A Concept for Design of Embedded Systems at Semantic Level

Frank Wawrzik, Javier Moreno Molina, Christoph Grimm
AG Design of Cyber-Physical Systems
Technical University of Kaiserslautern
Kaiserslautern, Germany
{wawrzik|moreno|grimm}@cs.uni-kl.de

Abstract—Until now, there is a gap of several weeks of modeling efforts between an initial block diagram sketched on a blackboard and first executable models for e.g. architecture exploration and virtual prototyping. Being able to simulate during or short after initial discussions would be of great benefit. We propose a concept that aims at putting the design of embedded systems at a higher level of abstraction. The idea is to provide “smart” models that fill missing details with context knowledge. In particular, communication between SW and peripherals is replaced by (semantic) references to ontologies. For demonstration, a model of an electronic throttle control is modeled at semantic level.

Index Terms—Ontology Library, Parameterization, Semantic Modeling in Simulation, TLM Abstraction, SystemC

I. INTRODUCTION

The design of embedded systems usually starts with a block diagram sketched on a blackboard; this usually happens within a meeting. Intuitively, all engineers understand how the system will work. For executable specification and architecture exploration, models are then created. The development of these models takes several weeks or longer. Even worse, these models are required to start development of software and hardware: As a contract between hardware and software development, at least a (validated) register map must be implemented.

We propose a new approach to put design at a higher level of abstraction: Using SystemC, a simple structure of a block diagram is instantiated. The blocks used come from a library of functional blocks that model typical peripheral units (e.g. ADC, DAC, PWM) at functional level. Communication is not modeled by addressing memory-mapped registers using TLM, but instead by using abstract XML commands that refer to abstract functionalities. This allows starting SW- and method development even without knowledge of a register map or details of peripherals. Instead, we exploit shared knowledge represented in ontologies. As binary communication is abstracted to commands that are references to semantic knowledge, we refer to the method as “design at semantic level”. The register map can – once it is fixed – be introduced later on e.g. by communication synthesis.

The paper is structured as follows: In the following paragraph, we give a brief discussion of related work. In Section II we describe the proposed methodology and ontology library. In Section III we give a first use case. Section IV concludes with a discussion and outlook to future work.

A. Related work

In particular in the synthesis of analog circuits, knowledge-based systems are used for configuration of analog circuits (e.g. OASYS [1], KANDIS [2]). In these tools, the knowledge-based design approach targets structural refinement of analog circuits, but not modeling of HW/SW interfaces.

IP-based design aims at supporting re-use considering parameterization changes [5], IP enhancement and database management [4] and simple ontologies to structure libraries [3]. In particular IP-XACT [10] allows describing register maps. While IP based re-use increases productivity, it has not been used for abstracting communication and generating new models.

Knowledge-based approaches and ontologies for modeling/simulation are a quite new topic [6]. Eriksson [7] proposes an approach for simulation of infectious diseases with ontologies, separating modeling from code and execution.

Communication abstraction and synthesis have not yet tackled the idea presented in this paper. [8, 9] infer various layer block components like buses and protocols; generate C code and abstract communication to the service level. However, [8, 9] generate general communication functionality, but not register maps of peripherals and the referring software.

Compared with previous work, this work allows modeling in an interactive way comparable with Matlab/Simulink, while being able to immediately start HW and SW development.

II. DESIGN ON SEMANTIC LEVEL: METHODOLOGY

To explain the methodology, we use a simple example: An electronic throttle control system. We will discuss models of this system on “blackboard-level”, and subsequent refinement steps that successively introduce semantic knowledge allowing execution and starting SW development based on this model.
A. Levels of Abstraction

Fig. 1 shows the example as a “blackboard model”. Engineers understand that this is a simple control loop; however, architecture level information is missing – but implicitly known to designers (at least the expert ones).

![Fig. 1: Blackboard-level model of an E-throttle.](image)

For example, the following information is missing, but known to a designer):

- The PI controller produces an 8-bit unsigned output that is programmed as duty cycle to the PWM generation peripheral unit.
- The PWM generation unit is initialized to be in edge aligned mode.
- The input of the PI controller is the throttle position that is sampled on request of the software, e.g. every computation of the PI controller.

Obviously, with the information only in mind of a designer, a model cannot be simulated.

To close the gap between a blackboard-level model and a model that can act as a virtual prototype, we use references to ontologies as shown in Figure 2.

![Fig. 2: Representation of knowledge at semantic level.](image)

To come from a blackboard-level model to an executable model, the block-diagram level structure is instantiated. Then, the functionality of the blocks is specified by references to an ontology using an “is-a” relation. This so far is exactly what would be done using existing modeling libraries: a block is replaced with an (or, as we do here, calling) a process from a library. However, the ontology represents not only a “reusable model”, but as well possible variants, parameters, modes, specialization, etc. of a component – the “expert knowledge” of a designer that is captured. In extension to the specializations the ontology also contains SystemC processes whose execution models or refines the model to behave in a particular way.

