A model driven based CAD tool for mppSoC design

M.Ammar, M.Baklouti and M.Abid
CES, National Engineering School of Sfax, Sfax, Tunisia
manel.ammar@ceslab.org, mouna.baklouti@ieee.org

Abstract— **The mppSoC architecture is a parametric massive parallel architecture which aims to support modern data-intensive applications. In this paper, we explore different concepts that facilitate the design of such system from the specification to the application and the architecture implementations.**

I. INTRODUCTION

Current data-intensive processing applications are various including both multimedia and high complex numerical and scientific applications. These applications share many common charactestics which are mainly: the high complexity of the data structures, the intensive parallelism available in the application functionalities, the high data storage and the computational requirements. The data-parallel structure of the data-intensive applications exposes the logical parallelism present in the algorithm. An important part of handling the processing needs of these modern applications refers to finding the best execution platform which guarantees an intensive physical data-parallelism. The data-parallel behavior of such applications is well realized on *Single Instruction Multiple Data* (SIMD) computer architectures. These systems are well suited to the modern data-intensive applications as they can afford high performance computations. But most of them are application-specific SoC which lack flexibility: changing a SoC configuration may necessitate extensive redesign. In order to address these issues, an innovative SIMD machine is proposed [1]. This machine is named mppSoC which stands for massively parallel processing System-on-Chip. mppSoC has a parallel architecture that results from an assembly of different components and may be implemented on a single chip. In addition, mppSoC proves very fruitful in massively parallel applications domain. However, the design and implementation of such systems become critical due to their long design and development cycles. In fact, the mppSoC design is facing today a strong pressure on reducing time-to-market while the complexity of this system has been increasing. In this climate, there are various places where tools facilitating SoC design can be successfully used before the SoC physical realization. Nowadays, Computer-Aided Design (CAD) tools are imperative to automate complex SoC design and reduce the time-to-market. They are also used to automate the Design Space Exploration (DSE) of alternative solutions. A part from these tools, design reuse is required to improve the design productivity. In fact, IP (Intellectual Property) reuse and platform-based design [2] are used to maximize the reuse of pre-designed components and to allow the customization of the system according to system requirements. Design abstraction offers also a possible solution to address the above issues concerning the time-to-market and complexity dilemma. Embedded systems are designed at a high abstraction level with the MARTE (Modeling and Analysis of Real-Time and Embedded systems) [3] profile. Different design steps are guided using Model-Driven Engineering (MDE) [4] approach. It is a well structured approach that has been introduced to raise the design abstraction level and to reduce design complexity. It stresses the use of models in the embedded systems development life cycle and argues automation via model transformation and code generation techniques. To facilitate and accelerate the design of an mppSoC configuration dedicated to a given parallel application, our contribution consists in proposing an mppSoC CAD tool. This tool uses an extension of the MARTE profile for high-level specifications of SoC applications and architectures. The resulting abstract models are afterwards deployed towards specific implementations. These implementations are finally generated based on automatic exploration step.

In the following sections, we present the mppSoC CAD tool and the two meta-models created to guide the exploration. We also introduce the DSE methodology defined to automatically perform the exploration step. Finally, we present an experiment to explain and validate our proposition.

II. MPPSOC ARCHITECTURAL MODEL

MppSoC is an IP-based massively parallel architecture [5]. It is composed of a number of processing elements (the PEs) working in perfect synchronization. A small amount of local and private memory is attached to each PE. Every PE is potentially connected to its neighbors via a regular network. The whole system is controlled by an Array Controller Unit (ACU). The configurable neighborhood interconnection network is implemented to assure inter-PE communications depending on its configurable topology (Mesh, Torus, Xnet, Linear array and Ring). Furthermore, each PE is connected to an entry of mpNoC, a massively parallel Network-on-Chip that potentially connects each PE to one another, performing efficient irregular communications. The mpNoC is integrated to manage point-to-point communications through different types of connections. In fact, the mpNoC includes a configurable router which can be of different types (Shared Bus, Crossbar, Delta MIN (omega, baseline and butterfly)). The mpNoC can perform different communication modes (PE-PE, PE-ACU, PE-Device) since it

