ANDRES - ANalysis and Design of run-time REconfigurable, heterogeneous Systems

Andreas Herrholz[⋆], Frank Oppenheimer[⋆], Andreas Schallenberg[†], Wolfgang Nebel[†]
[⋆]OFFIS Institute – [†]Carl v. Ossietzky University – Oldenburg, Germany
andreas.herrholz@offis.de

Christoph Grimm, Markus Damm
Technical University of Vienna, Austria

Fernando Herrera, Eugenio Villar
University of Cantabria, Spain

Ingo Sander, Axel Jantsch
KTH Stockholm, Sweden

Anne-Marie Fouilliart
Thales Communications, France

Marcos Martinez
DS2, Spain

Abstract

In this paper we will present the ANDRES project. The main objective of ANDRES is the development of a seamless design flow for adaptive heterogeneous embedded systems based on the modelling language SystemC. The methodology and tools will enable early integration and exploration of system specifications using different Models of Computation as well as automatic synthesis of hardware and software implementations. The project explores different aspects of adaptivity and will provide means to efficiently use and exploit adaptivity in embedded system design.

1. Introduction

Today, highly integrated embedded systems have a wide range of usage in many innovative European industries, such as telecommunications and automotive. These systems are usually heterogeneous in nature by including up to four different domains: software, analogue hardware, static hardware, and dynamically reconfigurable hardware. Especially the latter one is gaining more importance as it enables adaptive systems with increased flexibility and a wide range of applications.

Up to now, there is no methodology that allows to seamlessly specify, simulate, synthesise and verify such adaptive heterogeneous embedded systems (AHES), because each domain comes with its own computational models, languages and design tools. This prevents early holistic system validation and postpones the system verification to the system integration phase causing long, costly and time consuming design reiterations.

ANDRES is developing solutions to overcome these incompatibilities by creating an integrated modelling approach for AHES. This approach builds on the open-source modelling language SystemC already adopted by many European companies. As a result ANDRES will provide a modelling framework for designing embedded hardware/software systems on a high level of abstraction emphasising in particular the integration of adaptivity. This includes the development of concepts and tools for automatic synthesis of hardware and software implementations.

Achieving these objectives is a major challenge and requires resources and expertise on the European level. Two leading European companies in the field of telecommunication, DS2 and Thales Communications, are joined by four research institutions, OFFIS, Technical University of Vienna, KTH Stockholm and University of Cantabria, providing experiences in modelling and synthesis of embedded systems.

ANDRES is a specific targeted research project (STREP), co-funded by the European Commission within the Sixth Framework Programme. It has started in June 2006 and will last for three years. Additional and future information about ANDRES can be found on the project website [1].

The rest of the paper is structured as follows: We will first present the industrial motivation for the project arising from current issues and problems involved with embedded system design and from the expectations directed towards adaptive systems. Then we will present a general view on adaptivity in embedded systems including the types and architectures of adaptivity that are being considered in ANDRES. Following an overview of the planned design flow, we will briefly present the domain specific modelling libraries. The paper concludes with a look at the synthesis concepts for hard- and software.

2. Industrial Motivation

2.1. Thales

The design of today's embedded systems has to deal with the complexity implied by the combination of various technologies (hardware and software) and the increasing need for (re)configurability (e.g. communication systems have to support several protocols using the same hardware). Modern mobile phones combine for example a PDA, a GPS based navigation system, entertainment and multi-mode communication with all kinds of external devices. Furthermore, these systems have to manage performance and concurrent access on resources, which increases the time of tuning. Dynamically reconfigurable systems have the potential of realising efficient systems as well as providing adaptability to changing system requirements. Another motivation for reconfiguration is resource optimisation of the physical layer; one of the crucial issues today is power consumption.

