Design Environment for the Support of Configurable Network Interfaces in NoC-based Platforms

Amin El Mrabti¹, Frédéric Rousseau¹, Frédéric Pétrot¹, Jérôme Martin², Romain Lemaire², Emmanuel Vaumorin³

¹ TIMA Laboratory, 46 Ave Felix Viallet, 38031 Grenoble CEDEX, France
{amin.elmrabti, frederic.rousseau, frederic.petrot}@imag.fr

² CEA, LETI, MINATEC, F38054 Grenoble, France {jerome.martin, romain.lemaire}@cea.fr

³ Magillem Design Services, 4 rue de la Pierre Levée 75011 Paris, France, vaumorin@magillem.com

Abstract— IP-based platforms with Network on Chip (NoC) are one solution to support complex telecommunication applications. In this context, NoC architectures targeting high throughput applications tend to have configurable Network Interfaces (NI) and routers for reuse and performance purposes and aims at providing advanced communication and computation services. Unfortunately, these Network Interfaces are increasingly complex to parameterize and to program, while the deployment tools taking into account the low level architectural details are still non existent.

This work focuses on providing methods and tools to easily and efficiently deploy applications on IP and NoC based platform with configurable NI. Configurable NI offer primitives to synchronize and schedule the communication and the behaviour of IPs. Our code generation flow takes as inputs an abstract model of the HW platform, of the application and of the mapping, and generates most of the required configurations.

The efficiency of the approach is illustrated by the deployment of a complex 4G telecommunication application on a heterogeneous IP-based platform.

I. INTRODUCTION

Networks on chip are designed to support the increasing complexity of the systems on chips architectures [1] by providing efficient, flexible and scalable communication infrastructures. In the current work, we assume that a NoC is composed of a set of nodes connected to routers, and that each node includes a Network Interface (NI) component which is connected in turn to a local sub system. This local sub system contains either a CPU or a dedicated IP resource. The classical role of a NI is to abstract the network communication protocol from the connected resources. A NI is used to packetize data coming from the connected resources to the router and to depacketize data coming from the router to the connected resources [18]. Most of these services are not specific to the application and are chosen to improve performances and to optimize the chip area. They usually consist of protocol transactors that translate an information (address, data) available within a packet into a bus-like protocol and vice-versa.

More advanced NIs have been proposed to improve the data communication latency and throughput through the NoC in a configurable way. This configuration capability is required to make the NoC reusable for multiple application deployments. There are some services that a NI must provide independently of any application, like the data routing scheme

and the data flow control. There are some advanced services that advanced configurable NI can provide to support a specific application or a model of computation. This includes communication scheduling and synchronization [5,7], which are implemented in software [21] when using general purpose NIs. Having a NI providing such distributed communication services can be an interesting solution when dealing with highly complex applications. This is always accompanied by complex configurations for NoC programming. The complexity of configuring NoC-based platforms containing configurable NI is due to several causes: (1) Most of the advanced proposed NoCs are based on configurable application-specific NI [5,6]. Thus, the configuration of the NI is often dependent on the application communication flow. (2) The NI configuration programming model and parameters differ from a NoC to another. There are no standard models or languages for programming NIs.

To overcome these problems, general methods and tools for NoC programming which support the automatic generation of configurations for configurable NIs are required. They should provide functionalities to help the system integrator efficiently to program the NoC-based platform, using an *a priori* knowledge of the application communication requirements and the resources mapping choices. If several applications are supposed to share the whole platform in a time-sharing manner (*i.e.* switching radio in a wireless device or codec in a multimedia one), the deployments by means of simple configurations needs to be supported.

This paper presents a general configuration generation flow that targets NoC-based platforms with configurable NIs. The flow aims at producing communication and computation configurations. In our solution, we consider that the NoC-based platform is already defined (topology, nodes). The proposed configurations generation flow is based on abstract descriptions of the application, the architecture and the resources partitioning. Our flow contains two main parts: a Front End part which is target-independent and makes the flow general, and a Back End part which offers details about the configuration programming model and allows the configuration code generation for a specific platform. We implement this flow in an IP-XACT design environment.

