TRANSACTION LEVEL MODELING WITH UML AND SYSTEMC

Jeovânio Bitencourte, Jorge de Carvalho, Ricardo Jacobi

Departamento de Ciência da Computação
Universidade de Brasília

jeovanio@bb.com.br, jorgeol@solar.com.br, ricardo@exatas.unb.br

SUMMARY

This paper addresses the application of UML (Unified Modeling Language) in modeling digital systems based on the object oriented approach. The resulting UML diagrams are then used as the basis to generate a SystemC structure in transaction level, with automatic mapping of ports, channels and signals. Communication among modules are implemented through tokens, through blocking channels. Those techniques are discussed and exemplified through FFT polynomial multiplier. The proposed methodology is supported by a tool integrated in Rational Rose platform.
TRANSACTION LEVEL MODELING WITH UML AND SYSTEMC

Jeovânio Bitencourt, Jorge de Carvalho, Ricardo Jacobi

Departamento de Ciência da Computação
Universidade de Brasília

jeovanio@bb.com.br, jorgeol@solar.com.br, ricardo@exatas.unb.br

Abstract - This paper addresses the application of UML (Unified Modeling Language) in modeling digital systems based on the object oriented approach. The resulting UML diagrams are then used as the basis to generate a SystemC structure in transaction level, with automatic mapping of ports, channels and signals. Communication among modules are implemented through tokens, through blocking channels. Those techniques are discussed and exemplified through FFT polynomial multiplier. The proposed methodology is supported by a tool integrated in Rational Rose platform.

I. INTRODUCTION

The increasing complexity of integrated systems are pushing the designers to look for new techniques to help system level modeling and simulation. Considering that around 70% of the design cycle [BER03] is spent in the test and verification of the system, modeling and simulation should be addressed at the very beginning of the design to reduce design cycle time.

Therefore, many researchers are adopting techniques to raise the abstraction level of modeling tasks, investigating and verifying the best alternatives to solve their problems in complex designs while minimizing the design time. Some techniques are based on Object-Oriented modeling approach, using the UML notation in the specification phase and also to guide the development of hardware modules on the following stages. On the other hand, another alternative commonly adopted is to specify the system directly in a hardware description language, like SystemC, to describe the system structure and behavior on a high abstraction level, and then proceed through successive refinements until a correct and synthesizable model is obtained.

However, the success of UML in the software community shows that increasing the abstraction level at early stages of the design process is highly recommendable in order to allow focusing on the more important aspects of the system, without dwelling into minor details of the implementation. Furthermore, UML is an excellent tool to facilitate the communication among designers of a big team.

In this sense, some works had been proposed, advocating the usage of UML for modeling and SystemC to implement hardware modules to address hardware issues very early in the development process. However, just a few works tackle problems derived in the translation from UML diagrams to SystemC specifications and, also, how to make a good use of UML notation in order to focus on the essential features of the system, without spending too much time in over specifications or forgetting to describe some important features of the design.

This work proposes an approach based on the adoption of standard UML diagrams to describe the system and on the generation of a SystemC template which maps classes and methods to modules, channels, ports and tokens. The SHAL (System design at High Abstraction Level) methodology models the intercommunication among components as tokens exchanged through blocking channels. A CAD tool was developed and integrated in Rational Rose to support the automatic generation of SystemC Transaction Level descriptions, which is called SHAL-I (SHAL Integrator).

The structure of this paper is as follows. Section 2 gives an overview on related works. Section 3 introduces some basic concepts concerning high abstraction level modeling, UML and SystemC. Section 4 discusses the SHAL methodology. Section 5 describes a case study. Section 6 presents the UML modeling of the case study using the basic concepts. Section 7 describes the process used to translate the UML diagrams to SystemC structure. In the section 8 the SHAL Integrator tool is presented. Finally the conclusions are presented in section 9.

2. RELATED WORKS

The use of UML for hardware description is related to the application of object-oriented techniques for hardware modeling. In this regard, it also could be mentioned some related works involving UML and object-oriented.