A software-defined-radio (SDR) design must meet today's reconfigurability requirements to give access to the variety of different standards as well as accommodate cost, power and performance demands. The architecture for an SDR system includes a microprocessor, a DSP and an FPGA performing the high-computational-load filtering and digital download conversions. Field Programmable Gate Arrays (FPGAs) can be configured to perform a specific task. Once configured, the FPGA contains an implementation of an algorithm which is executed much faster and with less energy consumption than a general purpose processor (GPP), mainly due to its inherent parallelism. More and more FPGAs have the ability to change their configuration at runtime. Even if FPGAs are still less flexible than general purpose processors, they are progressing in that direction.

2.2. DS2

One of the most difficult tasks in the design of modern complex embedded home networking systems like powerline communication modems is the validation phase of the devices. In this sense, the use of FPGA technology allows having an early analysis of the behaviour of the system under almost real conditions while providing an extreme flexibility for adding or modifying the design, making them the ideal tool for validating powerline developments.

Dynamic reconfiguration of FPGAs can provide even more capabilities for the validation of this kind of systems: different peripherals can be included on the FPGA on demand. This way, if a specific interface is needed in order to extract information from the device under test, it can be configured and removed on the fly using a reconfigurable object adapting to different test situations without having to stop the whole system. A device can then be run during long tests, where different operations are performed and different equipment is connected, without having to stop the whole test.

This concept can also be extended to the debug phase, where traditionally there has been a problem of visibility of internal signals and monitoring of internal modules on the FPGAs. This limited visibility can be handled properly through the use of reconfigurable objects, adding monitoring modules on the fly on the device under test reusing and sharing the available pins of the FPGA.

3. Adaptivity in Embedded System Design

While the heterogeneity of embedded systems has already been extensively investigated in the past, considering adaptivity is rather new to embedded system design. In particular through the availability of high-performance programmable logic devices, like run-time reconfigurable FPGAs, making systems adaptive has become very attractive by offering a whole new range of possible applications. For instance, if different operating modes are used mutually exclusive, the system may automatically adapt providing only one mode at a time, effectively saving ressources. Other possible applications are the adaptation to future, yet unknown needs or to increase fault tolerance.

3.1 Adaptive Architectures

In general, adaptivity is not limited to one special domain, e.g. the software or reconfigurable hardware domain, but may be expressed and implemented using a range of different adaptive architectures. In ANDRES we consider the following architectures exhibiting various forms of adaptivity:

Microprocessor The microprocessor is highly adaptive due to the possibility to load and execute different programs.

FPGA Some FPGA families support both full and partial reconfigurability during run-time.

Analogue circuits The operation of an analogue circuit can be adapted by changing parameters of analogue components.

Custom hardware Even custom hardware can provide some degree of adaptivity, if a component can be set to different modes.

3.2 Types of Adaptivity

Depending on the chosen architecture and application adaptivity comes in different flavours. In ANDRES we cover the full degree of adaptivity, from setting a few parameters up to reconfiguring the whole device. We mainly focus on *dynamic adaptivity*, where the adaptation of the system is done during run-time. Components can either be self-adaptive or their adaptation is controlled externally. We also consider channel adaptation, where a communication channel adapts to different interfaces of an adaptive component.

ANDRES does not cover the dynamic creation of new components and channels during run-time, e.g. we do not consider spawning of new objects or dynamic creation of processes.

