TR 01-046

Transaction-based Waveform Analysis for Functional Verification in IP Selection

Jian Liu and Eugene Shragowitz

December 13, 2001

Transaction-based Waveform Analysis for Functional Verification in IP Selection

Jian Liu and Eugene Shragowitz
Department of Computer Science and Engineering
University of Minnesota, Minneapolis MN 55455
Email: {jliu,shragowi}@cs.umn.edu

ABSTRACT

Functional verification is an important aspect of IP selection. Formal verification and logic simulation are two traditional approaches to this problem. Both techniques have substantial limitations. A method described in our work combines these two basic approaches to achieve effective verification. A formal regular expression technique is merged with the simulation to provide meaningful transaction level verification of IP suitability. Implementation is illustrated by examples.

Keywords
Simulation, functional verification, intellectual property, regular expressions

1. Introduction

IP reuse is becoming an essential part of SoC designs. A candidate IP is subjected to verification to answer the question whether or not functionality provided by the IP is consistent with the behavior required by the potential user.
Generally, simulation and formal verification are two major approaches to establishing equivalence between hardware models [1]-[4]. Each of these approaches taken alone has serious deficiencies. The simulation approach has problems in distinguishing important variations in waveforms from those, which could be easily adjusted. Formal verification approach to proving equivalence has the complexity problem on the top of the “acceptable” differences aspects.

A method proposed in our work combines these two basic approaches. It is based on application of simulation techniques with the results analyzed and interpreted with one of the formal techniques called “regular expressions”. Regular expressions serve as an input language for many systems that process strings. In our work, this technique is modified and extended for the purpose of interpretation and comparison of simulation results.
A proposed framework analyzes equivalence of hardware models by comparing waveforms at respective ports. Commercial hardware simulation tools have limited capabilities to compare waveforms [5][6]. Similarity is established only if one waveform presents a copy of another waveform displaced in time. Such capabilities are not sufficient. In spite of substantial visual differences, two models could be equivalent in a specific sense. 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. Even for the same behavioral model, different synthesis settings may lead to different hardware solutions [7]. In the existing design environment, it takes substantial time to identify divergences in waveforms and to evaluate their significance. The framework described here automates a substantial part of this process. The simulation-based verification of equivalence is illustrated in Fig. 1.
In this work, an abstraction level in a waveform analysis is raised from the signal event level presented by simulation to a transaction level, where the 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. Before simulation, the transaction level testbench is adapted to interfaces and timing relations in the models. The rest of the paper is organized as follows. The coding method is presented in Section 2. The outline of the framework followed by detailed description of each step is given in Section 3. Section 4 provides experimental data. Conclusions are presented in Section 5.

2. Coding simulation waveforms

Main objects of our study are sequential circuits with single or multiple clocks. Each signal is allowed to take only binary values 0 and 1. For brevity, we describe here the system only with the single clock, while extension to multiple clocks is also possible. Signal waveforms at the input/output are sampled at the active clock edge and the values “1” and “0” are registered. “1” is produced when the signal is rising from “0” to “1”, or when a signal retains the value “1” from the previous clock period. Alternatively, “0” value is registered if the signal is falling with the active clock edge or retains previously acquired value 0. Consider D-f/f from Fig. 2. For signal D, a component-string of coded signal values is (0,0,1,1,1,0,0,0). The i-th character of the component string stores the signal value at the rising edge of i-th clock cycle. Fig. 3 contains coded signals for the D-f/f.

