Generation of VHDL code from UML/MARTE sequence diagrams for verification and synthesis

Emad Ebeid, Davide Quaglia, Franco Fummi
Dept. of Computer Science — University of Verona, Italy
[emadsamuelmalkiebeid|franco.fummi|davide.quaglia]@univr.it

Abstract—Verification of real time embedded systems is becoming more and more complex in terms of maintaining the code size and keeping equivalence between the specification. It requires the simulation of the system and the checking of its timing and functional properties as well as constraints. The paper presents a methodology which starts from UML sequence diagrams with MARTE timing constraints and generates a VHDL model with checkers. The simulation of the model allows to verify the specified sequence of exchanged information between components while checkers allow to verify that properties and timing constraints are met. The application of the methodology to the design of a wireless sensor node shows the validity of the approach and its simulation overhead.

Keywords—UML, Sequence diagram, MARTE, VSL, VHDL, FPGA, timing constraint.

I. INTRODUCTION

In the last years, Model-Driven Design and UML [7] have gained an increasing interest for electronic system design as a platform independent modeling approach. Several UML profiles have been proposed, e.g., the UML for SoC [10]. The most comprehensive effort towards a UML profile supporting all the required concepts for real-time, embedded system design is MARTE (UML Profile for Modeling and Analysis of Real-time and Embedded Systems) [8]. The generation of executable code from UML diagrams is a growing area of interest due to its benefits essentially for design verification. Furthermore, there are some works and commercial tools [6] to generate source code from UML specifications to mainstream languages, such as C++ and Java.

Actually, the problem was firstly investigated in [4], which presented the mapping between some UML diagrams and VHDL [1], as a common and structured environment for system documentation and specification. The approaches in [5] cope with direct code generation by taking the UML model as Metadata Interchange file for the translation to VHDL. Most of the literature focuses on UML structural diagrams (e.g., composition diagrams) while behavioral diagrams (e.g., state charts and sequence diagrams) are not used to generate a VHDL model of the internal functionality of the system. Sequence diagrams are used to describe time-ordered sequences of data exchanges between system components. The work in [5] is a first contribution in the generation of a simulable model of the system from UML Class diagrams, however the generation of code describing system behavior is not enough for its verification since assertions should be also generated, i.e., code which checks system properties and eventually rises warning messages.

Open Verification Library (OVL) [2] consists of a set of assertion checkers that verify specific properties of a design and its checkers are instantiated in Verilog and VHDL modules/entities. Actually, OVL library does not cover all kinds of assertion and it lacks some VHDL assertions especially for time and duration constraints (i.e., ovl\_time checks the value of an expression for a specified number of cycles) which are useful in the transformation between UML sequence diagram and VHDL code.

UML sequence diagrams allow to specify “invariants”, i.e., properties about system variables; furthermore, MARTE profile introduces a way to express time constraints between points of the sequence diagram. What is missing in this context is a methodology:

- to generate VHDL code from sequence diagrams;
- to express timing observations and to check constraints inside VHDL;
- to generate VHDL assertions from invariants and time constraints.
The proposed work aims at filling these gaps by:

- showing the rules for the automatic generation of VHDL codes from UML sequence diagram interactions;
- creating VHDL components to manage time observations and their evaluation;
- finding a strategy to generate VHDL assertions from UML invariants and MARTE time constraints as well as to put them in the right position inside generated VHDL code.

The whole verification flow derived from this approach is depicted in Figure 1. The UML/MARTE sequence diagrams describe both the behavior of the system (in terms of data exchanging between components) and the constraints it must satisfy. From this description a VHDL model is generated containing the assertion calls to check constraints (this step is the contribution of the paper). The generated code is not enough for simulation since sequence diagrams specify data exchange between entities but not the processing code inside them; therefore, these entities should be linked to an external library of Register Transfer Level Intellectual Property (RTL IP) cores. Simulation takes as input a suitable set of test benches and generates a trace in which failed constraints are reported. As a result of the verification process, new model information are obtained to refine the UML model and the constraints.

The paper is organized as follows. Section II introduces sequence diagrams, time expression language and VHDL modeling. Section III presents the proposed transformation rules from sequence diagrams to VHDL code. Section IV experimentally validates the methodology by applying it to the design of a wireless sensor node and Section V draws some conclusions.