4 ANDRES Design Flow

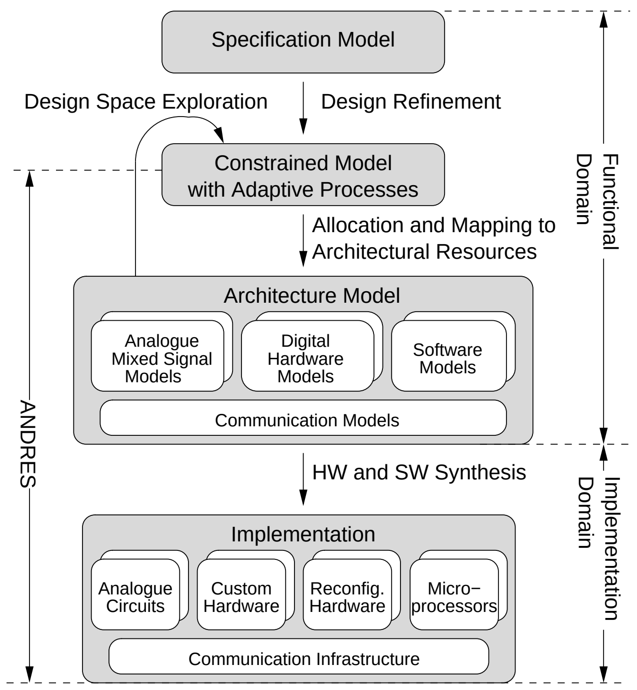

Figure 1. The ANDRES design flow

The design flow in ANDRES, as illustrated in Figure 1, starts with a constrained system model, which already models adaptivity and includes functional and non-functional properties. This constrained model is a SystemC model, whose design rules and guidelines are based on a formal approach (Section 5.1). This model can then be refined to different target domains using one of the three domain-specific modelling libraries that will be provided in ANDRES:

SystemC-AMS A library of building blocks based on SystemC-AMS will be provided to support the design of analogue/mixed-signal dominated communication systems.

HetSC A SystemC based library for system modelling using different Models of Computations (e.g. KPN, CSP) which also provides an entry point for automatic software synthesis.

OSSS+R A modelling library based on SystemC for object-oriented modelling of run-time reconfigurable hardware which will provide direct hardware synthesis capabilities.

Each of the libraries provides means to create executable specifications to simulate system components. Because all libraries are based on the discrete-event kernel of SystemC they can be coupled to provide system level simulation for analysis and validation of the overall system. Communication between system components is modelled using a concept called *polymorphic signals* (Section 5.5). These signals support fast design exploration as the Models of Computation (MoC) of different system components can be exchanged without explicitly changing the underlying communication infrastructure.

The general adaptivity concept already expressed in the initial model, is specialised within each of the libraries using domain specific techniques. Because not all target architectures provide the same types of adaptivity, e.g. analogue circuits only allow parametrisation, exploration of different adaptive architectures may be limited.

Finally the refined SystemC system model is the entry point for automatic synthesis. ANDRES is developing tools for automatic synthesis of digital reconfigurable hardware, software and communication infrastructure. However, ANDRES does not cover automatic synthesis of analogue circuits.

5. Specification of AHES

5.1 Formal Specification

The modelling framework in ANDRES is based on the existing ForSyDe [5, 8] framework developed at KTH. ANDRES extends ForSyDe by integration of adaptivity into the modelling framework. This is done by the concept of an *adaptive process*, which changes its behaviour depending on special input signals from the environment. The values carried by these input signals can be data values, but also functions or complete processes. Thus adaptation can be modelled at a varying degree of complexity.

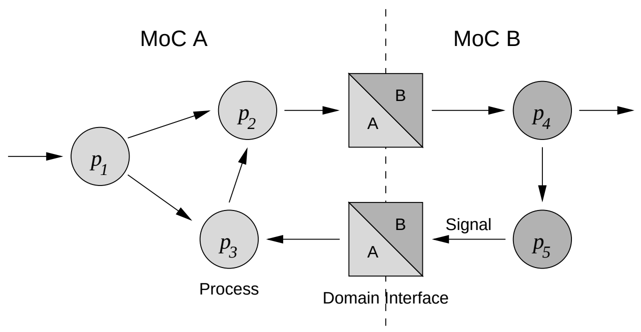

Figure 2. Processes of different Models of Computation can communicate with each other via domain interfaces

5.1.1 Model of Computation

ANDRES uses a formally defined, hierarchical heterogeneous MoC, which is illustrated in Figure 2. Processes communicate via signals. There are so-called domain interfaces to formally define the interaction between processes of different computational models. The following MoCs are used in ANDRES:

- untimed model
- synchronous model
- discrete time model
- continuous time model

Since the designer uses SystemC as a modelling language, the ANDRES project will define modelling rules and guidelines, which guarantee the SystemC models to be compliant with the ANDRES modelling formalism. Thus methods developed for this framework, e.g. for property analysis, verification or transformation, will also be applicable to the ANDRES SystemC models.