The paper is organized as follows. Section 2 reviews some related works on existing design flows that target classical and configurable NoC-based platforms. Section 3 introduces the

configuration code generation flow that supports configurable NI. Section 4 details the configuration generation flow and its implementation. Section 5 presents a case study to deploy a 3GPP-LTE application over the MAGALI platform. Section 6 concludes this paper and gives outlooks on future works.

II. RELATED WORKS

The number of networks on chip architectures is increasing while the design flows and tools for programming those architectures are still not sufficient. Most of the design methods and tools around NoCs focus on NoC synthesis [9,4,8]. In [9], a design flow called “Netchip” is presented for Xpipes [3] NoC synthesis. The flow includes tools for the topology selection [22], for custom topology synthesis [23] and for SystemC NoC generation [24]. The μ SPIDER CAD tool for NoC synthesis is presented in [8] taking into account latency and bandwidth constraints. In [4], the $\mathcal{A}$ ethereal [2] design flow is detailed. The flow takes as input a data flow model, the set of IP that could be connected to the NoC and some constraints to generate the topology model. The NIs in [4] are parameterized and programmed when the topology is generated. The NoC connections are configured via a memory-mapped configuration port on the NI [6,20] in case of multiple application deployment. This NI reconfiguration approach is used also in a QoS mechanism by splitting the available bandwidth between the different IP [19]. In [10], a methodology of run time task assignment over reconfigurable hardware tiles in a NoC is proposed. This reconfiguration concerns the IP resources for task migration between the different tiles connected to the NoC.

In the presented work, we are not doing *ad-hoc* synthesis of NoCs. We consider that the NoC-based platform and its topology are already fixed, and we provide methods and tools to reuse the platform for multiple application deployments. Our solution focuses on supporting advanced NoC architectures containing configurable NIs which could provide different configurable services like task and communication scheduling. The task scheduling is done on IPs, which can implement multiple tasks and switch from one behavior to another by mean of some registers configurations.

III. CONFIGURATION CODE GENERATION FLOW

A. Configuration code generation flow prerequisites

To program different types of configurable NI, which might provide control data flow services or task scheduling and synchronization services, an abstract model describing the data-flow is needed. We need also to know how the dataflow is partitioned over the resources connected to the NoC. Thus, a description model of the platform and a partitioning method are also needed. The partitioning model should be generic for multiple NoC-based platforms types and should support the description of architectural constraints. The final prerequisite concerns the genericity of the configuration generation flow for multiple NoC-based platform types. This can be obtained and ensured by introducing two independent parts in the generation flow: A general part which is platform-independent and a specific part which takes into consideration the

specificities of the target NoC-based platform in terms of programming models details.

B. Configuration code generation flow overview

To satisfy the prerequisites that we outlined in the previous section, a configuration code generation flow is defined in two parts: Front-End and Back-End. This compilation-like approach promotes the construction of flexible generation tools which are easy to maintain.

Figure 1. Configuration code generation flow for NoC-Based platforms

In the Front-End phase (Fig.1.A), the application description language, called APDL (APplication Description Language), provides the possibility to model complex data flows with arbitration possibilities. The architecture description language, called ARDL (ARchitecture Description Language), brings two main novelties. The first one concerns the description of the physical communication paths through the NoC to explore the platform. The second one is the description of the low level software layer provided by the architecture as a set of parameters which can ease configurations generation. The mapping solution that we present in our flow is defined in two steps in order to support the mapping of complex data flows over different types of NoC-based platforms with some specific constraints. The mapping constraints are defined in a previous step called meta-mapping. It defines the constraints on application and platform components for mapping. It is a specialization of the classical mapping to a given architecture. The mapping itself complies with the constraints and rules presented in the meta-mapping phase. The General Intermediate Model (GIM) is then generated and describes the application deployed over the architecture resources. At this level, no information are available on how we can program configurable NI.

In the Back-End part (Fig.1.B), the Specific Intermediate Model (SIM) contains specific information related to the configurable NoC-based platforms like NI configurations structures and parameters and the virtual components associated to the platform blocks for the simulation model generation. It is deduced from the GIM refined with additional details of the target NoC-based platform. Using the SIM model can ease the configuration code generation which is difficult to realize directly from the GIM that does not include any information about the target NoC-based platform and the potentially configurable NIs.