contains a mode manager. The mppSoC design approach aims to define an mppSoC configuration adapted to a given application. This customization is achieved with the parameterization as well as the extensibility and the configurability of the architecture. Moreover, mppSoC is parametric in terms of the number of PEs as well as the memories' sizes. It has three configurable aspects: the processor design methodology, the integrated neighboring network's topology and the mpNoC interconnection network's type. The processor design methodology is the manner to assemble processor IPs to build the SIMD system. We distinguish two methodologies: processor reduction and processor replication. The former consists on reducing an available open-core processor in order to build a processing element with a small reduced size. The replication methodology consists on implementing the ACU as well as the PE by the same processor IP so that the designing process is faster.

We have briefly demonstrated that the mppSoC is implemented as a flexible, parametric and configurable architecture. The designer can model an mppSoC configuration and a data-parallel application using our co-design tool. The mppSoC modeling concepts are explained in the following section.

III. A MARTE EXTENSION FOR THE CO-SPECIFICATION OF MPPSOC

To deal with the complexity of mppSoC design, two meta-models are developed to model both application and architecture.

A. Data-parallel application meta-model

The high-level specification of the mppSoC data-parallel application is based on the MARTE profile, the Array-OL language [6] and an mppSoC application meta-model.

The MARTE profile allows the modeling of data-parallel applications in an effective manner, but specific mppSoC details are needed to be represented at a high abstraction level. In fact, mppSoC specific instructions are used to manage communication and control in the architecture. These two mechanisms cannot be modeled based on the MARTE profile. To satisfy mppSoC application needs and to facilitate DSE, we introduce new stereotypes that help the designer to model well structured data-parallel applications.

To model mppSoC specific instructions, two stereotypes called *CommunicationTask* and *ControlTask* are proposed. The *CommunicationTask* stereotype allows the modeling of communication between the mppSoC components. The tagged value *Type* can take two values Load or Store indicating the sending or the reception of data flows respectively. In order to specify the communication mode, a tagged value named *Mode* is associated to the *CommunicationTask* stereotype. It can take five values according to the selected mode: 0 (PE-PE), 1(ACU-PE), 2 (PE-ACU), 3 (PE-Device) or 4 (Device-PE).

To facilitate the DSE, it is recommended to model the data-parallel application partition using a specific *DataParallelPartition* stereotype. This stereotype represents a task or a number of tasks which will be repeated. Making the repetitions independent and therefore parallel by construction, allows the data-parallelism expression. Three tagged values are associated with the *DataParallelPartition* stereotype. The first named *Priority* is used to indicate the order of the partition execution. It is needed when the application is composed of many data-parallel computations. Two other tagged values are defined to be used in the DSE step. The first, called *WorstExecutionTime*, is used to specify the maximum time needed for the data-parallel partition execution. The second, labeled *MemorySize*, expresses the needed instruction memory size occupied by this partition.

In a modeled data-parallel application, there are three kinds of tasks: elementary, compound and repetition. An elementary task is a black box. A compound task is a dependence graph, whose nodes are tasks connected via their ports, allowing the expression of task parallelism. A data-parallel partition is a repeated task which can be elementary or compound. Two stereotypes *CompoundTask* and *ElementaryTask* are used to model elementary and compound tasks respectively. A compound task contains elementary or other compound task(s). *WorstExecutionTime* and *MemorySize* tagged values are associated with the *CompoundTask* and the *ElementaryTask* stereotypes denoting time and memory space needed to execute these tasks. A repetition is in general expressed using the Shaped stereotype of the RSM package. The number of repetitions must be specified using the tagged value Shape. In our case, some shapes are needed to be parametric. Some ports' shapes and number of repetitive tasks are related to the number and the arrangement of PEs. Since this number is parametric and is specified after the DSE step, the repetitions must be themselves parametric. For this purpose, a tagged value named *Parametric* is added to the Shaped stereotype.