5.1.2 Modelling of Adaptivity

Adaptivity is modelled by means of formally defined adaptive processes. The functionality that the adaptive process computes can be changed from the environment depending on the value of an input signal.

Figure 3 shows a simple, but typical example for the modelling of adaptivity. The processes *Encoder* and *Decoder* are both adaptive processes and are fed with signals carrying functions. These functions change the behaviour of the encoder. In the first cycle the encoding function adds one, while in the second cycle two is added to the incoming value. The same adaptivity mechanism is used for the decoder. To yield an implementation this model is refined and

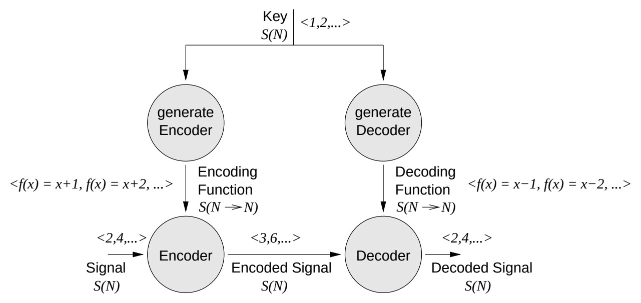

Figure 3. The Encoder/Decoder is a typical example for adaptivity.

non-functional characteristics, like reconfiguration time, are taken into account in subsequent design phases and the associated model.

5.2 SystemC-AMS

The OSCI working group SystemC-AMS is currently working on a prototype allowing designers to simulate systems that combine data-flow modelling (using the Synchronous Data Flow (SDF) MoC), analogue circuits (using the continuous time network (CT-NET) MoC), and rather control oriented digital circuits (using the discrete-event (DE) MoC). Compared with circuit simulators such as SPICE, the focus of SystemC-AMS is on executable specification, design space exploration and virtual prototyping of signal processing and analogue/mixed-signal systems [12]. All these use-cases require high simulation performance while less accurate simulation is acceptable.

An important feature of SystemC-AMS is its extensibility by other Models of Computation. Often, application or abstraction specific methods allow designers a more efficient modelling, or result in higher simulation performance. An example is the simulation of linear networks, which can be orders of magnitudes faster than the more general numerical integration for non-linear networks. New ”solvers” for alternative MoCs can be integrated easily into the synchronization layer. The synchronisation layer supports directed communication and only a simple (but efficient) synchronisation on user specified events or in fixed time steps. This reflects the requirements for high simulation performance at reduced simulation accuracy. This extensibility makes SystemC-AMS a very good integration platform for system level simulations.

In ANDRES for modelling AHES, like in the formal specification of adaptivity, different sets of possible functional behaviour can be expressed by overloading virtual methods. Parametrisation and other kinds of adaptivity can be modelled by parameters, and can be made more explicit by attributes. The means for specifying and mod-

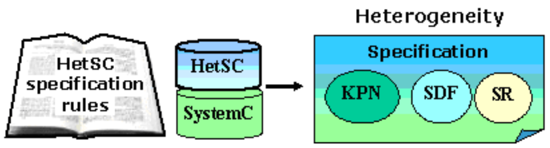

Figure 4. HetSC specification methodology

elling adaptivity will be encapsulated within a class that is inherited from an adaptive object class specified in the original system model. To support system level design of adaptive communication systems, a library of building blocks is being developed. This building block library will support designers in evaluating different variants of reconfigurable signal processing systems by comparing e.g. switched capacitor or parametrisable analogue circuits with digital hardware or software realisations.

5.3 HetSC

HetSC [3] is a methodology for enabling heterogeneous specification of complex embedded systems in SystemC (see Figure 4). Computational models supported include untimed MoCs, synchronous MoCs and the timed MoCs already supported by SystemC. Though HetSC aims at a complete system-level HW/SW codesign flow, in ANDRES HetSC will be integrated into the design flow for modelling and generation of embedded software (see Figure 5). Providing several abstract MoCs (e.g. KPN, PN, CSP, SR, etc.), HetSC enables a more intuitive and safer design of concurrent software systems adjusting to different specification needs of the designer. In addition, the HetSC methodology defines a set of specification rules and coding guidelines for each specific MoC making the design task more systematic. The fulfillment of MoC specification rules provides useful properties for concurrent software, such as determinism, deadlock protection, etc, which can easily be lost when using plain SystemC language elements.

