Holistic System Modeling and Refinement of Interconnected Microelectronic Systems

Jochen Zimmermann  
Oliver Bringmann  
FZI Karlsruhe  
zimmermann@fzi.de  
bringmann@fzi.de

Joachim Gerlach  
Robert Bosch GmbH  
joachim.gerlach@de.bosch.com

Florian Schaefer  
Cadence Design Systems  
schaefer@cadence.com

Ulrich Nageldinger  
Infineon Technologies AG  
ulrich.nageldinger@infineon.com

Abstract—In this paper, we propose an approach for modeling distributed embedded systems in a holistic way starting from an abstract specification of system requirements. We use the Unified Modeling Language (UML), which is very popular in software modeling and development, for describing both the target platform and the functionality, which has to be performed on that target platform, at a high abstraction level - even at system level. Therefore, we extended the existing UML profile MARTE to meet the requirements of distributed systems. These extensions enable to associate the elements of the model semantically and hence to use UML as a common underlying data model for system representation. Based on the holistic modeling approach an executable simulation model is generated in SystemC to facilitate exploration and verification of system behavior.

I. INTRODUCTION

In recent years it has turned out that the driving forces for new end-user applications are interactivity, mobility and individual need for safety, security, comfort and health in a highly interconnected world.

The applications rely on interconnected individual processing power combined with the networking of all information and communication technologies. Examples are not restricted to communication systems. For example, modern vehicles contain a network of interconnected electronic control units (ECUs) communicating via different busses using several protocols (e.g. CAN, MOST, FlexRay), and tomorrow’s automotive networks will have to support highly safety-critical and reliable systems, like X-by-wire. Current and future mobile phones resemble heterogeneous system conglomerates of both chip components, which may again contain complex subsystems (e.g. baseband, RF, application processor, memory) and advanced peripherals, like cameras or memory card interfaces, communicating over a number of both on-chip and off-chip interconnects (e.g. I2C, I2S, I2R). Thus, microelectronic systems migrate towards highly interconnected “systems of systems” resulting in a growing degree of complexity and interaction. An increasing number of subsystems are involved in providing a specific system function, and more and more system functions are sharing a heterogeneous network of distributed resources. In this context, interaction becomes even the most critical challenge of the design process.

In future, new system functionalities will be built less by isolated components and more by interaction of components within an interconnected system environment. Therefore, interconnection and networking will become the main drivers for innovation. The added value of a product will increasingly result from synergetic networking. As a consequence, the role of suppliers will also change significantly: The responsibility for the overall system behavior and reliability will be more and more devolved to the component supplier. In this new position, the supplier is responsible not only for individual component design but also for the integration of the component into the embedding system.

Following today’s design strategies, components are specified independently and integrated afterwards, environmental requirements are not considered sufficiently into the interconnected system model, and a seamless link to executable high-level system models is missing.

This also results in new challenges for microelectronic system design. As today’s design methodologies and tools are mostly oriented to design single components, they are hardly able to support the design of interconnected and distributed system solutions with high demands on flexibility and reliability. Thus, the various dependencies of system networking are considered insufficiently. As a consequence, future designs need to be done in the context of their embedding hardware and software system, and have to consider the communication characteristics and requirements of their embedding environment. This requires a paradigm shift in microelectronic system design methods. To cope with that challenge we propose a holistic modeling approach for distributed microelectronic systems based on the Unified Modeling Language (UML) which combines both hardware and software view in an integrated system development process.

II. PLATFORM MODELING

Generally, in a platform-based design of an embedded system functional units, which realize the requested system behavior, are mapped onto architectural components through a systematic process. Then, the resulting platform is optimized until all necessary system requirements are met.

In the following, the modeling of both behavior and target architecture in UML and the mapping of functional units onto platform components is described in details. In the
context of this modeling approach, target architecture means the underlying hardware platform.

A. Target Architecture Modeling

The Unified Modeling Language (UML) is commonly used for modeling software systems both in the specification and the development process. Whereas in the microelectronic world, UML is almost unknown because it’s considered to be too complex and time-consuming. Moreover, the inside of hardware blocks is usually not revealed to protect intellectual property (IP).

But as stated before, supplier of microelectronic systems are more and more not only responsible for individual component design but also for the integration of the component into the embedding system. Therefore, many supplier provide a component specification in a standard IP-XACT format to enable an independent system integration.

In our approach, IP-XACT \[2\] serves as a basis for the modeling of the underlying hardware architecture. IP-XACT provides a common specification mechanism for describing and handling intellectual property (IP), e.g. definitions of interfaces, ports, and address spaces. The component description used to be mainly at RT level but since version 1.4, IP-XACT methodology has been accumulated with ESL extensions which include module hierarchy and multiple views of different levels for a component supporting mixed IP modeling abstraction levels.