B. mppSoC architecture meta-model

The modeling of mppSoC configurations relies on the use of UML and the MARTE profile. A need for a MARTE extension is felt as all possible configurations share some components that are repeated through all designs. These components must be specified using their own parameters in order to facilitate the design space exploration step.

As we have seen in the second section, specific vocabulary is associated with the mppSoC architecture. PE, ACU, PE local memory, ACU instruction memory, ACU data and instruction memory and mpNoC are common vocabularies that can present a domain-specific modeling language for the mppSoC architecture. Because facilitating design space exploration and aiming to obtain optimal solutions are our primary goals, mppSoC architecture meta-model is defined. This meta-model extends some notions that are defined in the MARTE Hardware Resource Modeling (HRM) package. A number of stereotypes are added to model hardware mppSoC resources. These stereotypes are: ACU, PE, Local memory, Instruction memory, mpNoC and ACU memory. The *ACU* stereotype is used to model the unique control processor. Every processing unit in the grid of processors is defined thanks to the *PE* stereotype. We know that an elementary processor can be complete or reduced. For this purpose a tagged value named *isComplete* is added to this stereotype. It can take two values:

true if the designer chooses to implement a complete processor or false if he wants to integrate a reduced one. With each PE a local memory must be associated to manage local computations. This type of component is denoted through the *Local memory* stereotype and the data size is specified across the *adressSize* tagged value. The same thing is applied to the ACU memory. The designer has the choice to implement just one memory that contains instructions and data; in this case the *ACU memory* stereotype is used. He can also link the ACU with two memories. The first memory contains the program and is stereotyped Instruction Memory. The second stores data and is stereotyped *Data memory*. The last stereotype that has to be depicted is the *mpNoC* stereotype. It mainly gives a formal way to model the irregular interconnection network and to include it clearly in the design.

C. Deployment

To generate an executable low level model, the elementary modeled components should be associated with an existing implementation based on the provided mppSoCLib. The deployment enables to precise a specific implementation for each elementary concept among a set of possibilities. It concerns the processor IP, the instruction memory, the mpNoC interconnection network and the I/O device IP if it exists. At this stage the binary data-parallel program is specified as the memory initialization file of the main instruction memory. In fact, in our case we deal with a single data-parallel program so no mapping of tasks needs to be performed. Thus, the mapping of the application to the hardware (HW) architecture is systematic.

D. Architecutre and application code generation

Our tool allows the generation of the executable code depending on the processor type that is chosen automatically via the DSE tool. Two transformation chains are so defined. The first generates extended assembly for the miniMIPS processor. The second transforms mppSoC language to extended C code for the NIOS and OpenRisc processors. The synthesizable VHDL implementation is generated depending on the configuration specified by the DSE step.