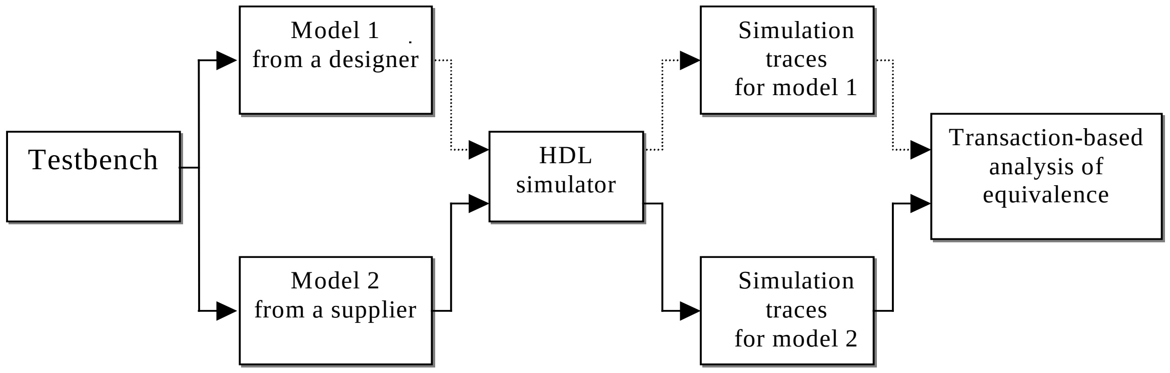
Testbench
Model 1 from a designer
Model 2 from a supplier
HDL simulator
Simulation traces for model 1
Simulation traces for model 2
Transaction-based analysis of equivalence

Figure 1: Simulation-based verification of equivalence on transaction level

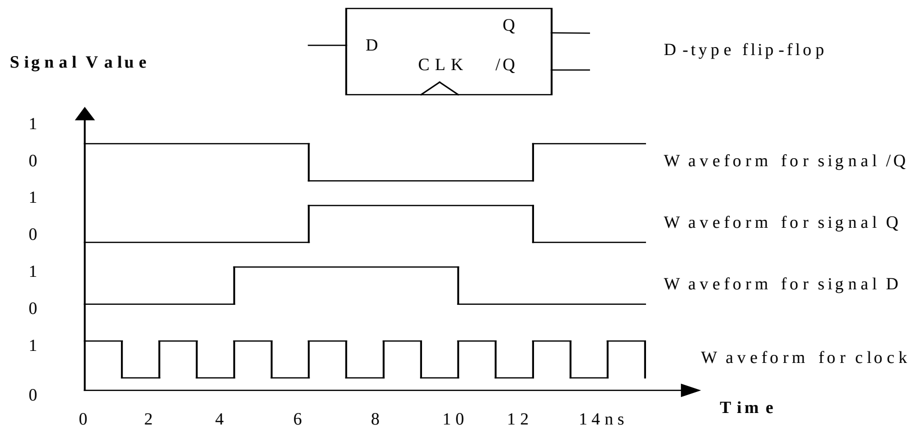

Figure 2: Simulation waveforms for D-flip/flop

0	0	1	1	1	0	0	0
0	0	0	1	1	1	0	0
1	1	1	0	0	0	1	1

Element-strings of D, Q, /Q

Figure 3: Waveform String derived from simulation waveforms

3. Transaction-based waveform analysis

In the proposed framework, analysis is conducted in three phases. At the first phase, designers describe the transactions of the same type for two investigated models as regular expressions. DFAs are automatically derived from these regular expressions and used to scan the coded waveform strings and recognize transaction tokens that are sequences of characters accepted by the DFAs.

At the second phase, the transaction tokens are labeled and sorted in ascending order of their starting positions. At the third phase, new transaction-based measures of distance between waveforms are introduced and applied to final evaluation of an IP. Fig. 4 illustrates the workflow of the process. In the following text details of the algorithm are provided.

3.1 Identifying transactions tokens using regular expressions (Phase 1)

As stated earlier, the same transaction executed by two models may produce two different signal event sequences. If users have prior knowledge about such differences, they can provide regular expressions for each model. The regular expressions describe the signal events for corresponding models.

Since the waveform string may contain signal waveforms at different ports, each character in the regular expression is a vector, every coordinate of which is associated with a particular signal. Assume that a vector includes *N* signals $S_1, S_2, ...S_n$, and the possible value set of each signal is $V_1, V_2 ..., V_n$, the alphabet for the vector character is $\{< a_{1,} a_2, ...a_n >| a_i \in V_i\}$.

Similar to meta-characters used in traditional regular expressions, meta-components could be used for each of the vector components. There are two types of meta-components in this application. One type is the predefined component over the alphabet, the other is 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 values. The second is a transaction delimiter. When a character “/” is used, the signal value depends on the previous transaction or on the next to the current transaction.