C. APDL: Application Description Language

We are facing complex data flow applications: data flows with arbitration possibilities, scheduling of communications on ports, and so on. Many system level languages that have been proposed in the past support application description with XML notation [11,12] but only for KPN or SDF representation.

We have developed a new application description language based on the XML notation, which allows the description of the application metadata. These information are difficult to extract from a C implementation. The novelty in our proposal, called APDL, is the possibility to model complex data transfers between tasks (Fig.2), so that the application description is not limited to KPN or SDF models. This language describes a set of tasks that run in parallel as well as the multi-input multi-output channels which ensure the transfer of data between tasks. An application described with APDL is composed of tasks and channels.

Figure 2. APDL basic components

An APDL task accesses channels via task ports. A task port is defined as a name, a direction, the size of data transferred through the port and the number of the sending and receiving data process done from the port. A task port models a call to a communication primitive (read or write) in the task implementation. The channels represent the communication medium between tasks. An APDL channel is a multi-input multi-output channel. It may model FIFO channels or channels based on shared memory.

An APDL channel can schedule accesses to inputs (resp. outputs) thanks to an input (resp. output) arbiter. A channel arbiter may be an input or output arbiter. It allows the scheduling of accesses to the channel by setting an access order for the channel ports. The access order is defined with an input or output matrix. The use of arbiters in channel descriptions is optional. APDL assumes that the amount of data transferred between the tasks is known in advance. More details on APDL are given in [13].

D. ARDL: Architecture Description Language

One way to express a structural model of a platform is to use IP-XACT [15]. The IP-XACT standard, provided by the SPIRIT consortium (now part of Accellera), is a XML-based language, for several purposes: IP packaging for platform assembly and IP integration, design and verification flow automation, etc. It has been initially developed for RTL level and has then been enhanced for TLM platform descriptions. IP-XACT schema is able to contain information about hardware structure, but does not fit totally our requirements to

describe the low level software and the communications paths. ARDL is the proposed language to describe configurable NoC-based platforms (Fig. 3). This language is an extension of the IP-XACT standard. We use this format to exploit the TGI API provided by the standard to extract the design data and to use the developed tools operating around the standard.

The architecture is modeled as a set of hierarchical systems. A system in ARDL is a set of computational components, communication components, storage components and devices. Computational elements are processing units which are an abstraction of CPUs or DSPs or configurable hardware IPs. ARDL is able to model communication components in a NoC-based platform and some related information like the NoC topology, size and routing algorithm. Two main novelties are presented in ARDL by comparison to languages like [11,14] in order to enhance the automation of the application deployment over NoC-based platforms with configurable NI by mean of configurations generation:

Figure 3. ARDL basic components

(1) Modeling the HAL (Hardware Abstraction Layer) API which is a software layer composed of a set of primitives to handle hardware components. In ARDL, a primitive could be described as a set of parameters and configurations. Thus, this format is useful to describe configurable NoC platforms and helps to generate configurations for diverse configurable components such as NI.

(2) Modeling the architecture physical communication links. A communication link between a source and a destination resource is described as the set of resources involved to realize it ("AccessType" Tags in ARDL), as well as some other required information (access mode: read/write, protocol, etc). In some hardware architectures, the number of communication links is large enough and the communications could just be deduced from the NoC topology and connectivity. In NoC-based platforms, this information is helpful to configure automatically NIs for the data flow control over the NoC. More details on ARDL are given in [13].

E. Meta Mapping and Mapping

In most of existing design and code generation flows, the mapping language is dependent on the type of application and architecture as it associates application components to architecture resources. The mapping rules to be applied may differ from one type of architecture to another. To support several types of NoC-based platforms in our generation flow, we introduced the mapping in two steps: The first step serves

to adjust the mapping of the application over the type of NoC platform by defining the mapping rules (meta-mapping step). The second step is to apply these rules in the mapping of the application components over the architecture resources.

