TR 01-019

Functional Verification in IP Selection by Black-box Logic Simulation

Jian Liu and Eugene Shragowitz

April 09, 2001

Functional Verification in IP Selection by Black-box Logic Simulation

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

Abstract

Reuse of IPs is an important feature of contemporary SoC design. To select an IP for reuse, it is necessary to verify that the proposed design solution satisfies the specification formulated by the SoC designers. This paper presents a framework for verification of functional equivalence between the IP solution and the specification in the form of constraints or HDL model. The methodology is based on the joint simulation of an IP and the user model. The waveforms of interested output signals are coded into strings. A set of metrics for waveform comparisons is proposed to determine the equivalence of waveforms. This methodology can be applied to a variety of digital designs.

1. Introduction

Design reuse has become an indispensable part of the SoC designs. One of the major problems facing designers is whether or not a candidate solution “fits” the design specification given in the form of constraints or in the form of HDL behavioral models of desired components. The other possibility here is that the searched IP should replace a component that is already in use. Therefore there is a necessity to establish equivalence between two designs for approximately the same functionality, where the models of designs could represent different stages of the design process.

Formal verification techniques are not applicable to this situation since the solutions in questions are not devised one from another. A framework described in this paper is based on the black-box logic simulation followed by result analysis steps. There are numerous reasons why the models written for the same design specification may produce different simulation results. The incomplete list of causes includes:

1) Differences in the interpretation of specifications. For example, some standards like MPEG-4 are not hardware specific and allow flexibility in implementations.
2) Internal structure of solutions. This includes such characteristics as internal buffer sizes, precision of computations, timing, etc.
3) Different definitions of ports and data types. As pointed in [3], many digital systems are implemented using fixed-point architecture, while at algorithmic level, the design could be captured in floating-point models.
4) Differences in simulation platforms and tools may result in mismatches in results. For example, cycle-based simulation versus event-based simulation.
5) Bugs in models that escaped the tests.

Application of simulation-based equivalence checking requires solving several problems that were considered in this context. What is a definition of equivalence between two models describing a function on different levels? What kind of algorithms could be used to establish equivalence? How to organize environment to make such investigations possible in the context of IP selection?

This paper provides our answers to these questions. Section 2 presents a review of methods used for design verification. Section 3 contains a problem formulation and an overview of solutions. Section 4 describes the details of techniques of equivalence verification by simulation. Results of our experiments are given in Section 5. Section 6 presents the conclusions and directions of future works.

2. Related work

In contemporary VLSI design flow, formal methods and simulation/emulation are the major validation techniques. Recently, significant progress has been achieved in the area of formal verification techniques. An excellent survey of formal verification in hardware design is given in [1]. An E-catalog of the tools for formal verification could be found on Internet [2]. One branch of formal verification is concerned with the problem of functional equivalence, but mainly in the formulation related to the equivalence between the RTL model and its modification [9]. Any of formal methods applied for purpose of establishing equivalence between models requires detailed knowledge about internal structure of compared solutions.

On the other hand, simulation is still a major validation methodology for equivalence checking [7]. Some IP providers started to release their protected models for download. Internet-based simulation environments had been introduced by [8]. The potentially huge amount of communication across the Internet may prevent distributed simulation from practical use. Among other works addressing the verification of equivalence issue, [4] presents a case study of automatic consistency checking between the golden model and the target HDL model. Contents of internal registers are used to compare the states of two models during the simulation. [5] introduces simulation-based equivalence checking of a behavioral model and its synthesized version.

3. Simulation approach

Our approach to evaluation of equivalence between the two models for design reuse is based on the joint simulation of two models. The same test benches are applied to the inputs of both models and the output signal waveforms are saved and compared. If the waveforms are deemed equivalent, the compared models are regarded functionally equivalent. This is not such a trivial task as it seems at the first glance because while the waveforms may differ, the models still could be recognized as equivalent with respect to the system requirements. In this problem definition, it is not required that compared models are just modifications of each other. The models could be developed independently from the same specification and may represent different hierarchical levels of design, such as the RTL-gate level combination or the behavioral-RTL combination. Figure 1 illustrates the process of joint simulation and comparison.

