Maude Specification Generation from VHDL

Fateh Boutekkouk

Department of Mathematics and Computer Science, University of Larbi Ben M'hedi, Route de Constantine, BP 358, Oum El Bouaghi, 04000, Algeria
fateh_boutekkouk@yahoo.fr

Abstract. In this paper, we present our flow that permits Maude specification generation from VHDL code. Firstly, a XML like Intermediate Format (IF) is created showing VHDL structures and statements in a hierarchical form. This format is an abstraction of the original VHDL code. Secondly, a Maude code is generated from this IF. Both Hardware system Maude specification and properties are then passed to the Maude model checker for verification purpose. Our idea is thus to combine between VHDL simulation and Maude based formal verification capabilities for hardware systems validation. The impetus behind this cooperation between simulation and formal verification is to enable hardware designers to discover errors that could not be detected by VHDL discrete event simulator.

Keywords: VHDL, Maude, Simulation, Formal specification, Formal verification.

1 Introduction

In order to increase productivity in electronic systems manufacturing, hardware engineers have developed Hardware Description Languages (HDLs) for describing the structure and function of integrated circuits at many levels of abstraction.

HDLs offer the necessary software and Hardware specific statements for sequenciality, concurrency, timing and synchronization expression. These languages are able to perform functional and timing simulations, design space exploration, performances estimation and synthesis. Among these languages, we find SystemC, Verilog, SystemVerilog and VHDL in particular. The latter is an industrial IEEE standard HDL [3]. It is designed to fill a number of needs in the hardware design process. Among these needs, ensuring the correctness of computer circuits is certainly the most difficult task. The most commonly used verification technique in HDLs is simulation. However, in many cases, simulation can miss important errors. A more powerful approach is the use of formal methods such as temporal logic model checking, which can guarantee correctness. In this context, we have developed an approach that permits Maude [4] code generation from VHDL code. Maude is based on rewriting logic [5]. In order to apply Maude model checker, we have to introduce in addition to system specification, the specification of the properties to be verified.

V.V. Das and Y. Chaba (Eds.): AIM/CCPE 2012, CCIS 296, pp. 377–380, 2013.

2 Our Flow

The entry of our proposed flow is a VHDL behavioural description. From this, a XML like Intermediate Format (IF) is generated. It captures the structure and the dynamic of VHDL design in an abstract manner. In other words, the IF abstracts the original VHDL code by taking into consideration only the statements having the impact on the global state of the system such as signals assignments and waits.

Before IF generation, designer should perform a functional and timing simulation to discover eventually errors. From the IF, a Maude code is generated following some correspondence rules. At this stage, we can simulate Maude code by executing rewriting rules. This step is very important to validate the correctness of the transformation from VHDL code to Maude code (i.e. execution of Maude code leads to same results as we simulate VHDL program). The last step of the flow is the formal verification that requires in addition to system specification, the properties specification. Figure 1 gives an example of a VHDL behavioural description. In this example, we have an entity named *exemple* with two bit ports: an input port *X* and an output port *Y*. The architecture (named *Description*) includes one process named *P* with signal *X* as a sensibility list. Here the process *P* tests the value after a mount of time. At the end, the process suspends on *X* (there is an implicit statement: *wait on X* at the end of the process). When an event occurs on *X*, the process *P* resumes its execution. The process *P* acts as an infinite loop. Figure 2 shows the corresponding IF description for the example of figure 1. Note that all implicit waits in the original VHDL code become explicit in the corresponding IF.

In order to transform the IF code to a Maude specification, a set of correspondence rules are defined. A VHDL process is declared as a Maude class with a set of attributes like the sensitivity list, the current state of the process and the name of the signal which is updated by the process. A VHDL signal is also declared as a Maude class. Each signal is characterized by its current value, its future value and the timeout. In order to preserve VHDL simulation semantic, we add two new attributes that are *isChanged* and *co*. *isChanged* is a Boolean to specify whether the value of the signal is changed or not. According to VHDL simulation semantic, the value of a signal can not be updated (modified) unless the process is blocked (i.e. it reaches a wait statement). If this is the case (i.e. the process is blocked) and if the current value of the signal differs from the future value then the signal is updated (i.e. the current value becomes the future value) and *isChanged* is set to true otherwise it is false. When updating a signal, all processes that are sensible to this signal resume their execution. We use *co* (counter) to specify the number of processes that are sensible to a specified signal. Whenever a process resumes its execution, *co* is decremented. When *co* is equal to zero (i.e. all processes sensible to this signal are waked up), *isChanged* is set to false (i.e. the end of a cycle) to start a new cycle. The progression of VHDL simulation time is specified by defining a class named *Horloge* with two attributes: a list to stock sorted signals timeouts and the current physical time. Using Maude, we define a sort named *Processstatevalues* to specify the set of a process states. For instance: *ops begin updatey1 waitX end : -> Processstatevalues* .