The meta-mapping model in our flow defines which type of APDL component might be mapped onto which type of ARDL component. This can be different from one ARDL platform to another. The meta-mapping defines two types of constraints: (1) general constraints concerning the general mapping rules. For example, a general constraint could allow the mapping of an APDL channel over a configurable IP or forbid the mapping of a task over the CPU if it is reserved to resources configuration. Tasks in this case will be mapped over hardware IPs. (2) Specific constraints concern some architecture components. For example, we can define which set of Hardware IPs could execute a specific application task. This can be expressed in the meta-mapping step and taken into consideration during the mapping. The meta-mapping is done manually and represents a first step in the exploration process.

Once the meta-mapping is achieved, the deployment of APDL application components over the ARDL components could be described while respecting the constraints defined in the meta-mapping step. The mapping is defined by “Task Binding” and “Channel Binding” manually. The task binding allows the mapping of APDL application tasks over ARDL processing units. Channel binding allows the mapping of APDL channels over ARDL communication links. This mapping is detailed by defining the correspondence between APDL channel subcomponents (arbiter, FIFO, buffers) and ARDL resources referenced as “AccessType” in the communication links definitions.

F. Intermediate models: GIM and SIM

The configuration generation flow is based on two intermediate models. The first is general and allows the description of a complex data flow mapped over an abstract model of the configurable NoC-based platform without mentioning any details about configurations structures and languages. Generally, at this level (Fig.4.A), we associate tasks to the IPs connected to the NoC and the data flow traffic over the configurable NIs which offer services for data flow control and scheduling.

The second intermediate model called SIM (Fig. 4.B) is a refinement of the GIM and adds some NoC-based platform specific information. A hardware component in the GIM is refined with information about an existing hardware component library (SocLib, SystemC/TLM library, etc ...). This information helps in code generation for simulation. A configurable hardware component in GIM is refined with information on how this component could be configured. This information details the configuration programming model of the target platform. This platform specific information is helpful to support the configurable NI in our configuration code generation flow. Other information could be expressed in the SIM which references software drivers, memory mapping and some tool chain details.

Figure 4. Intermediate models: GIM and SIM

IV. IP-XACT DESIGN ENVIRONMENT FOR NOC-BASED PLATFORMS DESIGN

A. Classical IP-XACT design environment

IP-XACT provides a design environment solution to manage ESL design flows. This solution is composed of three main parts:

(1) IP Packaging phase (Fig.5.A) in which we describe all the components of a platform with XML files in accordance with the IP-XACT schema. The IP-XACT Component [15] is the central concept. It is used to describe cores (processors, DSP ...), peripherals (memories, DMA controllers, timers ...) and busses. IP packaging facilitates future reuse of existing components.

(2) Design construction and configuration (Fig. 5.B): An IP-XACT “Design” instantiate components defined in the library described in phase (1). It describes the interconnections between these instances. To extract a specific view of the “Design”, IP-XACT proposes the “DesignConfig” [15] component to be applied to an IP-XACT “Design”.

Figure 5. IP-XACT design environment

(3) Generators (Fig. 5.C) are attached to IP-XACT designs and allow external tools to extract and use the design data. The data extraction for the generators is facilitated by an API provided by the IP-XACT standard called TGI [15] (Tight Generator Interface). This API promotes the portability of the developed generators across different IP-XACT design environments. The external tools could be code generators, simulation environments, etc ... We define a design flow (Fig. 5.D) connected to IP-XACT “Generators” components as a set of external EDA tools, input models and artifacts. An artifact

could be an intermediate model (GIM, SIM) or a final generated code (SystemC code, configurations, etc...).

B. Configuration generation flow implementation

We developed our configuration generation flow (Fig.1) in an IP-XACT design environment (Fig.5). We build our design environment around IP-XACT standard for different reasons: (1) Reuse of existing tools which handle this standard and implements the TGI API, (2) ARDL is implemented as an extended version of IP-XACT, (3) Use of the IP-XACT “Generator” and “ToolChainGenerator” components for the automation of the design flow.

