A Compilation Framework for a Dynamically Reconfigurable Architecture

Raphael David[1], Daniel Chillet[1], Sebastien Pillement[1], and Olivier Sentieys[1,2]

[1] LASTI - University of Rennes I, 6, rue de kerampont, F-22300 Lannion, France
name@enssat
[2] INRIA / IRISA, Campus de Beaulieu, F-35042 Rennes cedex, France
sentieys@irisa.fr

Abstract. In addition to high performance requirements, future generation mobile telecommunications brings new constraints to the semiconductor design world. In order to associate the flexibility to the high-performances and the low-energy consumption needed by this application domain we have developed a functional level dynamically reconfigurable architecture, DART. Even if this architecture supports the processing complexity of the UMTS while allowing the portability of the devices and their evolutions, another challenge is to develop efficient high-level design tools. In this paper, we discuss about a methodology allowing the definition of such development tool based on the joint used of compilation and behavioral synthesis schemes.

1 Introduction

Moreover the high performance requirements inherent to multimedia processings or to access technics such as the W-CDMA (Wide-band Code Division Multiple Access), a data processing sequence of third generation brings new constraints to the semiconductor design world. In fact, the success of the Universal Mobile Telecommunication System (UMTS) will be linked to a greater flexibility of the standard than that of the current generation mobile networks such as the Global System for Mobile Communication (GSM) or the north American Interim-Standard(IS)-95.

Hence, UMTS must be able to support various standards and algorithms, for each kind of processing, and their evolutions as well as the integration of new services which still have to be imagined. For example, a speech signal can be coded according to the GSM norm with an Enhanced Full Rate (EFR) coder but also with a more powerful and adaptative AMR (Adaptative Multi Rate) coder, which is recommended for third generation telecommunications. Moreover, from an architectural point of view, a multimedia terminal will successively have to ensure the execution of very different applications in terms of calculation and data access patterns, and which handles various data types. For example, a Viterbi coder working at the bit-level could follow an MPEG coder working on 8-bit data. These processing modifications are then much more problematics

M. Glesner, P. Zipf, and M. Renovell (Eds.): FPL 2002, LNCS 2438, pp. 1058–1067, 2002.

since it will be necessary, in order to be efficient, to dynamically adapt the architecture to these changes.

Traditionally, signal applications are implemented in hardware or in software. Hardware implementations are based on the use of dedicated piece of silicon which are optimized for one application (ASIC) or eventually one set of applications (ASIP). These solutions allow to support the processings in a very efficient way and the designer may optimize its design in area, time, power or even a combination of these three parameters. On the other hand, software implementations are not limited to few applications since they use a programmable processor which may support every kinds of processing. This flexibility is however obtained at the price of a large amount of energy and performance waste since, to be generic, a programmable processor has to integrate a lot of resources that will, most of the time, not be used during the execution.

Between these two kinds of implementations, a third way has been proposed by FPGAs. This kind of architecture allows to optimize the circuit for one type of application but, if the processing has to be changed, the circuit can be reconfigured, in some milliseconds, to be fitted to these new computation patterns. This solution allows new trade-off between performance, energy consumption and flexibility. However, even if it can be attractive for numerous applications, its flexibility (limited by the time of the reconfiguration) and its performances (limited by the extra-cost introduced for the genericity of the architecture) prohibits its use for high complexity tasks or for those that successively have to execute different calculation patterns. Hence, the concept of reconfigurable architectures has evolved in order to allow some far more quick reconfigurations while offering a very high-level of performance and consuming very few energy [1],[2].

Within the scope of the next generation mobile telecommunication systems, we have developed the DART architecture that is reconfigurable at the functional level. It deals with previously mentioned constraints by associating high-performance, flexibility and low-energy consumption [3]. This architecture will be quickly described in the next section in order to exhibit its programming model. Since the calculation power of DART will be exploited only if efficient high-level design tools can be developed, we defined a simple programming model during the design of DART. The development flow, based on the joint use of compilation and behavioral synthesis schemes, will be presented in the section 3. This paper will finally be concluded by discussing about the work in progress and the work to come.