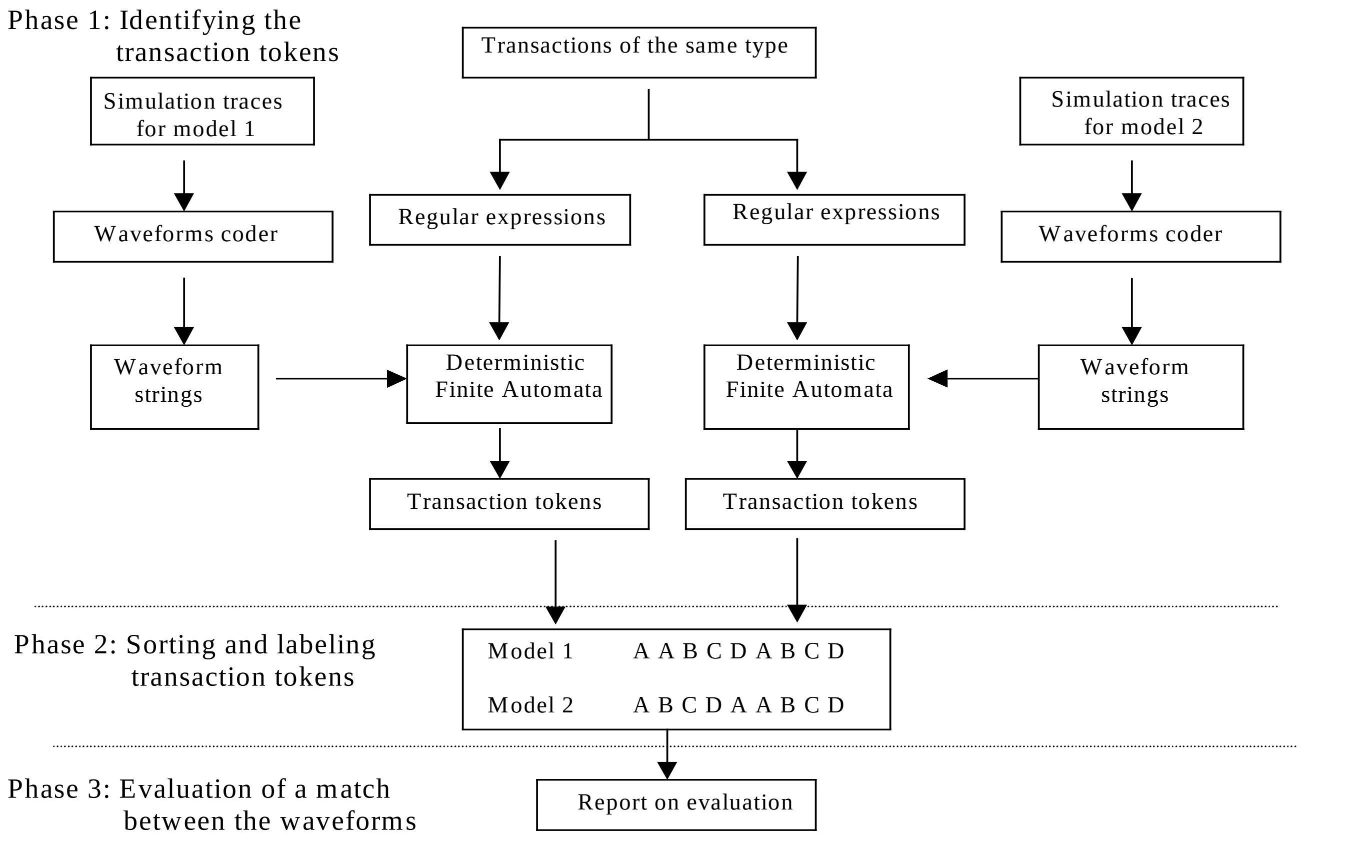
Phase 1: Identifying the transaction tokens
Transactions of the same type
Simulation traces for model 1
Simulation traces for model 2
Waveforms coder
Regular expressions
Regular expressions
Waveforms coder
Waveform strings
Deterministic Finite Automata
Deterministic Finite Automata
Waveform strings
Transaction tokens
Transaction tokens
Phase 2: Sorting and labeling transaction tokens
Model 1 A A B C D A B C D
Model 2 A B C D A A B C D
Phase 3: Evaluation of a match between the waveforms
Report on evaluation

