

A programming toolset enabling exploitation of reconfiguration for increased flexibility in future system-on-chips

Gilbert Edelin¹

Philippe Bonnot¹

Waelle Gouja¹

Koen Bertels²

Florian Thoma³

Axel Schneider⁴

Joachim Knablein⁴

Bernard Pottier⁵

Jean-Christophe Le Lann⁵

¹Thales Research & Technology, 91767 Palaiseau, France

²TU Delft, 2628 CD Delft, The Netherlands

³Universität Karlsruhe, 76131 Karlsruhe, Germany

⁴Lucent Technologies Network Systems, Nuremberg, Germany

⁵Université de Bretagne Occidentale, 29238 Brest, France

Abstract

This paper presents an integrated programming toolset for application implementation on an heterogeneous reconfigurable architecture. The objectives of the toolset are to optimize the application implementation productivity and to enable the dynamic reconfiguration on reconfigurable units of the target architecture. The proposed solution is based on the combination of a compilation concept permitting to call function implemented on these reconfigurable units, an operating system dynamically managing the units reconfigurations, a formal specification tool and a design tool for the implementation of the accelerated functions. The paper gives an overview of these tools, that are combined to take benefit of their facilitated reciprocal interactions.

Keywords: reconfigurable computing, retargetable compilation, dynamic reconfiguration, formal specification, data-parallel mapping, architectural synthesis.

1 Introduction

1.1 Objectives

We present in this paper a toolset whose objectives aim at satisfying embedded computing requirements within the context of Reconfigurable Computing architecture solutions. These objectives are portability (avoiding platform adherence), computing performance efficiency, flexibility and application programming productivity.

The presented MORPHEUS toolset targets an heterogeneous reconfigurable architecture based on several types of Reconfigurable Unit including fine grain and coarse grain RU. This shows the portability of the toolset and thus its potential large spectrum of architectural targets.

In the nominal target architecture, the RU are connected through a Network-on-Chip that leverages the scalability of the architecture and its programmability, which is seen as an advantage with respect to FPGA for example. In order to enable computing performance optimization, the toolset addresses parallelisation between RU and within RU.

Dynamic reconfiguration management offered by the toolset will be the way to achieve flexibility and will also bring a computation performance efficiency advantage.

Above all, the toolset targets a high application programming productivity through high abstraction programming level for a global application shared on the RU as well as for the implementation on each unit. Formal specification proposed in the toolset also contributes to productivity optimization.

Toolset specification has just been achieved and the module developments are progressing well as the following sections will show.

1.2 Overview and benefits of the integrated toolset

The toolset combines the large set of following technologies that are necessary to obtain an integrated design flow from high level application programming (such as C language and graphical interfaces) to hardware and software implementation: compilation, RTOS, data parallel reorganization, architectural and physical synthesis,

formal specification.

The toolset entry point for the application programmer follows an enhanced version of the Molen programming paradigm (see section 2). According to this paradigm, the programmer describes its application in a classical C-language program and just annotates the functions identified as preferably implemented on a RU. The compiler then generates code for the host processor of the system (ARM core), statically optimizing the scheduling of configurations and executions of the accelerated function on the RU. The compiler also generates a Configuration Call Graph that will be used as a basis for a more precise dynamic scheduling (see section 3). In this enhanced compiler implementation, several accelerated functions can run in parallel, the compiler and RTOS managing synchronization and scheduling. In the current version, HW/SW partitioning is not yet performed automatically and is left to the programmer choice.

Concerning the accelerated function implementation (presented in section 4), the toolset offers a high level

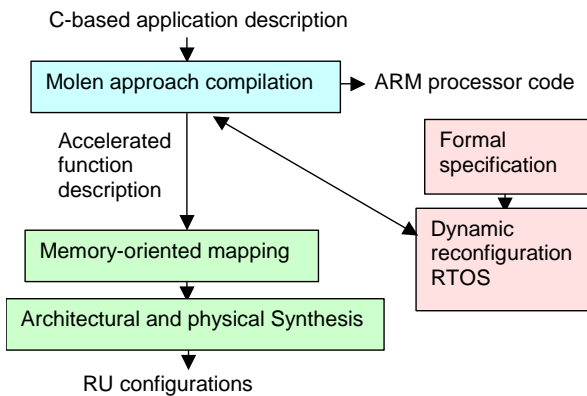


Figure 1 - Integrated design flow

graphical capture. The accelerated function is supposed to be of data-streaming computing type. The graphical interface thus offers a way to best express the parallelism inherent to the function. Then, the toolset aims at optimizing the parallel implementation within the selected RU. It performs an optimized local memory mapping followed by architectural and physical synthesis on a RU array. The toolset also generates communication code between RU and main internal memory.