Figure 6. Configuration code generation flow implementation

First, the ARDL and/or RTL component libraries are built in IP-XACT (Fig. 6.a). These components are instantiated and interconnected to compose platform designs (IP-XACT Design (Fig. 6.b)). A component could be described at the RTL or at the ARDL level. An IP-XACT “DesignConfig” is then linked to the design to show one view corresponding to the RTL or to the ARDL level. The information are extracted from the platform design using the IP-XACT TGI API (Tight Generator Interface) and will be used by the tools referenced by the IP-XACT “Generator” components (Fig. 6.c). To support our generation flow, we define an IP-XACT “GeneratorToolChain” (Fig. 6.d) which enchains a set of IP-XACT “Generator” components: GenGim, GenSim, GenConfigs and GenTop. Each IP-XACT “Generator” takes some parameters as entries to configure the linked tools and produce an artifact in output. The IP-XACT “GenGim” Generator (Fig. 6.e) considers the ARDL NoC model as entry in addition to the APDL, the meta-mapping and the mapping models. It configures the tool called “GIM Generator” with these parameters to produce the first intermediate model of the flow: the GIM model.

The GIM model is then considered as an input to the IP-XACT “GenSim” Generator (Fig. 6.g), in addition to the “SimDesignConfig” (Fig. 6.f) file. It contains additional information about the configurable components of the platform including the structure and the dependencies between the configurations parameters related to the ARDL configurable components (including configurable NIs). The “SIM Generator” tool is composed of a general part which is independent from the platform and treats the information coming from the GIM model. It includes also a platform specific part which treats the data of the “SimDesignConfig” file. The tool mixes the two models to produce the SIM artifact. This model represents the entry to all the others final code generator tools which are specific to the platform (“Configurations Generator” (Fig. 6.h) and the “Top Generator” (Fig. 6.i)).

The configuration code generation flow is integrated to an existent industrial design tool called Magillem [16]. Magillem tool suite is an IP-XACT complete design environment including debugging functions and runs “Generators”. The Front-End part of the flow is developed as an Eclipse plug-in added to Magillem environment. The main functionalities of this plug-in are the possibility to create a new APDL, ARDL, meta-mapping and mapping models with the appropriate wizards. A meta-mapping viewer and a Front-End project explorer are proposed. The IP-XACT “Generator” components are implemented in the Magillem environment and integrate our developed tools. The flow can also be used separately from Magillem.

V. EXPERIMENT

The objective of this experiment is to validate our flow by deploying a 3GPP-LTE application over the MAGALI platform through the generation of configurations. The configuration code is executed by a simulation model.

A. MAGALI platform overview

MAGALI [5] is a chip which targets the physical and MAC layers of telecommunication applications (3G, 3GPP-LTE, WiMax, etc...). It gathers multiple configurable resources (OFDM blocks which include FFT/iFFT, LDPC and Turbocodes blocks for coding/encoding, etc, ...), memory blocks (SME (Smart Memory Engine) which are similar to programmable DMA), DSP (Mephisto for channel estimation, synchronization, etc...) and an ARM11 processor to control the overall platform. All these different functional units are interconnected through an asynchronous and 2D-Mesh NoC. The links between routers are bidirectional. This structure enables flexibility for the routing of data exchanged between the resources. Each functional unit is connected to a router through a CCC (Communication and Configuration Controller) [5] block. It implements functionalities related to MAGALI like the configuration of the cores, the local scheduling of configurations, the control of data flow, etc. It is an extension to the classical NI and may include up to four input controllers (ICC) and four output controllers (OCC) for communication purpose. More details on architecture and protocols implemented in the CCC blocks are given in [5]. To

deploy a telecom application over MAGALI, we need to program it as a distributed application implemented as a set of configurations. Each CCC has to carry out a sequence of local configurations. The centralized processor is responsible for the starting and the orchestration of the execution of each resource.