II. BACKGROUND

This section provides a brief overview of UML sequence diagrams, MARTE expression language and VHDL modeling.

A. Sequence diagrams

Sequence diagram is a UML behavior diagram based on Message Sequence Chart and Interaction Diagram; it shows how data is exchanged between process instances and in what order [11]. As depicted in Figure 4 a sequence diagram shows objects interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between them to carry out the functionality of the scenario. A sequence diagram annotated with constraints expressed in a language like Object Constraint Language (OCL) [3] can be the basis for communication protocol verification.

B. UML/MARTE and VSL

Common Behavior package [7] is a UML package which defines a Simple Time model and provides a syntax for representing time and duration expressions as well as a mechanism to represent event observations with time marks. MARTE [8] is a UML profile intended for model-based development of real-time and embedded systems. MARTE provides a language named Value Specification Language (VSL) to specify the values of constraints, properties, and stereotype attributes particularly related to non-functional aspects such time. VSL is based on Tagged Value Language and OCL and provides specification for mathematical expressions (arithmetic, logical, etc) and time expressions (delays, periods, trigger conditions, etc). MARTE extends UML and deals with VSL to support specification expressions. In fact, VSL can be used in tagged values, body of constraints, and in any UML element associated with value specification.

Time expression package (VSL::TimeExpressions) is the one of VSL sub-packages as depicted in Figure 2. This package adds textual capabilities to represent time-related expressions and a specialized syntax for time value specifications and expressions. VSL’s Time Expression model improves UML with different capabilities; for instance, the i-th occurrence of a given event can be expressed by an occurrence index; duration observations and the jitter (i.e., the standard deviation) of a sequence of occurrences of the same event can be used inside expressions. Figure 3 shows the structure of the time expression package, including different kinds of time-related expressions, such as instants, durations, and jitter values. The idea of using MARTE expression language with UML sequence diagrams is to open the way for modeling real time expressions, observation concepts and constraint specification.

Figure 2. Structural of the VSL framework
C. VHDL

It is a programming language that has been designed and optimized for describing the behavior of digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program to generate the actual system. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. Furthermore, assertions are modeled and tested in VHDL based on boolean conditions and if the condition is false, a warning message is generated. It is worth noting that in the last years this language is being very used in the embedded systems community because it simplifies the hardware design and allows validating the behavior of the modeled systems by simulation.

III. UML/MARTE TO VHDL CODE GENERATION

This Section introduces the syntax elements of the sequence diagrams and describes how they are used to generate VHDL code.

A. Model and constraint specification

Figure 4 shows the syntax elements of a sequence diagram; numerical labels are used to show the items to be explained. Lifeline objects (label 1 and 2) interact by exchanging messages. Each message (3,4) represents either an information transfer or the corresponding reply. Transfers can be either synchronous (3) or asynchronous (4). Execution specification (5) represents the execution of a unit of behavior or action within the lifeline. Combined fragment is a logical grouping (6) which contain the control structures that affect the flow of messages. A combined fragment contains interaction operands and it is defined by the interaction operator (i.e., loop, alternative, sequence, etc). Furthermore, it is not allowed to transfer more than one message at the same time.

Figure 5 shows the modeling of durations and timing constraints; numerical labels are used to show the items to be explained. There are two kinds of constraints, local constraint for single observation point (10) and global constraint for different observation points (8). TimeObservation (7) is the reference to a given time instant during the execution. TimeExpression (8) is used to represent a time value. TimeConstraint (9) represents a constraint that refers to the range between two time expressions named TimeIntervalMin (10) and TimeIntervalMax (11). DurationObservation (12) represents the time spent to transfer the message to whom it is applied. DurationConstraint can be defined with respect to either a constant (13) or a range between two values named DurationIntervalMin (14) and DurationIntervalMax (15). StateInvariant (16) is a constraint placed on a lifeline that must be true at the given point. Timing expression package of VSL introduces the basis to write time-related expressions. For example, InstantExpression is a time expression that denotes a time instant value, DurationExpression is a time expression that refers to a duration value, JitterExpression denotes the time variation in a sequence of event occurrences that should be periodic. Section IV describes an example which shows the practical use of this package.
B. Generation rules

Based on the semantics of UML sequence diagrams [7], VSL [8] and VHDL [1], we propose transformation rules from UML sequence diagram with MARTE expression language to VHDL. As shown in Figure 6, each sequence diagram object is transformed into VHDL entity and interactions between lifelines with constraints are implemented as a Finite State Machine (FSM) controller with assertions while jitter constraint is modeled as a separate component. Moreover, FSM mechanism fits the same mechanism of the sequence diagram interactions and section IV describes this point clearly with actual case study. Class diagram has been used to describe the device functionality, [5] shows the rules of generating VHDL description from UML classes.

