Transaction-based Waveform Analysis for IP Selection

Jian Liu, Eugene Shragowitz

Department of Computer Science and Engineering, University of Minnesota
Minneapolis, Minnesota 55455 USA

ABSTRACT - In a process of IP selection, it is necessary to establish whether a candidate IP is equivalent to a behavioral model of a design proposed by a customer. It is desirable to perform this verification to exclude IPs, which "don't match" the rest of the design. This work combines a simulation approach to establishing equivalence between models with formal regular expression techniques to provide transaction-level preliminary evaluation of IP suitability. Such evaluation could precede a decision to acquire IP.

1. INTRODUCTION

IP reuse is becoming an essential part of SoC designs. In IP-based design flow, a designer may have his own model of component for quick prototyping. Then he needs to find out whether an IP core can be used to replace this model.

Generally, simulation and formal verification are two major approaches to establishing equivalence between hardware models [1]-[2]. Each of these approaches taken alone has serious deficiencies in the context of IP. The traditional formal verification approach is not applicable at this stage of IP selection because vendors usually don't release internal details of their models prior to a sale. Simulation via Internet is a practical approach to conduct evaluation in such circumstances. The waveform obtained form simulated IPs can be compared with the waveforms from behavioral models to evaluate their match.

Commercial waveform analysis tools [3] establish similarity between two waveforms only if one waveform presents a copy of the other displaced in time. It is very likely that two independently developed models for the same specification produce different waveforms at the specified ports for the same testbenches. Causes of differences in waveforms are numerous and are not expired by such factors as a different number of clock cycles per operation, different word length, SET/RESET conditions at asynchronous inputs, etc. In spite of substantial visual differences in waveforms, two models could be equivalent in a specific sense.

Framework proposed in our paper introduces a formal technique for the post-simulation waveform comparison. It performs evaluation of hardware models by comparing waveforms at respective ports. In this work, waveform analysis is raised from the signal-event level presented by simulation to a transaction level, where a transaction consists of a sequence of input/output signal events. For example, the memory read/write transaction typically consists of setting address, enabling memory and reading/writing data. Fig. 1 illustrates the proposed waveform analysis.

The rest of the paper is organized as follows. The waveform coding method is presented in Section 2. The outline of the framework followed by description of each step is given in Section 3. Section 4 provides examples.

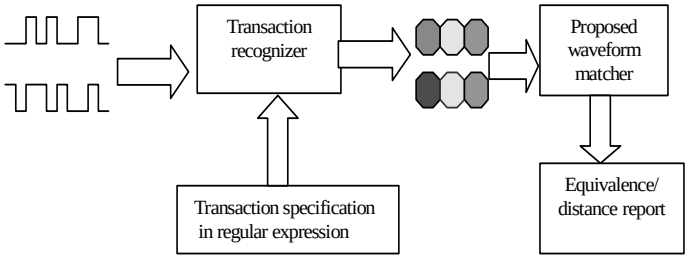

Figure 1. Traditional and proposed waveform comparison

2. CODING SIMULATION WAVEFORMS

In this section, we describe how to code simulation waveforms into data structures called waveform strings, which are compared to each other. We focus on synchronous sequential circuits with single or multiple clocks.

Given a set of interface signals, i.e. primary inputs and outputs. Each PI/PO is either a bit or a bit vector. The following terminology is used in the text.

Waveform character*:* A waveform character is a vector with components defined by signal values at ports of a model at any clock cycle.

Waveform string*:* A waveform string is a series of waveform characters produced at ports at consecutive clock cycles.

Waveform component string*:* A waveform component string is a series of values for a given port at consecutive clock cycles of simulation.

In the sequential circuits, a signal is either edge sensitive or level sensitive. For edge-sensitive signals, waveforms at the input/output are sampled at the active clock edge and the values are registered as an effective value. For level-sensitive signals, the last values during the active level of each clock period are effective.

For example, consider the waveform of D-f/f from Fig. 2, the correspondent waveform strings contain 8 waveform characters.

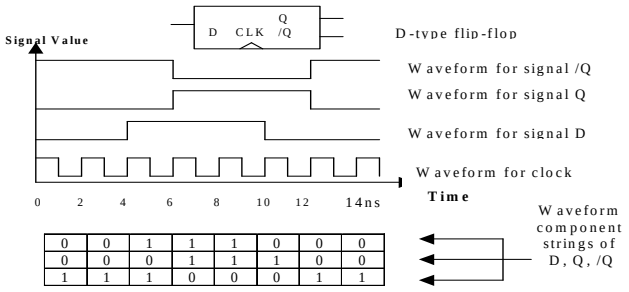

Figure 2. Waveform coding for D-flip/flop simulation

3. TRANSACTION-BASED WAVEFORM ANALYSIS

In the described framework, analysis is conducted in three phases. At phase 1, regular expressions are used as a vehicle to capture transactions. Designers describe the same type of transactions for two models as regular expressions. DFAs are automatically derived from these regular expressions and used to scan the coded waveform strings and recognize transaction tokens. At phase 2, the transaction tokens are labeled and sorted in ascending order of their starting positions. At phase 3, transaction-based measures of distance between waveforms are introduced; the minimum distance is found.