In [KUM94], the authors had mapped object-oriented techniques concepts to the hardware modeling tasks. That paper has focused in improving the hardware modeling process, using these concepts. In [GRI03], the authors present a method and a tool for object-oriented hardware design and synthesis based on SystemC. The focus on this work is converting object-oriented RT/behavioral level SystemC, generating SystemC or VHDL specifications that
could be further processable with existing off-the-shelf tools. [KEY04] presents a design flow to Object-Oriented Modeling and Synthesis of SystemC specifications using a JPEG encoder to illustrate the design flow and UML to write specifications in formal notation. The source code including the skeletons of SystemC modules is created.

In [TAN04] results concerning a translation from Real Time UML, a specific version of UML, to SystemC are presented. In order to generate the source code several tools are used, including a C++ code generator and C++ to XML parser. In [YU04] is proposed a design flow starting from a system specification written in transaction level using SLDL language. The specification is gradually refined until the C source code is generated.

The works in [KEY04, TAN04, YU04] are close to the topic of this paper, however, there are substantial differences as follows. At [YU04] the process starts from a specification written in SLDL, while our approach starts from high level UML models. In comparison to [KEY04], our process not include SystemC structure in the UML models, this structure is included only in the source code after the translation process. In [TAN04] is used a specific notation to modeling real time systems, while our approach uses basic notation of general diagrams.

3. BACKGROUND

3.1 Abstraction Levels in Modeling

Panagopolos [PAN02] and Lee [LEE01] define a “model” as a clear and formal description of a computational system or subsystem behavior. A model can represent a system with different amount of details depending on the level of abstraction adopted.

The use of RTL-based descriptions in VHDL or Verilog is still a necessary step in integrated systems design. However, depending on the complexity of the system, to start its description at this level is too cumbersome and error prone. Since SoC (System on Chip) designs today may integrate very complex systems on a single chip, including processors, microcontrollers and several custom modules possibly interconnected by a network on chip [BRA04], there is a need to model the system early in the design process to allow for functional and system verification.

At a higher level of abstraction, a designer can focus on key design issues and not with unnecessary implementation details at first [SCH04], simplifying the modeling process and improving the exploration of alternative solutions. Moreover, functional verification at system level is accelerated since simulation is much faster with abstract models.

An increasing number of works are adopting TLM (Transaction Level Modeling), for abstract hardware modeling. In a TLM, the details of communication among computation components are separated from the details of the implementation of computation components. Communication is modeled as channels and transactions requests take place by calling interface functions of these channel models. Unnecessary details of communication and computation are hidden in the TLM and may be worked out later [CAI03].

TLM design represents the system in a more abstract and concise way and simulates much faster than correspondent RTL designs [CAL03].

Considering the models of computation that could be employed at this level, the transaction modeling, based on components that exchange information through method calls, is one that closely matches the object-oriented notation. Although UML could be used to model even gate level descriptions, its notation, with a dozen diagrams, is better suited to system level descriptions.

The TLM models can then be refined into a sintesizable RTL description, by introducing structural and timing informations.

3.2. UML Modeling

The Unified Modeling Language (UML), the standard language for specification and documentation of systems adopted by OMG (Object Management Group) [OMG04], provides diagrams to describe the structure and behavior of a system, as well as diagrams to manage projects.

The UML diagrams present characteristics that can be used for hardware modeling. For instance, version 2.0 introduces the time diagram [UML03], which allows for waveform representations.

It should be noted that UML is a notation and not a methodology. There is no mandatory diagram and its use depends on the user needs. However, there are several advantages in adopting UML. It is a shared language that provides for team communication, where the diagrams that describe the system are written using a standard notation. The benefits of using UML appear when it is incorporated in a design process. In this context, UML diagrams can be used as a starting point to build systems, not only represent them.

There are many well-known methodologies that define formal procedures for modeling process [OMG04], for instance USDP (Unified Software Development Process) [JAC99] and RUP (Rational Unified Process) [RAT04]. Those methodologies are defined to increase the probabilities of a project succeed and improve the productivity of a design team.

In recent years, new UML modeling approaches had been used to perform functional verification and to check the system requirements in early stages of development. The xUML (executable UML) [MEL02] methodology uses, mainly, state and class diagrams in order to provide a way to “run” the model and check its functionality, like a UML
debugger. The MDA (Model Driven Architecture) methodology adopts xUML for design checking and provides support for the designer to verify the model requirements compliance in a platform independent model (PIM), before starting the coding process.