2 The DART Architecture

Since DART is to be embedded in portable multimedia devices, it has to deal with very different applications in terms of granularity, calculation patterns or timing constraints. Hence, this architecture has been broken up into clusters that may work independently the one with the others. This allows the partitioning of the application into distinct tasks (image processing, speech coding, W-CDMA)

which may be computed concurrently on different clusters. In order to ensure the cluster independence, each one of them have its own controller and its own storage resources. Thanks to this hierarchical organization, some time and energy efficient interconnect networks can have been designed inside the clusters and between them.

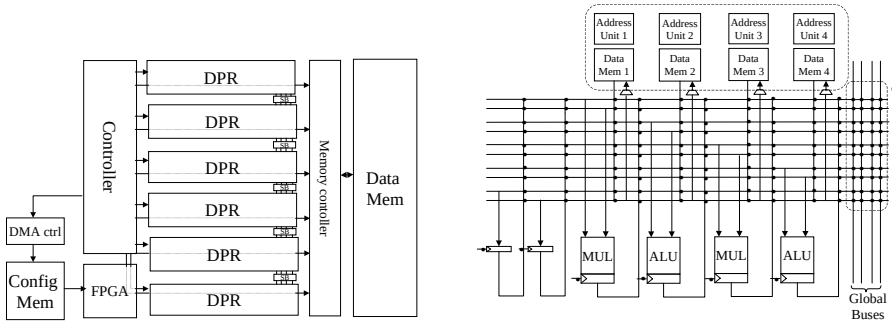

Fig. 1. Cluster (a) and DPR (b) architecture of DART

In the highest abstraction level of DART, a task controller have to distribute the tasks (and hence the configurations) to the clusters according to urgency and resource availability constraints. These reconfigurations are translated, at the cluster level (figure 1(a)), into modifications of the cluster controller program et by cluster internal data memory loading. The cluster controller program is constituted of configuration sequences, dynamically distributed towards two kind of targets: Reconfigurable DataPath (DPR) and FPGA. In fact, in order to have some calculation resources able to be in adequacy with every kind of processings we have integrated this two calculation primitives. The FPGA will be used for the bit-level manipulations which can be found in low-level processings such as the channel coding while the DPRs will be used for the arithmetic processings. Each DPR integrates one FPGA and 6 DPRs, as illustrated in figure 1(a). These DPRs are interconnected thanks to a segmented mesh network which allows some flexibility in their interconnection. According to the parallelism degree of the application to be implemented, the DPRs can be interconnected to compute it in a highly parallel fashion, or be disconnected to work independently on different threads.

Each DPRs (figure 1(b)) integrates 2 multipliers and 2 ALUs, interconnected according to a totally flexible interconnect network. These units are dynamically reconfigurable and are working on data stored in 4 local memories on top of which 4 local controllers are in charge of providing the addresses of the data handled inside the DPR. Moreover these 4 memories, 2 registers are also available in each DPR. These registers are particularly useful for data flow oriented applications where the different functional units are working on the same data flow but on

samples delayed from one iteration to the following. In that case, these registers will be used to build a delay chain to share the data in the time and hence, to minimize the number of data memory accesses.

One of the main feature of DART is to support two DPR reconfiguration modes which ensues from the familiar 80-20 rule. This rule asserts that 80% of execution time is consumed by about 20% of a program code, that 20% usually being nested loops. In fact, during this 20% of regular program code a same calculation pattern is used during long periods of time. On the other hand, for the 80% of remaining irregular code, the calculation pattern is changing very often, eventually at each cycle. From these two types of processing, have resulted two types of reconfiguration.