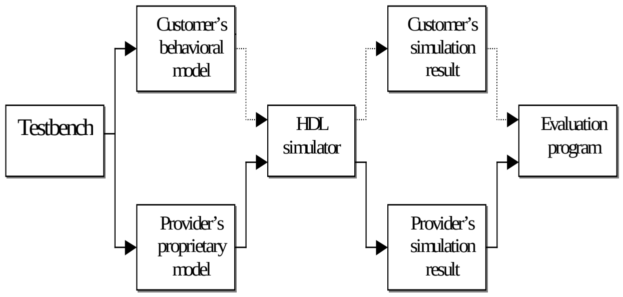
Testbench
Customer's behavioral model
Provider's proprietary model
HDL simulator
Customer's simulation result
Provider's simulation result
Evaluation program

Figure 1: Joint simulation and comparison

Typically in the IP selection process, the detailed information about the internal details of the marketed IPs is not available to the system designer. This is one reason why black-box based simulation is the major practical method to verify functional equivalence. Commercial simulation tools have a limited capability to compare waveforms [11]. They allow to recognize as similar only waveforms displaced in time with respect to each other. The commercial tools may identify waveform mismatch in simulations of one model and its optimized version, however such capability is not sufficient for the purpose of the IP selection.

4. Simulation results comparison

4.1 Overview

The process of waveform comparison is conducted according to a block diagram given by Fig. 2. First, the signal values at outputs of simulated models are converted into comparable formats. For example, signal values are presented in either floating point or in fixed-point format. A library is developed for conversion procedures between the commonly used data representations. Next, the waveforms are coded as waveform strings (see details later in this section). The matching algorithm is called to find if there is a mismatch between the string representations of simulation waveforms and a report to the designer is generated. As it will be defined in the text, equivalence of designs with respect to requirements may allow differences in waveforms. In the process of mismatch measurement, an algorithm conducts transformation of waveforms according to the user defined operations in an attempt to make them identical. If the transformation can be achieved under some constraints, the waveforms are deemed as equivalent. In this process, the user needs to provide information or options for format conversion, coding, metric for match and associated constraints.

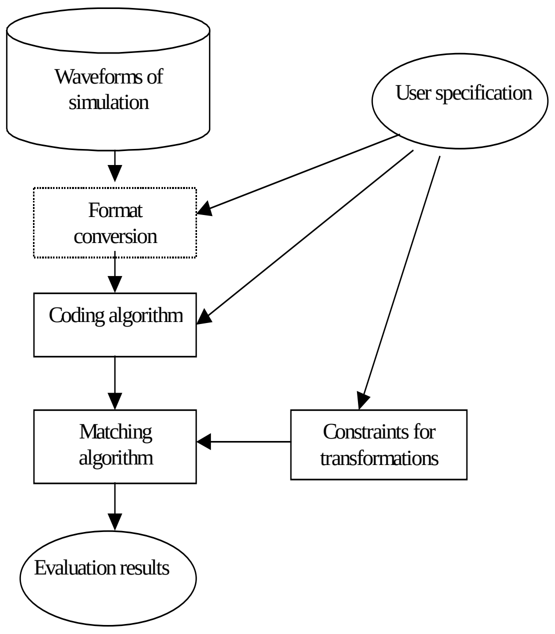

Figure 2: Workflow for simulation results comparison

4.2 Coding of simulation waveforms

Before the simulation results are processed using waveforms comparison, they should be coded first. In this work, a digital signal is allowed to take only binary values, either 0 or 1. The signal values are coded as distributed in time sequences of single bits
or as sequences of bit vectors. For each value of signal, time is provided, i.e., a set of pairs (time, value) completely describes a waveform.
The following coding schemes are applied:

- **Value stream**

The signal waveform is divided into fixed time intervals, producing a sequence of signal values at the beginning of the intervals.

- **Event stream**

Only transitions of signal values are considered. A sequence of alternating 0s and 1s values is produced when a waveform is coded according to this model.

