LOCALISATION OF THE FINITE STATE CONTROL IN THE IP CORES

Ing. Ján KUBEK, Doctoral Degree Programme (2)
Dept. of Computer Systems, FIT, BUT
E-mail: kubek@fit.vutbr.cz

Supervised by: Dr. Zdeněk Kotásek

ABSTRACT

This paper deals with methods of finite state machines (FSMs) localization in FSM-based, softcore intellectual property (IP) cores. An approach based on compilation techniques is presented. The main goal is to minimize test application time by offering a alternative way to test the control part of the FSM in the core. The proposed method was implemented using Savant VHDL compiler, 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 larger, more complex, integrated circuits. This, coupled with competitive pressures to improve both the design productivity and the time to market, are driving engineers toward new System on Chip (SoC) design methodologies and the growing use of predesigned embedded (intellectual property; IP) cores within their designs.

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 physical layout on the silicon mask.

1.1 IEEE P1500 TEST WRAPPER

Every SoC core is surrounded by standardized core test wrapper. This wrapper provides mechanisms for core test data access and core test isolation. It allows controlling core primary inputs and observing core primary outputs via the test access mechanism. It is also capable of providing test isolation of the core. The test isolation allows a core to be appropriately isolated when it or an adjacent core (or UDL) is tested, thus preventing any damage to the system chip [1].

Standard test wrapper can work in one of these four modes; *normal* – wrapper is transparent to the normal core input and output operations, the core works according to its specification; *internal test* – wrapper cells disable the core's normal mode I/O and enable access for source and sink of test pattern data, core itself is tested, *external test* – wrapper cells assume their scan-mode configuration and provide testability for the surrounding interconnection, and logic between cores, *isolation* – wrapper disables the core's normal mode, sets appropriate core's inputs to fixed constraints, and sets the appropriate core's outputs to predetermined conditions.

2 FSM-BASED IP CORE

Consider IP core FSM-based or FSM-driven (i.e. it is a controller). After locating the FSM in the behavioral notation of the core by finding its control structure, there is a possibility to design a specific core test. This test could check the correctness of the control part of the core (i.e. FSM's transitions), data part of the core has to be tested by other means. An extension of the standard test wrapper of the core by adding new states to the wrapper and suggesting changes in the core, which could allow access from the extended test wrapper to the control structure are needed for direct testing of the finite state machine. These suggested changes can be created algorithmically. Location of the FSM in softcore can done by compilation techniques, by a method which is independent on the specific coding style of IP core author.

2.1 LOCATING THE FSM IN CORE

Proposed technique is useful only when the core contains FSM, thus containing elements used as finite state control. This control has to have a central structure where the state information is stored in the core. Finding out, where this structure is located is crucial for the mentioned method to work.

FSM localization is not a new topic in this field of research. Some techniques for FSM extracting have been proposed [2]. This work is focused on FSM extraction from HDL model, but at the RT level, to improve functional verification, by converting the 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 on HDL coding style.

2.2 SEARCH METHOD

In the behavioral notation, central structure of the finite state control will be defined as a *signal* (or more *signal*s) in the core architecture. States will be defined as a set of possible values of those signals, e.g. noted as *type*s. Transitions between states can be then encoded by conditional statements like *if* and *case* with the central structure in the condition (current state affects the behavior of the core and the next state will mostly depend on the current state of the FSM), containing assigments to the *signal* or *signal*s of the next state, which can of course be identical with current state.

2.3 SAVANT VHDL COMPILER

Currently used method of finite state control localization employs the capabilities of Savant VHDL compiler [5].

By compiling the source file(s) the parse tree of the HDL code becomes available as a result of lexical and syntax analysis. Using the plugin component to Savant allows to traverse this parse tree, effectively analyzing the structure of the softcore.

2.4 SYNTAX TREE ANALYSIS

Two methods of syntax tree analysis, which came from my research, were evaluated.

Common part of both of them is that the entire *entity* and its *architecture* is searched in the syntax tree and *port*, *signal*, *if*, *case* and assignment statements are analysed. Both the methods have two attributes for each signal or port, which represents their hit rate as a part of the finite state control.

First one is the hit rate of the signal or port to hold the current state of the finite state control (CST value). The second one is the hit rate of the signal or port to hold the next state of the finite state control (NST value).

It is clear, that if the coding style of the HDL is the one that uses only one structure for current and next state of the FSM, then CST and NST value of this structure would be the biggest among all other signals and ports.

The first analysis method, called `1TR`, keeps a list of all ports (primary inputs and outputs) and signals. For each *if* or *case* statement with a signal or port in the condition, one point is added to this signal or port CST value. For each assignment statement of a port or signal, one point is added to this signal or port NST value.

The second analysis method, called `2CA`, stems from the `1TR` method, but the CST value computation is slightly different. A port or signal CST is increased by one point for each *if* statement with this signal or port in the condition and by one point for each *when* statement within a *case* statement with this signal or port in the condition.

More about analysis methods and their outcome is to be discussed later, in Sections 4 and 5.