Figure 4: Block-structure of transaction-based waveform analysis

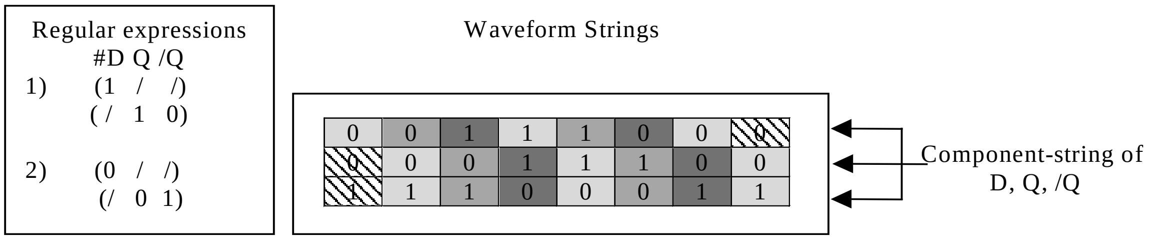

Figure 5: Example of regular expressions and transactions for D-f/f

As example, let us consider D-flip/flop and describe its work in terms of transactions. A transaction is defined by coded values of a signal D, which initiates a transaction and by coded values of signals Q and /Q produced as results of the transaction. Alphabet for description of the transaction includes normal characters "0" and "1" and a meta-character "/", which is a delimiter for a transaction. A vector of signal values on ports of f/f is considered as a character.

A regular expression for setting input D to 1 consists of two generalized vector-characters (1 / /) (/ 1 0)[1]. Similarly for setting input D to 0, the regular expression is (0 / /) (/ 0 1). With this definition in mind, we can identify seven transactions in the waveform in Fig. 5.
Also, 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 the strings.
Phase 1 consists of two steps.

1. Regular expressions are converted into DFAs [8].
2. Waveform strings are scanned by DFAs to identify the transaction tokens. This process is generally similar to the one described for compilers in [8]. A difference is in usage of meta-components in our implementation. If user-defined meta-components are used in regular expressions, the value for each of them is resolved in this step and stored internally.

For each regular expression, if the length of a waveform string is *l*, step 2 can be completed in $O(l^2)$ time. At the end of step 2, a set of transaction tokens is identified. Each token is characterized by the following items:

1. Position vectors for starting and ending positions in the waveform string.
2. An identifier for associated regular expressions pair.
3. A name-value table for user-defined meta-components.
4. A tag that will be specified later in the next phase of algorithm.

3.2 Labeling tokens (Phase 2)

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 each token in waveform *W1*, the tokens in waveform *W2* are scanned. 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.

Assuming that the number of tokens for two waveforms are *m* and *n* respectively, the time complexity of step 1 is $O(m*\lg(m))$ + $O(n*\lg(n))$. The step 2 has $O(n*m)$ time complexity.

Note that if two transaction tokens with the same identifier are associated with the same name-value table, it implies that transactions are instances not only of the same type, but also for the same data.
By the end of phase 2, two sequences of transactions TS1 and TS2 are formed. Each element in the corresponding token sequence is associated with one token in the token sequence formed by the previous phase for the input stream.

3.3. Finding minimum distance (Phase 3)

At this phase the algorithm produces quantitative measures of a distance between waveform strings for two models in the context of transaction tokens. It is assumed that the input sequences to these two models are equivalent at the transaction level, i.e., they are already adjusted to carry the same functionality with respect to differences in input ports requirements.

[1] Parentheses are not included in the alphabet and are introduced here for readability.

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. The algorithm for measuring distances is described and complexity is analyzed.

