From MARTE to Reconfigurable NoCs: A model driven design methodology

Imran Rafiq Quadri, Majdi Elhaji, Samy Meftali, Jean-Luc Dekeyser

From MARTE to Reconfigurable NoCs: A model driven design methodology

Imran Rafiq Quadri, Majdi Elhaji, Samy Meftali and Jean-Luc Dekeyser
INRIA-LILLE NORD EUROPE - LIFL - USTL - CNRS, LILLE – FRANCE
{FirstName.LastName}@lifl.fr

Due to the continuous exponential rise in SoC's design complexity, there is a critical need to find new seamless methodologies and tools to handle the SoC co-design aspects. We address this issue and propose a novel SoC co-design methodology based on Model Driven Engineering and the MARTE (Modeling and Analysis of Real-Time and Embedded Systems) standard proposed by Object Management Group, to raise the design abstraction levels. Extensions of this standard have enabled us to move from high level specifications to execution platforms such as reconfigurable FPGAs. In this paper, we present a high level modeling approach that targets modern Network on Chips systems. The overall objective: to perform system modeling at a high abstraction level expressed in Unified Modeling Language (UML); and afterwards, transform these high level models into detailed enriched lower level models in order to automatically generate the necessary code for final FPGA synthesis.

GENERAL TERMS: HW/SW Co-Design, Real-Time and Embedded Systems
KEY WORDS: SoCs, NoCs, FPGAs, ISP, MDE, MARTE, UML

1. INTRODUCTION

Since the early 2000s, System-on-Chip (SoC) has emerged as a new methodology for embedded systems design. In a SoC, the computing units (programmable processors, hardware functional units), memories, I/O devices, communication channels, etc.; are all integrated into a single chip. Moreover, multiple processors can be integrated into a SoC (Multiprocessor System-on-Chip, MPSoC) in which the communication can be achieved through Network on Chips (NoCs). These SoCs are generally dedicated to target application domains (such as multimedia video codecs, software-defined radio and radar/sonar detection systems) that require intensive computations. According to Moore's law, rapid evolution in hardware technology doubles the number of transistors in an Integrated Circuit (IC) nearly every two years. As the computational power increases, more functionalities are expected to be integrated into the system. As a result, more complex software applications and hardware architectures are integrated, leading to a *system complexity* issue which is one of the main hurdles facing SoC co-design. The fallout of this complexity is that the system design (particularly software design) does not evolve at the same pace as that of hardware due to issues such as development budget limitations, reduction of product life cycles and design time incrementation. This evolution of balance between production and design has become a critical issue and has finally led to the *productivity gap*. System reliability and verification are also the

other issues related to SoC industry and are directly affected by the design complexity. An important challenge is to find efficient design methodologies that raise the design abstraction levels to reduce overall complexity, while effectively handling issues such as accurate expression of inherent system parallelism: such as application loops; and hierarchy.

Network on Chips is considered as an emerging paradigm for resolving the problems related to current highly integrated complex SoCs (Benini, L., and Micheli, G 2001). A SoC may have tens or hundreds of IP (Intellectual Property) cores with each running at different clock cycles resulting in *asynchronous clocking*. NoCs thus adopt a globally asynchronous, local synchronous (GALS) approach and help to improve the performance: such as throughput; and scalability as compared to other communication structures such as point to point signal wires and shared buses. They are an ideal choice for MPSoC architectures as they allow separation of the *communication* and the *computation* concerns while allowing IP reuse by utilization of standard interfaces.

Currently High Level Synthesis (HLS) (or Electronic System Level) is an established approach in SoC industry. This approach raises the design abstraction level to some degrees as compared to traditional hand written HDL (Hardware Description Languages) implementations. The gap between the high abstraction levels and the low abstraction levels is often bridged using one or several *Internal Representations* (IRs) (Guo et al 2005). The behavioral (algorithmic) description of the system is written in a high level language such as SystemC (OSI 2007) or a similar language, and is then refined into a RTL (Register Transfer Level) implementation using HLS tools. An effective HLS flow and associated tools must be flexible to cope with the rapid hardware/software evolution; and *maintainable* by the tool designers. The underlying low level implementation details are hidden from users and their automatic generation reduces time to market and fabrication costs. However, usually the abstraction level of the HLS tools is not elevated enough to be totally independent from low level details. Normally, the set of concepts related to an IR are generally difficult to handle due to absence of formal definitions of key concepts and their relations. The text based nature of a system description also results in several disadvantages. Immediate recognition of system information such as related to hierarchy, data parallelism and dependencies is not possible; differentiation between different concepts is a daunting task in a textual description and makes modifications complex and time consuming.

Model Driven Engineering (Planet MDE 2007) (MDE) is an emerging domain and can be seen as a *High Level Design Flow* in order to resolve the issues related to SoC co-design. MDE enables system level (application/architecture) modeling at a high specification level allowing several abstraction stages (i.e. IRs). Thus a system can be viewed globally or from a specific point of view of the system, allowing to separate the system model into parts according to relations between system concepts defined at different abstraction stages. This *Separation of Views* (SoV) allows a designer to focus on a domain aspect related to an abstraction stage thus permitting a transition from *solution space* to *problem space*. Using a graphical modeling language i.e. UML (Unified Modeling Language) for system description increases the system comprehensibility. This allows designers to provide high-level descriptions of the system that easily illustrate the internal concepts (task/data parallelism, data dependencies and hierarchy). These specifications can be *reused*, *modified* or *extended* due to their graphical nature. Finally MDE's model transformations allow to generate executable models (or executable code) from high level models bridging the gap between these models and execution platforms.

FPGAs (Field Programmable Gate Arrays) are considered an ideal solution for SoC implementation due to their reconfigurable nature. Designers can initially implement, and

afterwards, reconfigure a complete SoC on FPGA for the required customized solution. Thus FPGAs offer a migration path for final ASIC (Application Specific Integrated Circuit) implementation. For FPGAs, the on chip interconnection can be either bus or NoC based.

MARTE (OMG 2007a) (Modeling and Analysis of Real-Time and Embedded Systems) is an industry standard of Object Management Group (OMG), dedicated to model-driven development of embedded systems. MARTE extends UML, allowing to model the features of software and hardware parts of a real-time embedded system and their relations, along with added extensions (for e.g. performance and scheduling analysis). Although rich in concepts, MARTE lacks a design flow to move from high level modeling to execution platforms.

Gaspard (DaRT team 2009, Gamatié et al 2008b) is a MDE based MARTE compliant SoC co-design framework dedicated specifically towards parallel hardware and software; and it allows to move from high level MARTE specifications to different execution platforms. It exploits the inherent *parallelism* included in repetitive constructions of hardware elements or regular constructions such as application loops. Gaspard also focuses on a limited application domain, that of *intensive signal processing* (ISP) applications.

The main contribution of this paper is to present a novel MDE based design methodology for implementing the aspects of Partial Dynamic Reconfiguration from an extended MARTE standard. This design flow successfully responds to the major issues, for both users and designers of a typical HLS flow. Applications are graphically specified at a high abstraction level with UML and factorized expressions of parallelism, multidimensional data arrays and powerful constructs of data dependencies are managed thanks to the use of the MARTE standard profile. The design flow allows to specify part of the reconfigurable system at a high abstraction level: notably the reconfigurable region and the reconfiguration controller. Afterwards, using model to model transformations, the gap between high level specifications and low implementation details can be bridged to automatically generate the code required for the creation of bitstream(s) for final FPGA implementation. Currently our approach focuses on a traditional SoC architecture; however it can be extended to include the aspects of NoCs also.

The rest of this chapter is organized as follows. Section 2 describes Network on Chip concepts while an overview of MDE is provided in section 3. Section 4 summarizes our MARTE compliant GASPARD framework and section 5 gives a detailed explanation of the deployment extension in MARTE. Section 6 details the related works and Section 7 illustrates our methodology related to implementing PDR supported FPGAs. A case study is present in section 8 followed by future works and perspectives. Finally section 10 details the conclusion.

2. NETWORK ON CHIPS

Recently, the term SoC (System on Chip) has replaced VLSI (very-large-scale integration) or ULSI (ultra-large-scale integration) as the key word in information technology (IT). The change of name is nothing but a reflection of the shift of focus from chip to system in the IT industry. Before the SoC, semiconductor technology and circuits themselves played the central role as a discipline and industry, and engineer's needed to master semiconductor circuits and technology to enhance the performance f components in the known target system. However, in the SoC era, the engineer is required to provide a system solution to the target problem with the final end application in mind. These SoCs are widely used in portable and handheld systems such as cell phones and portable game devices. Therefore, the discipline of SoC design is intrinsically complicated and covers a variety of areas, such as marketing, software, computing system, and

semiconductor IC design, as described in Figure 1.SoC development requires hexagonal expertise in not only technological areas such as IC technology, CAD, software, and algorithm but also in management techniques complicated team, project, and customer research. Figure 1 shows the general disciplines for the design of SoCs.