Formal specification is also used to generate RU configuration (through VHDL generation) and configuration call graph to ensure correct RTOS scheduling.

The benefit of such integrated toolset is to permit a fast and thus efficient design space exploration: partitioning trials (identification of accelerated parts) and allocation trials (implementation unit choice) with quick

feedback since implementation results are obtained with the toolset itself.

Another benefit of the proposed approach is the scheduling optimization coming from the combination of compilation design time scheduling and RTOS run time scheduling.

2 Molen-based approach and Retargetable Compilation

The Molen machine organization [1] that defines the way the different components interact and that supports the Molen programming paradigm is described in Figure 2.

The two main components in the Molen machine organization are the general purpose processor (CP) and the “Reconfigurable Processor” (RP) which can contain any number of custom computing units (CCU) corresponding to the RU mentioned above.

The Arbiter issues instructions to either processor by means of a partial decoding of the instructions. The support for the SET/EXEC instructions required in the Molen programming paradigm is based on **reconfigurable microcode**. This reconfigurable microcode (μ -code) is used to emulate both the configuration of the Custom Computing Unit and the execution of implementations

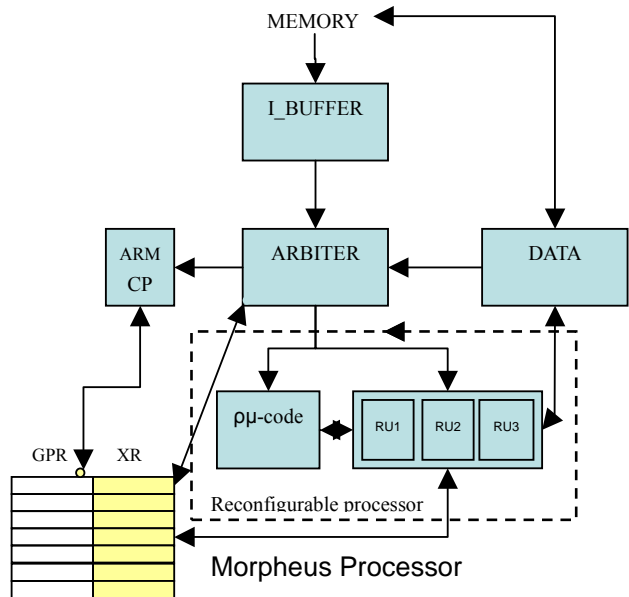


Figure 2 – The Molen organization

configured on the CCU.

The Molen programming paradigm is a sequential consistency paradigm for programming custom computing machines possibly including a general-purpose computational engine(s). The paradigm allows for parallel and concurrent hardware execution. For a given ISA, a

single architectural extension comprising 4 instructions suffices to provide an almost arbitrary number of operations that can be performed on the reconfigurable hardware¹. Each new operation can thus be considered an extension of the current instruction set. However, in order for the Morpheus processor to use these new instructions in a transparent way, we extend the current instruction set by introducing 4 new instructions.

These four basic instructions are SET and EXECUTE, MOVTX and MOVFX. The SET instruction causes a hardware implementation to be loaded and the EXECUTE allows for its execution in the reconfigurable processor. The MOVTX and MOVFX instructions are needed to provide the communications between the reconfigurable hardware and the general-purpose processor. They put parameters in dedicated registers, called exchange registers (XR). The result of a computation is also stored in such a register and passed back to the GPP.

In order to use such a machine organization, one of the required tools is a compiler, which can schedule the different SET and EXECUTE statements in an optimal way.² The compiler is a retargetable compiler as it needs to be able to include an arbitrary number of new instructions, each having their own latency and area requirements. This optimality however is only guaranteed within the execution flow of one application. Once multiple applications are running on the platform, it is the responsibility of the operating system to manage the timely configuration (SET) and execution of the respective CCU's.

3 Dynamic reconfiguration control

3.1 RTOS

The real-time operating system (RTOS) provides the interface between the applications and the hardware. The application can use basic operating systems services as known from other operating systems. Additionally the RTOS offers the Dynamic Reconfiguration Framework providing necessary services to support reconfigurable computing.

One service of the dynamic reconfiguration framework provides interfaces for the reconfiguration and execution control of operations on the RU to the compiler.