Phase 1: Identifying transaction tokens using regular expressions

Regular expressions (REs) REs are provenly equivalent to FSMs. They are used in many fields, including software engineering [6] and hardware design. Applications of RE in hardware design include interface synthesis [7] and formal model checkers specification [8]. In this work, REs are used for waveform transformations and comparisons. In this work, the technique is extended to the vector interpretation of a character.

According to definitions in section 2, each waveform character in the regular expression is a vector, every coordinate of which is associated with a particular signal. Assume that a waveform character is associated with N signals $S_1, S_2, \ldots S_n$, and the possible value set of each signal is $V_1, V_2 \ldots, V_n$, and the alphabet for the waveform character is $\{< a_1 a_2, \ldots a_n > | a_i \in V_i\}$.

Similar to meta-characters used in traditional REs, meta-components could be introduced for each vector component. There are two types of meta-components in this application. One type includes predefined meta-components. The other type includes meta-components with the values defined over the value set V_i or its subset defined by users.

There are two predefined meta-components, "-" and "/". The first one stands for *don't care* condition. It can be mapped to any signal value. The second one is a transaction delimiter. This means the signal value at this position is not a part of this transaction and it may have any value from the value set.

As an example, let us consider D-flip/flop and describe its work in terms of transactions. Generally, when D is set to 1/0 at certain clock cycle, Q and /Q will be 1/0 and 0/1 at the next clock cycle. A regular expression for the transaction of setting output Q to 1 consists of two generalized vector-characters (1 / /) (/ 1 0). Similarly for setting output Q to 0, the regular expression is (0 / /) (/ 0 1). With this regular expression specification, we can identify seven transactions in the waveform in Fig. 3.

In our work, users are allowed to define meta-components. Each meta-component is associated with a name and a value. The name is provided by the user, while the value is determined when scanning of strings is performed. Use of meta-components enables the symbolic representation of transactions. More details are given in section 4.

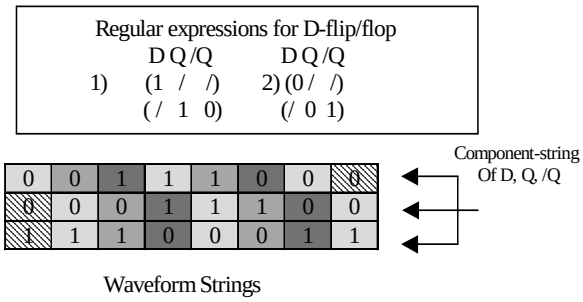

Figure 3: Example of identifying transactions using regular expressions

For two models M and M', the same transaction may produce different sequences of signal events. Therefore, the designer should provide a pair of regular expressions, each correspondent to the models M and M' respectively.

Phase 1 consists of two steps:

1. Regular expressions are converted into DFAs [4].
2. Waveform strings are scanned by DFAs to identify the transaction tokens. If user-defined meta-components are used in regular expressions, the value for each of them is resolved at this step and stored internally.

Each transaction token is a sub-waveform string that is an instance of a regular expression. At the end of step 2, a set of transaction tokens is identified for a waveform string of each model.

Phase 2: Labeling tokens

The transaction tokens are sorted in ascending order by their starting time in each waveform string. An identifier is assigned to the tag field of each token. The same identifiers are assigned to identical tokens in the same waveform string and in two different strings if the strings are matched. This phase can be broken into two steps.

1. For each of waveform strings, the transaction tokens obtained in the previous phase are lined up.
2. For any pair of transaction tokens <t1,t2>, if and only if the tokens are associated with the same regular expression pair and with the same name-value table, they are labeled by the same identifier. It implies that transactions are instances not only of the same type, but also associated with the same data.

By the end of phase 2, two sequences of transactions TS1 and TS2 are formed. Each element in the corresponding transaction sequence is associated with one token from the token sequence formed by the previous phase for the input stream.

Phase 3: Finding minimum distance

At this phase the algorithm produces quantitative measures of a distance between the waveform strings for two models in the context of transaction tokens.

In the following text, we first introduce quantitative metrics for the distance between waveform strings. Then we describe how the metrics are adapted to the context of transaction tokens.

