizing on-chip parallelism. *ISCA95*, 1995.
13. L. Vanzago, B. Bhattacharya, J. Cambonie, and L. Lavagno. Design space exploration for a wireless protocol on a reconfigurable platform. *DATE'02*, 2002.
14. S. Vassiliadis, G. Gaydadjiev, K. Bertels, and E. Panainte. The Molen programming paradigm. In *SAMOS'03*. Springer LNCS, 2003.