FINITE STATE MACHINE LOCALISATION BASED ON IP SOFTCORES ANALYSIS

Ján Kubek
Doctoral Degree Programme (3), FIT BUT
E-mail: kubek@fit.vutbr.cz

Supervised by: Zdeněk Kotásek
E-mail: 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. An approach based on compilation techniques is presented. The main goal of the FSM localisation is to supplement additional information about the softcore for the core user. Three methods and experimental results are presented in the paper together with the perspective of the 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 designer's 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.

1.1 TESTING OF CONTROLLERS

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 [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.

1.2 MOTIVATION

I cooperate with company developing intellectual property softcores. An idea emerged to analyze behavioral VHDL code of arbitrary (either created or supplied) softcore to (1) identify memory structures of the core, (2) identify control structures (FSMs) of the core, if present and (3) design 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 has to consist of two parts: the control part, which covers the combinational logic and the state register 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 means.
One option to create an alternative controller test is to extented the standard test wrapper ([4]) (TW) of the core by adding new states to the TW and suggesting changes in the core, which could allow access from the proposed TW to the control structure. This provides a 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 [2]. 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 the IP core, which can be 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.

3 LOCATING THE FSM IN CORE

Locating the state control in the behavioral notation of arbitrary IP core must consist 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`). 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.
Dividing state registers into two (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 3.6) 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.

3.1 VHDL COMPILATION

Savant is a project of freely redistributable VHDL analysis tools solved by the University of Cincinnati's Experimental Computing Laboratory. Savant has been released under one of the GNU public licenses. By compiling the source file(s), the extended syntax tree in the IIR intermediate form becomes available as a result of lexical, syntax and semantics analysis. Using the Part plugin [1] to Savant, the syntax tree with additional information is saved in the DOT format 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

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

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 *if* statement, (2) *iir_casestatement*, which stands for *case* statement, (3) *iir_casestatementalternativelist* and (4) *iir_casestatementalternative-byexpression* which are used in the subtree of the *case* statement, (5) *iir_signalassignment-statement* for signal or port assignment statement, (6) *iir_signalinterfacedeclaration* for port declaration and finally (7) *iir_signaldeclaration* for signal declaration.
Names of the lexem types comes from VHDL language description.

3.2 PORTS AND SIGNALS EXTRACTION

Input/output ports of the core are of *iir_signalinterfacedeclaration* type, internal signals are of *iir_signaldeclaration* type. Register nodes are searched in the list of all vertices and recorded into a register table.

3.3 1TR EVALUATION METHOD

This method is based on the identification of transitive closures in an extended syntax tree. Evaluation method compute 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:

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

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:

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

and where u is of the *iir_signalassignmentstatement* type.

3.4 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:

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

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.

3.5 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 neither current nor next state registers and, according to this heuristics, the n_C or n_N attribute of the register are reset, so it cannot be chosen as the state control of the core. Because this method is still under research, there is no formal algorithms, but anyway there are some experimental results mentioned in the Section 4.

3.6 NAC CORE DETECTION

NAC (which stands *not a controller*) is a core, which has no FSM to control it. Because the input of analysis can be arbitrary core, even that one, which is not a controller, the analysis have 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 sane core) will be addressed as controller with one of the signals selected as state control. The result of this analysis is evidently wrong. There are some heuristic NAC detection tricks currently used in the `3PE` method, which are under research. If the core is a controller, then there are three types 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 types of controller cores: E, I, EE, EI, II, IE (the first letter is for the current state).

3.7 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's other 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 description may change.

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.

5 CONCLUSIONS AND FUTURE RESEARCH

Proposed methods 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 ranges between 14 and 43%, 2TR method between 29 to 57% and `3PE` method 71 to 86%. The goal of my research is that the target success rate should be 95% on the set of twenty selected cores. More softcores are yet to be analysed, which will make the success rates of the proposed methods more accurate. 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 formalisation so the implementation of this method will be available soon.

REFERENCES

[1] 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*, page 263, New York, NY, USA, 2005. ACM Press.

[2] C.-N. J. Liu and J.-Y. Jou. An automatic controller extractor for hdl descriptions at the rtl. *IEEE Design and Test of Computers*, 17(3):72–77, 2000.

[3] D. Moundanos, J. A. Abraham, and Y. V. Hoskote. Abstraction techniques for validation coverage analysis and test generation. *IEEE Transactions on Computers*, 47(1):2–14, 1998.

[4] Y. Zorian. *IEEE Standard Testability Method for Embedded Core-based Integrated Circuits*. IEEE, 2005.