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 string distance measurement. The first one is the edit distance and the second is the block distance.
Edit distance is the minimal number of edit 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 [9][10]. The distance can be found by dynamic programming technique efficiently [11], the time complexity is $O(m*n)$, where *m* and *n* are the lengths of string *s1* and *s2* respectively.
Block distance is the minimal number of block operations that are needed to convert one string into the other. Each block operation consists of a series of consecutive edit operations of the same type. Block distance is defined for the minimal number of edit operations equal to edit distance. For example, given two strings "111" and "1110000", the edit distance is 4, and the block distance is 1. Since a small difference in the designs may result in consecutive mismatches in waveforms, block metric can interpret such situations more effectively. Computation of a block distance is integrated with that of edit distance in one pass without increasing the time complexity.
In this work, the edit and block distances are derived for each pair of common signals in two compared models of interest.

Distances in the context of transaction tokens

In this part of text, we describe how the above metrics are used in the context of transactionalized waveform strings.
As it was stated earlier, inputs to the two models are identical in terms of transactions. Assume that the complete transaction sequence during the 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 only describe the transactions which may produce differences in waveforms, both *TS1* and *TS2* are the 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 contain errors. Secondly, the models themselves may contain bugs. Therefore transactions will not be recognized. In order to measure the divergence between waveforms in such context, the following definition is introduced:

Definition: Mapping sequence
Assume that the tags of transaction token sequences *TS1* and *TS2* are $< x_1, x_{2,} ...x_m >$ and $< y_1, y_{2,} ...y_n >$ respectively. A mapping sequence is an indexed common subsequence of *TS1* and *TS2* $< m_1, m_{2,} ...m_k >$. 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 mapping sequence into m_i correspond to the same token in *TS*.
Fig. 6 gives examples of mapping 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 mapping 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*.

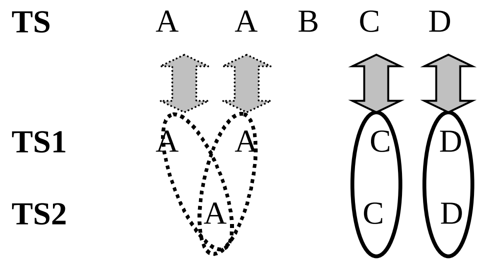

Fig. 6 Example of mapping sequences

If a given mapping sequence contains *X* tags, 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 a given segment is defined by the sum of these distances. Fig. 7 illustrates this ideology.

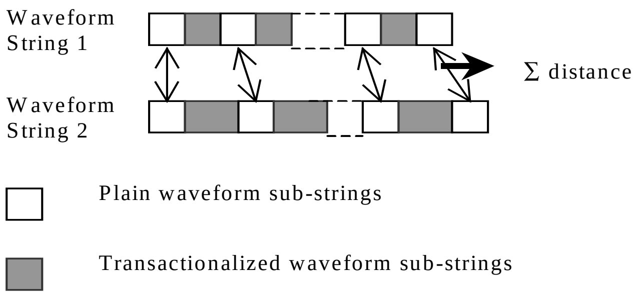
Waveform String 1
Waveform String 2
Σ distance
Plain waveform sub-strings
Transactionalized waveform sub-strings

Fig. 7 Distance for a given mapping sequence

After we eliminated from consideration the difference in waveforms produced by equivalent transactions, the remaining distance not captured by the regular expressions could be close to zero. To check whether this is the case, we still need to find minimal edit and block distances between the waveforms for the optimal mapping sequence.

Though the number of mapping sequences can be very large, we proved in the following text that the optimal mapping sequence can be found in polynomial time.

***Theorem*:** Given two waveform strings and the derived transaction token sequences. If the optimal mapping sequence is $< m_1, m_{2,} ...m_k >$, then $< m_{2,} m_3 ...m_k >$ is the optimal mapping sequence for the sub-string remaining after removal of transaction tokens associated with m_1.

Proof: The proof is by contradiction. If there exists another optimal mapping segment $< m'_{2,} m'_3 ...m'_l >$ that leads to the smaller distance for the above sub-strings, then the distance for $< m_1, m'_{2,} ...m'_l >$ is smaller. This contradicts our initial assumption.

The optimal substructure for mapping sequence is implied by this theorem. Therefore the optimal mapping sequence could be found by using dynamic programming technique.