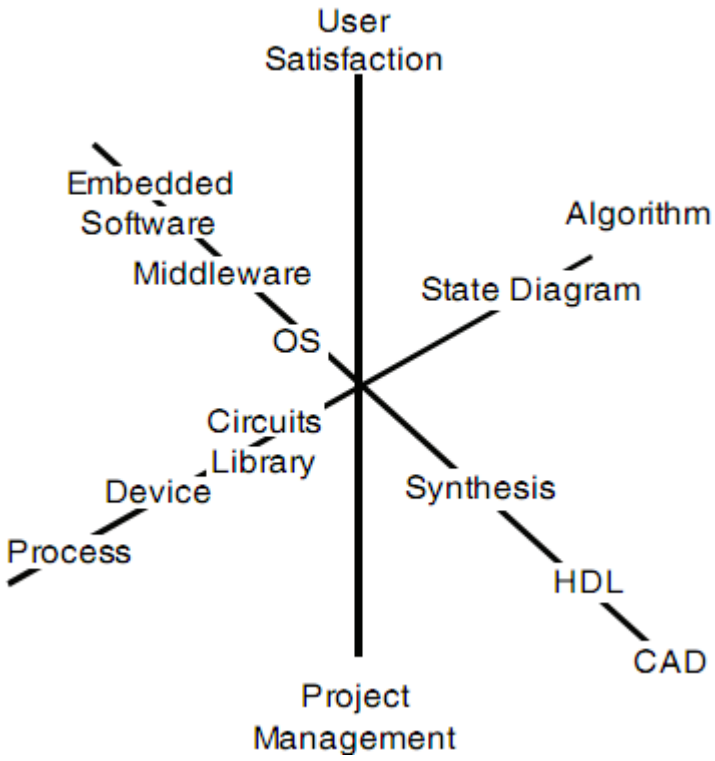


Fig.1: Disciplines for the design of SoCs

The design of a system is a sequence of activities to implement its functional requirements by assembling the physical components and algorithmic software. The entire design process of a system can be split into many stages; each stage has a design input and a design output. Every intermediate design activity requires inputs for simulation, analysis, or synthesis. In the top-down design, the design process starts from the system function requirements; these are the design inputs to the first design stage. Its design output is more detailed and more complicated than the system requirement and one step closer to the physical implementation. Each stage needs a model of system or model of computation (MoC) to map its design input to the design output (see Figure 2). The model is composed of a set of simpler subsystems and a method, or the rules for integrating them, to implement the system functions. Of course, each stage has a different model with a different level of abstraction. On the one hand, it should contain all relevant characteristics, but on the other hand it should be as simple as possible.

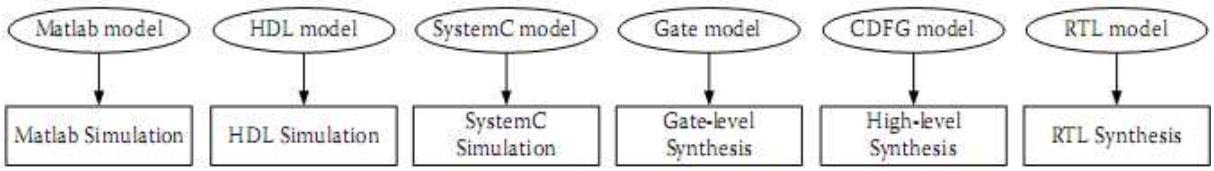


Fig.2: Model of Computation

As the chip scale grows, current System on Chip (SoC) designs generally incorporate a number of processing cores to answer high performance requirements with reasonable power consumption [Vangal et al 20007, Lattard D et al 2007]. This design methodology has the advantage of achieving high performance with moderate design efforts because, once a processor is designed and verified, it can be replicated and reused. However, integrating a number of processors on a single chip does not necessarily mean a SoC design. Depending on the applications, SoC also requires the integration of numerous peripheral modules, such as on-chip memory, external memory controller, I/O interface, and so on. As a result, it is getting more and more important to provide efficient interconnections among numerous processing cores and peripheral modules of SoC. Traditional interconnection techniques, such as on-chip bus or point-to-point interconnections are not suitable for current large-scale SoCs because of their poor scalability. In recent years, a design paradigm based on a Network on Chip (NoC) was proposed as a solution for interconnection techniques of large-scale SoCs [Dally et al 2001, Benini, L and Micheli, G.. 2002]. The modular structure of NoC makes chip architecture highly scalable, and well controlled electric parameters of the modular block improve the reliability and operation frequency of the on-chip interconnection network. After the proposal of the NoC design paradigm, research concerning NoC has fairly advanced.

NoCs bring the networking principles for data transfer, such as those used in large area networks (e.g., the Internet), to the on-chip domain. Developing NoC-based systems tailored to a particular application domain, satisfying the application performance constraints with minimum power-area overhead is a major challenge. With technology scaling, as the geometries of on-chip devices reach the physical limits of operation, another important design challenge for NoCs will be to provide dynamic (run-time) support against permanent and intermittent faults that can occur in the system. Traditionally, bus-based architectures have been used to interconnect the various cores of the MPSoCs. To meet the increasing communication demands, the bus based architectures have evolved over time from a single shared bus to multiple bridged buses and to crossbar-based designs. Current state-of-the art bus architectures, such as the AMBA multilayer, enable the instantiation of multiple buses operating in parallel, thereby providing crossbar architecture. However, such architecture is inherently no scalable for large number of cores in the design. A communication-centric design approach, Networks on Chips (NoCs), has recently emerged as the design paradigm for designing such scalable micro networks for MP SoCs. A typical NoC consists of switches, links, and Network Interfaces (NIs). A NI connects a core to the network and coordinates the transmission and reception of packets from/to the core. A packet is usually segmented into multiple FLow control unITS (flits). The switches and links are used to connect the various cores and NIs together. The use of a NoC to replace bus-based wiring has several key advantages:

- Better scalability at the architectural and physical levels. NoCs can add bandwidth as needed and segment wires as required.
- Better performance under high loads. NoCs can operate at high frequencies, cope with large bandwidth demands, and parallelize traffic streams

- NoCs facilitate modularity by orthogonalizing the design of the communication architecture from the computation architecture, thereby leading to reduced design efforts.

- Quicker design closure. NoC are more predictable: They intrinsically provide wire segmentation, which helps ensuring that design will not be needed in the last phases of the design flow, when they are more costly.

- Higher energy-efficiency. To support the same traffic load, NoCs can operate at a lower frequency than bus-based systems and the data transfer can be finished faster. These can lead to a reduction in energy consumption of the system.

2.1 OSI to NoC Model

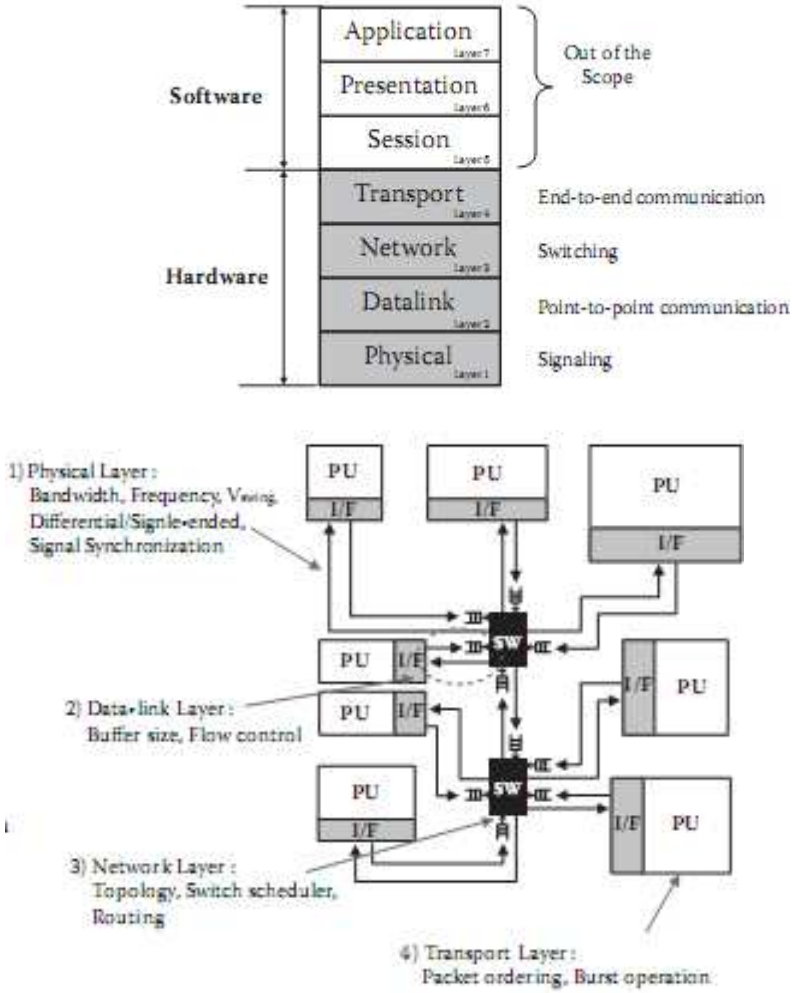


Fig.3: OSI Layer Model (top) and Layered Architecture of NoC (bottom)

The NoC exploits a layered-stack approach to communicate between integrated processing elements (PE), which may be processors, dedicated functional blocks, or memories. The NoC decouples the communicational part from the computational part efficiently from an early design stage. This dissociation enables a parallel design of PEs and interconnection structures without any interference between them. In this subsection, we focus on the communicational part, the functions of which are provided by NoC. Figure 3(a) represents the OSI seven-layer model

[Zimmerman, H. 1980] and its correspondence to the building blocks of NoC. The NoC involves the four bottom layers of the OSI seven-layer model, realized as hardware on a fabricated chip.

Figure 3(b) describes design issues to be considered for each of the OSI model layers. Because the physical layer defines electrical and physical specifications, its design issues include operational frequency of the wire link, signaling scheme, clock synchronization, and so on. In contrast to the traditional personal computer (PC) networks, NoC physical layer design has significant impact on power consumption and performance of SoC because a large volume of on-chip data transactions among PEs occur frequently and concurrently, compared to inter-PC communications.

Therefore, inefficient design of the physical layer easily results in a huge amount of wasted power, and poor performance. In addition, maximum clock frequency and wire width of the physical layer determine the theoretical bandwidth limit of the NoC design. After the first prototype chip fabrication of NoC [Lee, S et al 2003], there have been researches concerning the physical layers of NoC [Lee, K et al 2004, Panades and Greiner, A 2007].