The HetSC library, associated to the HetSC methodology, provides a set of facilities to cover the deficiencies of the SystemC core language for heterogeneous specification. The support of some MoCs requires new specification facilities with specific semantics and abstraction levels. In addition, some facilities of the HetSC library help to detect and locate MoC rule violations and assist in debugging concurrent specifications.

Though an important part of the work on the software design flow based on HetSC has already been developed [2], several tasks are being carried out for integrating the HetSC methodology into the AHES design framework:

- Formalisation of the HetSC methodology under the ForSyDe formal model of AHES.

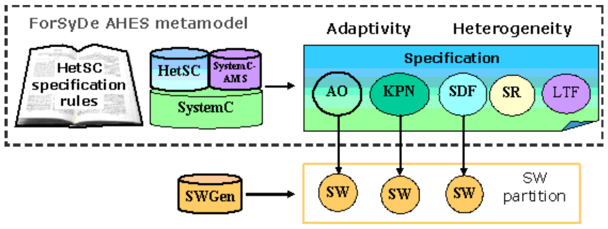

Figure 5. HetSC in ANDRES

- Providing the HetSC methodology with the capability for specifying Adaptive Objects (AO), thus adaptivity.
- Connecting HetSC with other SystemC based specification methodologies.
- Defining and providing specific features for software modelling.

The ForSyDe formal model provides means to analyse, verify and transform system specifications and is being extended to support AHES. To be able to apply these methods to HetSC specifications, the HetSC methodology needs to be formalised in ForSyDe. To assist this task the ForSyDe formal model is being implemented using HetSC.

Regarding the specification of adaptivity, HetSC will be able to capture adaptive objects (AO). For this, beside considering the different types of adaptivity defined in ForSyDe, the MoC employed has to be taken into account, too. That is, the MoC affects when and how adaptation is performed. For instance, given the untimed KPN MoC, adaption is synchronized with the inputs and outputs of a process by means of an uc_fifo channel, a specialized type of channel provided by HetSC. The only information about the adaptation time is an order relationship between the adaption and other process computations. However, in a clocked synchronous MoC, adaptation can take place at explicit clock ticks. This is a more precise timing information about adaptation than in the untimed case.

Finally, since the HetSC methodology will be integrated into the AHES modelling framework, other tasks deal with the interoperability and connection with other ANDRES specification methodologies. Specifically, the connection with SystemC-AMS is being studied and enabled.

5.4 OSSS+R

The latest generation of FPGAs can be partially reconfigured during run-time without interfering with the rest of the design. Though this enables the creation of dynamically adaptive hardware, current design tools lack the support for this kind of application at higher levels. OSSS+R is a SystemC based modelling library providing high-level language constructs to model (self-)reconfigurable hardware systems. Additionally OSSS+R keeps a well-defined synthesis semantics providing a direct entry point for automatic hardware synthesis (Section 6.2).

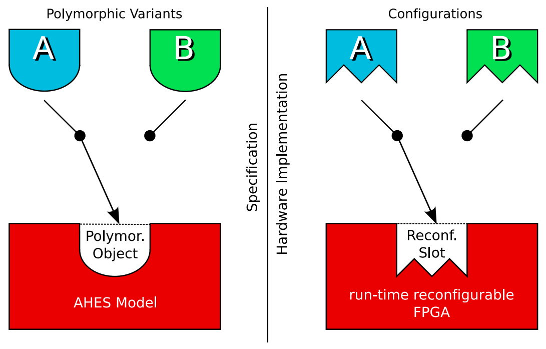

Figure 6. Polymorphism and configurations

Based on the original OSSS approach of combining object-orientation and hardware design [6], in OSSS+R object-orientation is used as an adequate abstraction mechanism for dynamically reconfigurable hardware. The concept is based on the assumption that changing functions of parts of a hardware system largely resembles the use of polymorphism in object-oriented software design [9].