2.1 Hardware Reconfiguration

These configurations are used for a long time and are adapted to the regular processings such as loop kernels. Since their modifications are very occasional, they can require a large amount of data without disturbing the execution of the entire processing. Hence, in that case, a total flexibility of the DPR will be ensured thanks to 4 52-bit instructions.

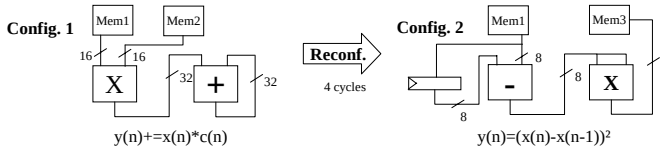

Fig. 2. Hardware reconfiguration

This kind of configuration can for example be illustrated by the figure 2. In this figure, the datapath is optimized at first in order to compute a filtering based on Multiply-ACcumulate operations. Once this configuration has been specified, the computation model is of dataflow type and no other instruction memory readings are done during the filtering. At the end of the computation, after a reconfiguration step which needs 4 cycles, a new datapath is specified in order to be in adequacy with the calculation of the square of the difference between x(n) and x(n -1). Once again, no control is necessary to conclude this computation.

2.2 Software Reconfiguration

For irregular processings that need to change the configuration of the DPRs at each cycle without particular order and in a non repetitive way, a software reconfiguration is used. In order to be able to reconfigure the DPR in one cycle with an instruction of reasonable size, their flexibility has been limited. It has been decided to adopt a calculation pattern of *Read-Modify-Write* type, such as

that of conventional DSP. In that case, for each operator useful for the execution, the data are red, computed, then the result is stored in the memory associated with this operator at each cycle.

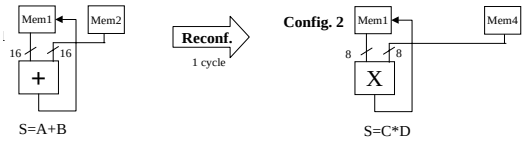

Fig. 3. Software reconfiguration

This software reconfiguration thus concerns only the functionality of the operators, the size of the data and their origin. Thanks to these flexibility limitations, the DPR may be reconfigured at each cycle with only one 52-bit instruction. This is illustrated on figure 3 which represents the reconfiguration needed to replace an addition on data stored in the memories 1 and 2 by a multiplication on data stored in memories 1 and 4.

2.3 Performance and Energy Efficiency of DART

A DART cluster have been synthesized with the *Synopsys* design tool framework with a 1.95V ST Microelectronic 0.18μm technology. Thanks to the DART simulator (see §3.5), the overall energy consumption of the architecture is evaluated from the activity of the different architectural modules (functional units, memories, glue-logic, ...) and their average energy consumption per access, estimated by *Design Power* with 10.000 test vectors randomly generated.

The synthesis leads to an operating frequency of 130 MHz. Running at 130MHz, DART is able to provide up to 1.56MMACS/cluster while consuming 0.34W which can be translated in a worst case energy efficiency of 9.2MIPS/mW. In fact, since an instruction includes an address generation, a memory access and up to 2 operations per multiplier (1 shift and 1 multiply(+saturate)) or 3 operations per ALU (2 shift + 1 ALU operation), this figure is equal to 32MOPS/mW @10.9GOPS. Practically, previous implementations from the application domain (DCT, autocorrelation, complex despreading [4]) show that DART provide between 11.6 and 16.7 MIPS for each mW consumed.

3 Development Flow

To exploit the computation power of DART, the conception of an efficient development flow is the key to enhance the status of the architecture. Hence, we develop a compilation framework based on the joint use of a front-end allowing the transformation and the optimisation of a C code, a retargetable compiler and

a behavioral synthesis tool, as described in the figure 4. As in most development methodologies for reconfigurable hardware, the key of the problem has been to distinguish the different kinds of processing. In fact, this approach had already been used with success in the PICO (Program-In Chip-Out) project developed at HP labs in order to distinguish regular codes, implemented in systolic array, and irregular ones, executed in a VLIW processor [5]. Other related works such as the Pleiades project [6] or GARP [7] are also distinguishing regular processings and irregular ones to implement massively parallel processings on circuits respectively reconfigurable at the functional and at the gate level.