The third layer of the OSI seven-layer model is the network layer. The major design issues of the network layer involve NoC topology selection, packet routing schemes, and quality-of-service (QoS) guarantee. Topology of the NoC should be very carefully selected because of its significant impact on overall performance and power consumption. To reduce communication overheads, PEs with frequent data transactions need to be placed at a close distance, although sufficient bandwidth for every node should be supplied to avoid performance degradation. As for low-overhead routing schemes, source routing is generally adopted in NoC because only simple decoding logic is needed for packet routing at each router instead of large look-up tables. Finally, guaranteeing QoS is also important for efficient utilization of bandwidth. In the NoC, QoS guarantee is implemented by supporting priorities or constructing different classes of virtual channels. The Æthereal NoC [Rijpkema, E 2003] implements both Guaranteed Throughput (GT) and Best Effort (BE) router for worst case QoS guarantees. An arbitration look-ahead scheme, which aims at reducing packet switching latency, is also reported [Kwanho, K et al 2005].

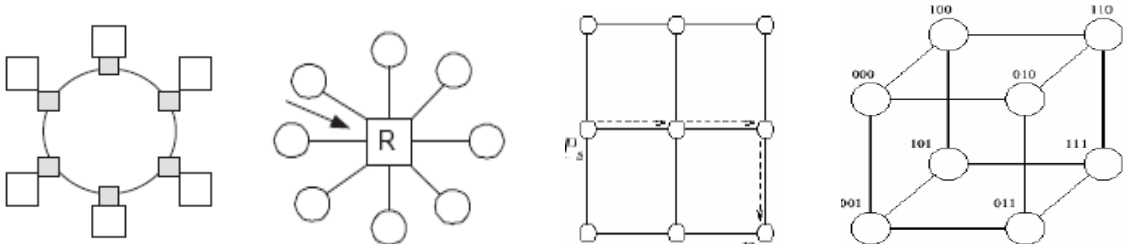
The transport layer is the highest level of the OSI seven-layer model implemented by the NoC. In the NoC-based SoC design, each of the PEs and functional blocks should be designed according to NoC protocols to support interfaces to the on-chip network.NI modules, which perform packet generation and parsing, provide abstractions of end-to-end data transactions between PEs and other functional modules. In many NoC implementations, out-of-order packet transmission is not supported because of limited buffer resources on a chip.

2.2 Concepts related to Network on Chip

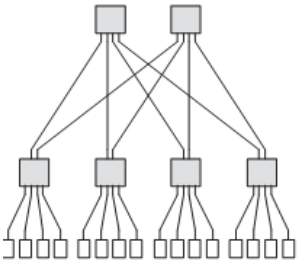
The main objective of a network on chip to provide a system for exchanging data between different processing resources (or IP core). The network consists of nodes (also called switches or routers). The data passing between nodes through links (or channels) from point to point communication. The combination of nodes and links is the communication system. Each node includes a set of communications ports. These ports allow connection to the links connecting the nodes between them. Next architectures, resources are to nodes connected directly or through links. The nodes make routing decisions (where to send data) and arbitration (which transmit data).

2.2.1 Topology

A topology is the connection map between the constituent processing elements (PEs). There exist many network topologies, and the simplest one is a ring or a shared bus. A three-dimensional cube or torus is a more complicated topology, but a mesh topology is widely considered as a typical NoC topology, especially for the homogeneous SoC. The mesh topology, however, is neither a unique nor an optimal solution for the heterogeneous multiprocessor. You should choose an optimum topology for your system and application in terms of the system performance, power-consumption, performance/power, and area cost. Traditionally, interconnection networks can be categorized into two classes: direct and indirect network. The former provides a direct connection between processing nodes. On the other hand, in an indirect network, the communication between any two nodes has to be carried through some switches. Most of the NoC topologies are indirect networks even for the mesh topology. However, as the distinction between these two classes of networks is blurring, the categorization becomes meaningless. Many network topologies have been proposed in terms of their graph-theoretical properties. Most of them were proposed for minimizing the network diameter for a given number of nodes and node degrees [Duato, J et al 2005]. However, very few of them have ever been analyzed and implemented in NoCs. Figure4 shows a few of the most famous topologies and also an application-specific one as examples. There has been research on NoC topology exploration. Murali et al. have developed a tool for automatically selecting an application-speciic topology for minimizing average communication delay, area, and power dissipation [Murali, S. 2004]. Wang et al explore the technology-aware topology of various meshes/tori [Wang, H 2005]. Kreutz et al. present a topology evaluation engine based on a heuristic optimization algorithm [Kreutz et al 2005]. In these works, the candidate pool of topologies was limited to the typical, regular and homogeneous topologies such as a mesh, torus, hypercube, tree, or multistage network.



RING STAR MESH HYPERCUBE



TREE

Fig.4: an example of network topologies

Designers of large-scale SoCs must be aware of the advantages and disadvantages of each architecture in order to select an appropriate candidate for their implementations. The metrics that are of interest can be broadly categorized as:

- Performance (latency, throughput, cross-section bandwidth).
- Energy consumption.
- Reliability (error detection and/or correction).
- Scalability.
- Implementation cost (area).

The topology of a network is described by a set of vertices, or nodes V connected by a set of edges, or channels. The set of edges E, where an edge

$$e_{x,y} = (x, y) \in E \mid x, y \in V$$

Connects a source node x to a destination node y. A network topology can therefore be represented by the graph:

$$G = (V, E)$$

Representing network topologies as graphs does not take into account physical implementation issue, but it does allow interesting properties to be studied such as:

- Node degree: The number of channels that connect a node to its neighbors.
- Diameter: The maximum distance between two nodes in the network.
- Regularity: A network is regular when all the nodes have the same node degree.
- Symmetry: A network is symmetric when it looks alike from every node.

2.2.2 Routing Algorithms

Routing algorithms define the path followed by each message or packet. The list of routing algorithms proposed in the literature is almost endless. The routing algorithm used influences many properties of interconnection networks. Hereinafter the most important ones are reported:

- *Connectivity*. It deals with the capability to route packets from any source node to any destination node.
- *Adaptivity*. Ability to find alternative paths for packets in the presence of contention.
- *Deadlock and livelock freedom*. Ability to guarantee that packets will not block or wander across a network forever.

Routing algorithms can be classier as:

- Deterministic routing always selects the same path between two nodes, even if there are multiple paths.
- Oblivious routing does not consider the state of the network when making decisions.
- Adaptive routing uses information about the state of the network to make routing decisions. These algorithms attempt to circumvent congestion points in the network in an effort to more evenly distribute traffic.

Packet-switched routers, based on the information in the packet header, determine an input →

output configuration of the switch to forward the packet. Formally, a routing function is defined as:

$$R[p, i]: P \times I \rightarrow O^n$$

Given the routing information in the packet header p (P is the ensemble of possible packet headers) and the corresponding input port i from the ensemble of the input ports I of the router, it maps to an ensemble of output ports O {0,….., n-1} of the same router. A deterministic routing function has n =1 and $\forall$ (p, i) $\in$ (P×I), $\exists!$ o$\in$ O / R(p, i)= o. An adaptive routing function has 1 <n $\leq$ card (O), so for a given packet header p, multiple output ports O{0,...,n−1} are possible. A selection function is then required to choose a single output port o from the routed ensemble O {0,..., n−1}. The selection function typically does not depend on p, but on an internal state of the router, so that at different moments in time, a different output port can be selected for the same (p, i) $\in$ (P×I) input pair.

Let us consider a 3 × 3 mesh and the XY routing algorithm. Function L_{xy} represents the routing logic of each node. It decides the next hop of a message depending on its destination. In the following definition, s_x or d_x denotes the coordinate along the X-axis, and s_y or d_y denotes the coordinate along the Y-axis.

function $L_{xy}(s,d)$
if $s = d$ then return d
else if $s_x < d_x$ then return $(s_x + 1, s_y)$
else if $s_x > d_x$ then return $(s_x - 1, s_y)$
else if $s_x = d_x$ and $s_y < d_y$ then return $(s_x, s_y + 1)$
else return $(s_x, s_y - 1)$
end if
end function

2.2.3 Problems on Routing

Deadlock, livelock are potential problems on routing algorithms. Routing is in deadlock when two packets are waiting each other to be routed forward. Both of the packets reserve some resources and both are waiting each other to release the resources. Routers do not release the resources before they get the new resources and so the routing is locked. Livelock occurs when a packet keeps spinning around its destination without ever reaching it. This problem exists in non-minimal routing algorithms. Livelock should be cut out to guarantee packet's throughput.

2.2.4 Switching techniques

The switching technique determines how data flows through a router, from its input port to its output ports. There are three switching techniques, circuit, packet, and wormhole switching.

In *circuit switching*, a physical path consisting of a series of links and routers is reserved from the sending node to the destination node. The setup time refers to the time required to reserve the resources, and the tear-down time refers to the time required to release them. Circuit switching has a high initial latency due to the setup time, but it exhibits high throughput because the bandwidth is guaranteed due to the reserved resources. The disadvantage is that during the setup and tear-down times, when data is not being transmitted, the network resources are underutilized.

In *packet switching*, large messages are broken up into smaller pieces called packets. Each packets flows through the network independently, possibly along different routes, from sender to receiver. Each packet must be stored in its entirety before being forwarded to the next node on the network, called store-and-forward, which can result in large buffer requirements. Since no resources are explicitly reserved, there is the possibility that two or more packets may wish to use the same resources at the same time, called contention. When contention occurs, one packet is granted the resource, and all others must wait. The delays caused by contention are variable, and depend largely on the amount of traffic on the network.

2.2.5 Network flow control

Network flow control, also called as routing mode, determines how packets are transmitted inside a network. The mode is not directly dependent to routing algorithm. Many algorithms are designed to use some given mode, but most of them do not define which mode should be used.