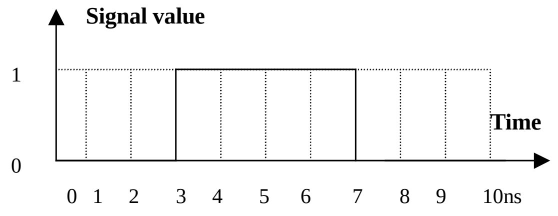

Figure 3: Signal waveform in value stream model

Fig. 6 depicts a signal waveform in a binary code. The signal starts at value ‘0’ at time 0, and jumps to ‘1’ at time 3 ns and falls back to ‘0’ at 7 ns. For the value stream model, with the time interval 1 ns, the following sequence, {0, 0, 0, 1, 1, 1, 1, 0, 0, 0} is generated. For the event stream model the code is {0,1,0}. Such coding mechanism can be extended to signals of bit vectors. Virtually several signals can be bounded together to form a composite signal for coding and comparison. For example, assume a design has two output: port1, port2, in the case when it is desirable to code them as one, a composite signal [port1, port2] can be defined.

4.3 Metrics for waveform comparison

After the waveforms are coded, for each pair of compared signals, the corresponding waveform strings are derived.

Source String = {a1, a2, …, am}

Target String = {b1, b2, …, bn}, where *ai, i=1,2…,m* and *bj, j=1,2,…n,* are *Tokens* that represent possible signal value: {0,1} or bit vectors.

Hereafter, waveform strings are used to denote the coded waveforms. As stated earlier, our goal to verify the functional equivalence by comparison of the output waveforms. Therefore the problem is now reduced to validate the equivalence of two waveform strings. In order to evaluate the match between the waveform strings, the framework uses three metrics: identical waveform strings, matching with errors and matching based on regular expression patterns. They are described one by one in the following text. In particular, regular expression patterns matching is described in a separate section 4.4.

A) Identical waveform string

Two signal waveforms S and T are called identical (*S=T*) if and only if the same transitions of logic values (0->1 or 1->0) take place in the same time.

B) Matching with errors

In many cases, two strings have some divergences. One can be derived from another (*S->T* or *T->S*) by sequence of operations:

- *Insertion(String, Position,Token):* This operation inserts *Token* into *String* at *Position*.
- *Deletion(String, Position)*: This operation eliminates Token at Position from String.
- *Substitution(String, Position, newToken):* This operation replaces the *Token* at *Position* in *string* with the *newToken.*

Each operation has an associated cost. This metrics is used to find the minimal cost to transform one string into the other. Similar work in software engineering can be found in [12]. In our work default, each operation has the same unit cost. Under this assumption, the minimal cost is equal to the number of operations used in the transformation. By adjusting the weights of costs for the operations, users can describe the preferences of the operations. For an instance, when the cost(substitution) > cost(Insertion)+cost(Deletion), the substitution is never used in the transformation. User supplies a predefined cost for the transformation. If the minimal cost is below the value, the waveforms are deemed equivalent.

C) Matching based on regular expression patterns

The above two metrics are used when the user has a little prior knowledge about the possible divergence of two implementations. Complementary to these two metrics, we also propose another new metric, which is based on pattern match. This metric is applied when the user knows the possible difference at the output between two designs. For example, in different implementations of a UART module, one model can send out bytes without any parity bits, while the other may append parity bits at the end of raw data. This can lead to some mismatch at outputs that could be tolerable with respect to the specification. The constraints about acceptable divergence in the term of patterns shall be supplied by users. Details of this approach are described in the following section.

4.4 Equivalence of waveforms based on patterns matching

To accommodate flexibility in implementation, a new equivalence metric based on patterns matching, is proposed. First let us define some basic terms.

- **Definition: Pattern**

Pattern is a sequence of tokens in the string. It is used to define some segments in signal waveform which have specific meaning, such as a transaction occurred at the output signals. Pattern can be represented in various forms, one of them is based on regular expression [13].