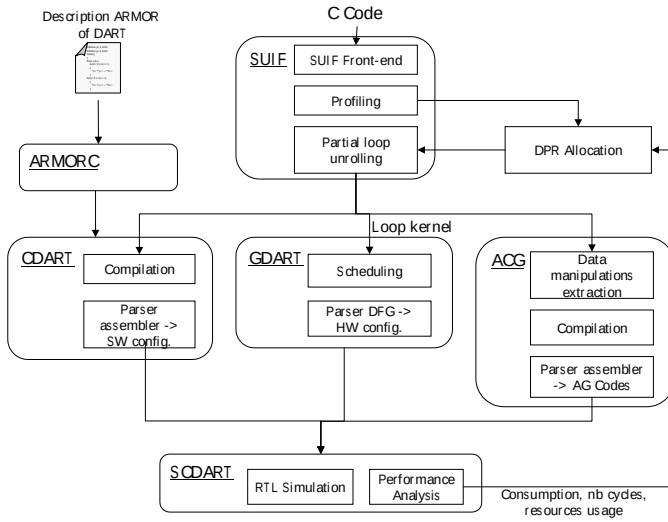

Fig. 4. DART Development flow

The development flow depicted in the figure 4 allows the user to describe its applications in C. These high-level descriptions are first translated into Control and Data Flow Graph (CDFG) from which some automatic transformations (loop unrolling, loop kernel extractions, ...) are done in order to optimize the execution times. After these transformations, the distinction between regular codes, irregular ones and data manipulations permits to translate, thanks to the compilation and the architectural synthesis, a high level description of the application into binary executable codes for DART.

3.1 The Suif Front-End

The Front-end of this development flow is based on the Suif [8] framework developed at Stanford. The main goal of this module is to generate a CDFG from which other modules can operate. Moreover, this module has to extract the loop kernels inside the C code. In fact, these loop kernels will be mapped as Hardware

reconfigurations whereas the rest of the code will be implemented as Software reconfigurations and address generation instructions. For example, the C code of the figure 5(a) have to be translated into the modified C code of the figure 5(b) in order to bring to the fore the three kinds of processings of the task:

1. The irregular codes (x(n)=A.B; h(0)=B+D;) must be translated in Software reconfigurations by cDART (see §3.2);
2. The data manipulations (Mem1=x(i); Mem2=h(N-i-1); y(n)=Mem3;) must be translated in address generation instructions by ACG (see §3.4);
3. The loop kernels (Mem3=Mem3+Mem1*Mem2;) must be translated in Hardware reconfigurations by gDART (see §3.3).

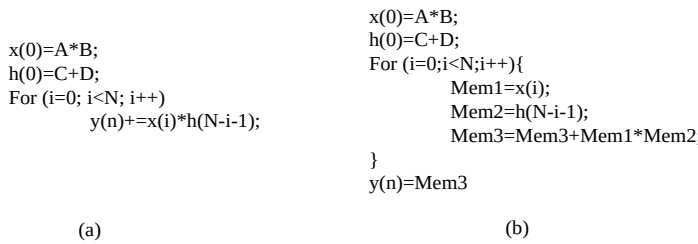

Fig. 5. Suif entry C code (a) and Modified C code (b)

Moreover these code extractions, Suif is also in charge of unrolling the nested loops. In fact, the only information that is transmitted to gDART is the loop kernel description and hence, this tool does not have any information about the parallelism degree of the application. Thus, to exploit the parallelism of an application, it must be extracted before the Hardware code generation. This is notably done by unrolling the loop. This unrolling depends on a module in charge of specifying the number of DPRs allocated to the application in the cluster. According to this information, Suif is thus able to establish the number of calculation units available in the cluster and so the unrolling order that can be realized. For example, on the previous pseudo FIR filter code, if 4 DPRs are allocated to the application, the loop will be unrolled by a factor of 8, in order to extract 8 multiplications and 8 additions.