B. Configuration scheme and configuration files details

Computation and communication configurations are defined and executed, for each IP, via the CCC unit. The computation configurations specify and orchestrate the execution of tasks over the resource. The communication configurations specify and orchestrate the exchange of data between the IPs. The configurations are modeled as a set of configuration layers (Fig.7.a). The “Global Scheduling” layer describes the application behaviors based on the orchestration of the elementary behaviors of the configurable nodes defined in the “Resource Scheduling” layer. It is implemented in a file called “cpu.sno”. The “Resource Scheduling” layer defines the node behavior by scheduling a set of computations (HAL layer) and communications (COM layer) configurations. This layer is implemented in a file called “ip.sno” for each IP. The “COM” layer contains configurations for sending and receiving data and for synchronizing the data flow. The configurations are associated to the ICC and OCC elements. An ICC is configured to receive data from the network to the target IP and an OCC element is configured to send data from the source IP to the network. An OCC configuration, defined in the “occ.cfg” file, explains how to send data from the IP directly connected to the OCC, to other IPs. It contains information such as the number of data to be transmitted, the path to follow through the NoC and the identifier of the ICC that receives the data. An ICC configuration, defined in the “icc.cfg” file, explains how the IP directly connected to the ICC, receives data. A file called “ctx.cfg” is used to schedule the ICC and the OCC configurations for an IP, using primitives such as RC: Request Configuration, LL: Local Loop, etc. The “HAL” layer describes the set of tasks that the IPs implement. Each task is defined as a set of parameters that initializes the registers of the IP. The HAL layer is implemented in a “core.cfg” file, for each IP. The “ip.str” file is also defined for each IP and contains parameters to initialize the NoC (number of ICCs and OCCs for example). The “ctx.cfg” file schedules the basic configurations defined in the “core.cfg” file to describe the IP behavior.

A configuration in (core.cfg, icc.cfg, occ.cfg) is used and referenced by a micro program (ctx.cfg). The difficulty of manually configuring MAGALI to deploy an application is due to the high number of configuration files to develop, of parameters to initialize in each file, and the dependencies between them. Moreover, adding a new IP to the platform, or changing an IP location, require the development of new configuration files and the modification of others as well. We can also have several resources connected to the NoC that implement the same tasks. Switching a task from an IP to another requires also a new network programming. This highlights the interest of developing a method and tools to

automatically generate configuration files for this kind of platform.

Figure 7. configuration files to program the MAGALI platform

C. ARDL description of MAGALI platform

In this section, we describe the reception chain of the MAGALI platform using ARDL. The “top_magali” (Fig. 8.b) includes IPs for specific functions (mep, asip, rx_bit) and IPs which model programmable storage devices (SME).

Figure 8. MAGALI architecture modeled with ARDL

“top_3gpp” (Fig. 8.a) is the main design used to test the “top_magali” sub system. It contains the processor for the orchestration of the configurations and two components for data generation and recording (recgen_00w, recgen_00s). The Fig. 8.c is a graphical ARDL view of a MAGALI node (mep_10) designed in the Magillem environment.

All the physical communication links through MAGALI are allowed. They are listed in a file generated during ARDL model loading (“comlink.cl”). The file contains about 400 exploitable communication links for the architecture described in Fig.8. A communication link in MAGALI is described as a couple made of one OCC corresponding to the sender IP, and one ICC corresponding to the receiver IP: IP_sender → OCC → ICC → IP_receiver. The description of these links in the architectural model helps in generating communication configurations for the CCC (icc.cfg, occ.cfg, ctx.cfg).

A HAL primitive, for an IP, is described as a set of parameters initialized to run a specific function. The “recgen_00w” IP is used to extend the “top_magali” design in order to simulate data inputs which are used in the considered 3GPP-LTE application. This IP offers two primitives called “send_pilot” and “send_data”. The “send_pilot” allows the sending of the OFDM pilot symbols in order to estimate the transmission channel. The “send_data” primitive allows the sending of the OFDM data symbols which contains the payload data to be transferred over the NoC. The description of these primitives in the architecture model is useful when generating the computation configuration files for the IPs (core.cfg, ctx.cfg).

D. APDL description of 3GPP-LTE application

The application we want to map over the architecture is a digital base band demodulation chain for wireless communication [17].

Figure 9. 3GPP-LTE application modeled with APDL

Fig. 9.a shows an example of the use of an APDL channel in the description of this complex dataflow application. The channel includes an input arbiter (Fig. 9.b). The input matrix defines a single data input scenario (number of column of the matrix) which is repeated 24 times. The conduct of the scenario is defined by writing 4 data into the FIFO of the channel from the first port of the channel coming from “sme_21_coef13” task, then 4 data are written into the FIFO via the second port coming from “sme10_coef24” task. We described the overall application with 16 APDL tasks and 19 APDL channels including 6 multiple-input single-output channels.