In the field of VLSI CAD, Interface synthesis based on regular expression has been proposed in [14]. Regular expression notations are used in this work to describe the equivalent patterns. Only some standard regular expression operators, including * for Kleene closure (0 or more of the reference tokens), + for semi-closure (1 or more), | for choice in our work. Regular expressions can be expressed hierarchically using regular grammar [15]. In our work, since only the divergence of output waveform is desired to describe by the regular expressions, recursion and multiple level definitions are not allowed for simplifying the implementation. The tokens in the

regular expressions are the possible values of signal waveforms. In addition to normal tokens, a special token DONTCARE is used as a wildcard that can be used to match any possible signal value.
Here we use the example given in [14] as an instance to illustrate the pattern of output waveforms. This example defines two equivalent protocol segments of communication interface: handshake and serial. The interface has two output ports. The finite automata describing the two protocols are illustrated by Figure 4, where the don't care is represented by '-'. Then the correspondent output patterns based on the finite automata protocols are
a) (0/-)*(1/a)+(0/b)+, and
b) (0/-)(1/a)(0/b)(0/-)* respectively.

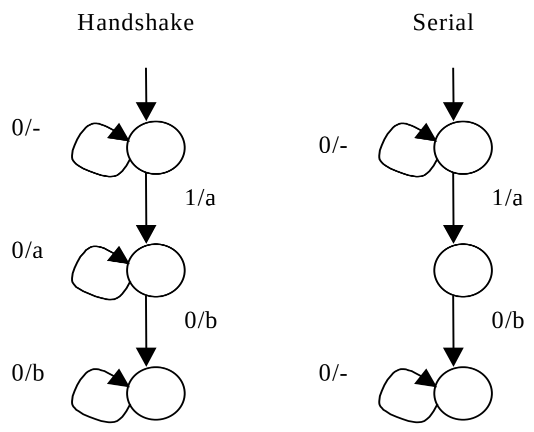

Figure 4: Finite automata of communication protocols

- **Definition: Equivalent patterns**

A equivalent pattern is defined as a pair of (A,B) where both A and B are patterns and A contained in the source string is equivalent to B contained in the target string. In the following text, A is called *original*, and B is called *replacement*. In the above example, (a,b) are deems as equivalent patterns in that they reflect the equivalent transaction under different protocols.

A user is expected to supply a set of equivalent patterns. It is possible that one *original* may have multiple *replacements*. Also, the user may define a set of equivalent patterns. The waveform comparison program then attempts to match two waveforms by replacing some segments of waveforms by their *replacements* defined in equivalent counterpart.

- **Definition: Equivalent waveforms**

Two signal waveforms *S* and *T* are called equivalent ($S \approx T$) if one string can be transformed into another by replacement some segments in the string with their equivalent part.

In pseudo code, the major steps of the above algorithm are outlined as below.

***Algorithm**: Match two strings using equivalent patterns*
***Input**: Source string, Target string, Equivalent Patterns*
***Output**: Return "yes" if the Source string can be translated into the Target by replacing some segments by the equivalents. Otherwise, return "no".*
1) *Construct DFAs for the originals in the equivalent patterns.*
2) *Scan the Source string and identify all the subsequences that are originals of equivalent patterns.*
3) *Attempt to transform the Source string to the Target string by substituting the originals identified in the previous step with one of their equivalents. If successful return yes, else return no.*

At the step 1, the regular expression is first transformed into the NFA (Non-deterministic Finite Automata), next the NFA is transformed into the DFA (Deterministic Finite Automata).

At the step 2, whether or not a string can be recognized as an instance of regular expression can be checked in linear time by traversing the DFA. A check can be completed in polynomial time to locate all the segments, which are instances of the originals.
Assume that after the step 2, we found M *originals* in the source stream. As stated earlier, every original may have multiple correspondent replacement. Each original can either be substituted by one of possible replacements or stay unchanged when we attempt to transform the source string into the target string. In the worst case where there is no overlap among these original patterns, the number of the total combinations is $\prod_{i=1}^{m} (X_i + 1)$, when X_i is the number of candidate replacements for *original i.*
At the step 3, it is not necessary to go through all the combinations to find a possible match. Instead of that, A tree exploration can be used to improve the search efficiency. The exploration of a tree for a matching process is conducted in a depth-first manner.
During the traverse of the generated tree, some branches can be pruned. For example, if one original being replaced by its equivalent immediately makes two strings not match, the replacement for remaining originals will not be considered.