Polymorphism, as it is used in object-oriented programming, enables calling methods on an object, whose exact type is unknown to the caller. The only known reference to the object is its *interface*. Depending on the actual class of the object, the corresponding implementation of a method is executed. This technique enables changing parts of the software at run-time without modifying the static part of the code.

Considering a digital hardware system consisting of a static and a dynamically reconfigurable part, it is obvious that the interface between the two parts needs to be fixed. However, the implemented functionality of the reconfigurable hardware may change. Hence, the key idea of OSSS+R is to model the reconfigurable area of a hardware system as an adaptive (polymorphic) object with a fixed interface. This *interface* is defined by a base class, while its possible variants belong to different subclasses. During runtime, different variants of the adaptive object can be configured and used (see Figure 6).

To handle the management of different object configurations and to ensure persistance OSSS+R introduces *Named Contexts*. A context represents all relevant information of an object, including its current type and state. From the designer's point of view, a context is used similar to a C++ pointer, so objects can be assigned to it and methods can be called arbitrarily. However, automatically instantiated infrastructure ensures that a context is enabled (i.e. configured) only if it is accessed. Additionally its state is automatically saved and restored during consecutive reconfigurations. Because a context can be accessed concurrently, incoming request are serialised using a built-in scheduler. Contexts hide the complexity of configuration management and state preservation and enable the designer to use adaptivity transparently.

The concept of OSSS+R focuses on run-time reconfigurable FPGAs and resembles the concept of adaptivity through configuration. Simulating a design provides a cycle-accurate simulation of an FPGA implementation including annotated reconfiguration times. In ANDRES OSSS+R is being further extended to support a wider range of adaptive architectures at higher levels of abstractions enabling faster and easier exploration of different adaptive architecures.

5.5 Polymorphic Signals

When modelling heterogeneous systems, typically several different MoCs are used for different subsystems. Moreover, the MoC used for a specific system part may change during system development, e.g. when passing over (due to a top-down refinement) from an abstract description of a low pass filter using a transfer function within the SDF-MoC to an explicit one using an RC circuit within a continuous time MoC.

Therefore, two problems arise: to use appropriate converter modules when connecting systems parts described within different MoCs, and to exchange these when changing the MoC for a specific subsystem. This has to be done manually and is therefore time consuming and potentially error-prone. To overcome these problems, the concept of *polymorphic signals* was developed and prototypically implemented in SystemC-AMS [10].

Polymorphic signals are able to convert value types (e.g. real to bit-vector) and semantics (e.g. SDF-MoC to DE-MoC) automatically. This is achieved by instantiating appropriate converter modules depending on the types of the ports the signal is connected to, such that the designer has not to be concerned with this. Another application of polymorphic signals is simulator coupling [11].

Polymorphic signals will be used within ANDRES to support the interactive performance analysis of heterogeneous systems by mixed level simulation.

6. Synthesis of AHES

6.1 Synthesis of Software

In ANDRES, HetSC will be the entry point for software development and synthesis. In [2] an automatic software generation methodology based on HetSC has been presented based on a library called SWGen. This library enables the generation of target code (source and binary) corresponding to the software partition of a HetSC specification. Because there is no modification of the original specification code needed, this technique is called *single-source software generation* [7]. In ANDRES, this work will be reused, improved and adapted to the AHES design flow (see Figure 5).

In the SWGen methodology, the SystemC implementation of the HetSC specification facilities (provided by the HetSC and SystemC libraries) is substituted by a C/C++ implementation provided by the SWGen library, which is based on real-time operating system (RTOS) calls belonging to one of the supported RTOS APIs. The generated software code is equivalent to the specification code in a sense that it preserves the properties guaranteed by the specification. Most of this is provided by preserving the structures of concurrency and hierarchy of the specification in the generated code. Thus, most of the MoC rules, which impose constraints on structure, are kept. However, the SWGen methodology also involves some kind of MoC refinement since the underlying DE simulation kernel of SystemC is substituted by a RTOS runtime library. This refinement preserves the semantics of the user view, but uses the resources and assumptions of the target MoC, i.e. the RTOS runtime library. In ANDRES, using ForSyDe will be helpful for formalising the refinements performed by the software generation methodology.

