Finite State Machine Localisation Based on IP Softcores Analysis

Ján Kubek, Zdeněk Kotásek
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, 612 66
Email: kubek@fit.vutbr.cz, kotasek@fit.vutbr.cz

Abstract. This paper deals with techniques of finite state machines (FSMs) localization in FSM-based softcore intellectual property (IP) cores described in VHDL. The main goal of the FSM localisation is to supplement additional information about the softcore for the core user, to ease design of the diagnostic test of the core. Three methods and experimental results are presented in the paper together with the perspective for future research.

1 Introduction

Continued advances in both the semiconductor technology and the design automation tools are enabling engineers to design more complex integrated circuits. This, combined with competitive pressures to improve both the design productivity and the time to market, are driving engineers towards new System on Chip (SoC) design methodologies and the growing use of predesigned embedded, intellectual property (IP) cores within their designs.

With an SoC methodology, IP cores are integrated with the designers own User Defined Logic (UDL) in order to create complex SoC designs. Examples of embedded cores used in SoC design include functional blocks such as memory, processors (general purpose, graphics and DSP), and complex standard logic such as industry standard bus interfaces.

It can be seen that core-based SoC methodologies take advantage of design reuse, thereby significantly increasing designer productivity of complex electronic systems. The designer can build a SoC using various cores (similar to using lower level cells in a library) and connect them using UDL.

According to the level of abstraction IP cores can be divided into (1) *softcores*, which are cores in behavioral notation, (2) *firmcores*, in the form of netlist or register transfer level (RTL) and (3) *hardcores*, which are cores in the form of final layout on the silicon mask.

With the increasing complexity of controller (IP cores with control FSM) designs, testing of these cores has become a bottleneck in the design process. To cope with the exponential state-space growth, some techniques have been proposed [2, 3] in order to reduce this state space, but only at the RT level.

To date, no technique, which uses behavioral level of the softcore for FSM analysis, is known to the author.

1.1 Motivation

Last year a coooperation between our department and a professional company developing intellectual property softcores was established. As indicated by professionals, it was seen as reasonable contribution to the design of IP cores if the design could be accompanied with recommendations concerning the diagnostics and testing of the developed core. This was the main motivation for us to start the research

in this area. Thus, our objective was to develop a software tool for the analysis of a behavioral VHDL code of a softcore (either created or supplied) which is able to: (1) identify memory structures of the core, (2) identify control structures (FSMs) of the core, if present and (3) define basic principles of VHDL-synthesized circuit test application.

2 Analysis of the IP cores

Let an FSM based IP core be considered (i.e. a controller). The core consists of two parts: the control part, which covers the combinational logic and the state control of the FSM; and the data part, which covers data paths used in the core.

After locating the FSM in the behavioral notation of the core, by finding its state control, a possibility to design an alternative core test appears. This test could check the correctness of the control part of the core (i.e. FSM states and transitions), data part of the core has to be tested by other techniques.

One option to create an alternative controller test is to extend the standard test wrapper (TW) [1, 4] of the core by adding new states to the TW and suggest modifications in the core which could allow access from the proposed TW to the control structure. This provides an alternative way to test the finite state machine by changing and/or observing the machine states directly.

The location of the FSM in softcore can be done through compilation techniques, by a method which should be independent of the specific coding style of IP core author.

FSM localization is not a new topic in this field of research. Some techniques for FSM extracting have been already proposed [5]. This work is focused on FSM extraction, but again at the RT level, to improve functional verification, by converting the HDL model to a hierarchical *process-module* (PM) graph. Typical FSM patterns are to be searched in the PM graph afterwards. The authors claim that their technique is independent of HDL coding style.

The goal of this method is that the behavioral notation of an IP core, which is delivered to the core user, will be supplied with the results of this analysis. The user will therefore be provided with a detailed information about the alternative way of testing the core. The information will be useful within the design of the test patterns in the subsequent steps of test design. To provide a user with this information, FSM must be located in the code first.

3 Locating the FSM in core

Locating the state control in the behavioral notation of arbitrary IP core consists of these steps:

(1) Syntax analysis of the core done by VHDL compiler. This creates the syntax tree of the core.

(2) Extraction of the input/output ports of the core and extraction of the internal signals of the core from the syntax tree. These are listed in the table called register table.

(3) Syntax tree analysis by one of the proposed methods: `1TR`, `2CA` or `3PE` (described later in the text). These methods add two attributes to each of the items from register table, called n_C as the value of the current state register candidate and n_N as the value of the next state register candidate (both parameters described are later in the text).

Dividing state registers into two registers (current and next state registers) are the specifics of analysis at the behavioral level. After the synthesis into the RTL these two registers are merged into one.

(4) Using the results from the previous step, determine, whether the core under analysis is really a controller and determine the type of the controller, or it is a NAC (see 3.3) core.

(5) Using the results from step 3, determine which register (or registers) represent the state control of the core, and provide the core user with this information.

All five steps of the procedure are described in detail in this section.

3.1 VHDL compilation

The goal of the first step is to convert the core behavioral description into an abstract syntax tree. To do so, Savant VHDL compiler was used [6].