Starting from standard component description of microelectronic processor and interconnection resources in IP-XACT these components are integrated into existing and well-known UML design environments. The result is a UML component library which can be used for structural modeling in various UML diagrams. Therefore, we transform IP-XACT component descriptions via XMI to environment-specific UML components. This UML components encapsulate UML classes as attributes to represent component elements, e.g. address spaces, registers, and ports. Furthermore, UML components and classes can hold operations which are specified through an IP-XACT file set, e.g. C functions of an encoder.

By applying a domain-specific UML profile in the export process the semantics of the modeling elements is defined. This forms a common modeling basis for a connection between target architecture and behavior, and also between hardware and software development.

The UML profile MARTE \[3\], which currently resides in the standardization process of the Object Management Group (OMG) \[4\], has been developed for modeling real-time and embedded systems. Nevertheless, it has turned out that the MARTE profile is not sufficient to model microelectronic systems in a complete manner – especially not in the System-on-Chip (SoC) area.

So, we had to extend MARTE by a domain-specific UML profile which also allows the modeling of registers, protocols, and so on. We added appropriate stereotypes and tagged values to adapt the UML meta model to the SoC application area.

During the import process of IP-XACT component descriptions to the UML design environment the generated UML classes, which are going to be generated, are denoted by stereotypes. Specific recognition features are used to perform a semantic classification, e.g. whether there are bus interfaces, or registers, or internal channels defined in the component description. Finally, the assignment of stereotypes is done by regarding a pre-defined mapping table.

In a UML class diagram these generated classes are put into relationship to each other, e.g. associations or generalizations, representing the internal structure of the modeled system. Since UML version 2.0 the coherence of complex system architectures and especially the interconnection of system components are modeled in a component diagram, which can be created using our generation approach.

![Component Diagram for High-Level Platform Modeling](image)

Figure 1 depicts a multi processor platform which consists of two bus units from the AMBA family, two Leon2 processors, a hardware RAM and an ASIC. The abstraction level of the representation depends on the view of the model and is therefore implicitly defined by using a certain type of UML diagrams. In a structural view, the component ports and their connection is basically relevant. For this reason a component diagram abstracts away all internal attributes and simply shows the components including their component ports and the dedicated communication network. Whereas in a component-centric view, the model is rather focused on component attributes, and tagged values respectively. Hence, this diagram type forms an interface to the behavioral modeling which is described in section II-C.

This UML platform model can also be encapsulated in a component to be available in further models as a parameterizable subsystem. This so-called blackbox approach of platform templates is very practical if subsystems are already verified or included in system design as intellectual property.
B. Architecture Refinement

To manage the complexity of today’s microelectronic designs, both the architecture and the communication of distributed system elements have to be modeled at a high abstraction level.

Figure 2 depicts a processor and a bus which are linked together by a simple connection modeling a high-level communication. Details of this communication, e.g. types of ports, are abstracted away to allow to connect system components without the need to deal with low level signals, appropriate interfaces, protocols, and so on. Designer are enabled to model distributed systems fast and easily.

As a matter of course, these high-level models are not accurately enough when implementation of the modeled design is concerned. In an IP-XACT component, bus interfaces are groups of ports that belong to an identified bus type, and represent the connection of the component to the outside. Group of ports that together perform a function are defined by the busDefinition and abstractionDefinition, which contains the low-level attributes of an interface. A component bus interface refers to a certain abstractionDefinition and therefore indicates that the corresponding port implements the specified port function.

Figure 3, the interface concept of IP-XACT is mapped to UML interfaces. Port definitions of the abstractionDefinition, namely RW_TRANSACTION, are transformed to UML classes and get associated with the specified component interfaces which they refer to. As a consequence, component ports of type RW_TRANSACTION, which are represented as UML ports, can be provided with the appropriate interfaces.

With this approach, communication channels which are modeled at a high abstraction level (cf. figure 2) are refined to a lower abstraction level adding defined interfaces to the model. In the component diagram these interfaces are linked together representing component interconnection. The refined communication interconnect is depicted in figure 4.

The target platform including communication refinement is displayed in figure 5.

C. Modeling of Behavior

To model functional behavior UML provides behavioral diagrams like activity diagrams, sequence diagrams, and final state machines. The export process of IP-XACT component description, which is described in section II-A, and the generation of UML component library in the UML design environment, enables system designers to work directly on instances of these components. Component dependencies are implicitly defined by using component representations in UML class diagrams.

As an example, operation rd_mem() of a processor reads some data out of a memory and therefore implements a protocol which consists of several protocol steps. This operation can be modeled precisely using a sequence diagram which embraces instances of CPU1, AHB, and RAM in figure 1.