One particular task in ANDRES regarding software synthesis is considering what adaptivity actually means in software. A basic interpretation is to view adaptation as multiplexing different functionalities over a shared execution resource. In software, these are usually the processor and the memory hierarchy. This drives to already well-known software concepts and techniques, such as process scheduling, context switching, virtual memory, etc. Most of these are already handled by RTOS. However, in ANDRES, the SWGen methodology will be extended to support software generation from a HetSC adaptive object (AO). Assigning two or more exclusive functionalities to an AO in the HetSC specification will induce the SWGen methodology to map those functions to the same resource, i.e. the same processor, using the underlying services of the RTOS. This might be useful, for instance, to save processing resources in a multiprocesor SoC architecture (MPSoC). However this requires the selected RTOS to support multiprocessor architectures and to provide corresponding system calls.

6.2 Synthesis of run-time reconfigurable digital Hardware

One of the major goals of the ANDRES project concerning digital hardware is the automated synthesis of reconfigurable hardware architectures. In this case "Synthesis" covers the translation of a given OSSS+R model to RT-level VHDL, which in turn serves as input for third-party backend tools, e.g. Xilinx' *Early Access Partial Reconfiguration Design Flow* [13] (see Figure 7). The synthesis tool FOSSY (Functional Oldenburg System SYnthesizer) for OSSS will be extended to support the language constructs for reconfigurable components introduced with OSSS+R.

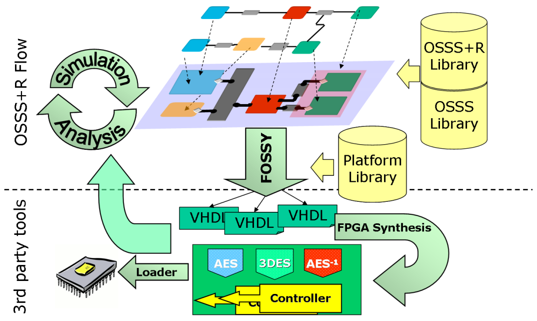

Figure 7. OSSS+R synthesis flow

The major transformation step towards a pure RTL design from a OSSS+R model consists of the generation of various management structures. Additional to the application and annotations given by the designer, different arbitration mechanisms, structural information (e.g. FPGA types) etc. need to be considered. The generated infrastructure consists of a set of hierarchically organised controllers. A set of distributed controllers for each reconfigurable area handles access requests by the static design parts. Each access controller uses a central reconfiguration controller per device to accomplish reconfigurations. That unit resolves conflicts between different distributed controllers and provides an interface to the FPGA's configuration port.

The required interfaces to the reconfigurable areas can be determined during synthesis by analysing the interfaces of the corresponding *Named Contexts* that are bound to the area. This even allows the synthesis of static signal-level interfaces for unrelated interfaces bound to a single reconfigurable area on the application layer.

For each possible functional content of a *Named Context*, a VHDL implementation of the behaviour is generated separately. In the later steps of the synthesis flow each of these functional blocks can be used for the generation of the required partial bitstreams.

6.3 Synthesis of Communication Interfaces

Beside the automatic synthesis of hard- and software components, ANDRES is developing means for automatic synthesis of communication infrastructure between those components. In particular, the communication between adaptive objects will be studied. One primary goal is the automatic generation of hardware/software interfaces, which will enable, for example, to access and to control adaptive hardware objects from software and vice versa. However, also the interaction between the digital hardware/software and the analogue part is investigated.

The basis for a methodology and corresponding tools for communication synthesis are on one hand the communication modelling framework provided by the *polymorphic signals* (see Section 5.5) and on the other hand the approaches for communication refinement and synthesis developed by the ICODES project [4]. While the polymorphic signals provide means to connect models with different computational models, they might not provide enough information to efficiently synthesise communication infrastructure between them. Here the methods developed by ICODES will be adopted for communication refinement in adaptive heterogeneous architectures. As this affects all domain specific modelling libraries, it will require a close cooperation of the involved research partners.

