MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

KHARKOV NATIONAL UNIVERSITY OF RADIOELECTRONICS

ISBN 966-659-113-8

Proceedings of IEEE East-West Design & Test Workshop (EWDTW'06)

Sochi, Russia, September 15 – 19, 2006

CONTENTS

Dynamic Register Transfer Level Queues Model for High-Performance Evaluation of the Linear Temporal Constraints

Vladimir Hahanov[1], **Oleg Zaharchenko**[1], **Sergiy Zaychenko**[2]
[1] *Kharkiv National University of Radioelectronics, Design Automation Department, Kharkiv, UKRAINE*
[2] *Aldec Inc., Aldec-Kharkov division, Kharkiv, UKRAINE*
E-mails: hahanov@kture.kharkov.ua, sergeiz@aldec.com.pl

Abstract

Today the Assertions Based Verification (ABV) is by all means the most effective verification technology for SoC designs. Assertions provide basic blocks for building functional verification concept. Assertions simply catch a lot of design errors on early phases. This paper suggests new effective algorithmic model for assertions checking within the testbench-based simulation. The algorithms for handling key temporal operators from Property Specification Language (PSL) are described. Paper demonstrates the advantages of the suggested model over existing equivalents - in simulation performance, verification efficiency and model extensibility.

Keywords: *functional verification, assertions, linear temporal logic, system-on-chip, PSL, simulation.*

1. Introduction

Obviously, the verification process is a very complex and a very expensive part of the modern SoC design cycle. This process consists of searching the model for mistakes, causing the design to violate the functional specification, localizing the problems reasons and applying the fixtures. According to the EDA industry experts opinion, the cost of verification in ASIC [1] designs often overheads the 70% of the entire project budget [2]. Such high cost of the system quality is driven by several factors, in particular:

- A large amount of missed details and mistakes in the work of SoC designers in the RTL code, verification engineers mistakes in the testbenches, also, the inevitable ambiguities of the original design specification;
- drawbacks in the choosen design flows, complicating the bugs localization and fixtures, missing the possibilities for early discovery of the typical problems;
- relatively low performance and bugs within the selected automation tools, which reach the quality and performance goals much slower than the input design complexity raises.

Resolving these problems altogether and degrading the SoC verification cycle cost is currently a primary goal for the entire EDA world [3]. Leading EDA companies and industry experts are focused on developing the new generation of complex design verification methods, which will be able to:

- minimize the human participation in the routine design and verification procedures, which will obviously decrease the probability of mistakes in several times;
- lead to catching the largest amount of problems on the early design phases, reducing the average fixture cost;
- upgrade the performance and stability of the design verification systems by raising the abstraction level both for the SoC models and for the testing stimulus.

There are two basic directions in modern SoC verification methods – dynamic methods [2,4], based on the simulation, and static, or formal methods [5,6], based on the mathematical proof of certain system properties without testing stimulus. There are also hybrid methods [7] used, which assume usage of the simulation and functional coverage results to improve the performance of formal methods. This work is focused on the assertions-based verification technology [8,9], playing its role both in dynamic and formal methods.

Assertions declare system behavior in the terms of temporal properties. Properties define the desired and forbidden event sequences, reflect the specification requirements, check the internal data and protocols integrity. The expressiveness of the temporal semantics provides effective ways to describe the most complex processes and protocols within the system being

verified. Equivalent pin-to-pin style testbenches are simply several times larger than the compact properties descriptions. The most popular properties specification languages are PSL [10], System Verilog [11] and OVA [12].

Formal properties checking is a fundamental mechanism in the static verification methodology. Assuming that the input properties represent the specification with maximum possible precision, the formal methods allow to check the system quality with 100% functional coverage and 100% guarantee without providing any input testing stimulus. However, realistic industry-level usage of the formal property checking is limited to the functional verification of only certain design blocks. Applying formal methods to larger blocks is simply not practical, as these advanced methods acquire too heavy computation resources [6,7].

This work focuses on using the temporal properties with the circuit simulation with testing stimulus [13,14]. Due the expressiveness of the temporal logic and the ability to access any internal system signals at all abstraction levels, simulation with assertions provides an effective strategy for detection and diagnosis of the errors and regressions during all design cycle phases. Typically preparing the assertions basing on the functional specification has at least 10 times smaller engineering cost, than developing the testbenches covering the same amount of described functionality. Also the assertion monitors are able to easily localize the source of problems deeply in the design by displaying the exact hierarchical paths, where the specification violation is detected.

The ground for the assertions-based verification solutions existing on the market is modeling the temporal constraints with the finite state machines (FSMs). Some of the implementation models are based on special decision diagrams [15-17]. The primary problem of all FSM-based simulation tools is a non-linear structural complexity, a state explosion, relatively to the temporal formula size. Also, these implementations are often limited only to a simple languages subsets [9], as they are not able to implement the evaluation of few important temporal constructs with the acceptable computation complexity.