Store-and-forward is the simplest routing mode. Packets move in one piece, and entire packet has to be stored in the router's memory before it can be forwarded to the next router. So the buffer memory has to be as large as the largest packet in the network. The latency is the combined time of receiving a packet and sending it ahead. Sending cannot be started before the whole packet is received and stored in the router's memory.

Virtual Cut-Through Routing is an improved version of store-and-forward mode. A router can begin to send packet to the next router as soon as the next router gives permission. Packet is stored in the router until the forwarding begins. Forwarding can be started before the whole packet is received and stored to router. The mode needs as much buffer memory as store-and-forward mode, but latencies are lower.

Wormhole Routing. In wormhole routing packets are divided to small and equal sized flits (flow control digit or flow control unit). A first flit of a packet is routed similarly as packets in the virtual cut-through routing. After first flit the route is reserved to route the remaining flits of the packet. This route is called wormhole. Wormhole mode requires less memory than the two other modes because only one flit has to be stored at once. Also the latency is smaller and a risk of dead-lock is larger. The risk can be reduced by multiplexing several virtual ports to one physical port, so the possibility of traffic congestion and blocking decreases.

2.3 Noc Design Challenges

Designing an efficient NoC architecture, while satisfying the application performance constraints is a complex process. The design issues span several abstraction levels, ranging from high-level application modeling to physical layout level implementation. Some of the most important phases in designing the NoC include: modeling NoC topology for the application, mapping of cores onto the topology, finding paths and reserving resources, verifying performance of the system, developing simulation and synthesis models, and achieving reliable operation of the interconnect. In order to handle the design complexity and meet the tight time-to-market constraints, it is important to automate most of these NoC design phases. To achieve design closure, the different phases should also be integrated in a seamless manner. The NoC design challenge lies in the capability to design hardware-optimized, customizable platforms for each application domain.

Computer-aided synthesis of NoCs is particularly important in the case of application-specific systems on chip, which usually comprise computing and storage arrays of various dimensions as

well as links with various capacity requirements. Moreover, designers may use NoC synthesis as a means for constructing solutions with various characteristics that can be compared effectively only when a detailed model is available. Thus, synthesis of NoCs can be used for comparing prototypes. Needless to say, synthesis may also be very efficient for designing NoCs with regular topologies. NoC architectures are pushing the evolution of traditional circuit design methodologies to deal effectively with functional diversity and complexity. At the application level, the key design challenge is to expose task-level parallelism and to formally capture concurrent communication in models of computation. Then high-level concurrent tasks have to be mapped to the underlying communication and computation resources. At this level, an abstract model of the hardware architecture is usually exposed to the mapping tool, so that area and power estimates can be given in the early design stage, and different objective functions (e.g., minimization of communication energy) can be considered to evaluate the feasibility of alternative mappings.

The design of a complicated system requires high-level abstraction to reduce its design complexity. External specification is required for supporting the system application, whereas internal specification is necessary for a clear definition of the design scope and hardware implementation. Recently, an object-oriented approach has been applied to the analysis of the system specification to enhance the readability and reusability of the design. UML has unified the existing object-oriented design methodologies and design documentation methods, and is widely used in software engineering. It was standardized by OMG (Object Management Group) in 1999 as UML 1.1 and, in 2004, as UML 2.0. Here, we would like to introduce UML as a good vehicle to analyze the design of the SoC, too. Although UML 1.1 has nine types of diagrams, only three (use case diagram, class diagram, and sequence diagram) are useful in deriving the Behavioral specification from system requirements for the SoC design. UML can be used in the development of SoC in three different ways. The first is to use UML to get a rough sketch of a system's behavior. In this case, the UML diagrams are drawn for a better understanding of the overall system behavior, to provide efficient communication in the design team, and for a clear documentation format. The second approach is to regard UML as a design language and use it actively in the detailed hardware and software design. The third way is to use UML and the source codes in parallel to design a SoC. The UML generates a skeleton model of a target source code C++, and later the designer completes the detailed program by adding the required scripts to the skeleton model. This specification of a UML profile adds capabilities to UML for model-driven development of Real Time and Embedded Systems (RTES). This extension, called the UML profile for MARTE (in short MARTE), provides support for specification, design, and verification/validation stages. This new profile is intended to replace the existing UML Profile for Schedulability, Performance and Time. MARTE consists in defining foundations for model-based description of real time and embedded systems. These core concepts are then refined for both modeling and analyzing concerns. Modeling parts provides support required from specification to detailed design of real-time and embedded characteristics of systems. MARTE concerns also model-based analysis. In this sense, the intent is not to define new techniques for analyzing real-time and embedded systems, but to support them. Hence, it provides facilities to annotate models with information required to perform specific analysis. Especially, MARTE focuses on performance and schedulability analysis.

3. MODEL DRIVEN ENGINEERING

MDE revolves around three focal concepts: *Models*, *Metamodels* and *Model Transformations*. A model is an abstract representation of some reality and has two key elements: *concepts* and *relations*. Concepts represent "things" and relations are the "links" between these things in reality. A model can be observed from different abstract point of views (views in MDE). The abstraction mechanism avoids dealing with details and eases re-usability. A metamodel is a collection of concepts and relations for describing a model using a model description language; and defines syntax of a model. This relation is analogous to a text and its language grammar. Each model is said to *conform* to its metamodel at a higher definition level. A metamodel can be viewed as an IR in an HLS flow. Finally, MDE permits to separate the concerns in different models, allowing reutilization of these models and to keep them human readable.

The MDE development process starts from a high abstraction level and finishes at a targeted level, by flowing through intermediate levels of abstraction via *Model Transformations* (MTs) (Sendall and Kozaczynski 2003); by which concrete results such as an executable model (or code) can be produced. MTs carry out refinements moving from high abstraction levels to low levels models and help to keep the different models synchronized. At each intermediate level, implementation details are added to the MTs. A MT as shown in figure 5 is a compilation process that transforms a source model into a target model and allows to move from an abstract model to a more detailed model. Usually, the initial high level models contain only domain specific concepts, while technological concepts are introduced seamlessly in the intermediate levels. The source and target models each conform to their respective metamodels, thus respecting *exogenous* transformations (Mens, T., and Van Gorp, P 2006). A model transformation is based on a set of rules (either declarative or imperative) that help to identify concepts in a *source* metamodel in order to create enriched concepts in the *target* metamodel. New rules extend the compilation process and each rule can be independently modified; this separation helps to maintain the compilation process. The advantage of this approach is that it allows to define several model transformations from the same abstraction level but targeted to different lower levels, offering opportunities to target different technology platforms. The model transformations can be either unidirectional (only source model can be modified; targeted model is re-generated automatically) or bidirectional (targeted model is also modifiable, requiring the source model to be modified in a synchronized manner) in nature. In the second case, this could lead to a model synchronization issue (Stevens, P 2007). For model transformations, OMG has proposed the Meta-Object Facility (MOF) standard for metamodel expression and Query/View/Transformation (QVT) (OMG 2005) for transformation specifications.

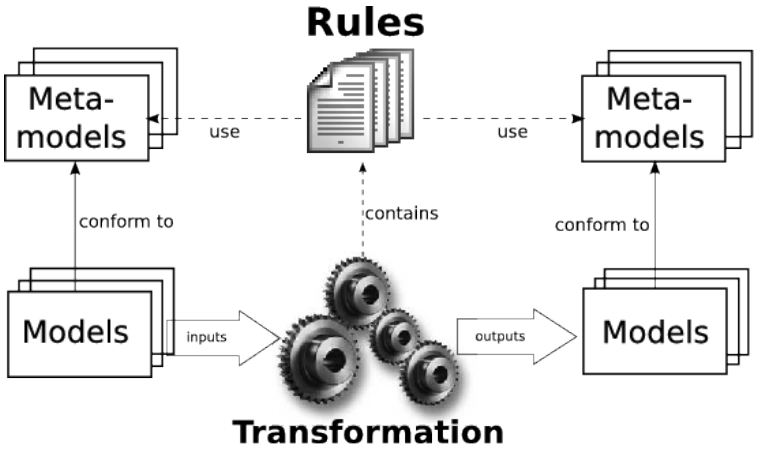


Fig.5: an overview of Model Transformations

4. GASPARD: MARTE COMPLIANT CO-DESIGN FRAMEWORK

Gaspard (DaRT team 2009, Gamatié et al 2008b) is a MDE oriented SoC co-design framework that utilizes a *subset* of the MARTE standard currently supported by SoC industry. In Gaspard as in MARTE, a clear *separation of concerns* exists between the hardware/software models, as shown in figure 6.

Gaspard has also contributed in the initial MARTE conception. One of the key MARTE packages, the *Repetitive Structure Modeling* (RSM) package has bee inspired from Gaspard. Gaspard, and in turn RSM, is based on the Array-OL (Boulet, P 2007) model of computation (MoC) that describes the *potential parallelism* in a system; and is dedicated to intensive multidimensional signal processing (ISP). Array-OL itself is a specification language and not an execution model. In Gaspard, data are manipulated in the form of multidimensional arrays. The absence of limited number of dimensions in data arrays allows to represent data in a manner typical of their manipulation in ISP applications. For example, video processing applications handle two spatial and one temporal dimension. Sonar chain is another kind of application, which handles spatial, temporal and frequency dimensions. RSM allows to models such applications.

RSM permits to describe the regularity of a system's structure (composed of repetitions of structural components interconnected in a regular connection pattern) and topology in a compact manner. Gaspard uses the RSM semantics to model large regular hardware architectures (such as multiprocessor architectures) and parallel applications. For application functionality, both data parallelism and task parallelism can be expressed easily via RSM. A *repetitive* component expresses the data-parallelism in an application (in the form of sets of input and output patterns consumed and produced by the repetitions of the interior part). A *hierarchical* component contains several *parts*. It allows to define complex functionalities in a modular way and provides a structural aspect of the application: specifically, task parallelism can be described using such a component.