At first step, a set system call will be used by the compiler to request the configuration of an operation on a reconfigurable unit. The Configuration Call Graph provided by the compiler allows prediction of near-future

requests of operations. Depending on these predictions and the current configurations of the RUs the RTOS chooses a RU and an implementation of the requested operation on this RU from a given set of configuration bitstreams. This allocation process has to consider configuration time / execution time trade-offs if the preferred RU is already in use. The necessary information is taken from a library which contains the available operations, their implementations and the properties of these implementations like throughput, delay, size and power. The allocation decision leads to the prefetching of the configuration bitstream, but not necessary yet to the configuration of the RU.

The second step is the execution of an operation. For the transfer of the parameters, the RTOS provides virtual exchange registers, which are mapped to physical register files inside the RUs. After the filling of these registers by the compiler, the execute system call is used by the compiler to request the execution of an operation. The scheduling of this operation has to consider constraints set by other pending operations.

For the synchronization of parallel running operations the RTOS provides a break system call which waits for the completion of the referenced operations. The compiler can assure data consistency between parallel operations by using of this system call.

If an operation is no longer used, for example the calling loop has been left; the allocated resources are freed by the compiler with the release system call.

The second service of the dynamic reconfiguration framework is the control of the Network-on-Chip. It provides abstract communication channels to the application designer. He defines the properties of a pipeline structure between functions independent of the used RU. The RTOS programs the NoC according to these definitions and the current allocation. In the case of limited communication bandwidth the network scheduler can use the priority features provided by the NoC.

3.2 Scheduler formal specification and verification

Run-time reconfiguration scheduling needs to be absolutely reliable. Formal specification and verification techniques enable proving that the scheduler satisfies its required properties. Further it provides means to analyze and optimize the scheduling. For this purpose Lucent's specification platform SpecEdit and the formal specification language ADeVA (Advanced Design and Verification of Abstract Systems) [2] is used. SpecEdit provides a platform for the structured management of specification data, proving their consistency and plausibility and rendering them into different models e.g. VHDL, C or SMV. These models allow further processing

¹ These 4 instructions constitute the minimal ISA extension. The full extension is described in [1]

² In this project, the COSY compiler is used.

by other tools for code generation, model checking, timing analysis etc.

The semantics of ADeVA primarily follows from the interpretation of the tables as asynchronous parallel state transition systems. Each table represents a finite state machine whose states are defined by the modes (MTT) or by the values (DTT). ADeVA has been successfully applied to telecommunication protocols [3] but may also be used for other applications like the reconfiguration scheduler. An ADeVA model consists of two types of tables, called: Mode Transition Tables (MTT) and Data Transformation Tables (DTT). Table 1 shows an MTT.

From	To	Condition1	Condition2	Condition3
state_a	state_b	@T	F	-
state_a	state_c	@T	T	-
state_b	state_c	F	@F	T

Table 1: Mode Transition Table

The symbols T for True, F for False, and “-” for “don’t care” all indicate whether a condition applies, and @T/@F indicate an event that changes the value to True/False. Multiple concurrent events are allowed – an

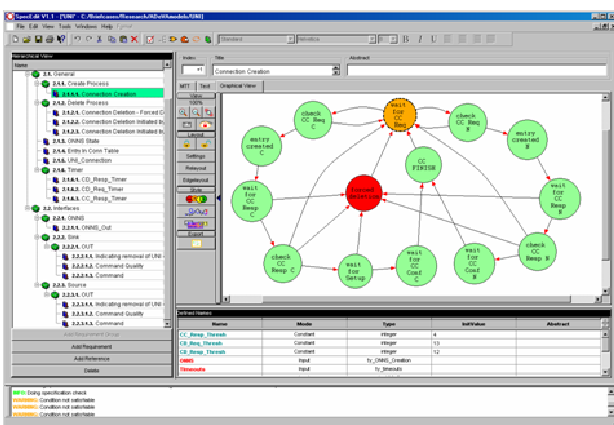


Figure 3 - SpecEdit Graphical User Interface

essential feature for hardware specifications. The rows of an MTT describe state transitions, whereas the rows of a DTT describe the value changes of the signals named in the first column. Timing information may be annotated to each row of an MTT or DTT. Figure 3 shows the GUI of SpecEdit with its graphical MTT editor.

SpecEdit proves that all state transitions are deterministic, all transition conditions are satisfiable and detects dead end states. Furthermore, SpecEdit generates output representations of the specification, including VHDL [4], C [5], Esterel [5], SystemC, SMV and Promela which may be processed by a wide range of commercial or non-commercial design, verification and simulation tools. The VHDL models are of particular interest, because these

models may be used to generate the RU configuration once the correct scheduling has been verified with model checking and timing analysis.