3.3 SystemC
SystemC is a powerful HDL (Hardware Description Language) used for modeling, simulation and synthesis of digital hardware devices. It is based on C++ and, due to the high abstraction it provides, it is useful to model complex designs at system level, such as Systems on Chip. Its first implementation is basically a C++ library, composed by several classes, macros and data types suitable to model hardware issues such as time and parallelism [GRO02]. SystemC brings the power of object-oriented design to the hardware arena.

One of the main advantages of SystemC is its capability of modeling complex designs that combine hardware and software functionality. In this case, the simulation of the processor executing code should be very fast, in the order of hundreds of thousands of cycles per second.

4. THE SHAL METHODOLOGY

Usually, the first concern about modeling hardware in UML is related to intrinsic hardware features like clock events, buses, signals, ports, parallelism and synchronization.

Our initial experiments have shown that introducing all those characteristics in UML diagrams is cumbersome and makes the design difficult to read by the development team. This approach raises the complexity of description, obligating the designers to know deeply UML and HDLs specific languages structures and primitives. In addition, over-specifications generate negative impact on simulating model flexibility.

Our approach focus on behavioral modeling. The UML diagrams are built using software engineering development techniques. Only UML basic notation is used to write the diagrams, leaving the hardware implementation details, such ports and channels, to be automatically generated. This approach allows the designer to focus on the system functionality, reducing the design and the verification task’s complexity. In this first version, only Use Case, Class and Sequence UML diagrams were used. However, the Component diagram and the State diagram have been analyzed to develop the interconnection modules and to describe the modules internal behavior.

The system description begins with the Use Case diagram, where the system functionalities are discovered and presented as use cases. The users are identified and represented as actors. Each actor is connected with the use cases that represent the system functionalities, as shown in figure 1. In the context of a SoC design, actors represent external modules that interact with the system. If we are modeling a processor, for instance, the memory system and the I/O systems would be represented as actors.

For each identified use case, the details around interactions among actors and the system are described, using a sequence diagram. In the sequence diagram classes representing modules are created and messages among them are established. The class that interacts with the actor is called a boundary class. The messages among actors and boundary classes define the system limits and later will be translated to external ports. Figure 2 depicts a sequence diagram for the FFT example.

All classes identified on sequence diagrams are put on class diagram where their relationships must be created in the following way: if a class is contained in other, the relationship is an aggregation, otherwise the relationship is a simple association. The kind of relationship is very important because their determine the responsibilities of the channels. A class diagram can be seen in figure 3.

After describing the system, the information contained in the diagrams is extracted to provide elements in to support the translation of the UML notation to a SystemC description in transaction level, using the algorithm summarized in section 4.

During all process, the correlation between structures of UML and SystemC is maintained. It is possible to locate UML structures in SystemC code and also to reverse engeneer the UML description from it.

Synchronization in TLM is obtained by using blocking channels to connect modules. When one component calls a method (or send a message to another component), the caller remains blocked until the answer arrives. The message itself can be carried out by tokens. Tokens are interesting for the reason that all arguments (signals set, for instance) are transferred in the same time, avoiding deadlocks in sequential blocking function calls and avoiding the use of global variables. Due to blocking messages, circular reference must be avoided.

Additionally, tokens can contain all transaction arguments or return values along with their data types, which can be used to refine the channel into separated ports and signals in the RTL description.

Notice that, in this context, each service provided by a module will be represented in UML by a method and its signature (prototype) will denote the necessary data (arguments) and return value. On the SystemC side, it will be necessary define a message channel (MessageChannel class) which connects the source module port to the target module port. Tokens passing through that port-channel-port (request and return directions) will contain data structures representing the arguments or return values.

It is important to keep in mind that the final product of the methodology is not a simple SystemC skeleton: the channels that provide for module communication are automatically generated and sender and receiver methods are put in the right place. Thus, all information contained in
classes and sequences diagrams are translated to SystemC, leaving the module behavior to be described by the designer.