The algorithm for finding minimum distance for optimal mapping sequence is described in the following text.

Algorithm for Phase 3 and complexity analysis

The algorithm for phase 3 consists of two major steps.

1) In the first step, one Longest Common Subsequence (LCS) is derived and used as a mapping sequence. The distance is computed under this mapping sequence. The algorithm for this step can be found in [11].

2) In the second step, the exploration of all possible mapping sequences is conducted by memoization techniques [11], which is a variation of dynamic programming. It is performed in a top-down manner. A table is constructed during the search process. This table records the solutions for sub-problems. Each entry in the table includes 3 items:

- the starting position *S1* in *W1*,
- the starting position *S2* in *W2*,
- the distance between the sub-string of *W1* starting at *S1* and the sub-string of *W2* starting at *S2*.

The distance for a given mapping sequence is the sum of distances between plain waveform sub-strings. When a mapping sequence is developed constructively, the distance increases with each additional element. This property can be utilized to speed up the process. Since we are interested to find the minimal distance, explorations during the recursion can be stopped each time when the partial distance for a mapping sequence is already larger than the current minimal distance. Initially the distance obtained in step 1 is used as current known minimal distance.

The detailed description of algorithm for step 2 is given below.

Algorithm: Find the minimal distances between two waveform strings.

Input: two waveform strings *W1, W2*,
starting positions for the two strings *S1, S2*
transaction token sequences *T1, T2,*
partial distance *PD,*
current minimal distance *CD.*

Output: the minimal distance under an optimal mapping sequence

```
BEGIN
1. Look in the table described above, if the distance is already registered, return the distance.
2. Find the token X in T1 whose starting position is at the smallest distance after S1.
```

```
3.  If the above step fails, compute the distance between sub-string of W1 after S1 and sub-string of W2 after S2, update the table and return.
4.  Find the tokens in T2 positioned after S2 with the same identifier as for token X and put them into a list L;
5.  If L = NIL, compute the distance between sub-string of W1 after S1 and sub-string of W2 after S2, update the table and return.
6.  For each token Y in L {
        Add a new tag Z into mapping sequence which associated X and Y;
        Compute the plain waveform sub-string distance D' when Z is added;
        If D' > CD, mark the linked entry in the table as VOID, remove the Z from the mapping sequence and continue the loop;
        PD = PD+D';
        S1 = X.end, S2= Y.end; and recursively call this algorithm to find the minimal distance RD for the remaining waveform sub-strings;
        Total distance D = PD + RD;
        If D < CD then CD = D;
        Remove Z from the mapping sequence;
        PD = PD-D';
        }
7.  S1=X.end; and recursively call this algorithm to find the RD';
8.  D = PD + RD';
    If D < CD then CD = D
9.  Register min(RD, RD') into the table and return it.
END
```

Assume that the number of transaction tokens is t_1 and t_2 respectively at the end of the phase 2, and the length of two waveform strings is *m* and *n* respectively.

In the first step, the LCS computation can be finished in $O(t_1 * t_2)$ time and the time complexity for the distance computed under the LCS mapping sequence can be bounded by $O(m*n)$.

In the second step, the table size for distance is at most $t_1 * t_2$. The computation complexity of computing distance between plain waveform strings is bounded by $O(m*n)$. In each iteration, one transaction may be associated with at most t_2 transaction tokens in a *TS2*, therefore the total time complexity is bounded from above by $O(t_1 * t_2^2 * m * n)$.

4. Experimental results

The algorithm for waveform analysis is implemented in C++ at SUN SPARC Ultra 5/10 workstation with 128 MB memory. The ModelSim [12] event-driven simulation tool had been used in experiments. The ModelSim has a facility to produce waveforms in a text form. The data in text format were coded into strings and compared for a string match and a sequence distance.

Case 1. PIC15C5X compatible microcontroller

In this experiment, we used the Free-RISC8 core available at the website [13] as a Model 1, and Model 2 is SILICORE [14] product. Both models are written in VHDL and are completely synthesizable.