Let *P* be a process with a sensibility list *(X)*. In Maude, such a process is declared as: *< P: Process | listeSensibility: (X), state: begin >* .

Similarly, the specification of a signal *X* in Maude looks like:

< X: Signal | currentValue: N, futurValue: F, timeOut: T, isChanged: false, Co: m >

The horologe is declared as a Maude class named *Horloge*:
< H: Horloge | times : L, currentTime : C >
We use the functions *head* and *throw* to extract the first element and to delete this element from a list respectively. The elements of L are sorted in a descendant fashion.
Figure 3 gives an example of a rewriting rule.

```
0entity Exemple is
       port (X : in bit;
             Y : out bit );
 end Exemple;
architecture Description of Exemple is
  begin
    P:Process (X)
      begin
        if X ='1' then
           Y <= "1" after 10;
        else
           Y <= "0" after 5;
        endif;
    end process
 end Description;
```

Fig. 1. Example of VHDL behavioural description

```
< ENTITY Name : Exemple >
   < Port>
     < X: in   Type: bit >
     < Y: out  Type: bit >
   </Port>
 </ENTITY Exemple>
< ARCHITECTURE Name: Description    Entity: Exemple >
 < BODY>
   < Process Name: P  Param: X >
   <BODY>
     < IF (X='1') >
       < AFFEC Name:  Y  Value: "1" afterTimes: 10 ns >
    < ELSE >
      < AFFEC Name: Y Value: "0"  afterTimes: 5 ns >
    < /IF >
    <Wait on X>
     < /BODY>
    < /Process>
   < /BODY>
  </ARCHITECTURE>
```

Fig. 2. The generated IF code for the example of figure 1

```
crl [rl2] :
< P: Process | listeSensibilite : (X), state : updatey1, SIGNAL : Y>
< X: Signal | currentValue : N, futurValue : F, timeOut : T,
isChanged : false, Co : m >
< Y : Signal | currentValue : N1, futurValue : F1, timeOut : T1,
isChanged : false, Co : n >
< H : Horloge | times : L, currentTime : C >
=>
< P : Process | listeSensibilite : (X), state : waitX, SIGNAL : X>
< X : Signal | currentValue : N, futurValue : F, timeOut : T,
isChanged : false, Co : m>
<Y : Signal | currentValue : F1, futurValue : F1, timeOut : T1,
isChanged : true, Co : n >
<H : Horloge | times : throw C from L, currentTime : head (L) >
if T1 == head (L) and F1 =/= N1 .
```

Fig. 3. Example of a rewriting rule in Maude

3 Conclusion and Future Work

The process of formal verification passes by many steps: first an IF is generated from VHDL code then a Maude code is generated from this IF following a set of well defined rules. IF is an abstraction of the original VHDL code. In order to formally verify the correctness of the Hardware functionality, we have to specify in addition to hardware system specification a set of properties. At the end, we call Maude model checker. As a perspective, we plan to generate Maude code for VHDL structural and data flow styles. We also intent to discover some pertinent properties for verification.

References

1. Brayton, R.K., Clarke, E.M., Subrahmanyam, P.A.: Formal Methods in System Design. Special Issue on VHDL Semantics 7(1/2) (1995); Borrione, D. (Guest ed.)
2. Breuer, P.T., Fernndez, L.S., Delgado Kloos, C.: Clean Formal Semantics for VHDL. In: European Design and Test Conference, pp. 641–647. IEEE Computer Society Press (1993)
3. IEEE Standard VHDL Language Reference Manual. IEEE, IEEE Std 1076 (2000)
4. McCombs, T.: Maude 2.0 Primer, Version 1.0. International report. SRI. International (2003)
5. Meseguer, J.: Rewriting as a Unified Model of Concurrency. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 384–400. Springer, Heidelberg (1990)
6. Reetz, R., Schneider, K., Kropf, T., Verysys, H.: Formal specification in VHDL for hardware verification. In: Proceedings of Design, Automation and Test in Europe (DATE), February 23-26, pp. 257–263 (1998)
7. Wilsey, P.A., Davis, K.C.: Modeling VHDL's Execution Semantics in EXPRESS, Department of Electrical and Computer Engineering, University of Cincinnati (November 13, 1993)