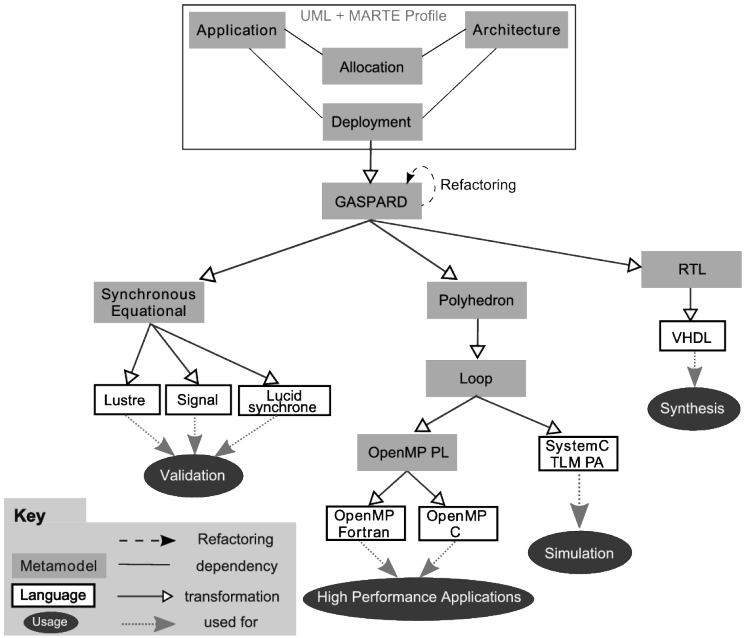


Fig.6: GASPARD framework with deployment added at the MARTE specification level

The MARTE Hardware Resource Model (HRM) concepts are inspired heavily from the preexisting hardware concepts in Gaspard. Finally the Generic Component Modeling (GCM) concepts are used as the basis for component modeling. Gaspard currently targets a limited application domain, namely *control and data flow oriented* ISP applications (such as multimedia video codes, high performance applications and anti-collision radar detection applications). The applications targeted in Gaspard are widely encountered in SoC domain and respect Array-OL semantics (Boulet, P 2007).

Gaspard also integrates the MARTE allocation mechanism (*Alloc* package) that permits to associate the applicative part of the system onto the available hardware resources (for e.g. mapping of a task or data onto a processor or a memory respectively). An example of an allocation is present in figure 7. The figure clearly illustrates the utilization of the MARTE concepts presented before. The RSM package represents the hardware repetitions and the application loops concisely in a declarative way, while the *Alloc* package allows to map the application on to the hardware resources.

Although MARTE is suitable for modeling purposes, it lacks the means to move from high level modeling specifications to execution platforms. Gaspard bridges this gap and introduces additional concepts and semantics to fill this requirement for SoC co-design.

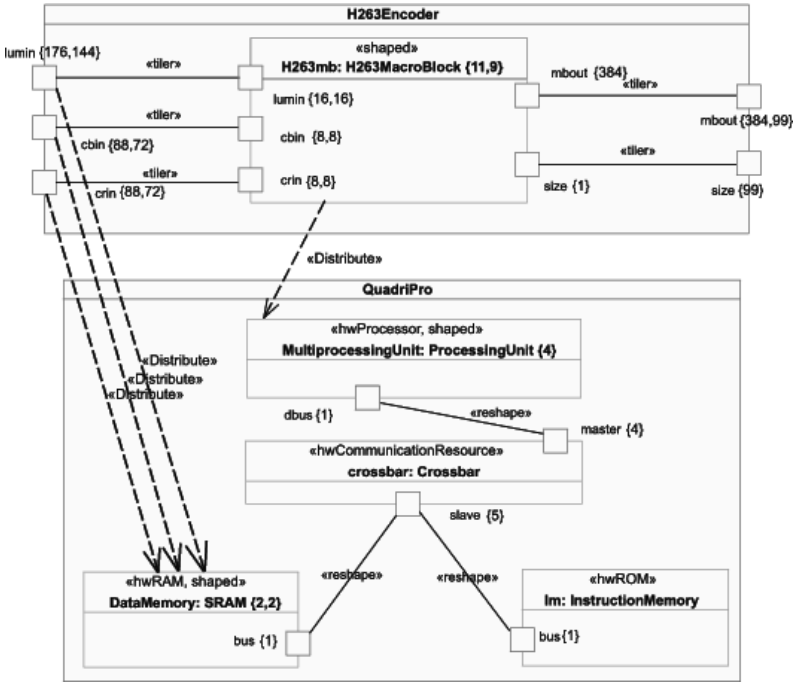


Fig.7: An Allocation: mapping a part of a H.263 codec onto a QuadriPro architecture

Gaspard also defines a notion of a *Deployment* specification level (Atitallah et al 2007) in order to generate compilable code from a SoC model. This level is related to the specification of *elementary* components (ECs): basic building blocks of all other components having atomic functions. Although the notion of deployment is present in UML, the SoC design has special needs, not fulfilled by this notion. Hence, Gaspard extends the MARTE profile to allow deploying

of ECs. To transform the high abstraction level models to concrete code, detailed information must be provided. The deployment level associates every EC (of both the hardware and the application) to an implementation (code) hence facilitating Intellectual Property (IP) reuse. Each EC ideally can have several implementations: e.g. an application functionality can either be optimized for a processor (written in C/C++) or written in hardware (HDL) for implementation as an hardware accelerator. Hence this level is able to differentiate between the hardware and software functionalities; and allows to move from platform independent high level models to platform dependent models for eventual implementation. Deployment provides IP information to model transformations to form a compilation chain in order to transform the high abstraction level models (application, architecture and allocation) for different domains: formal verification, simulation, high performance computing or synthesis. Hence deployment can be seen a potential extension of the MARTE standard to allow a complete flow from model conception to automatic code generation. It should be noted that the different transformation chains: simulation, synthesis, verification etc., are currently unidirectional in nature.

Once Gaspard models are specified in a graphical environment, model transformations are carried out via a transformation tool. However, since the standardization of QVT, few of the investigated tools are powerful enough to execute large complex transformations such as present in the Gaspard framework. Also none of these engines is fully compliant with the QVT standard. An alternative solution to QVT is the Eclipse Modeling Framework or EMF (Eclipse a), that allows to create and modify models. In order to solve this dilemma, in 2006, an initial transformation tool called MOMOTE (MOdel to MOdel Transformation Engine) was developed internally in the team that was based on EMFT QUERY (Eclipse b). MOMOTE is an enhanced Java framework that allows to perform model to model transformations. It is composed of an API and an engine. It takes source models as input and produces target models with each conforming to some metamodel. Another advantage of MOMOTE over the then existing transformation tools was that it supported external black box calls: e.g. native function calls, rule inheritance, recursive rule call and integration of imperative code. However, since that time, new tools such as QVTO and smartQVT have emerged that implement the QVT Operational language and are effective for handling the Gaspard model transformations. Currently, in order to standardize the model transformations and to render them compatible with the future versions of the MARTE standard; we have chosen QVTO as the future transformation tool for Gaspard. Current all the existing MOMOTE based transformation rules for each execution platform are being converted into QVTO based transformation rules.

MOCODE (MOdels to CODe Engine) is another internal Gaspard integrated tool that allows automatic code generation and is based on EMF JET (Java Emitter Templates) (Eclipse c). JET is a generic template engine for code generation purposes. The JET templates are specified by using a JSP (JavaServer Pages) like syntax and are used to generate Java implementation classes. Finally these classes can be invoked to generate user customized source code, such as Structured Query Language (SQL), eXtensible Markup Language (XML), Java source code or any other user specified syntax. MOCODE offers an API that reads input models, and also an engine that recursively takes elements from input models and executes a corresponding JET Java implementation class on them.

5. DEPLOYMENT LEVEL: A DETAILED OVERVIEW

In order to generate an entire system from a high level specification, all implementation details of every EC have to be determined. Low level details are much better described by using usual programming languages instead of graphical UML models. As explained before, the deployment

level in Gaspard enables one to precise a specific implementation (IP) for each EC (of both application and architecture) among a set of possibilities.

The reason being that in a complex system on chip design, one functionality can be implemented in different ways. This is necessary for testing the system with different tools, or at different abstraction levels. For instance, different IPs can be provided for a given application component and may correspond to an optimized version for a specific processor or a version compliant with a given language. As compared to the earlier deployment concepts specified in (Atitallah et al 2007), the current deployment level has been modified to respect the semantics of traditional UML deployment diagrams.

The concept of *VirtualIP* has been introduced to express the behavior (functionality) of a given EC, independently from the compilation target. It links to all the possible implementations (IPs) for one EC. Finally, the concept of *CodeFile* is used to specify, for a given IP, the file corresponding to the source code and its required compilation options. The CodeFile thus identifies the physical path of the source code. It should be noted that the modeling of a CodeFile is not possible in the *UML composite structure diagram* but is carried out in the *UML Deployment diagram*. The desired IP is then selected by the SoC designer by linking it to the EC through the *implements* dependency.

