UML IN DESIGN OF ASIP

Karel Masařík*, Tomáš Hruška*

** Faculty of Information Technology, BUT, Czech Republic*

Abstract: There are many possibilities how to describe the architecture of the embedded system eventually of the whole microprocessor. The description is realized usually using any special language, which only rarely provides a formal description of the architecture. The formal description is useful for an easily verification of the most critical parts of the architecture. Our task deals with the formal description of the Application Specific Instruction set Processors (ASIP) in UML. Due to the lower complexity of this type of processor, it is possible to generate automatically the software toolset from the model, which provide fluent development of the microprocessor's applications. The task is to provide the methodology for fully automatized design of microprocessor base on the UML description. *Copyright © 2006 IFAC*

Keywords: ASIP, architecture description language, UML.

1. INTRODUCTION

Embedded systems are getting more important in all of the branches of human activities, for example in the medicine, in the form of the small control systems, or in the form of massive distributed computer systems, which consist of hundred thousand embedded computers. Each of these small embedded computers can be produced for lower price. Recently, the production of microprocessors has grown rapidly.

Application Specific Instructionset Processors are used mostly often in the embedded systems. Particular functions of the hardware of these processors are available through the application specific instructions. This means, that each processor or its family has own instruction set. Specific software tools like assembler, disassembler, eventually the whole development system, are bounded with the specific instruction set. This automatic generation of software toolset from the microprocessor description is required. This can be achieved by using of Architecture Description Languages (ADL). They provide automatic generation of software tool on the basis of the microprocessor model. Usually the design of microprocessor is supported by specific development framework, which supports the development using specific ADL.

Besides ADL it is possible to use for description other higher-level programming languages like C++ or UML (OMG 2004). Mainly the UML becomes importance in community of embedded systems because it helps to overcome the gap between the specification and implementation. Our research concentrates on use of the UML as an architecture description language and the motivation is to provide the methodology for fully automatized design of microprocessor which should cover the formal verification of the microprocessor model too. However, it remains an open question how best to use object-oriented notations for architectural description and whether they are sufficiently expressive as currently defined.

The development supported with the UML has following advantages: the architecture is easy to understand, is consistent during development, and can be supported by existing tools. Research to represent microprocessor architecture in UML can be generally approached in two ways. The first way maps architecture concept to existing UML notations just as they are. The second way extends the vocabulary and semantics of UML to match its modeling capabilities to the concept of architectures. The second way can be divided again into the heavyweight way and lightweight way. The heavyweight way adds new modeling elements or replaces existing semantics by directly modifying the UML metamodel. The lightweight way defines new modeling elements by means of the extension mechanism of UML and does not modify the UML metamodel. We are using the last mentioned approach.

This article focuses firstly on the base concept of UML then deals with the design of ADL based on the UML notation. The research is going out from the architecture description language ISAC (Hruška 2004) and the name of the ADL based on the UML notation is UMLISAC.

2. UML AS ADL

UML is general modeling language and does not provide all concepts that are important for architecture description. UML must be extended in order to precisely model architecture and to automatically generate the software toolset from the model. In our work we try to overcome the semantic gap between object oriented notation and the architecture description.

UML2.0 has much more useful concepts for architecture description than UML1.x. The UML2.0 defines new constructs such as component, connectors, and port that make the UML more suitable for architecture description. The UMLISAC uses the base concepts of UML. These concepts consist of the composition and connection, abstraction, interaction, encapsulation and scalability. When these concepts are applied to the domain of architecture description the structure of the microprocessor is defined implicitly in the class diagram, the behavior of the architecture in the state diagram and additional nonfunctional requirement are defined in the deployment diagram in case of UMLISAC language, that will be explained in later. The lightweight extension will be used for the generation of the software toolset.

3. UMLISAC

3.1 ISAC EXTENSIONS

This section introduces new extensions with regards to our latest research (Masařík 2005) which includes a design of the architecture description language ISAC. ISAC was improved by the version of language LISA (Hoffmann and Meyr 2002). This language was used mainly for the simulation of the microprocessor's instructions. In the current research we concentrate on modeling of the microprocessor structure and its behavior. Without sufficient structural information it is not possible to generate an optimized silicon chip. The language ISAC doesn't contain sufficient structural information, for example which function unit (FU) executes which instruction, time constrains of FU, which are necessary by scheduling of instructions etc. Without information like this the compiler of Very Long Instruction Word (vliw) (Srikant and Shankar 2003) microprocessor can't be created. Besides this, the ISAC doesn't allow for example sharing of the attributes among more instructions. New structural extensions are going out from the work of Cichon (Cichon 2004). His language was successively used for design and production of microprocessors and was successively tested by electronic companies.

As other extension we suggest grouping of functional units. Each functional unit belongs into one group. The group prescribes common assembler and coding syntax of instructions, which belong to the functional units classified with the same group. This makes the model simpler and more flexible, because there is always a set of instructions with the same syntax and coding. Base on the idea of instructions' grouping it is possible to perform some optimizations of the instruction decoder, which lead to the reduction of power consumption (Glökler and Meyr 2004).

3.2 UMLISAC STRUCTURAL MODEL

TEMPLATES OF BUILDING BLOCKS

This section presents structural model of UMLISAC. The architecture consists of building blocks. One building block is for example functional unit, memory etc. The class of the class diagram should represent one template of this building block. Created instances of templates are used later in larger units, e.g. core of microprocessor. The UML concept of port and interfaces can be used successfully in interconnecting among more building blocks. The ports are mapped onto the input and output ports of the physical entity of the architecture. The interface and connectors are mapped onto the some type of communication channel, for example bus.

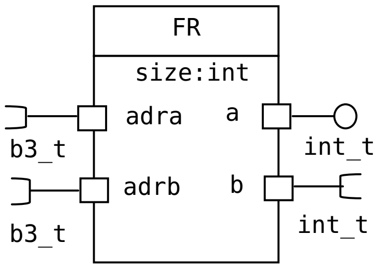

Figure 1. Class of File register

The figure 1 represents the class of a file register. The file register contains several registers (their number is determined with the attribute of *size*) that are selected using two address ports: *adra* and *adrb*. The first mentioned port together with its required interface (represented as half circle) requires an address which is used for selecting of register those data are provided through the provided interface (represented as circle) of port *a*. The address *adrb* selects the register, into which the new data is stored. The data is received through the port *b* that has an interface which requires data. Each type of interface provides methods, which manipulate with data of a specific data type, e.g. *b3_t* interface manipulates with instance data type *b3_t*. Each data type has a specific