5. DESCRIBING THE CASE STUDY

In order to analyze the mapping process from UML diagrams to SystemC structures a case study was developed, which consists in an implementation of a fast polynomial multiplier using the FFT [KNU80]. The basic idea behind this multiplication technique is to reduce the number of multiply operations, making a transformation of classic coefficient-represented form to point-values representation. Multiplying coefficients of two polynomials takes, in the worst case, \( O(n^2) \) operations, where \( n \) is the number of coefficients of each polynomial, since each coefficient of the first polynomial needs to be multiplied by each coefficient of the second polynomial. But, in point-values representation, two polynomial can be multiplied in time \( O(n) \) [COR01].

However, the transformation of the coefficient form to a point-values representation, also called evaluation, using typical techniques, like the Horner rule, takes \( O(n^2) \) [COR01]. The inverse transformation, which gives the coefficient values of a polynomial from its point-value representation, also called interpolation, takes another \( O(n^2) \) time, using the Lagrange Formula.

Using the properties of the FFT (Fast Fourier Transform), [KNU80] demonstrated that evaluation, pointwise multiplication and interpolation can be done in \( O(n \log n) \) time. Notice that, despite the mathematical correctness of the method, the interpolation can insert numeric instability because of the limited number representation capacity of digital machines. But, this limitation can be ignored in this implementation, which intends just to demonstrate the methodology.

The entire FFT-fast polynomial multiplication method can be described with the following steps:

- Take two polynomials, \( P \) and \( Q \);
- Call FFT routine to evaluate \( P \), giving \( P' \);
- Call FFT routine to evaluate \( Q \), giving \( Q' \);
- Call PointWise multiplication routine with \( P' \) and \( Q' \), giving \( R' \);
- Call Interpolation routine to make inverse transformation over \( R' \), giving \( R \), which is the final multiplication result of \( P \) and \( Q \).

The polynomial \( P, Q \) and \( R \) are in coefficient representation form and \( P', Q' \) and \( R' \) are complex values, representing the point-values of corresponding polynomials, since FFT and its inverse (used in interpolation routine) work with complex roots of the unity [COR01].

It is interesting to adopt the “Use Case” Diagram to understand the entire context, in which the multiplier is inserted, allowing a clear identification of the required functionalities. It also allows identifying who are the users of each module. Figure 1 presents the “Use Case” diagram illustrating the five steps of the multiplication process (the use case “Evaluate Points” occurs twice). The actor in this diagram represents the I/O operations which supply the data and collect the result.

6. MULTIPLIER MODELING

Following object-oriented modeling techniques, we have identified in the multiplier its modules with specific responsibilities. Therefore, those entities can be modeled as classes. The multiplier itself is a top-level class, which was named “FastMultiplier”. It aggregates a set of classes (SystemC modules) that collaborate to perform the fast multiplication.

Now, it is necessary to investigate the other modules that define the entire structure, figuring their intrinsic functionalities out. This task is done with the following steps:

- build the sequence diagram;
- analyse existing operations;
- indentify which modules are involved;
- ordering the calls to the functions (methods invoking) and
- find which classes call those methods.

At this moment we do not care about how the methods work. We are just interested on investigating what kind of data each method will need and what kind of result it has to return back. Doing that, we will obtain the classes with their methods, and each method with its signature, according to figure 2.

Once the sequence diagram is ready, the next step is to build the class diagram. Usually this task is done using UML modeling tools like Rational Rose, by dragging classes created in sequence diagram to a specific scenario and making the relationships among class. Figure 3 illustrates the class diagram of the case study.
a) For each class in the UML class diagram:
   a.1) Creates the SystemC module body (SC_MODULE).
   a.2) For each public method:
      a.2.1) Create tokens classes, which represent the signature of all public methods, inherited from MessageToken base class.
      a.2.2) Instance an adequate Message Port (duplex or simplex) object, parameterized (template) by message tokens, created above, like a public attribute;
      a.2.3) Declare a public method using the name defined in UML class diagram, which will not receive any argument; This method must be declared as SC_THREAD in the constructor (SystemC-styled);
   a.3) Identify and make instances of all aggregated modules method:

b) Search for each different interaction on UML sequence diagram in order to:
   b.1) Create source Message Port (simplex or duplex) in the module that is starting the interaction;
   b.2) Create the Message Channel (simplex or duplex) to interconnect the source to target modules in the module, which is a common owner of both;

8. SHAL TOOL

For small models, the translation techniques described above can be done easily by hand. But, when there are several classes and interactions like the most complex projects that designers are supposed to cope with, the translation is not a simple activity. So, this methodology is supported by a tool aimed to automate the UML to SystemC translation.