4 Spatial design

This part of the toolset concerns the accelerated function implementation on RU from high level specification to retargetable implementation synthesis. This includes data movements management since the RU are defined to work on the Main Memory Data Structure. The major architectural fact is indeed that data circulate on a loop from MMDS to local memories, then back to MMDS. This data movements management is produced by local or shared DMA engines that transfer data to local memories according to address patterns.

4.1 Memory-oriented task mapping

The design flow starts with the high level specification of an accelerated function required by Molen (section 2). This specification (usually in C language) is currently translated manually into an array transformation formalism [6] through the interactive SPEAR [7] graphical interface as entry point. This graphical view contains processes (elementary functions) connected together to form the accelerated function.

The SPEAR tool generates automatically the appropriate communication processes between the computing processes. A mechanism implements the communication processes and manages specific data arrays manipulation, communication and scheduling. This mechanism, named “connector”, has been first elaborated in VHDL for FPGA implementation purpose before the presented one. Connectors role is to optimize internal RU scheduling performance and data movements as shown in Figure 4.

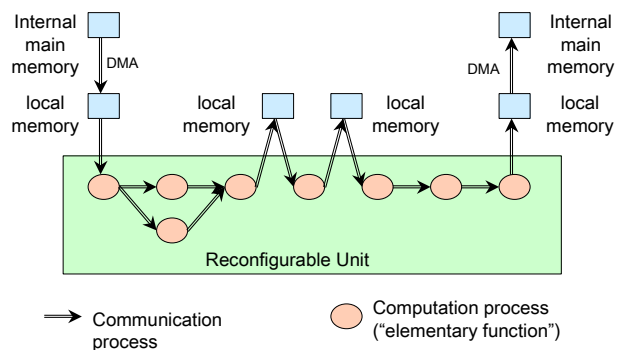


Figure 4 - Chain of computation and communication processes implementing an accelerated operation

Connectors provide fine grain synchronizations where the decision to access shared data in memory (read and write modes) is done statically by the tool and known at design

time (triggered on elementary data). The size of the buffer implemented on local memories and supporting the synchronization and the data reorganization is optimized. The chain of computation is a chain of data array transformations based on nested loop kernels. The outer loops are (splitted if required and) merged when this is found to be possible. This permits to build the pipeline and to identify a minimum buffer size ensuring the required data reorganization.

Computation data synchronizations issue (fine or coarse grain) is a different notion from scheduling of operations on the circuit. Scheduling can lead to replicate a process for example. It is the job of architectural synthesis tool (MADEO [10]) to provide feedbacks to SPEAR (implementation area, latency, computation bandwidth, etc).

Coarse grain synchronization is also managed. SPEAR can notably suggest a way to partition the different processes on different RU and to manage their synchronization. However, we first focus on dynamic coarse grain synchronization between the internal main memory and the RU. Main memory access decision is done dynamically and triggered on data blocks. SPEAR plays a part with programming the DMA unit (Figure 4) through parameters definition.

SPEAR, as a high level specification and programming tool, handles important information (memory allocation optimization, data reorganization, process pipelining, ...) which have to be furnished to lower levels for synthesis and feedbacks. Feedbacks serve as a way to provide quick evaluation on human driven decisions. Synthesis of course serve to finalize the design with the implementation on the chip.

To flow down from this high level specification to the architecture and then physical levels, we use a common high level synthesis intermediate description Control Data Flow Graph (CDFG) defined for our specific purpose. The global CDFG, as the output of SPEAR, is elaborated thanks to an API build through a Squeak environment called Platypus [8]. The global CDFG is the representation of the complete application mapped and scheduled on a particular RU. This CDFG includes the connectors mentioned above and some elementary CDFG representing the computation processes written in C language. The front-end analysis module of CASCADE [9] (being part of the presented toolset) elaborates those CDFG.

There is a strong interaction between SPEAR and CASCADE tools since the latter provides the library of functions (elementary CDFGs) necessary to elaborate the global scheduling of the application. There is also a strong link between CASCADE and MADEO since MADEO evaluates the elementary functions constraints. These evaluations are used by CASCADE heuristics-based

optimization tool to furnish the best implementation regarding occupied spaces, delays, consumption ...

The spatial design flow is not straight forward but must be seen as a succession of contributions coming from lower and upper levels to the high specification level called SPEAR. The elaboration of a design through human driven approach leads to successive attempts to finalize the right entry point to synthesis tools.