IV. HIGH LEVEL DESIGN SPACE EXPLORATION

The suggested approach allows dealing with the increasing parallelism and the huge number of HW resources. Furthermore, it allows decreasing exploration costs by performing a high-level exploration. This approach is flexible and helps the designer to choose the best configuration at an early stage of the mppSoC design flow. The MDA-based methodology focuses on models rather on implementation and supports a fast DSE in the early design steps where modeling decisions can lead to superior improvements. The proposed DSE methodology necessitates the high-level specification of the parts needed to build the embedded system. Once the user models the application algorithm on the one hand, and the HW architecture on the other hand, a high-level analysis of the application and the architecture is performed in order to do a multi-objective DSE. Our goal is to generate an adequate architecture that meets the application needs and responds to a number of performance criteria mainly: execution time, energy consumption, memory size and logic resources. The data-parallel application models are automatically analyzed in term of structure. Structural analysis is performed using class diagram and composite structure diagram. Information about classes, ports, data types and other are extracted. The initial application model is a black box representation of a set of stereotyped communicating tasks via different ports. Memory size and execution time needed for each task to be run are either specified using the tagged values *WorstExecutinTime* and *MemorySize* or calculated using an engine that is integrated to the framework. This engine takes as input application IP's. These IPs must be mapped to the real instruction set of available processors in order that execution time and memory size may be obtained for each task. To estimate the data memory size, all data flows are analyzed in the application's diagrams and in the reused IPs. The program memory size is estimated by listing the instructions for each task. An mppSoC exploration library is available. It contains different parameters needed to perform the DSE such as instruction set of processors, mppSoC HW components performances, etc. The current framework takes several design decisions: it selects the number of PEs in the mppSoC, data-parallel partitions between processors, maps processors to a communication structure (regular communication network or mpNoC) and selects the application parameters such as ports shape and number of data-parallel task repetitions. Therefore, it automatically explores the platform, the mapping and the application domains. Our ultimate objective is to propose a multi-objective resolution approach that permits to find the best candidate solutions for a given data parallel application. It is multi-objective because we have mainly four performance criteria to optimize which are the total execution time, the total energy consumption, the data and program memory size and finally the total area. A mathematic model is then necessary to calculate the objective functions related to those performance criteria. For each criterion we have attached one objective function to be minimized and a number of constraints. The objective function related to the area is calculated as follows. The total area resources occupation is the summation of the area occupied by each resource.

$$A = \sum a_i \times shape_i \qquad (1)$$

The area estimation is based on resources area measurements available in the exploration library. In fact, areas of all the components that can exist in an mppSoC configuration are presented in this library. In this case we are facing a multi-objective problem that needs multi-objective exploration algorithm to be solved. For this purpose the SPEA2 [7] genetic algorithm was selected to deal with such NP-complete problem.

V. EXPERIMENTS

A simple RGB to CMYK color conversion application is chosen to illustrate the design process with the presented mpp-SoC framework. The Cyclone III video development board has been chosen as a target platform to prototype the video processing based mppSoC. It is equipped with an EP3C120 FPGA device which has 119,088 Logic Elements (LEs) and

432 M9K embedded memory blocks. The used software tools are the Quartus II V9.0 that allows the synthesis and the prototyping of the design on the FPGA, and the ModelSim-Altera v.6.4a that allows the simulation and debug of the design. The input video frames are captured by a TRDB D5M camera and then processed by mppSoC and displayed on a 800×RGB×480 pixel TRDB LTM LCD displayer. In this example, each pixel processing should not exceed 10.42 ns in order to assure real-time processing. Therefore, a 800×480 video frame must be processed within 4 ms. To assure parallel reading and writing of data pixels, two devices are used with the mppSoC configuration. In fact, the camera is directly connected to the SDRAM external memory to store pixels. An SDRAM device driver is so needed to assure parallel data transfers from the SDRAM to all PEs. In the same manner, an SRAM device driver allows parallel data writing from all PEs to an SRAM memory connected to the displayer. These devices are provided in the HW mppSoCLib.