3 CORE DESIGN MODIFICATION

Using standard methods to test FSMs, algorithms exist to create test patterns. These test patterns are there to set the state of the automata and to identify the state in which the FSM was [4]. The disadvantage of this method is, that if only one transition is to be tested, possibly many cycles of the state-preset pattern have to be executed, then the requested transition is to be made, followed by many cycles of the state-identification patterns. This extends the time needed for testing.

For this generic testing method, location of the control structure is irrelevant, it works with the FSM inputs and outputs only. Having the position of the control structure already available, it is possible to design dedicated test patterns for testing of the FSM internal control logic only.

With modifying the core by creating a shortcut from core primary inputs to the control structure of the FSM and from the control structure to the core primary outputs, the state-preset sequence and the state-identification sequence can be shortened to only one test cycle.

Sequent changes has to be arranged to the standard core test wrapper, which now has to handle presetting and identifying of states as the part of core internal test – test of core's controller (or internal core's FSM) logic.

3.1 STRUCTURE MODIFICATIONS

By connecting the core's primary inputs and outpus with the control structure through multiplexer controlled by modified test wrapper, time needed to set or identify the state of control structure will be shortened to one cycle. Total time needed to test one transition of the FSM will be cut to only three cycles then: (1) cycle to preset requested state of control structure, (2) setting the appropriate inputs and executing the transition and (3) a cycle to identify new state of the control structure.

Test wrapper needs to be extended to cope with new operations, presetting and identifying state of the control structure. They are called *set FSM state* and *get FSM state*.

Switching test wrapper to the *set FSM state* allows requested state to be loaded to the core through multiplexers defined in previous section. On the other hand, switching test wrapper to the *get FSM state* makes current state of the machine available on the primary outputs of the core.

3.2 SOFTCORE MODIFICATION

Core itself needs to be modified in order to use new test method.

For presetting the value of status structure its input multiplexing is required, so a regular function of the core is allowed when in normal mode, and presetting of the value through the primary inputs is possible in the *set FSM state* mode.

The same operation needs to be applied by analogy for status structure output, primary outputs and *get FSM state* mode.

4 EXPERIMENTAL RESULTS

Seven different cores from different sources was selected for the experimental testing of the proposed analysis methods. Cores marked with asterisk are not real-life cores, they are synthetic cores created for testing purposes only. Core `TC02` is not a FSM-based core, so it has no central structure (NAC stands for *not a controller*). Results are summarized in Table 1. First column is the name of the core, second the number of input/output ports and core signals, third the proper location of the central part of the cores FSM. Fourth and fifth column are the results of the core analysis methods.

As a result of this experiment, it is visible that the `2CA` analysis method is same or better in the FSM detection than `1TR` (CST detection failures at `STMT` and `ITTC` cores). However, there are still problems with detection of cores not containing any FSM (like core `TC02`) and the success rate of CST or NST detection is actually not sufficient.

IP core	ports, signals	CST/NST	1TR result CST/NST	2CA result CST/NST	outcome
TC01*	3, 1	S1/S1	S1/S1	S1/S1	analysis OK
TC02*	3, 4	NAC	P1/S2	P1/S2	NAC detection problem
TC03*	3, 0	P3/P2	P1/P2	P1/P2	NST fail
STMT	5, 4	S1/S1	P3/P1	S1/P1	NST fail, 1TR CST fail
ATAC	11, 5	S1/S2	S3/S2	P3/S2	CST fail
BUSC	16, 7	S2/S3	S14/S3	S14/S3	CST fail
ITTC	17, 17	S1/S1	P1/S1	S1/S1	2CA OK, 1TR CST fail

Table 1: Experimental results

5 CONCLUSIONS

By examining analysis results, it becomes evident that the better analysis methods of the syntax trees are needed, with better success rate and NAC core detection capabilities. Different evaluation methods are under consideration, including oriented graph of signal and port assignments.

For better detection statistics more cores have to be analysed, focusing on real-life IP cores (controllers), which are currently used in SoC design. Even NAC cores are useful for testing of the NAC detection. Next phase of research covers creation of algorithms, extending test wrapper and modifying the core, described in Section 3.

ACKNOWLEDGEMENTS

This research was supported by the Czech Ministry of Education – FRVŠ grant No. 3198/2006/G1 and the Grant Agency of the Czech Republic under grant No. 102/04/0737 “Modern Methods of Digital Systems Synthesis”.

REFERENCES

[1] Marinissen, E., Goel, S., Lousberg, M.: Wrapper design for embedded core test, in Proceedings of the International Test Conference ITC’00, IEEE, 2000

[2] Liu, C.-N. J., Jou, J.-Y.: An Automatic Controller Extractor for HDL Descriptions at the RTL, IEEE Design and Test of Computers, vol. 17, no. 3, pp. 72–77, 2000

[3] Moundanos, D., Abraham, J., Hoskote, Y.: Abstraction Techniques for Validation Coverage Analysis and Test Generation, IEEE Trans. Computers, 1998, pp. 2–14

[4] Yannakakis, M., Lee, D.: Testing finite state machines, ACM symposium on Theory of computing, New Orleans, Louisiana, ACM Press, 1991, pp. 476–485

[5] Savant VHDL compiler project, www.cliftonlabs.com/savantp.htm