Quantitative metrics for distance between waveform strings

Given two strings *s1* and *s2*, different metrics can be used to evaluate the similarity of two strings. In our work, we propose two metrics for the string distance measurement. The first one is the edit distance and the second is the block distance. The edit distance is the number of operations (insertion, deletion, and substitution of one symbol) necessary to convert one string into the other. This metric has been adopted by many different applications [5]. The block distance is the minimum number of block operations that are needed to convert one string into another for the given edit distance. Each block operation consists of a series of consecutive edit operations of the same type. For example, given two strings "111" and "1110000", the edit distance is 4, and the block distance is 1. Since a small difference in designs may result in consecutive mismatches in waveforms, the block metric can interpret such situations more effectively. In this work, the edit and block distances are derived for each pair of interested common component waveform string.

Distances in the context of transaction tokens

Assume that the complete transaction sequence during simulation is *TS*. After phases 1 and 2, two transaction sequences *TS1* and *TS2* are derived and the identical transactions are labeled with the same identifiers. Since we describe only the transactions which may produce differences in waveforms, both *TS1* and *TS2* are subsequences of the input stream *TS*. It is possible that *TS1* and *TS2* are not identical. The reasons are twofold. Firstly, the regular expressions may not be accurate or complete to capture the transactions. Secondly, the models themselves may contain bugs, therefore transactions are not executed as expected and are not recognized. In order to measure the divergence between waveforms in such context, the following definition is introduced:

Alignment sequence: Assume that tags of transaction token sequences *TS1* and *TS2* are $< x_1, x_2, \ldots x_m >$ and $< y_1, y_2, \ldots y_n >$ for two models respectively. An alignment sequence $< m_1, m_2, \ldots m_k >$ is an indexed common subsequence of *TS1* and *TS2*. Each element m_i of this sequence is associated with a pair of indexes (a_i, b_i) defined by positions in *TS1* and *TS2*. It is assumed that tokens in *TS1* and *TS2* that combined by the alignment sequence into m_i correspond to the same token in *TS*.

Fig. 4 gives examples of alignment sequences. Assume that *TS* is {*A, A, B, C, D*}, *TS1* and *TS2* are {*A, A, C, D*} and {*A, C, D*} respectively. This figure illustrates two possible alignment sequences for *TS1* and *TS2*. In the first one, the *A* of *TS2* is associated with the first *A* of *TS1*. In the second one, the *A* of *TS2* is associated with the second *A* of *TS1*.

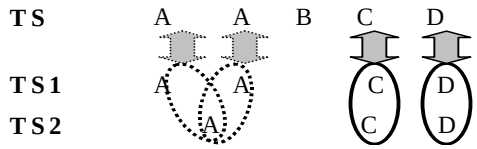

Figure 4. Example of alignment sequences

If a given alignment sequence contains *X* transactions, then the corresponding transaction tokens partition each of the original waveform strings into the *2X+1* sub-strings that includes *X* transactionalized sub-strings and *X+1* plain waveform sub-strings that are not recognized as tokens. For each pair of plain sub-strings, the edit and block distance can be derived. Finally the distance for two waveform strings is defined by the sum of these distances. Fig. 5 illustrates this ideology.

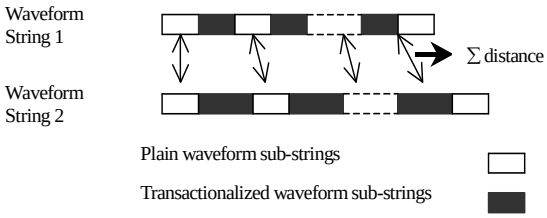

Figure 5. Distance for a given alignment sequence

For each alignment sequence, a pair of edit and block distance (*ED, BD*) is derived. As the edit and block distance is dependent on the alignment sequence, it is desirable to find minimum edit and block distances between the waveforms and the correspondent alignment sequence is called optimal alignment sequence. Our experiments show that in most practical cases, the Longest Common Subsequence of TS1 and TS2 is the optimal alignment sequence. In short, phase 3 consists of two steps to find the minimum distance.

1. Find the Longest Common Subsequence of TS1 and TS2.
2. The LCS is used as the alignment sequence, distance upon LCS is calculated.

4. Examples

The algorithm for waveform analysis is implemented in C++ at SUN SPARC Ultra 5/10 workstation. A set of experiments has been conducted. But for the sake of brevity, only two cases are used below to demonstrate the effectiveness of the approach.

Case 1. PIC15C5X compatible microcontroller

In this experiment, the Free-RISC8 core available at the website [9] is used as the Model 1, and the Model 2 is SILICORE [10] product. In this example, the waveforms of signals at the instruction port, ROM address bus, and IO data ports, that are common in the interfaces of both models, were compared. For a

microcontroller, each instruction can be deemed as a transaction. The instruction set includes 32 instructions, most of them require one clock cycle to complete and only 3 instructions require 2 or more clock cycles. For example, one of such instructions is *GOTO* instruction. The models differ for the multiple clock cycle instructions. Each multi-cycle instruction requires 2 clock cycles in the first model. For the second model, this transaction requires 3 clock cycles. Table 1 presents regular expressions for this instruction for both models.