For example, a PWM has different modes, among them single-shot mode, and edge aligned mode. The mode has to be set prior to execution of a model. This happens usually as part of firmware initializing the peripheral units. This is represented by specialization (refinement) of the PWM concept from a general one to e.g. an edge aligned one. This can be done by giving the ontology a command “PWM mode edge aligned”. The command can be given

1. directly from within the module itself during instantiation, or
2. externally from the “software” module.

Regarding the example of an E-throttle, we would get a semantic-level model as shown in Figure 3. It contains sufficient information to be simulated; we call it model at semantic level.

![Fig. 3: Semantic-level model of an E-throttle.](image)

On semantic level, we instantiate a block diagram like on blackboard-level. As signals we use

- Timed data flow for modeling simple method-level block diagrams like in Matlab/Simulink.
- TLM signals that – as generic payload – transport parameters, modes to the “digital” side of peripheral units as XML-formatted references to the ontology.

On semantic level, development of hardware-related software can be started, using the model as a virtual prototype. Then, software can program peripherals using XML commands instead of a register map.
Register maps can be introduced later in the design flow; then, the commands at semantic level are replaced with binary data written in specific addresses. While the semantic-level models are independent from a concrete architecture, register maps are very specific properties of platforms; changing the address of a peripheral’s register or misinterpreting it’s semantics is a major issue and cause of many errors in the development of embedded software.

For example, the register map of an ARM Cortex M3 is shown in Fig. 4; at the level of TLM/Programmer’s view accuracy, the registers of the register map have to be programmed in all peripheral components by software.

A model at semantic level can be used as a starting point towards synthesis of embedded software.

**B. Component and Structure of the SICYPHOS Library**

SICYPHOS is a SystemC-based framework for Simulation of Cyber-Physical Systems. It provides a modeling library for creating virtual prototypes for early software development and architecture exploration. The software part of embedded systems is modeled by either compiled C code, or by machine code running on instruction set simulators; we currently use ArchC for this purpose. The modeling library is focused on modeling peripheral components and the physical environment. The library of SICYPHOS provides the following models:

- PWM, ADC, DAC, PLL
- Various transceivers, receivers and its components.
- Mathematical functions and analysis tools.

Each module of the library provides:

- Data flow (analog, physical, RF) interfaces towards the parts that are modeled in a data-flow like way,
- A TLM interface via which it is typically configured or initialized from the system’s software,
- A non-ideal behavioral model that can be parameterized prior to the design. The non-ideal behavior includes distortions and other inaccuracies, but also power consumption.
- Some modules are hierarchical models; e.g. a PWM might be a PWM with a control loop inside that has a controller (e.g. PI, PID) and an ADC (of different types).

The library’s ontology extends a SystemC model that is capable of parameterization with information about parameter names with means to set the parameters via the TLM interface (for programmable parameters), or before simulation (for parameters that just model non-ideal behavior).

However, the ability to structure a library, and to allow setting parameters via TLM alone do not justify an ontology. Ontologies offer the power of concept management and allow these concepts to be *mixed* and *combined* in a dynamic way. The concepts in the SCYPHOS library’s ontology are intended to be a part of a model, and to be extended within the design process by newly acquired knowledge of designers. Since the concept structure is modular, it allows new hardware modules to be created by mixing multiple old and/or new concepts, representing top-level modules of a system, and composition of new simulation modules (and new concepts) by a structural composition (has-parts) of existing concepts.

Figure 5 gives an example for the relation between parameterization, and the ontology. The dashed lines indicate the is-a relation between (SystemC) modules and concepts of the ontology that assist in interpreting incoming messages (e.g. for setting parameters).
III. STATUS OF IMPLEMENTATION AND EXAMPLE

We have developed the concepts described in section II for a number of models of peripherals. The PWM is a quite challenging component. Many microcontrollers contain PWM and a lot of functionalities and parameters can be programmed. Figure 6 shows an excerpt of the PWM ontology. It also shows the benefit of ontologies, such as use of ADC inside a PWM.

For creating the ontology, Protégé 5.0.0 was used. The XML representations of “TLM messages” are identical to the format of the parameters in this tool. For proof-of-concept we modeled the E-throttle system as shown and described in section II. Simulation was possible and showed a control behavior like in Fig. 7. The Figure shows an altered simulation result by variations of PWM plateau values (after a third of simulation time) and period time (after two third of simulation time). A reference value of 7.5V indicates the ideal behavior ($i_{ref}$), $i_{meas}$ is the measured value in the control feedback and $i_{err}$ the control loop error.

IV. DISCUSSION AND FUTURE WORK

The use of ontologies helps closing the gap between an informally drawn blackboard model and executable specification. The advantage is that no knowledge of hardware details is required when developing hardware-related software. To set parameters or get/set information from peripheral components, just it’s names can be used.

Beyond putting design at a higher level, this also allows SW development to start earlier, and with less likelihood of complex re-development cycles in case of later changes in the register-map addresses.

V. REFERENCES