The IEEE standard 1076.6 is defined, which is a suitable subset of the VHDL simulation language for VHDL to RTL synthesis, along with its semantics. The purpose of our work is to locate the state control, which is independent on the core synthesability, so the compilation can be done over the entire set of the VHDL standard. It can be stated that Savant software tool can analyze VHDL descriptions even when they are not in accordance with the IEEE standard 1076.6.

By compiling the source file(s), the abstact syntax tree in the IIR intermediate form becomes available as a result of lexical, syntax and semantics analysis. Using the Part plugin [7] to Savant, the syntax tree with additional information is saved in the DOT format [8] for the next steps of analysis.

Syntax tree Z, $Z = (V, E)$ is formally defined by a set of its vertices $v \in V$ and edges $e \in E$, where

$$E = \{(u, v, i) \mid u, v \in V;\ i \in \mathcal{N}^+\} \tag{1}$$

the symbol i indicates the index of the edge. All vertices have its type assigned from the lexems type set A, which consists about 250 lexem types, which can be formalised as a surjective mapping $f(v) \rightarrow a$, where $v \in V$, $a \in A$. From the set A the following lexem types a are used in this paper: (1) *iir_ifstatement*, which stands for the VHDL *if* statement, which is used for conditional branching, (2) *iir_casestatement*, which stands for *case* statement, used as conditional multiple branching, (3) *iir_casestatementalternativelist*, which stands for list of *case* variants and (4) *iir_casestatementalternativebyexpression* which are used in the subtree of the *case* statement as one of the variant branches, (5) *iir_signalassignmentstatement* for signal or port assignment statement, which is used to change the value of signal or port, (6) *iir_signalinterfacedeclaration* for port declaration, which is used to define an input/output port, and finally (7) *iir_signaldeclaration* for internal core signal declaration.

The names of the lexem types come from the VHDL language description.

3.2 Ports and signals extraction

The information on the state control location can be identified from ports or signals of the core. As a result of this step, a table is formed which contains ports and signals recognized in the core. Input/output ports of the core are of *iir_signalinterfacedeclaration* type, internal signals are of *iir_signaldeclaration* type. Both of lexem types are searched in the vertices set V, creating a new subset $W \subset V$ called register table.

3.3 Evaluation methods

For the analysis, two evaluation methods were used, the third one is under development. As a result of applying this step on the table and abstract syntax tree (formed in the previous two steps), two attributes are added to each row of the table which reflect the possibility that the particular port/signal belongs to the state control. As a conclusion of this step, the state control of the core is determined.

1TR evaluation method

This method is based on the identification of transitive closures in an abstract syntax tree. Evaluation method computes the values of n_C and n_N attributes. The n_C value of arbitrary register s ($n_C(s)$) is

computed as follows:

$$n_C(s) = |T_s| \tag{2}$$

where T_s is a set of ordered n-tuples of the set E:

$$T_s = \{[(u, t_1, 1)(t_1, t_2, x_1)(t_2, t_3, x_2) \dots (t_i, s, x_i)]\} \tag{3}$$

where u is of the *iir_ifstatement* or *iir_casestatement* type.

Value n_N of arbitrary register s ($n_N(s)$) is computed as follows:

$$n_N(s) = |A_s| \tag{4}$$

where A_s is the set of all items from E, where:

$$A = \{(u, s, 1)\} \tag{5}$$

and where u is of the *iir_signalassignmentstatement* type.

2CA evaluation method

`2CA` method is an extension over `1TR` method with different algorithm for the current state register (n_C) computation. The extension takes into account the number of *case* statement alternatives which reflect the nature of the statement. Because `2CA`'s results are always better than that of the `1TR` method, it supersedes the `1TR` method. The algorithm for n_N is same as in the `1TR` method, and the n_C for arbitrary register s is computed as:

$$n_C(s) = |T_s| + |V_s| \tag{6}$$

where T_s is the set of ordered n-tuples of items from E, where:

$$T_s = \{[(u, t_1, 1)(t_1, t_2, x_1)(t_2, t_3, x_2) \dots (t_i, s, x_i)]\} \tag{7}$$

where u is of the *iir_ifstatement* type. V_s is the set of ordered triplets of items from E, where:

$$V_s = \{[(u, s, 1)(u, t, 2)(t, v, x)]\} \tag{8}$$

where u is of the *iir_casestatement* type, t is of the *iir_casestatementalternativelist* and v is of the *iir_casestatementalternativebyexpression* type.

3PE evaluation method

At the moment, the third method is under research which uses similar algorithms for n_C and n_N atributes as `2CA` method. In addition to this, it uses heuristics to determine which registers are uncapable to be either current or next state registers and, according to this heuristics, the n_C or n_N attribute of the register are reset, so they cannot be chosen as the state control of the core. This heuristic is based on locating the assignment operator, from core next state register to the core current state register. Nonexistence of this assignment classifies this core as NAC. Because this method is still under research, there are no formal algorithms.

State control types

NAC (*not a controller*) is a core, which has no FSM be controlled by. Because the input of the analysis can be any core, even that one, which is not a controller, the analysis has to identify this kind of cores.