Algorithm: *determine whether Source string can be transformed into the Target string with equivalent pattern substitution*
Input: *Source string, Target string, equivalent patterns, originals identified in the Source string*
Output: *return "yes" if the transformation is successful, otherwise return "no".*

```
1)  Initially result = "no";
2)  For the first original in the source, repeat
    {  substitute it with i-th replacement.
        If the substitution leads to immediate mismatch between Source and Target, set result to no and break;
        If there is no other original in the Source, break.
        Recursively call this algorithm to determine whether the tail of source string can be transformed into the
    tail of Target string.
        If the above call returns "yes", set result to "yes",  break;
        }
        until all possible replacements are tested.
3)  return result..
```

4.5 Overall algorithm and its complexity

The user should specify either the equivalent pattern or maximal cost allowed in the string match with error. Optionally, the user can select the coding mode and set the cost for each operation for the string match metric.
The pseudo code of overall algorithm for waveform comparison is described as following:

Algorithm: *determine whether two signal waveforms are equivalent.*
Input: *Source waveform, Target waveform, coding mode, metric of equivalence.*
Output: *return yes if the waveforms are deem equivalent, otherwise return no.*

```
1)   Code the waveforms into the strings in either the value stream or the event stream mode according to the
     coding mode.
2)   Check if the strings are identical. If they are identical, return yes.
3.1) CASE 1: If the user selects string match metric, the two strings are compared using string match algorithm,
     report the minimal cost.  If the cost is less than predefined cost, return yes, otherwise return no.
3.2) CASE 2: If user selects equivalent pattern metric, the two strings are compared using the equivalent pattern
     metrics. If they are equivalent, return yes, otherwise return no.
```

Step 1 requires in O(n) time if the value stream is used, where n is the simulation time divided by the fixed time interval. Otherwise it can be done in O(nlogn), where n is the total number of transitions occurred at the signals, if the event stream mode is used since it requires to sorts all the transition of signal values in ascending order of time.
Step 2 can be finished in O(n) time, where n is the length of the strings.

Step 3.1 can be solved using dynamical programming techniques [16] in O(n*m) time, where m and n are the length of two strings respectively.
In the step 3.2, the worst case complexity of the algorithm is exponential in the number of symbols in the regular expression. However, in the most practical cases, such situation is unlikely to encounter.

5. Experimental results

Experiments were conducted for different types of circuits, such as memory models, arithmetic circuits, encryption engine, micro-controllers and others. In the following text, a comparison of two PIC16C5x compatible micro-controllers is described as an example. In this experiment, Free-RISC8 core available at the website [20] is used as a reference model, and the IP product of SILICORE [19] is used as the candidate model. Both models are written in VHDL and are completely synthesizable. The test benches given for the reference model were used to evaluate the IP from SILICORE.
The ModelSim [17] event-driven simulation tool has been used in experiments to simulate both models. The ModelSim has a facility to present waveforms in a numeric form. The numeric data were coded into value-stream and event stream and compared for stream match and stream distance by the algorithms described in the text. In this example, the waveforms of signals at data bus, ROM/RAM address, and ports were compared. A composite signal waveform is formed as [DATA_BUS, ADDRESS, PORT]. Extra clock cycles are allowed in the simulation of waveforms for instructions that may need one or more clock cycle to complete. The correspondent equivalent patterns can be defined using regular expression as
([*INS*, DONTCARE, DONTCARE], [*INS*, DONTCARE DONTCARE][DONCARE DONTCARE DONTCARE]*), where the *INS* is the tokens for some instruction codes such as *branch* which may need 2 or more clock cycle to execute.
The results of equivalence evaluation are given in Table 1.