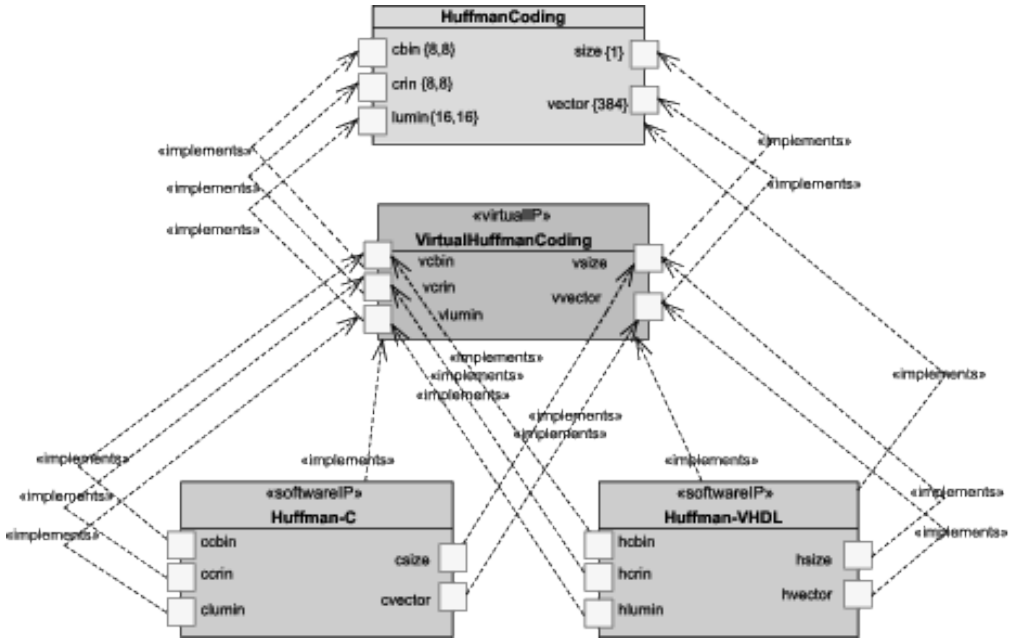


Fig.8: Deployment of the HuffmanCoding elementary component

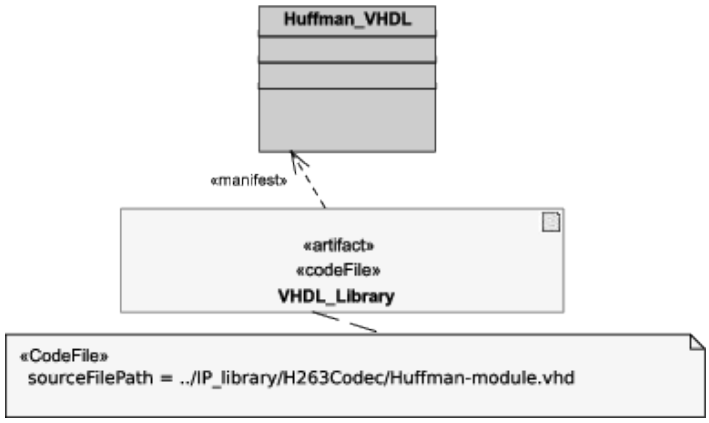


Fig.9: The CodeFile artifact determines the physical path for the code related to an IP

Figures 8 and 9 show a clear description of the deployment level. The component *HuffmanCoding* is an elementary component of the Gaspard application (H.263 codec) present in figure 6. At the deployment level, this elementary component has several possible implementation choices. These choices can be for the same execution platform (same abstraction level) in a given language, or can be for different ones. In the illustrated example, the component can be implemented for simulation in SystemC or can be implemented as hardware functionality: a hardware accelerator in an FPGA by synthesizable VHDL. The final *implements* dependency from the *Huffman-VHDL* component to the *HuffmanCoding* illustrate that this is the targeted implementation choice and the execution platform.

6. RELATED WORKS

6.1 From UML to Synthesis

ROSES (Cesario et al 2002) is an environment for Multiprocessor SoC (MPSoC) design and specification, however it does not conform to MDE concepts and as compared to our framework; starts from a low level description equivalent to our deployment level. (Atat, Y., and Zergainoh, N 2007) provides a simulink based graphical HW/SW co-design approach for MPSoC but the MDE concepts are absent. In contrast, (Gailliard et al 2007) uses the MDE approach for the design of a Software-Defined Radio (SDR), but they do not utilize the MARTE standard as proposed by OMG and use only pure UML specifications. While works such as (Damasevicius, R., and Stuikys, V 2004) and (McUmber et al 1999) are focused on generating VHDL from UML state machines, they fail to integrate the MDE concepts for HW/SW co-design and are not capable of managing complex ISP applications. MILAN (Mohanty et al 2002) is another project for SoC co-design benefiting from the MDE concepts but is not compliant with MARTE. Only the approach defined in (Le Beux et al 2007) and (Le Beux, S 2007) comes close to our intended methodology by using the MDE concepts for SoC co-design. Yet the disadvantage is that in reality it only generates the ISP application which is implemented as a black box in a targeted FPGA; and there is no notion of a heterogeneous SoC and the MARTE and PDR aspects are absent. MOPCOM (Koudri et al 2008) integrates MDE and MARTE but is not oriented towards PDR. In (Berthelot et al 2008), the authors present a design flow to manage partially reconfigurable regions of an

FPGA automatically using SynDEx. A complete system (application/architecture) can be modeled and implemented, however the MDE concepts are strikingly absent. Similarly (Boden et al 2008) present an HLS flow for PDR, yet it still starts from a lower abstraction level as compared to MDE. **My reference FPGA**

7. PROPOSED DESIGN FLOW

Currently in our design flow related to the synthesis and implementation at the RTL level, we are able to model the application part of the system, which is modeled via the MARTE profile. Afterwards the model transformations allow to generate that modeled application into the VHDL code, thus transforming the application into a hardware functionality that is afterwards implemented onto the targeted FPGA (while keeping the multidimensional arrays and repetitions specified at the modeling level). Fig 10 shows the model transformation view of our current design flow.

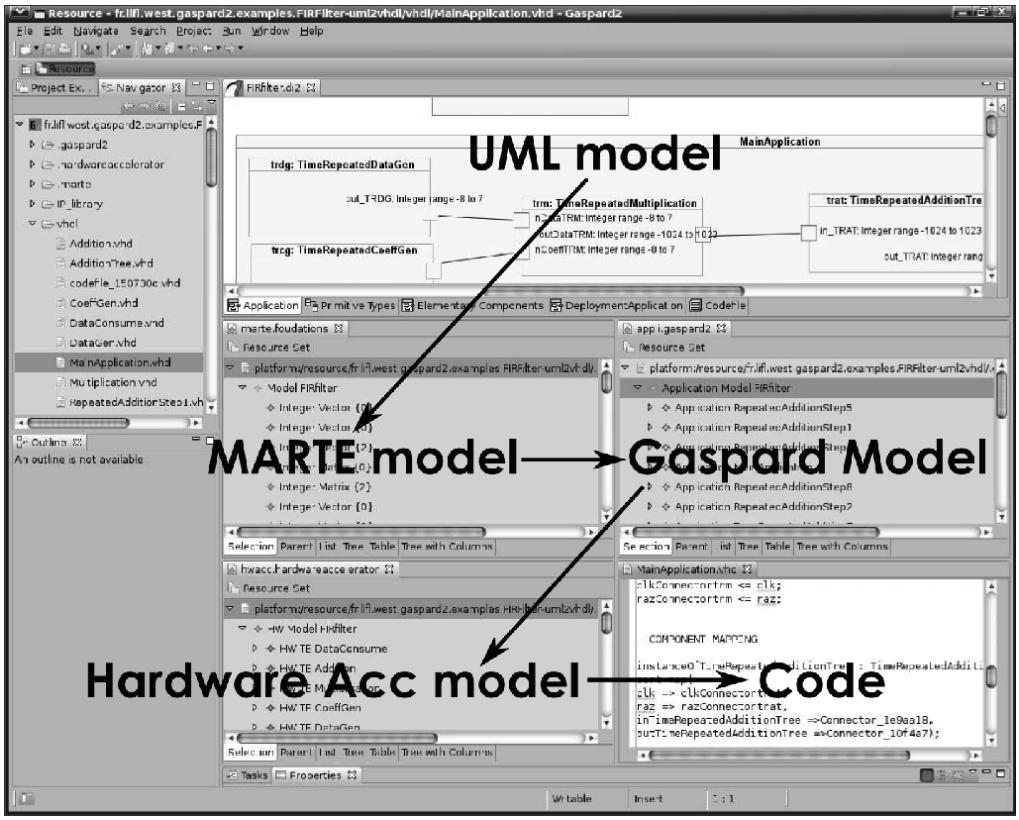


Fig.10: Current model transformations flow

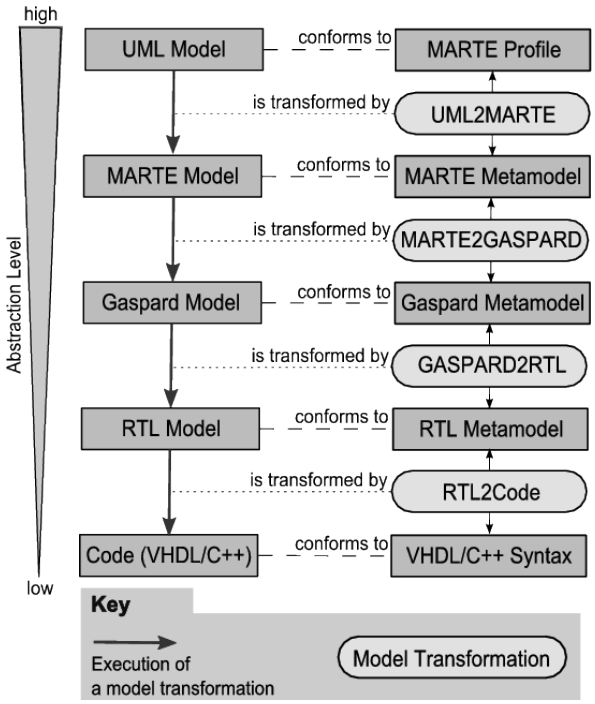


Fig.11: The proposed model driven design flow for NoCs

The limitation of the current design flow is that the hardware part of a SoC is not modeled at the high abstraction level; and the model transformations do not exist to automatically generate the code from the modeled examples.