The 1TR and 2CA methods have no concept of detecting NAC cores, so using defined algorithms on any core with at least one port and/or one internal signal (that means any practical core) will be addressed as controller with one of the signals selected as state control. The result of this analysis is evidently wrong.

If the core is a controller, then there are more possibilities of FSM coding in the core. The state register can be a port (external; E) or a signal (internal; I). The state register can be coded as one register, or two registers, one for the current state and one for the next state of the FSM. Therefore there are six variants of controller cores: E, I, EE, EI, II, IE (the first letter is for the current state register, the second for the next state register).

State register selection

In the 1TR and 2CA methods, the register with the greatest n_C or n_N attribute is selected as the register of the current or next state, respectively.

In method 3PE the register a with the greatest n_C ($n_C(a)$) is selected as the register of the current state. If the greatest n_N is $n_N(a)$, the state register proposed is a, if it is another register b, the tree is searched for an n-tuple starting with *irr_signalassignmentstatement* and eventually ending in the b register. If the tuple is recognized, a is the current state register, b the next state register, otherwise the core is NAC. This method is still under research, so the presented approach may be modified.

4 Experimental results

The methods described in this paper were evaluated in terms of successful state register identification. *Success rate* is given by the ratio of successful identifications to the total number of experiments. There are two methods of success rate computation, the first one counts successes only if both current and next state registers are correctly identified (called pessimistic), and the second one which counts partial success, i.e. when only one of the registers is correctly identified (called optimistic). Optimistic success rate logically equals or is greater than pessimistic rate.

Five different cores from different sources were selected for the experimental testing, along with three synthetic cores created for experimental testing purposes only (these cores are marked with asterisk). Results are shown in Table 1. Table columns show the name of the core, number of core input/input ports, number of core internal signals, type of the controller as defined in 3.3, and the result of the core analysis by respective evaluation methods, where Y means correct analysis and N wrong analysis.

5 Conclusions and future research

The methods proposed in this paper are able to analyse controller softcores with the success rate mentioned, the first two methods (1TR and 2CA) are already implemented. Success rates of the 1TR method are up to 13% pessimistic and up to 38% optimistic, 2CA method is up to 38% pessimistic and 56% up to optimistic and finally 3PE method is between 71 to 86% optimistic. The goal of the research is that the target success rate should be 95% on the set of twenty selected cores.

It is expected that the 3PE method will be able to analyse the set of softcores with the requested 95% success rate and it will be able to identify NAC cores. This method is currently in the state of testing and

IP core	ports signals	controller type	1TR n_C	1TR n_N	2CA n_C	2CA n_N	3PE n_C	3PE n_N
TC01*	3 + 1	I	Y	Y	Y	Y	Y	Y
TC02*	3 + 4	NAC	N	N	N	N	NAC, Y	
TC03*	3 + 0	EE	Y	N	Y	N	NAC, N	
UART	16 + 14	I	N	Y	N	Y	?	
STEP	5 + 4	I	N	N	Y	N	Y	N
ATAC	11 + 5	II	N	Y	N	Y	Y	Y
ITTC	17 + 17	I	N	Y	Y	Y	Y	Y
ZBYS	8 + 2	II	N	Y	Y	Y	?	

Table 1: Experimental results

formalisation so the implementation of this method will be available soon.

Based on the cooperation with a company developing IP cores, it is supposed that the methodology will be further developed in accordance with the needs of practical design area.

Acknowledgements

This work was supported by the Research Project No. MSM 0021630528 – Security-Oriented Research in Information Technology.

References

[1] F. da Silva, T. McLaurin, T. Waayers: The Core Test Wrapper Handbook Rationale and Application of IEEE Std. 1500. Frontiers in Electronic Testing. Springer, 2006. ISBN: 978-0-387-30751-0.

[2] D. Moundanos, J. A. Abraham, and Y. V. Hoskote, "Abstraction techniques for validation coverage analysis and test generation," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 2–14, 1998.

[3] B. K. Sikdar, A. Sarkar, S. Roy, and D. K. Das, "Synthesis of testable finite state machine through decomposition," Proceedings of the 14th Asian Test Symposium, pp. 398–403, 2005.

[4] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper design for embedded core test," Proceedings of the 2000 IEEE International Test Conference, pp. 911–920, 2000.

[5] C.-N. J. Liu and J.-Y. Jou, "An automatic controller extractor for hdl descriptions at the rtl," *IEEE Design and Test of Computers*, vol. 17, no. 3, pp. 72–77, 2000.

[6] "Savant vhdl compiler," 2006, this is an electronic document. Date retrieved: January 18, 2007. [Online]. Available: http://www.cliftonlabs.com/vhdl/savant.htm

[7] L. Kafka, R. Kielbik, R. Matousek, and J. M. Moreno, "Vpart: an automatic partitioning tool for dynamic reconfiguration," in *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM Press, 2005, p. 263.

[8] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo, "A technique for drawing directed graphs," Journal of Software Engineering, vol. 19, no. 3, pp. 214–230, 1993.