As a result, a synthesis tool, based on FOSSY, will be developed that enables automatic generation of signal-level and software interfaces from abstract interface descriptions for integration of adaptive objects into a heterogeneous system environment. As these abstract interfaces can efficiently be synthesised to fit to different communication structures (buses, networks-on-chip, etc), different architectures can be explored and evaluated.

7. Conclusion

This paper presented motivation, goals and ongoing work of the ANDRES project. While the heterogeneous nature of embedded systems still yields many problems in current design flows due to different computational models, languages and tools, these methodologies particularly lack support for adaptive architectures. To resolve this issues, ANDRES is developing a seamless design flow for such adaptive heterogeneous embedded system and corresponding tools including automatic synthesis of software and runtime reconfigurable digital hardware, like FPGAs.

Based on a formalism to express and analyse adaptivity in different Models of Computation, three SystemC based modelling libraries are used for specification, simulation and analysis of adaptive system designs: SystemC-AMS for analogue/mixed-signal components, HetSC for software and OSSS+R for run-time reconfigurable digital hardware. An overall framework includes and connects these libraries using a concept calls *polymorphic signals*. This modelling framework provides direct entry points for automatic and efficient synthesis of hardware, software and communication interfaces. These synthesis concepts particularly consider different types of adaptivity and adaptive architectures and will be implemented in corresponding synthesis tools.

The concepts and tools of ANDRES are evaluated using industrial use-cases. As a result, ANDRES will provide a complete design flow and tools based on SystemC. At the end of the project, the modelling framework is planned to be released to the public.

References

[1] ANDRES *project*. `http://andres.offis.de`.

[2] V. Fernandez, F. Herrera, P. Sanchez, and E. Villar. *Embedded Software Generation from SystemC*, chapter 9, pages 247 – 272. Kluwer, March 2003.

[3] F. Herrera and E. Villar. A framework for embedded system specification under different models of computation in SystemC. In *Proceedings of the Design Automation Conference*, 2006.

[4] ICODES *project*. `http://icodes.offis.de`.

[5] A. Jantsch. *Modelling Embedded Systems and SoCs*. Morgan Kaufmann, June 2003.

[6] H. Kleen, T. Schubert, and C. Grabbe. A tutorial for OSSS. Technical report, OFFIS Institute, Oldenburg, Germany, January 2006. http://icodes.offis.de.

[7] H. Posadas, F. Herrera, V. Fernandez, P. Sanchez, and E. Villar. Single source design environment for embedded systems based on SystemC. *Transactions on Design Automation of Electronic Embedded Systems*, 9(4):293 – 312, December 2004.

[8] I. Sander and A. Jantsch. System Modeling and Transformational Design Refinement in ForSyDe. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(1):17–32, January 2004.

[9] A. Schallenberg, F. Oppenheimer, and W. Nebel. OSSS+R: Modelling and Simulating Self-Reconfigurable Systems. In *Proceedings - 2006 International Conference on Field Programmable Logic and Applications*, pages 177–182, Aug. 2006.

[10] R. Scholl, C. Grimm, and K. Waldschmidt. Heaven: A Framework for the Refinement of Heterogeneous Systems. In *Proceeding of the Forum on Specification and Design Languages (FDL '04), Lille, France*, Sept. 2004.

[11] R. Schroll. *Design komplexer heterogener Systeme mit Polymorphen Signalen*. PhD thesis, Institut für Informatik, Universität Frankfurt am Main, 2007.

[12] A. Vachoux, C. Grimm, and K. Einwich. Towards Analog and Mixed-Signal SoC Design with SystemC-AMS. In *IEEE International Workshop on Electronic Design, Test and Applications (DELTA'04)*, Perth, Australia, 2004.

[13] Xilinx, Inc. *Early Access Partial Reconfiguration User Guide (UG208)*, Mar. 2006.