4.2 Retargetable architectural and physical synthesis

• Execution framework

Architectural efficiency of the RU is based on a balance between local computing power and data availability. As explained above, the pipeline-based mechanism allows to manage either data circulation, and the balance of communication and execution delays. The RU is seen as a Load-Execute-Store processor operating on data chunks defined as a queue of DMA transfer blocks, address computations and transfers are executed in parallel with the execution kernel based on two bounded programs.

The RU becomes a coarse grain processor which programs can be produced by rewriting tools with seamless access to high level data.

• Kernel representation

An important challenge is the adaptation of the kernel program to different reconfigurable units.

The program is expected to be produced by external tools/compiler in the form of a hierarchical graph embedding control structures and operations with dependencies denoted by edges (CDFG). The adaptation will result in a similar graph carrying nodes denoting hardware primitives known to exist in a target architecture (low level CDFG).

Reconfigurable architecture are specified using a generic model extrapolated from fine grain FPGAs (Madeo [10]) and an extensible set of classes for resources (LUT, operators, memories, ...). Process based structuration is allowed, with access to local memories too.

This platform central layer is also where interoperability takes place. Due to its semi-formal representation of data structures, a STEP-based³ set of tools (Platypus mentioned above) produces automatically API to read/write CDFGs, visitors, in different common syntax. This is critical for quick evolution of models shared between different actors, as well as application language support.

• Program transformations

An expected good result is also that CAD tool

³ STEP is an ISO-10303 standard for data exchange.

designers will be able to setup transformation tools at syntax level, hardware primitive level, or architecture synthesis tools.

Beside imperative languages such as C, it is expected that wider arithmetic support will be made available, and also type inference over the CDFG due to the symbolic capabilities of this format. Source languages include formal and object-oriented languages.

In the case of Morpheus, two back-ends are proprietary tools in which the LL-CDFG will be rewritten, while one back-end allow architecture description on fine grain resources.

5 Conclusion

The innovation brought by each part of the presented toolset is expected to contribute to the target objectives achievement (Molen approach, mapping approach and synthesis regarding productivity and portability, formal specification regarding productivity, dynamic control regarding flexibility, etc).

The presented approach, beyond the interest of each part, is interesting for its integration aspect. On each interaction interface, this integration presents an advantage for heterogeneous reconfigurable architecture programming.

For instance, the interoperability between the Molen approach and the proposed Spatial Design environment permits to efficiently explore solutions for the partitioning between an implementation on processor or reconfigurable units thanks to the feedback that can be quickly provided.

The complementarities of the Molen approach performing a static design-time scheduling and the dynamic reconfiguration control managed at run-time by the proposed RTOS permits to benefit from a two-step optimized scheduling.

Dynamic reconfiguration naturally benefits from formal specification for a perfect reliability.

Finally the tight interaction within the Spatial Design environment between task mapping, graph building and synthesis is expected as a decisive differentiator in term of design productivity and implementation efficiency.

References

- [1] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G.K. Kuzmanov, E. Moscu Panainte, The Molen Polymorphic Processor, pp.1363- 1375, *IEEE Transactions on Computers*, November, 2004.
- [2] W. Haas, S. Gossens, U. Heinkel, "Semantics of a Formal Specification Language for Advanced Design and Verification of ASICs (ADeVA)", *11th E.I.S.-Workshop, Erlangen*, April 2003.

- [3] A. Schneider, T. Bluhm, T. Renner, U. Heinkel, J. Knaeblein, R. Zavala, "Formal Verification of Abstract System and Protocol Specifications", *30th IEEE/NASA Software Engineering Workshop, Columbia MD*, April 2006
- [4] U. Heinkel, T. Dinkel, S. Schock, T. Schlichter, C. Haubelt, J. Teich, "Comparison of Techniques for the Automatic Verification of ADeVA Specifications", *Dresdner Arbeitstagung Schaltungs- und Systementwurf, Dresden*, April 2005.
- [5] A. Thomas, J. Becker, U. Heinkel, K. Winkelmann, J. Bormann, "Formal Verification of a Sonet/SDH Framer", *GI/ITG/GMM Workshop Methoden und Beschreibungs-sprachen, Kaiserslautern*, February 2004.
- [6] A. Demeure, Y. Del Gallo *An Approach for Signal Processing Design In SAME, System On Chip Session*, 1998
- [7] E. Lenormand, G. Edelin. "An industrial perspective : A pragmatic high end signal processing design environment at Thales", *SAMOS 2003*
- [8] <http://cassoulet.univ-brest.fr : 8000/squeak>
- [9] <http://www.criticalblue.com>
- [10] <http://as.univ-brest.fr/>