Furthermore, state machines can directly refer to configuration registers, which are also defined in IP-XACT format. With these reference to configuration registers the control flow of the component or the system is modeled. Depending on current register values different component algorithms are triggered, e.g. filter algorithms to process a continuous data flow in mobile communication.
Another way to specify functional behavior are pure function calls to existing source code, e.g. in C/C++ or VHDL. The methodology of behavior from source code is thereby independent from being implemented in hardware or software.

Modeling of behavior depending on specified system component resources plays a key role in the generation process of virtual prototypes starting from an abstract UML system model. This process is described in section IV.

III. SYSTEM MAPPING

By the use of UML as a common data model for both the target architecture and the functionality there exists a model-based relation between different views of system integration, e.g. hardware and software development. This facilitates the composition of components to a target platform at a high abstraction level.

Additionally, by applying further UML profiles to the system model, e.g. SysML [5], non-functional properties like performance, reliability, and timing behavior can be factored in the model. Thus, an explicit interface from system modeling to requirements management and the verification of pre-defined requirements occurs.

The junction of the different views mentioned above is also one part of the coherent modeling methodology based on UML as underlying common data model. In this process, it has to determined which functionality is mapped, and executed respectively, on which available resource in the model. UML provides deployment diagrams to realize this mapping function.

Figure 5 shows the mapping process of structural and behavioral diagrams in general. The modeling of the target platform has already been described in section II-A. This target platform model is represented by XML code, which is automatically generated by the UML design environment. Both hardware and software functionality is encapsulated into UML artefacts.
which are mapped onto structural elements in a deployment diagram. These structural elements, e.g. UML devices, represent abstract system components. By this mapping process of functionality, which is extracted from behavioral diagrams, or directly from source code, and represented by communicating processes, an abstract system model is generated. In further steps, this abstract model can be transferred into an executable model which is used to perform the refinement of the entire system and to explore the design space regarding different optimization strategies.

To combine platform modeling and function mapping UML artefacts can be mapped directly on components in a component diagram. This is depicted in figure 7. Therefore, holistic system modeling can be done in an integrated modeling design flow.

Fig. 7. Mapping of Behavior Elements onto Target Architecture Components

IV. GENERATION OF EXECUTABLE MODEL

The methodology of our holistic system modeling approach enables to perform model transformations leading to both a model-based platform refinement and an automatic generation of virtual prototypes starting from the common data model in UML. The transformation steps can be made on different levels, e.g. a model-to-model transformations processing XML model code, and depending on further analysis and exploration processes which are based on component characteristics. This design flow is depicted in figure 8.

By combining the system modeling and concrete component models the macro-architecture, which has been developed through the UML modeling and function mapping process, is transferred into a micro-architecture. This micro-architecture, which depends on the transformation steps mentioned above, and of course the desired abstraction level, is transformed into SystemC.

In listing 1, a SystemC module is generated to implement a UML processor component. For each function, which has to be executed on this processor, a SystemC process is being created. Further elements of the component model, e.g. registers, are mapped onto SystemC data structures with suitable data types. Additionally to that, a convenient API is generated to offer a uniform access to component elements. This approach provides dedicated interfaces from the development of functionality to the underlying target platform.
#include <systemc.h>
#include "ahb_master_interface.h"

class cpu1 : public sc_module
{
    public:
    sc_port<AHB_MASTER> initport;
    ... 
    sc_bv<32> reg1; // register 1
    ...
    void function1();  // specified function
    ...
    sc_bv<32> getValueOfReg1; //convenient method
    ...
    cpu1(sc_module_name name) : sc_module(name)
    {
        SC_METHOD(function1);
        ...
    }
};

Listing 1. Generated Component Module in SystemC

To manage component interconnections at an abstraction level just below considering only communicating processes, SystemC channel communication is generated to implement both SystemC and IP-XACT interface concepts. An excerpt of the generated code is shown in listing 2.

#include <systemc.h>
#include "ahb_master_interface.h"

template<typename T>
class RW_TRANSACTION : public sc_channel, public AHB_MASTER,
{
    public:
    void read(T&);
    ...
    RW_TRANSACTION(sc_module_name name) :
    _sc_module(name)
    {
        ...
    }
};

Listing 2. Generated Communication Channel

V. CONCLUSION

In this paper we have presented an approach to model both the target architecture and the behavior of an embedded distributed system in a holistic way. Starting from a component library, which is generated from IP-XACT component descriptions, UML serves as an underlying common data model for this system modeling process. The component generation approach can be easily integrated into UML design environments. By mapping the system behavior onto instances of the component library we are able to generate an abstract system model which can be refined to an executable model in SystemC.

REFERENCES