The transformation rules between UML sequence diagram with VSL and VHDL are described below in details and numerical labels in Figures 4 - 5 are used to show the relevant item:

For lifeline objects of sequence diagram, VHDL data path components have been created for modeling these objects.

- **Rule 1**: Each UML lifeline object is represented by VHDL COMPONENT consisting of ENTITY for object (1) and ARCHITECTURE for the lifeline (2).

- **Rule 2**: Communication flows (i.e., asynchronous and synchronous message) between lifelines are represented by FSM transitions. Therefore, the message direction from one lifeline to another is represented by two VHDL signals one named SIGNAL current_state for the source lifeline and the other named SIGNAL next_state for the destination lifeline.

- **Rule 3**: Synchronous message synchCall (3) is represented by 2 transitions one from SIGNAL current_state to SIGNAL next_state and the other from SIGNAL next_state to SIGNAL current_state while asynchronous message asynchCall (4) is represented by one transition from SIGNAL current_state to SIGNAL next_state.

- **Rule 4**: ExecutionSpecification (5) is represented by a PROCESS() function.

For constraint modeling, VHDL control path component has been created for observation and constraints checking.

- **Rule 5**: For time observation, time point observation t1 (7) is modeled by FSM SIGNAL. Each observation point presented as a signal or bus of FSM ENTITY with counter signal to count the observation time.

- **Rule 6**: For duration observation (12), in UML it is defined by two points i.e., MessageSend and MessageRecv. In VHDL it is represented by two internal signals of FSM architecture, one for the starting observation point and the other for the ending observation point. Therefore, MessageSend is modeled as signal t1_0: unsigned of FSM architecture and MessageRecv is modeled as signal t1_1: unsigned of FSM architecture and the duration observation is the difference between those signals.

- **Rule 7**: Assertions are used to model constraints, VHDL assert statement assert()severity ; is used to check the constraint validity (e.g., assert Not(delaySPI>5) severity failure;) and it triggers when the condition is false. Furthermore, assert() is used to model UML StateInvariant (16).

- **Rule 8**: UML time and duration global constraints (8) are modeled as VHDL global action statements.

- **Rule 9**: Each time unit is normalized by the system clock signal which has the highest frequency in the whole system; for instance, if the frequency of system clock signal is 1 MHz and the constraint time is (10,us) then this constraint will be normalized as 10.

- **Rule 10**: Jitter is computed by a separate component which takes data from jitter observation point inside...
controller, counts the number of iterations and it provides the controller with jitter value for this observation point.

These rules allow a complete transformation of sequence diagram elements with time constraints into VHDL code. Table I summarizes the mapping between UML/MARTE sequence diagram elements and VHDL elements.

<table>
<thead>
<tr>
<th>UML/MARTE</th>
<th>VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object (1)</td>
<td>Component structure</td>
</tr>
<tr>
<td>Lifeline (2)</td>
<td>ARCHITECTURE</td>
</tr>
<tr>
<td>ExecutionSpecification (5)</td>
<td>PROCESS()</td>
</tr>
<tr>
<td>Message(3)</td>
<td>FSM states / transitions</td>
</tr>
<tr>
<td>TimeObservation (7)</td>
<td>FSM SIGNAL t1 with counter SIGNAL (t1 - t0)</td>
</tr>
<tr>
<td>DurationObservation (12)</td>
<td>Jitter COMPONENT</td>
</tr>
<tr>
<td>Jitter (t1)</td>
<td>assert()</td>
</tr>
<tr>
<td>Constraints</td>
<td>assert() at global action statement</td>
</tr>
<tr>
<td>Global constraint (8)</td>
<td>assert(p=2)</td>
</tr>
<tr>
<td>StateInvariant (16)</td>
<td></td>
</tr>
</tbody>
</table>

### IV. EXPERIMENTAL VALIDATION

The approach has been validated on the model of a wireless sensor node (WSN). Each WSN is implemented by connecting a Radio Frequency (RF) module to the Data Source through the Serial Peripheral Interface (SPI). RF module implements the IEEE 802.15.4 stack and exposes an application programming interface through the SPI interface to let Data Source send data to other wireless sensor nodes.