E. Description of meta-mapping and mapping

The meta-mapping for MAGALI corresponds to define which application element could be mapped over which platform resource. Fig. 10 shows two mapping constraints. The first one concerns the possibility of mapping an APDL task over an ARDL IP (Fig. 10.a). The second constraint is

about the mapping over the ICC/OCC elements, which are defined as ARDL devices. The meta-mapping allows the mapping of the APDL arbiter or the APDL task port over the ICC/OCC devices (Fig. 10.c 10.b). The mapping model is in conformance with the meta-mapping constraints. The metadata around the data traffic in tasks ports and in arbiters are used to generate communication configuration (icc.cfg, occ.cfg).

Figure 10. Meta-mapping for the MAGALI platform

F. Intermediate models

The GIM model models an abstract deployment of the application over the NoC according to the mapping described above. In Fig. 11 the tag “TaskPortRef” (line 6) explains that the task port “gen00_s_data” is mapped over the OCC element “occ0_rec_gen00s” (line 1).

Figure 11. GIM and SIM models

Figure 12. Communication and computation configurations in MAGALI

This model does not explain how this mapping is translated into configurations for MAGALI. These details are explained in the SIM model which is a refinement of the GIM. In Fig. 11.b we show the information added to the GIM to obtain the SIM. The “simConfiguration” tag (line 8-14) appears comparing to the GIM description of the OCC. Each

“simConfiguration” tag is defined with a type (OCC type for the example in Fig. 11) and a set of parameters. Configurations generation for MAGALI uses a set of tools which extracts directly information from the SIM model.

G. MAGALI configuration files generation

The generated configurations are the computation and the communication configurations detailed in 5.2. In Fig. 12 we display the generated configuration files for the “recgen_00s” node. The “occ.cfg” file (Fig. 12.a) details a data transfer by defining the path to follow through the NoC (EAST EAST RES: sme_21), the amount of data to transmit (7600), the size of each packet (8) and the identifier of the receiver ICC (num_icc 0). This configuration is called by a micro program defined in “ctx.cfg” file (Fig. 12.b). The RC (Request Configuration) primitive is called to execute the referenced configuration detailed in “occ.cfg”. The functions executed by the “recgen_00s” IP are defined as a set of initialized parameters. We illustrate in “core.cfg” file (Fig. 12.c) the configurations of the functions “send pilots” and “send data”. Those configurations are called by a micro program defined in the file “ctx.cfg” through the RC primitive. The micro-program calls “send pilots” once, and then it calls the “send data” function also once.

Some micro programs and configuration files are not generated yet (the CPU and a part of the IP scenarios). The generated configurations for all the IPs are validated over a generated SystemC simulation model of the platform. The number of configuration files to run the 3GPP-LTE application over MAGALI is about 80 configuration files and we are now able to generate about 65 files. Developing and debugging configuration files manually take several weeks, even months. Our approach improves this time to one or two weeks.

VI. CONCLUSION

In this paper, we present a general flow for configuration code generation which target NoC-based platforms and support configurable NI. The generation flow is based on abstract description of application (APDL), architecture (ARDL) and partitioning. We propose a novel way to define mapping rules and how to deploy an application over the architecture. We show in a case study how to model a complex application called 3GPP-LTE with APDL and a complex NoC-based platform called MAGALI with ARDL. We show the benefit of the meta-mapping phase to support the mapping of such a complex application over the MAGALI platform and particularly how to support the configurable NI (including CCC units). To prove the relevance of our solution, we present a part of the final generated configuration files to run the 3GPP-LTE application over MAGALI. Our future works will focus on multi applications and dynamic applications deployments over MAGALI platform.