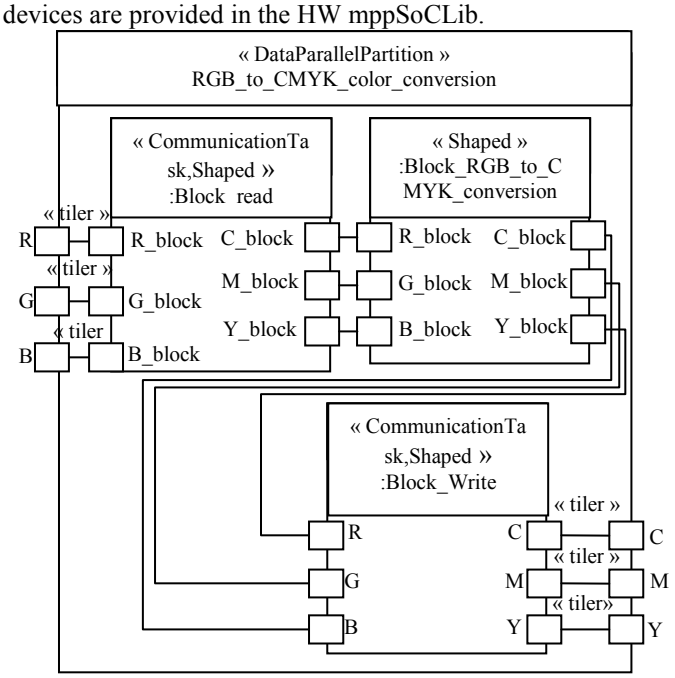

Figure 1. RGB to CMYK color conversion component

As a first step in the design flow, models of the application are defined using the mppSoC application meta-model. Stereotypes described in the section 3 are used to model the data parallelism of the application. Figure 1 presents the RGB to CMYK color conversion component which is responsible of converting RGB input pixels of the captured data to CMYK pixels for printing. Three tasks are performed in parallel representing a data-parallel partition. First, a communication task is employed to read one block then the color conversion algorithm is applied on this block and finally another communication task finds its place to write the resulting converted data. The architecture is also described using the architecture meta-model. Elementary components are firstly modeled then the instantiation mechanism permits to instantiate these components in order to model the main architecture. An example of elementary component is depicted in figure 2. It models the control processor using the *ACU* stereotype. An optimal solution is given thanks to the use of our design space exploration framework. In this case, the DSE defines an mppSoC configuration which includes 16 NIOS PEs communicating via a crossbar based mpNoC. The data memory size of each PE is 24000 bytes.

As a final step of the design flow, an implementation of the optimal solution is automatically generated.

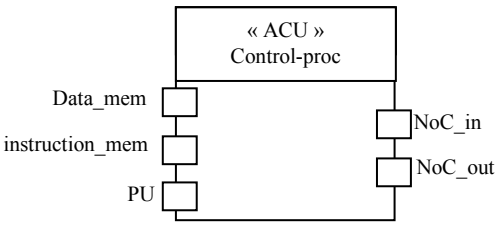

Figure 2. Control processor model

VI. CONCLUSION

A Model-Driven Engineering (MDE) approach for mpp-SoC design was presented. The proposed design flow is composed of five steps: data-parallel application modeling, SIMD SoC modeling, deployment, design space exploration and parallel configuration and application generations. The MDE fundamental notion of transformation between models is used to generate the data-parallel binary as well as a SIMD configuration at register transfer level from their models at a high abstraction level. Classical codesign flows aren't efficient enough to design mppSoC systems composed of a big number of processors. Our new methodology is innovative and it permits to deal with challenges imposing by these systems and embedded systems in general.

REFERENCES

[1] M. Baklouti, "A rapid design method of a massively parallel System on Chip: from modeling to FPGA implementation", Available in http://tel.archives-ouvertes.fr/tel-00527894/en/

[2] A. Sangiovanni-Vincentelli, L. Carloni, F. D. Bernardinis, and M. Sgroi, "Benefits and challenges for platform-based design," in Proc. DAC, 2004, pp. 409–414.

[3] L. Rioux, T. Saunier, S. Gerard, A. Radermacher, R. de Simone, T. Gautier, Y. Sorel, J. Forget, J.-L. Dekeyser, A. Cuccuru, C. Dumoulin, and C. Andre, "MARTE: A new profile RFP for the modeling and analysis of real-time embedded systems," in UML-SoC'05, DAC 2005 Workshop UML for SoC Design, Anaheim, CA, June 2005.

[4] D. Schmidt, "Model-driven Engineering," IEEE Computer, vol. 39, no. 2, 2006.

[5] P. Bonnot, F. Lemonnier, G. Edelin, G. Gaillat, O. Ruch, and P. Gauget, "Definition and SIMD implementation of a multi-processing architecture approach on FPGA," in Proc. of DATE, 2008.

[6] P. Boulet, "Array-ol revisited, multidimensional intensive signal process ing specification," INRIA, Tech. Rep. RR-6113.

[7] E. Zitzler and M. Laumanns and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm", Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 2001.