Table 1: Regular expressions for Multiple Cycle instruction for two models

Model 1	Model 2
Inst ROM_addr IO_ports	Instr ROM_addr IO_ports
(x y z)	(x y z)
(- - -)	(- - -)
	(- - -)

Along with this regular expression pair, we introduce following definitions for meta-components:
META_COMPONENT MUL_CYCLE_OPR {101XXXXXXXX [1], *1001XXXXXXXX 1000XXXXXXXX} x;*
META_COMPONENT y, z;
The above regular expression pairs combined with the definitions of meta-component shows that in the simulation, if the instruction X of the ROM(y) is executed, the model 2 will take one more cycle to finish. For a testbench segment that contains 37 instructions, Table 2 summarizes the following distances between waveforms.

Table 2: Edit and block distances for signal waveforms with & without transaction recognition

Signals	Edit/Block Distance Without Transaction Recognition	Edit/Block Distance With Transaction Recognition
Instruction	3/3	1/1
ROM_address	3/3	1/1
IO_port	3/1	1/1
Total	9/7	3/3

Case 2: DES engine
In this experiment, we compared two DES models, both of them implemented standard DES encryption/decryption algorithm and are compatible with NIST-800-17. The first one was available at [11], and the other is downloaded from [10].
For this design we can consider encryption of input data as a transaction. The major difference is that the second design can operate in a pipeline mode, while the first one can only process a new encryption after the previous transaction is completely finished. Such differences are reflected by the regular expression in Table 3, where x, y and z are user-defined meta-components.

Table 3: Regular expressions for encryption for two models

Model 1	Model 2
Decrypt d_in key d_out	Reset decrypt d_in key d_out
(0 x y -)	(0 1 x / /)
{15}[2]	(0 1 / y /)
(0 x y z)	(0 1 / / /) {15}
	(0 1 / / z)

For the first model, when a plain text and a key are inputs, they must be kept stable for 15 clock cycles, the cipher is available until at the 16th clock cycle. For the second model, data and key are stable for the first 2 clock cycles, then after 15 clock cycles, the cipher will be available. Meanwhile, a new plain text and key pair serve as the input for following transactions.
90 test vectors were used to simulate the both IP models. Table 4 presents the distances for waveforms with and without transaction recognition.

Table 4: Edit and block distances for signal waveforms with & without transaction recognition

Signals	Edit/Block Distance Without Transaction Recognition	Edit/Block Distance With Transaction Recognition
D_in	1335/16	17/1
Key	1351/57	17/2
D_out	1351/90	17/1
Total	4037/163	51/4

As it is confirmed by experiments quantitative measures of distance on the transaction level drastically reduce the amount of work on interpretation of simulation results and allows a designer to concentrate on a few mismatches instead of very many. Provided with the exact location and source of mismatches in waveforms on the transaction level, a designer has a choice to ignore the differences as not significant, or introduce a glue logic to eliminate differences.

4. REFERENCES

[1] C. Kern and M. R. Greenstreet, "Formal Verification in Hardware Design: A Survey", *ACM Transactions on Design Automation of Electronic Systems*, Volume 4, Issue 2, 1999.

[2] F. Corno, M. S. Reorda, G. Squillero, "Simulation-based Sequential Equivalence Checking of RTL VHDL", *proc. of 6th IEEE Intl. Conf. On Electronics, Circuits, and Systems*, 1999.

[3] DAI Comparescan homepage, http://www.designacc.com/products/comparescan/index.html, May, 2000.

[4] A. V. Aho, R. Sethi, J. D. Ulman, "*Compilers: Principles, Techniques, and Tools"*, Addison Wesley Publisher, 1988.

[5] D. Sanko, "Edit Distance for Genome Comparison Based on Non-local Operations", *proc. of 3rd Annual Symposium on Combinatorial Pattern Matching*, pp. 121-135, 1992.

[6] F. Lustman, "Specifying Transaction-based Information Systems with Regular Expressions", *IEEE trans. on Software Engineering*, Vol. 20, Issue 3, pp. 207-217. March 1994.

[7] R. Passerone, J. A. Rowson, "Automatic Synthesis of Interface between Incompatible Protocols", *proc. of Design Automation Conference*, 1998.

[8] IBM Formal Checkers Specification: http://www.haifa.ibm.com/projects/verification/RB_Homepage/ps/checkers.ps, 2001.

[9] Homepage of Silicore. http://www.silicore.net, 1999.

[10] Free IP project, available at http://www.free-ip.com/risc8/index.html, 2001.

[11] Open Cores project, available at http://www.opencores.org. 2001.

[1] In the binary op-code, the sequence 101 is a code for GOTO instruction, the following part is the address of the next instruction.
[2] {15} indicates this item is repeated for 15 times.