In order to model complex NoC systems; and for their automatic code generation, we propose a design flow as shown in Fig 11. This flow will be able to model NoC algorithms (the application part) and the hardware structure of the NoC via the MARTE profile. Afterwards, the algorithms can be allocated to the hardware components (such as processors, network routers etc). Also the elementary components related to the modeled systems can be deployed onto user defined or third party IPs.

Our aim is not to replace the commercial FPGA tools but to aid them in the conception of a system. While tools like ISE and PlanAhead are capable of estimating the configurable FPGA resources (CLBs and in turns the slices) required for implementing the hardware design, this resource estimation is only possible after initial synthesis. In our design flow, the elementary components can be synthesized independently to calculate the consumed FPGA resources. This information can be then incorporated into the model transformations, making it possible to calculate the approximate number of consumed FPGA resources of the overall application and architecture (at the RTL model) before final code generation and eventual synthesis. Thus the designer is able to compare the resources consumed by the modeled system and the total resources available on the targeted FPGA resulting in an effective Design Space Exploration (DSE) strategy. If the system is too big to be placed on the FPGA, the designer can carry out a *refactoring* of the system. It should be noted that a refactored Gaspard application (or architecture) remains a Gaspard application (or architecture).

8. CASE STUDY

In this chapter we present some modeling examples of NoCs which can be modeled via the MARTE profile, taking advantage of the RSM package specifically. For this we first present some basic concepts of the RSM package.

The shape of a pattern is described according to a *Tiler* connector which describe the tiling of produced and consumed arrays. The *Reshape* connector allows representing of complex link topologies in which the elements of a multidimensional array are redistributed in another array. The difference between a *Reshape* and a *Tiler* is that the former is used for a connector that links two parts while the latter is used for a delegation connector: between a port of a component and ports of its parts. Another point to remember is that the ports (interfaces) of a component modeled in Gaspard have the MARTE *FlowPort* stereotype by default.

The *interrepetition* dependency is used to specify an acyclic dependency among the repetitions of the same component, compared to a tiler, which describes the dependency between the repeated component and its owner component. The interrepetition dependency specification leads to the sequential execution of repetitions. A *defaultlink* provides a default value for repetitions linked with an interrepetition dependency, with the condition that the source of dependency is absent. The introduction of an interrepetition dependency serializes the repetitions and data can be conveyed between these repetitions.

8.1 Modeling of a Hypercube topology

The logical hypercube overlay network topology organizes the applications into a logical n-dimensional hypercube. Each node is identified by a label (e.g., "010"), which indicates the position of the node in the logical hypercube. In an overlay network with N nodes, the lowest N positions of a hypercube are occupied (according to a Gray ordering). One advantage of using a hypercube is that each node has only $\lceil \log N \rceil$ neighbors [Liebeherr, J 1999], where N is the total number of nodes. Also, the longest route in the hypercube is $\lceil \log N \rceil$. A disadvantage of hypercube is that the physical network infrastructure is completely ignored. Another disadvantage is that the hypercube construction must be done sequentially, i.e. one node at a time. Therefore, for large groups it can take a long time before the overlay network is built. Also, the departure of a single node may require substantial changes to the overlay topology. Figure12 shows modeling of a three dimensional hypercube topology as shown in Fig 4 via the MARTE profile. This modeling approach utilizes two interrepetition link dependencies. The *Router* component in this figure represents one PE in the hypercube topology and contains three ports, one for each axis. The shape value on the router expresses the repetition of the component that is repeated 8 times. The interrepetition link dependencies connect one instance of the component to another consecutive neighboring instance on the relative axis. The values on the interrepetition dependencies thus determine the exact connection in relation to the axis.

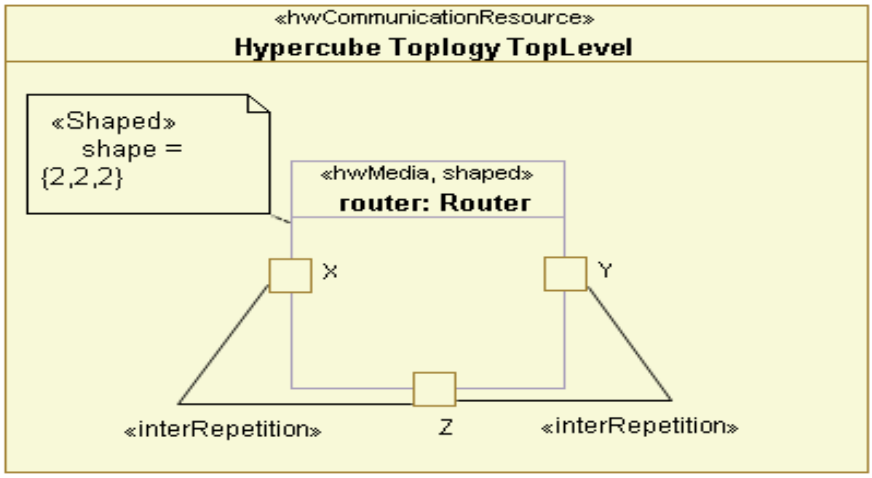


Fig.12: Modeling of a Hypercube topology

8.2 Modeling of a Mesh Topology

A mesh-shaped network consists of m columns and n rows. The routers are situated in the intersections of two wires and the computational resources are near routers. Addresses of routers and resources can be easily defined as x-y coordinates in mesh. Regular mesh network is also called as Manhattan Street network. It represents a topology for on-chip networks, which is often proposed in NoC related literature [Jantsch, A 2003]. For a mesh network, routers have at maximum four neighbours, the maximum distance in hops on an n × m mesh network is (n − 1) + (m − 1). Figure 13 show modelling of a Mesh topology of a Mesh network as shown in Fig 4 via the MARTE profile.

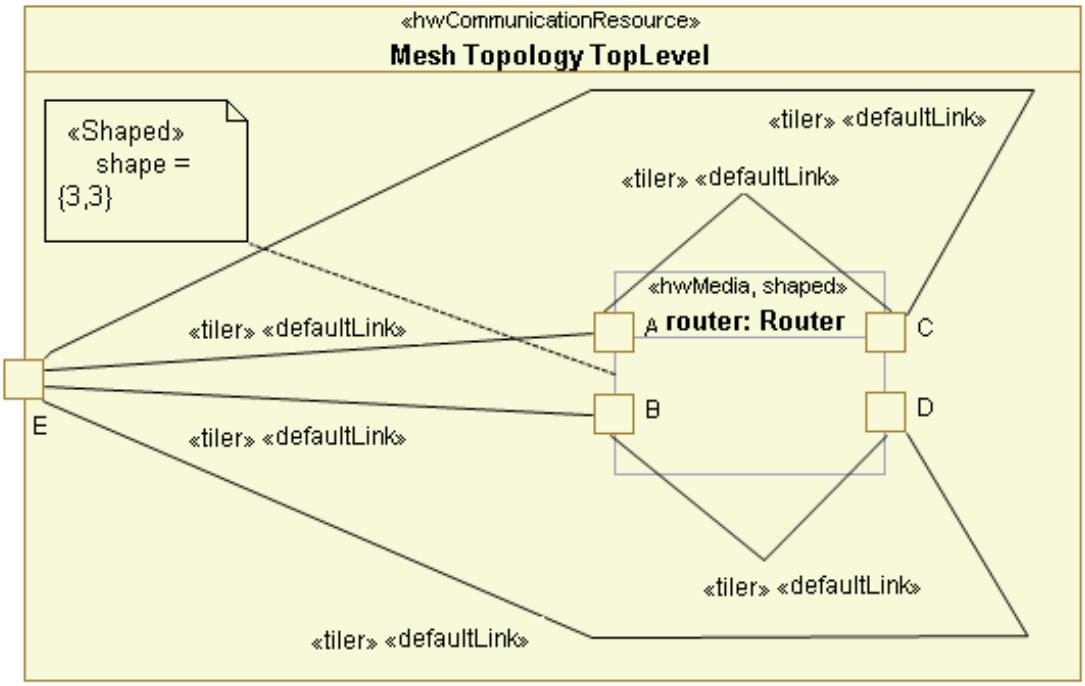


Fig.13: Modeling of a Mesh topology

As similar to the modeling approach in Figure 12, the RSM dependencies allow to express the mesh topology in a compact manner. The shape of 3,3 on the router component illustrates that this component is repeated 9 times.

8.3 Modeling of a Star topology

In a star topology, the hop count is always 1 and every transaction goes through the central crossbar switch. The star graph of order N, sometimes simply known as an "n-star" is a tree on n nodes with one node having vertex degree n-1and the other n-1 having vertex degree 1. The central switch has a number of N I/O ports and the average distance between two PEs via the central switch is $(N)^{1/2}$-1. Figure 14 shows the modeling of a Star topology in MARTE.