Among different tools for UML modeling, Papyrus [9] has been used in this work for its support of the latest MARTE sub-profiles. As depicted in Figure 7 a sequence diagram with MARTE VSL is used to model the specification and time constraints of WSN application, respectively; numerical labels are used to denote the item to be explained. At the beginning, Data Source sends data to SPI with a non-blocking transfer whose maximum duration is constrained to 50 ms (1). SPI receives data and sends bits to RF with a blocking transfer in a time interval which is constrained to be between 1 to 3 times the duration of the data transfer (2). The RF module receives a sequence of bits from SPI with at least 33 microseconds between them (3). In addition, jitter constraint is also attached to the RF unit limiting the deviation of bit arrival time to a value less than or equal to 10 microseconds (4).

VHDL code has been generated from two UML diagrams, class diagram which describes the structure of functional requirements of each device unit and sequence diagram which describes the interactions between them. Figure 7 shows the sequence diagram of the case study and Figure 8 shows the generated structural VHDL block diagram with constraints. Generated structural VHDL consists of two parts, one for data path and the other for control path with checkers. Jitter computation is modeled as a separate component which takes inputs from fsm_controller component and provides the jitter value as an output to the fsm_controller. Numbers are used to show the mapping of each constraint from UML to VHDL as depicted in Figures 7, 10. Each state inside fsm_controller controls device unit and checks the validity of its constraint as shown in Figure 10. Counters are added to each state for counting the delay time of corresponding observation point e.g. delayDS <= delayDS +1; as shown in Figure 10.

Some portions of the generated code with comments labeled by item number are reported as follows:

```
-- Data path part.
ENTITY SPI IS --SPI object.
ARCHITECTURE rtl OF SPI IS -- SPI Lifeline.
BEGIN -- SPI Specification.
END PROCESS;

-- Control path part:
SIGNAL current_state : STATE_TYPE ; -- Message.
SIGNAL next_state : STATE_TYPE ; -- Message.
SIGNAL t1,t2:unsigned (9 downto 0); -- Time Observation (@t1).
SIGNAL t1_0,t1_1:unsigned (9 downto 0); -- Duration observation (@d1).
SIGNAL count: unsigned (9 downto 0); -- Counter for observation point.

-- Constraints part:
Begin
assert (( t2 - t1 )>50 ) severity failure; -- Global Constraint #5.
WHEN Data_Source <=
assert (delayDS=0 AND delayDS<=50000) severity failure;-- Local Constraint #1.
WHEN SPI <=
d3<=d * "0011";
assert (c>=d AND c<=d3) severity failure;-- Local Constraint #2.
WHEN RF <=
assert ((t1-1-t1_0)>33) severity failure; -- Local Constraint #3.
WHEN RF <=
assert ((Jitter1 <= 10)) severity failure; -- Local Constraint #4.
```

A. Constraint checking

Experiments have been performed on Intel core i5-2520M CPU @ 2.50GHz running the Windows 7. The testing environment consists of HDLdesigner tool, ModelSim,
To evaluate the overhead brought by constraint checking, the simulation time of the generated VHDL model was evaluated without constraints and with different constraints assertions with the corresponding time or duration observations. Results are shown in Figure 9; the execution time of the VHDL model without constraint checking takes about 8.815 ms while there is an overhead due to time observations and assertion checking which depends on the number of constraints and on the number of observation variables for each of them; in the experiment the execution time trend which is less than linear and it substantially increased when constraint #4 of jitter component is checked. The architecture was synthesized by using Quartus II tool and the design used 5262 cells of cyclone EPIC6Q240C8 FPGA chip which corresponds to 88% of its chip area.
V. CONCLUSIONS

We presented a methodology to generate VHDL models from UML sequence diagrams with the purpose of verifying them through simulation. Lifeline objects of sequence diagram are separated from their interactions and they are modeled in VHDL as data path and control path, respectively. UML invariants and MARTE time constraints are translated into VHDL checkers which sample system properties as well as time and duration values and check them according to the original UML expressions. The methodology has been validated on the design of a wireless sensor node to show the checking overhead. Future work aims at releasing a full prototype tool which implements the methodology.

REFERENCES