REFERENCES

- [1] P. Guerrier, A. Greiner, “Generic Architecture for On-chip Packet-switched Interconnections”, DATE 2000, Paris, France, pp. 250-256
- [2] K. Goossens, J. Dielissen, A. Radulescu, “AETHERAL network on chip: concepts, architectures, and implementations”, Design & Test of Computers, IEEE Sept.-Oct. 2005 Volume: 22, Issue: 5 pp. 414-421
- [3] D. Bertozzi, L. Benini, “Xpipes: A Network-On-Chip Architecture for Gigascale Systems-on-Chips” IEEE Circuits and Systems Magazine, v.4(2), 2004, pp. 18-31
- [4] K. Goossens, J. Dielissen, O.P. Gangwal, S.G. Pestana, A. Radulescu, E. Rijpkema, “A Design Flow for application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification”, DATE 2005, pp. 1182-1187
- [5] F. Clermidy, Y. Thonnart, R. Lemaire, P. Vivet, “A Communication and Configuration Controller for NoC based Reconfigurable Data Flow Architecture”, 3rd IEEE International Symposium on Networks-on-Chip (NoCs), May 2009
- [6] A. Hansson, K. Goossens, “Trade-Offs in the Configuration of a Network on Chip for Multiple Use-Cases”, IEEE International Symposium on Networks-on-Chip (NoCs), May 2007, pp. 233-242
- [7] F. Clermidy, D. Varreau, D. Lattard, “A NoC-based communication framework for seamless IP integration in complex systems”, Proceedings of Design and Reuse IP-SOC’2005, France, pp. 279-283
- [8] R. Dafali, J-Ph. Diguët, S. Evain, Y. Eustache, E. Juin, “μSpider CAD Tool: case study of NoC IP generation for FPGA”, in Proc. of Workshop on DASIP’07, Grenoble, France, November 27-29
- [9] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, G. De Micheli, “NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip”. IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 2, February 2005, pp. 113-129.
- [10] V. Nollet, T. Marescaux, P. Avasare, J-Y. Mignolet. “Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles”. DATE 2005, Munich, pp. 234-239
- [11] S. Stuijk, M.C.W. Geilen, T. Basten, “SDF3: SDF For Free “ACSD 2006, pp. 276-278, Turku, Finland, June 2006
- [12] L. Thiele, I. Bacivarov, W. Haid, K. Huang, “Mapping Applications to Tiled Multiprocessor Embedded Systems”, Application of Concurrency to System Design, ACSD 2007, pp. 29-40.
- [13] A. El Mrabti, A. Sheibanyrad, F. Rousseau, F. Pétrot, R. Lemaire, J. Martin, “Abstract Description of System Application and Hardware Architecture for Hardware/Software Code Generation”, DSD’2009, 27-29 August, Patras Greece, pp. 567 – 574
- [14] R. Marculescu, J. Hu, U.Y. Ogras, “Key research problems in NoC design: a holistic perspective” CODES+ISSS, Sept. 2005, pp. 69-74
- [15] SPIRIT consortium Mars 2009, IP-XACT Draft/D3: A specification for XML meta-data and tool interfaces -User Guide v1.5
- [16] Magillem Design Service, web site: www.magillem.com
- [17] D.T. Phan Huy, R. Legouable, D. Kténas, L. Brunel, M. Assaad, “Downlink B3G MIMO OFDMA Link and System Level Performance”, VTC Spring 2008. IEEE, pp. 1975-1979
- [18] S. P. Singh, S. Bhoj, D. Balasubramanian, T. Nagda, D. Bhatia, P. Balsara “Network interface for NoC based architectures”, International Journal of Electronics, vol 94, No. 5, May 2007, pp. 531-547
- [19] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, “Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip”, DATE, 2003, pages 350-355.
- [20] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, P. Wielage, “An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration”, DATE 2004, pp. 878-883
- [21] H. Charlery, A. Greiner, “A Systemc Test Environment For Spin Network”, MIXDES 2006, pp. 449-453
- [22] S. Murali, G. De Micheli, “SUNMAP: a tool for automatic topology selection and generation for NoCs”, DAC 2004, pp. 914-919
- [23] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, G. De Micheli, “Network-On-Chip Design and Synthesis Outlook”, Integration The VLSI journal, vol. 41, num. 2, February 2008, pp. 340-359
- [24] A. Jalabert, S. Murali, L. Benini, G. De Micheli, “xpipesCompiler: a tool for instantiating application specific networks on chip”, DATE 2004, pp. 884-889