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. Both models are binary compatible, though there are some divergences in implementations. 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. *GOTO* instruction requires 2 clock cycles in the first model. While such instruction is still being executed, the following instruction is fetched but will not be executed. For the second model, this transaction requires 3 clock cycles. It is obvious that difference in the number of clock cycles between two models for the same instruction will result in a difference between simulated waveforms and in a large edit distance if simple metrics for distance are used. On the other hand, if from the user's perspective difference in the number of clock cycles for GOTO instructions is not significant, a regular expression based analysis will show that the distance between "transactionalized" waveform for the models is reduced. Other multi_cycle instructions can be treated similarly.

Table 1 presents regular expressions for this instruction for both models.

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

Model 1 Inst ROM_addr O_ports	Model 2 Instr ROM_add 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 [2]*, 1001XXXXXXXX 1000XXXXXXXX} x;*

META_COMPONENT y, z;

In the first statement above, the subset of signal *Inst* values is defined for those multiple cycle instructions. 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.
In the experiment, the signals (instruction, ROM_ADDR, IO_ports) are compared. Each of the signals is a bit vector. For a testbench segment that contains 37 instructions, the following distances between waveforms are produced by the algorithm from simulation traces.

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

Signals	Edit/block Distance Without Transaction Recognition	Edit 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 [15], and the other is downloaded from [14]. The major difference is that the second design can operate in a pipeline mode. It is natural to consider encrypting of a plain text as a transaction. Table 3 provides a pair of regular expressions for two models.

Table 3: Regular expressions for encryption for two models

Model 1 Decrypt d_in key d_out	Model 2 Reset decrypt d_in key d_out
(0 x y -) { 15 }[3] (0 x y z)	(0 1 x / /) (0 1 / y /) (0 1 / / /) { 15 } (0 1 / / z)

90 test vectors were used to simulate both cores. Table 4 presents the distances 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
Din	1335/16	17/1
Key	1351/47	17/2
Dout	1351/90	17/1
Total	4037/153	51/4

Case 3: GCD

[2] In the binary op-code, the sequence 101 is a code for GOTO instruction, the following part is the address of the next instruction.

[3] {15} indicates this item repeats for 15 times.

In this experiment, two models that implemented Greatest Common Divider function were compared. One is a behavioral model from [17], and the other is an RTL model from [18]. The first model can compute the GCD in one clock cycle, while the other design load data first, and computes the GCD after several clock cycles. The number of clock cycles depends on the combinations of input data. In this case, the computation of GCD for a given pair of inputs is considered as a transaction, Table 5 provides the pair of regular expressions for these two models.

Table 5: Regular expressions for encryption for two models

Model 1 a b reset gcd	Model 2 Reset load a b gcd done
(x y 0 /) (/ / / z)	(0 0 - - - 1) (0 1 x y - 1) [(1 0 - - - 0)]+ (1 0 - - z 1)

26 test vectors are used to simulate both cores. Table 6 reports the distances with and without transaction recognition.

Table 6: Edit and block distances for signal waveforms with & without transaction recognition

Signals	Edit/block Distance Without Transaction Recognition	Edit/block Distance With Transaction Recognition
A	344/10	23/2
B	343/9	21/1
GCD	342/9	24/4
Total	1028/32	68/7

Case 4: CORDIC
In this example, two models for computing sin/cosine function were compared. One is using CORDIC algorithm from [14], the other is a behavioral model written in C. The first model presents iterative approximation algorithm, the other is using the standard function from the C library.

Table 7: Regular expressions for arithmetic models

Model 1 Theta sine cosine	Model 2 Theta sine cosine
(x - -)+ (x y z)	(x y z)

45 test vectors were used to simulate both cores. Table 8 summarizes the distances with and without transaction recognition. Note that in this case, the binary values are converted into real values before comparison. Two real values are considered to be equivalent if the difference between them is within the allowed range. For the different errors, the distance could be different.