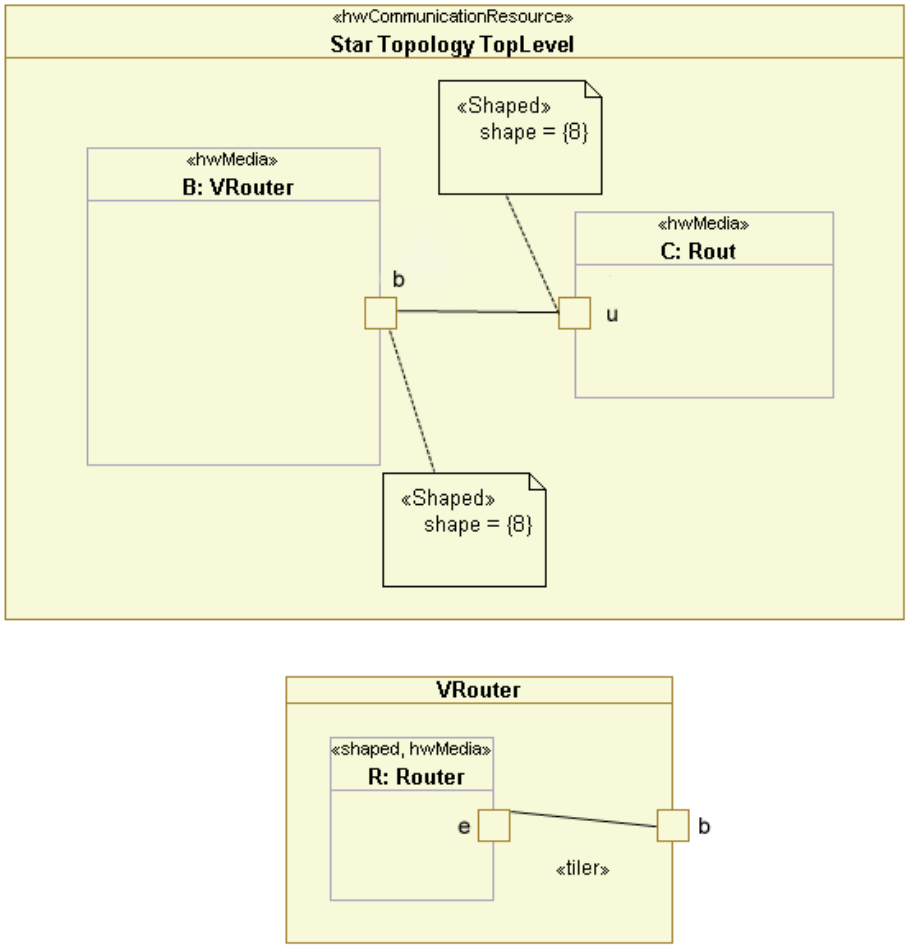


Fig.14: Modeling of a Star topology

The *VRouter* component contains a *Router* component that is itself repeated 8 times. Each repetition of the Router component has a port with a dimension of 1, while the VRouter component has a port with a dimension of 8. Thus a tiler connector is utilized to connect the

different instances of the Router component to the VRouter component. Finally at the highest level of modeling, this VRouter component is connected to another component, the *Rout* component which also has a port with a dimension of 8. A simple connector is sufficient to connect the components VRouter and Rout to each other.

9. CONCLUSION

This paper presents a novel model driven methodology to move from high level MARTE specifications to complex Network on Chip architectures. Our methodology allows to specify complex NoC intensive signal processing applications such as a multimedia codecs and digital filters in a graphical language, which via model transformations, are implemented as hardware functionalities in a targeted FPGA. These functionalities retain the inherent task and data parallelism specified at the modeling level. Currently we are in the process of developing the model transformations which will be able to take the modeled systems as input and generate the code which corresponds to the user requirements and the high abstraction level models.

REFERENCES

Atat, Y., and Zergainoh, N. 2007. Simulink-based MPSoC Design: New Approach to Bridge the Gap between Algorithm and Architecture Design. In *ISVLSI'07*. 9–14.

Atitallah et al. 2007. Multilevel MPSoC simulation using an MDE approach. In *SoCC 2007*.

Benini, L., and Micheli, G. 2001. Network on Chips: A new SoC Paradigm. *IEEE Computer*, 2001.

Berthelot et al. 2008. A Flexible system level design methodology targeting run-time reconfigurable FPGAs. *EURASIP Journal of Embedded Systems* 8, 3, 1–18.

Boden et al. 2008. GePARD - a High-Level Generation Flow for Partially Reconfigurable Designs. In *ISVLSI 2008*.

Boulet, P. 2007. Array-OL Revisited, Multidimensional Intensive Signal Processing Specification. *Tech. rep*., INRIA. http://hal.inria.fr/inria-00128840/en/.

Carver et al. 2008. Relocation and Automatic Floor-planning of FPGA Partial Configuration Bit-Streams. *Tech. rep*., Microsoft Research.

Cesario et al. 2002. Component-Based Design Approach for Multicore SoCs. *Design Automatic Conference, DAC'2002*, 789.

Dally et al. 2001. Not Wires: On-Chip Interconnection Networks, IEEE Proc. Design Automation Conf., pp. 684–689.

Damasevicius, R., and Stuikys, V. 2004. Application of UML for hardware design based on design process model. In *ASP-DAC*'04.

DaRT team. 2009. GASPARD SoC Framework. http://www.lifl.fr/DaRT.

Dorairaj et al. 2005. PlanAhead Software as a Platform for Partial Reconfiguration. *Xcell Journal* 55, 68–71.

Eclipse. Eclipse Modeling Framework. http://www.eclipse.org/emf.

Eclipse. Eclipse Modeling Framework Technology. http://www.eclipse.org/emft.

Eclipse. EMFT JET. http://www.eclipse.org/emft/projects/jet.

Gailliard et al. 2007. Transaction level modelling of SCA compliant software defined radio waveforms and platforms PIM/PSM. In *Design, Automation & Test in Europe, DATE'07*.

Gamatié et al. 2008b. A model driven design framework for high performance embedded systems. *Research Report RR-6614*, INRIA. http://hal.inria.fr/inria-00311115/en.

Guo et al. 2005. Optimized generation of data-path from C codes for fpgas. In *Design, Automation & Test in Europe, DATE'05*. 112–117.

Hubert Zimmermann. 1980. OSI Reference Model—the ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on Communications, Vol. 28, no. 4, pp. 425–432

Jantsch, A. and Tenhunen, H. 2003. Will Networks on Chip Close the Productivity Gap?, chapter 1, pages 3 – 18. Kluwer Academic Publishers

Panades, I and Greiner, A. 2007. Bi-synchronous FIFO for synchronous circuit communication well suited for Network-on-Chip in GALS architectures, Proc.1st IEEE/ACM Int. Symp. On Networks-on-Chip, pp. 83–92.

Kangmin et al., A 51 mW 1.6 GHz on-chip network for low-power heterogeneous SoC platform. IEEE Int. Solid-States Circuits Conference. Digest of Technical papers, pp. 152–518.

Koch et al. 2006. An adaptive system-on-chip for network applications. In *IPDPS 2006*.

Koudri et al. 2008. Using MARTE in the MOPCOM SoC/SoPC Co-Methodology. In *MARTE Workshop at DATE'08*.

Kim, K. et al. 2005. An arbitration look-ahead scheme for reducing end-to-end latency in networks on chip, Proc. IEEE Int. Symp. on Circuits and Systems, pp. 2357–2360.

Kreutz, M. et al. 2005. Energy and Latency Evaluation of NoC Topologies, in Proc.Int. Symp. on Circuits and Systems, pp. 5866–5869

[11]Jose Duato, Sudhakar Yalamanchili, and Lionel Ni, Interconnection Networks, Morgan Kaufmann.

Lattard et al. 2007. A Telecom Baseband Circuit based on an Asynchronous Network on Chip, Digest of Technical Papers, IEEE Intl. Solid State Circuits Conf., pp. 258–601.

Liebeherr, J and Beam, T. K. 1999. HyperCast: A protocol for maintaining multicast group members in a logical hypercube topology. In *Proceedings First International Workshop on*

Networked Group Communication (NGC '99), Lecture Notes in Computer Science, Volume 1736, pages 72-89

McUmber et al. 1999. UML-based analysis of embedded systems using a mapping to VHDL. In *IEEE International Symposium on High Assurance Software Engineering*, HASE'99. 56–63.

Mens, T., and Van Gorp, P. 2006. A taxonomy of model transformation. In *Proceedings of the International Workshop on Graph and Model Transformation, GraMoT 2005*. Vol. 152. 125–142.

Mohanty et al. 2002. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *LCTES/Scopes 2002*.

Murali, S. et al. 2004. SUNMAP: A Tool for Automatic Topology Selection and Generation for NOCs, in Proc. Design Automation Conf., 2004, pp. 914–919.

OMG. 2005. MOF Query /Views/Transformations. http://www.omg.org/cgi-bin/doc?ptc/2005-11-01.

OMG. 2007a. OMG MARTE Standard. http://www.omgmarte.org.

OMG. 2007b. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/.

OSI. 2007. SystemC. http://www.systemc.org/.

Planet MDE. 2007. Portal of the Model Driven Engineering Community. http://www.planetmde.org.

Quadri et al. 2009. A Model Driven design flow for FPGAs supporting Partial Reconfiguration. *International Journal of Reconfigurable Computing*. Hindawi Publishing, To Appear.

Rijpkema, E. 2003. Trade offs in the design a router with both guaranteed and best-effort services for networks on chip, Design, Automation and Test in Europe Conference and Exhibition, pp. 350–355

Se-Joong Lee et al. 2003. An 800 MHz star-connected on-chip network for application to systems on a chip, IEEE Int. Solid-States Circuits Conf., Digest of Technical papers, pp. 468–469.

Sendall, S. and Kozaczynski, W. 2003. Model transformation: The heart and soul of model driven software development. *IEEE Software* 20, 5, 42–45.

Stevens, P. 2007. A landscape of bidirectional model transformations. In *Generative and Transformational Techniques in Software Engineering 2007*, GTTSE'07.

Vangal et al. 2007. On An 80-Tile 1.28TFLOPS Network-on-Chip in 65 nm CMOS. Digest of Technical Papers, IEEE Intl. Solid State Circuits Conference, pp.98–589

Wang, H. et al. 2005. A Technology-aware and Energy-oriented Topology Exploration for On-

chip Networks, in Proc. Conf. on Design Automation and Test in Europe, pp. 1238–1243.