This tool was called SHAL-I (SHAL - Integrator) and has the responsibility to build SystemC modules from UML models, automatically, giving a source code ready to compile with SystemC libraries.

However, the specific behavior of modules, represented in UML by public methods, must be specified by hand in a operation called “back-annotation”, after the translation process.

The model generated can be used in simulation and verification steps in the System On Chip modeling process. Later refinements can be done directly on SHAL model, without loosing important information acquired in earlier modeling stages.
The SHAL-Integrator obtains the UML diagrams directly from XMI files, generated by UML CASE tools. XMI (XML Metadata Interchange) is a popular pattern to interchange information from UML diagrams between CASE tools [OMG04]. The overall SHAL-Integrator working schema is presented below in the figure 5.

![SHAL-Integrator Screen](image)

Figure 4 – SHAL-Integrator screen.

In order to employ the SHAL methodology on the case study, the UML Class, Use Case and Sequence diagram was designed with Rational Rose. After that, the UML diagrams were exported to XMI file, using a plug-in installed on Rational Rose. So, the SHAL-I tool loaded the XMI file and generated the source-code ready to compile with SystemC libraries. The case study model was translated automatically in a cpp file with 523 lines, before back-annotation phase.

The complete model was implemented in a cpp file with 645 lines, inserting a typical FFT iterative in-place algorithm called from FFT8 SystemC module, like described in [COR01], in the source code generated by SHAL-I. The simulation was successful, providing correct results from reference test values. The case study was considered suitable to demonstrate the capacity of SHAL methodology to modeling complex System on Chip projects.

9. CONCLUSION AND FUTURE WORKS

In this paper we presented the SHAL methodology and your corresponding tool: the SHAL - Integrator. The methodology addresses the complexity of large computational system projects, adopting high-level abstraction approach and a set of “best practices” that lead designers to build models, simulate and verify the design very early in the development process. The designer is encouraged to adopt the methodology proposed, which is an extension of USDP (Unified Software Development Process) approach, as a suitable technique to investigate the requirements and the best alternative solutions. Additionally, UML is proposed as a modeling language to specify the structure, modularization and behavior (including time constraints) of computation system being modeled.

Moreover, the SHAL methodology employs a spiral development approach following the model-driven architecture, where the modeling abstraction get more and more detailed, without discarding any preceding information from UML diagrams. To do this, SHAL adopts the PIM (Platform-Independent Model) technique, avoiding designer to care about specific implementation details and allowing increasing the search space before choosing the final solution.

The SHAL-Integrator tool is responsible for the translating UML diagrams to SystemC structures, inserting special token classes and converting member-functions calling style in the channel-port-tokens schemas. The tool automatically creates the set of necessary tokens, parallel operations as SystemC threads and instances of all channels and ports, transforming the PIM model denoted with UML in a Platform Specific Model (PSM), implemented with SystemC. This transformation and insertion of implementation elements makes the SHAL – Integrator quite different from others common UML-skeleton software generators, since none of them are capable to implement the MOC (Model Of Computation) employed by the SHAL to simulate the hardware features.

In order to demonstrate the application of the SHAL methodology, a case study was developed that presented the characteristics of a complex computational system, which can be implemented like an IP module in a SOC component. So, we applied the SHAL methodology in modeling a fast polynomial multiplier component, which uses FFT iterative in-place routine to perform in \(O(n \log n)\) real multiplications, instead of classical \(O(n^2)\). We explored the study case, showing all necessary UML diagrams and the process we employed to conclude and simulate our multiplier IP module. The SHAL-Integrator was considered
a valuable tool, translating quickly and without errors the UML model to a SystemC compilable source code, ready to the back-annotation process. The last task was done through a simple copy-and-paste from a software implementation of the classical FFT iterative in-place algorithm, described in the references. The simulation performed successfully.

For further work, we intend to implement an interactive mechanism for the SHAL – Integrator to simplify the spiral development, allowing us to make back-annotation and to return back to the UML model. In addition, we are investigating a manner to implement the simulation directly on UML diagrams, using the SHAL – Integrator, allowing one to “debug” the model in the UML design.

10. REFERENCES