3.2 Software Code Generation

In order to generate the Software reconfiguration instructions, we have integrated a compiler, cDART, into our development flow. This tool have been generated thanks to CALIFE [9] which is a retargetable compiler framework, developed inside the COSI project of the IRISA/INRIA, aiming at providing a compilation tool for a processor described in the ARMOR language.

In order to generate cDART, we first have to model DART in the ARMOR language. The main purpose of this step is to provide the informations needed

by the flexible code generator. These informations come up from the inherent needs of the three main compiling activities which are the code selection, the allocation and the scheduling, and from the architectural mechanisms used by DART. It has to be noticed that the software reconfigurations implies some limitations about the DPRs flexibility and hence, the architecture subset concerned by this reconfiguration is very simple and orthogonal since it is constituted by 4 independent functional units working on 4 memories in a very flexible manner, i.e. there is no limitations about the use of the instruction parallelism.

The next step to generate cDART have been to translate the DART ARMOR description into a grammar able to analyze expression trees in the source code, thanks to the ARMORC tool. Finally, to built the compiler, the CALIFE framework allows us to choose the different compilation passes (e.g. code selection, resource allocation, scheduling, ...) that have to be implemented in cDART. In CALIFE, while the global compiler structure is defined by the user, module adaptations are automatically performed by the framework. Thus, thanks to this tool we have been able to generate a compiler optimized for DART, since each compiler structure efficiency can easily be checked and new compilation passes can quickly be added or subtracted from the global compiler structure.

3.3 Hardware Code Generation

If the software reconfiguration instructions can be generated thanks to classical compilation schemes, the hardware reconfiguration instructions have to be generated according to more specific synthesis tasks. In fact, as it has been said before, a hardware reconfiguration can be specified by a set of instructions that exhibit the DPRs structure. Hence, the tool developed, gDART, has to generate a DataPath in adequacy with the processings to be implemented during a loop. The entry point of this tool is thus a Data Flow Graph (DFG) representing the computation pattern that must be synthesized. The parallelism exhibition has been done during the Suif transformations and the only task that must be done by gDART is to find the datapath structure allowing the DFG implementation.

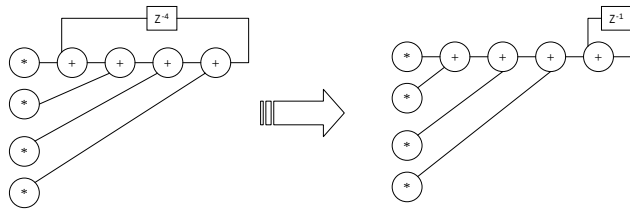

Fig. 6. Critical loop reduction

However, even if the DPR architecture is very flexible, it has some limitations. The main constraint to schedule the DFG is that it is impossible to share

intermediate results between several cycles. Thus, the critical loops must be computed in one cycle. This problem can be illustrated by the FIR filter DFG represented on the figure 6 and mainly concerns the accumulations. In this particular case, the solution is to transform the DFG in order to reduce the critical loop timing to an only one cycle by swapping the additions. This solution can be generalized by swapping the operations of a critical loop according to the associativity and distributivity rules associated to the operators.