Table 1: Equivalence evaluation of the SLC1655 vs reference model

	Functions under test	**Result**
Test bench 1	Increment / Decrement	Pass
Test bench 2	Add / Subtract	Pass
Test bench 3	Rotate	Pass
Test bench 4	Timer	Conditional pass (*)
Test bench 5	Logic instruction	Pass
Test bench 6	Subroutine	Fail
Test bench 7	Register file	Not applicable
Test bench 8	Port r/w	Pass

The comparison results fall into these four categories:

- *Pass* means that either the waveforms are equivalent based on the pattern matching.
- *Fail* means that the waveforms are not equivalent, implying that the required capabilities present in the reference models are absent in the IP. In this particular case, design A does not implement 2-level stacks present in the reference model. The waveforms at the address bus cannot be matched.
- *Conditional Pass* means that values of compared signals depend on the internal implementation. Acceptability is not certain. In this example, the waveform is not equivalent based on pattern matching. But the mismatch is small.
- *Not applicable* means the comparison cannot be conducted due to certain reasons, such as unmatched interfaces (i.e., there are no corresponding signals for those in the reference model).

6. Conclusion and future work

This paper presents a framework for logic simulation based functional verification of IP solutions. Simulated IP designs are viewed as black boxes and no prior knowledge about the internal structure of a candidate IP is assumed. The major contribution of this work is a framework for experimental verification of equivalence of models by joint logic simulation. The framework includes definitions of a set of metrics, the associated constraints and algorithms for comparison of coded signal waveforms. In particular a regular expression pattern match is

proposed for evaluation of the output waveforms. This kind of methodology can be applied to a wide range of digital designs.
Some extensions are left for future work. Other constructs may be included in the regular expression definition to make this approach practically useful for real world design comparison. Also other metrics may be introduced later in the framework to support more types of digital designs.

Acknowledgements

This work is supported in part by the NSF Award No. MIP9628022.

References

[1] C. Kern and M. R. Greenstreet, "Formal Verification in Hardware Design: A Survey", *ACM Transactions on Design Automation of Electronic Systems, Vol. 4, No.2*, April 1999, pages 123-193.

[2] E-catalog of EDA tools: http://ecat.ibsystems.com/EDA_Design_Tools/, Oct 2000.

[3] H. Keding, M. Willems, et al, "FRIDGE: A Fixed-Point Design and Simulation Environment", *Proceedings of Design, Automation & Test in Europe Conference,* 1998.

[4] Y. S. Chang, S. Lee, et al. "Verification of a Microprocessor Using Real World Applications", *Proceedings of Design Automation Conference*, 1999.

[5] C. Hansen, A. Kunzmann, W. Rosenstiel, "Verification by Simulation Comparison Using Interface Synthesis", *Proceedings of Design, Automation & Test in Europe Conference,* 1998.

[6] *Synopsys Behavioral Compiler User Manual,* 1999.

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

[8] A. Fin, F. Fummi, "A Web-CAD Methodology for IP-Core Analysis and Simulation", *Proceedings of Design Automation Conference,* 2000.

[9] C. A. J. Van Eijk, "Sequential Equivalence Checking Based on Structural Similarities", *IEEE transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 19, No. 7,* July 2000.

[10] S. G. Govindaraju and D. L. Dill, "Verification by Approximate Forward and Backward Reachability," *Proceedings of ICCAD,* 1998.

[11] Cadence Affirma signal waveform viewer: http://www.cadence.com/datasheets/affirma_sig_wav_view.html, Oct 2000.

[12] J. E. Cook, A. L. Wolf, "Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model". ACM Transactions on Software Engineering and Methodology, Vol.8, No.2, April 1999, Pages 147-176.

[13] F. Lustman, "Specifying Transaction-based Information Systems with Regular", IEEE Transactions on Software Engineering, Vol 20, No. 3, March 1994, pages, 207-217.

[14] R. Passerone, J. A. Rowson, A. S. Vincentelli, "Automatic Synthesis of Interfaces between Incompatible Protocols", proceedings of DAC 1998.

[15] A. V. Aho, R. Sethi, J. D. Ullman, "Compilers: Principles, Techniques, and Tools", Addison-Wesley Publisher, 1988.

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

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

[18] P. J. Ashenden, *"The Designer's Guide to VHDL"*, Morgan Kanfmann Publishers Inc. 1996.

[19] Homepage of Silicore. http://www.silicore.net, May, 2000.

[20] Free-RISC8 core at http://www.free-ip.com/risc8/index.html, May, 2000.