Table 8: Edit and block distances for signal waveforms with & without transaction recognition

signals	Edit/block Distance Without Transaction Recognition			Edit/block Distance With Transaction Recognition		
	Error Range			Error Range		
	0.02	0.01	0.005	0.02	0.01	0.005
Theta	689/23	689/23	689/23	0/0	210/8	404/14
Sine	689/14	690/25	692/39	0/0	211/11	407/27
Cosine	689/8	689/11	690/19	0/0	210/6	405/12
Total	2067 /45	2068 /59	2071 /81	0/0	631/25	1216 /53

Case 5: DAC
In this experiment, two Digital Analog Converters were compared. One is available at [14], the other is described at [16]. Both of them are Delta-Sigma DAC but the algorithms are slightly different, however the waveforms at the outputs vary greatly. Since the converters are expected to generate the same voltage, the high frequency components in the output bit stream can be ignored. Based on this assumption, we can use the following table to capture a difference produced by two algorithms.

Table 9: Regular expressions for encryption for two models

Model 1 (Reset din dout)	Model 2 (reset din dout)
[(0 x 1) (0 x 0)]+	[(0 x 0)] +

Both models were simulated for 1000 clock cycles with the same input. Table 10 summarizes the distances with and without transaction recognition.

Table 10: Edit and block distances for signal waveforms with & without transaction recognition

Signals	Edit/block Distance Without Transaction Recognition	Edit/block Distance With Transaction Recognition
Din	0/0	1/1
Dout	500/490	1/1
Total	500/490	2/2

5. Conclusion

While simulation still remains a main approach to evaluation of suitability of IPs for target applications, it is very difficult and time consuming to compare and to interpret simulation waveforms from different models. Traditional simulation tools do not have capabilities to analyze such situations. The proposed techniques makes the analysis more informative and effective by raising level in waveform comparison from the signal event to the transaction level. This technique does not require modification of the source HDL code as in [3], nor writing some *ad hoc* code to monitor the simulation process as in [19]. Simulation of compared models can be conducted independently and analysis could be performed over simulation traces. The proposed technique allows not only to identify functionally equivalent segments of waveforms, but also quantitatively measures differences between segments of waveforms which may not be functionally-equivalent. The proposed method is based on the regular expression technique and equivalent Deterministic Finite Automata. It allows to transform waveforms into strings of tokens recognized by DFAs, to compare these strings, and to measure "distances" for this purpose. The methodology was implemented in C++ code and tested on several designs. The experimental results given in the paper show that some designs, which produced visually absolutely different waveforms functionally are very close to each other with respect to new metrics.
The methodology is flexible and allows designers introduce their own definitions of a "distance" used in comparison of waveforms.
The techniques can help to simplify and speed up a difficult task of IP selection and solve other problems where equivalence of models is an issue.

6. Acknowledgements

This work was supported in part by NSF grant MIP-9628022.

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] S.Y. Huang, K.T. Cheng, *"Formal Equivalence Checking and Design Debugging",* Kluwer Academic Publishers, 1998.

[3] C. Hansen, A. Kunzmann, W. Rosenstiel, "Verification by Simulation Comparison using Interface Synthesis", *Proceedings of DATE'98*.

[4] F. Corno, M. S. Reorda, G. Squillero, "Simulation-based Sequential Equivalence Checking of RTL VHDL" *ICECS'99: 6th IEEE Intl. Conf. On Electronics, Circuits, and Systems, 1999.*

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

[6] Novas homepage, http://www.novas.com, May 2000.

[7] Synopsys Behavior Compiler User Manual, 1999.

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

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

[10] G. Seni, V. Krip Asundar, and R.K. Srihari, "Generalizing edit distance for handwritten text recognition". *Proceeding of SPIE/IS&T Conference on Document Recognition,* San Jose, CA, 1995.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *"Introduction to Algorithms"*, McGraw-Hill, 1989.

[12] ModelSim EE/SE 5.3 User's Manual, 1999.

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

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

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

[16] Xilinx Delta-Sigma DAC, http://www.xilinx.com/xapp/xapp154.pdf

[17] GCD Behavioral model in VHDL, available at http://cs.uci.edu/pub/hlsynth/HLSynth92/gcd), Sept, 2001.

[18] GCD RTL model in VHDL, available at http://www.erc.msstate.edu/~reese/EE8993/verilog_vhdl.htm, Sept, 2001.

[19] P. J. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann Publishers, Inc. 1996.