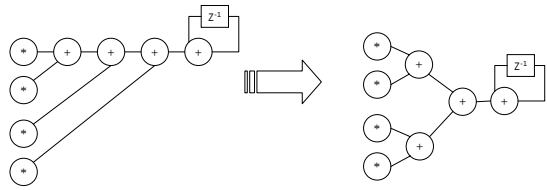

Fig. 7. DFG optimization

Moreover, the figure 7 shows a quite high latency of execution pipeline and if the iteration number of innermost loop is small, this feature is quite prejudicial. Hence, another scheduling algorithm has been developed in order to exploit the implementation parallelism and to obtain the DFG represented on figure 7. Thanks to this new algorithm, execution pipeline can now be filled in only 4 cycles while 5 cycles was previously needed. If unrolling factor is extended to 8, latency will be decreased more drastically from 10 cycles to 5 cycles.

3.4 Address Code Generation

if gDART and cDART allow the definition of the datapath, they do not take into consideration the data supplying. Hence, a third tool, ACG (Address Code Generator), has been developed in order to obtain the address generation instructions which will be executed on the address generators of each DPRs [3]. Since the address generators architecture is similar to a small RISC (a small controller manage a datapath built around a register file and an ALU), the generation of these instructions can be done by classical compilation steps thanks to CALIFE. The entry of the compiler is this time a subset of the initial entry code (Mem1=x(i), Mem2=h(N-i-1) and y(n)=Mem3 on the figure 5) which corresponds to the data manipulations.

3.5 Simulation

The different configurations of DART can be validated thanks to a bit-true and cycle-true simulator (SCDART), developed in SystemC. This simulator also generates some informations about the performance and the energy efficiency of the implementation realized. In order to have a good relative accuracy, the DART

modeling have been done at the Register Transfer Level and each operator has been characterized by an average energy consumption per access. These energy consumption figures are coming from logical synthesis realized with *Synopsys* and from gate level consumption estimations realized via *Design Power* with randomly generated test vectors.

4 Conclusion

In this paper we have presented a dynamically reconfigurable architecture, which aim at supporting the next generation telecommunication constraints, and its associated development methodology. We have first verified that this architecture is in adequacy with our application domain then we have focused on the development tools. Since this architecture exhibits a quite simple execution model we have been able to partition the methodology into reasonable complexity tasks that are issued from compilation and behavioral synthesis tools. Hence, by discussing about 5 modules, from which some are still in development (ACG, cDART), we have described the overall methodology. The next step in our study is thus to complete the specification of this development tools, to validate it, and to study the system architecture of DART that will support a Real-Time Operating System.

References

[1] Rabaey, J.: Reconfigurable Processing : The Solution To Low-Power Programmable DSP. In: IEEE International Conference on Accoustics, Speech and Signal Processing (ICASSP). (1997)
[2] Hartenstein, R.: A Decade of Reconfigurable Computing : A Visionary retrospective. In: Design Automation and Test in Europe (DATE). (2001)
[3] David, R., Chillet, D., Pillement, S., Sentieys, O.: DART : A Dynamically Reconfigurable Architecture dealing with Next Generation Telecommunications Constraints. In: RAW. (2002)
[4] David, R., Chillet, D., Pillement, S., Sentieys, O.: Mapping Future Generation Mobile Telecommunication Applications on a Dynamically Reconfigurable Architecture. In: ICASSP. (2002)
[5] Schreiber, R., Aditya, S., al.: PICO-NPA : High-Level Synthsis of NonProgrammable Hardware Accelerators. Technical Report HPL-2001-249, Hewlett-Packard Labortories (2001)
[6] Wan, M.: Design Methodology for Low Power Heterogeneous Digital Signal Processors. PhD thesis, Berkeley Wireless Design Center (2001)
[7] Hauser, J.: Augmenting a microprocessor with reconfigurable hardware. PhD thesis, University of California, Berkeley (2000)
[8] Wilson, R., al.: SUIF : An Infrastructure for Research on Parallelizing and Optimizing Compilers. Technical report, Computer Systems Laboratory, Stanford University (1994)
[9] Charot, F., Messe, V.: A Flexible Code Generation Framework for the Design of Application Specific Programmable Processors. In: International Symposium on Hardware/Software Co-design (CODES). (1999)