The *goal* of this paper is to develop a new algorithmic model for checking the temporal assertions during the SoC simulation, combining the effective problems diagnosis, high properties evaluation performance and the largest assertions description languages constructs coverage.

Main *research tasks* include:

- definition of the suggested assertions checking model;
- description of the algorithms for handling key temporal operators on the base of the suggested model;
- comparison with existing equivalents by means of the verification efficiency and computation performance.

2. DRTLQ Model Definition

The Dynamic Register Transfer Level Queues (DRTLQ) model is oritented on high-performance evaluation of the linear temporal logic and the efficient bugs diagnosis. Model can be logically divided on 4 interacting levels, corresponding to 3 well-known assertions-based verification levels [8]:

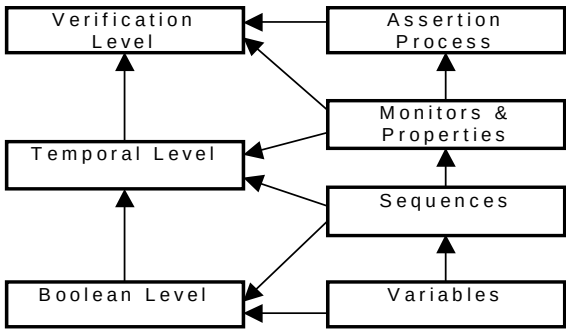

Figure 1. DRLTQ model levels in the context of ABV levels

The top-level object in the DRTQ model is an assertion process – an aggregate of boolean, temporal and verification level assertions semantics, implementing the verification of the temporal declarations group, describing certain functional block. PSL verification unit (vunit) construct is similar to the DRTLQ assertion process. Formally the assertion process can be defined in the following way:

Definition 1. Assertion process AP in the DRTLQ model is an aggregate of verification variables B, currently handled events E, sequential functions F, temporal properties P and assertion monitors M, checking the functional correctness of a certain functional block or communication protocol:

$$AP = \{B, E, F, P, M\}. \quad (1)$$

The atomic object in DRTLQ is a verification variable, which generalizes the immediately countable expressions.

Definition 2. Verification variable $b \in B$ in the DRTLQ model is a design object, which current value or state is used as an operand of the temporal relations in this assertion process, which is convertible to the Boolean data type.

The set of all possible variable values corresponds to a well-known conception – a set of atomic propositions [6]. The set of all variable states is an

alphabet for more complex temporal relations, and a set of atomic propositions is a set of all possible input testing stimulus.

Many design objects can be represented in the DRLTQ model with the verification variable – a constant (parameter), scalar or vector signal, memory word, variable of the built-in type, and also complex objects – expressions, function calls, class objects. Unlike the existing assertion checking models, DRTLQ completely isolates from the details of concrete data types, description language or verification environment. Providing the conversion of the object value to the Boolean type, and also the notification about value changes is a responsibility of the surrounding simulation system. Such abstraction layer results in several important bonuses:

- abstraction from data type speeds up the analysis, as only the simulation environment, explicitly defining the value storage formats and conversion algorithms, can convert the value to Boolean type with the optimal performance;
- resigning from non-Boolean values (X and Z) also speeds up the analysis, as any expression, requiring to match the 4-valued objects finally results in a Boolean value;
- abstraction from data types makes the DRTLQ universal in the relation to the data domain; this idea is used for verification of the high-level SoC models, in particular, with the SystemC technology, where the transactor-object state can be used as an atomic verification variable [18].

The verification variables represent other assertions description constructs, which do not have a direct representation in the design modeling languages, f. ex:

- an abstract conception of a permanent truth and lie (internal Boolean constants, used implicitly for modeling more complex temporal operators);
- sequence endpoints;
- local assertion variables (System Verilog only).

The computations procedures for assertions evaluation are performed only when the value of one or more verification variables changes. A clock verification variable is a special variable, which edge initiates the assertion analysis iteration. In general case, the assertion process can contain several clock variables, which serves the needs of complex ASIC designs with multiple clock domains. However, in the majority of cases only a single clock variable is used in the concrete functional block. If there are no clocks defined, the computations are triggered by a change in any variable in the temporal relations.

During activation the verification variables initiate events.

Definition 3. DRTLQ event is a relation (2)

$$e = \{v_e, x_e, t_{be}, t_{ae}\} \quad (2)$$

where v_e is a current Boolean event value, x_e – an optional event extension, depending on the sequential function, which handles the event, t_{be} and t_{ae} – a creation and activation times.

The event is called activated, if it has reached the assertion process element, creating an activation thread.

Definition 4. An activation thread in DRTLQ is a set of all events, corresponding to a single computation path π.

Computation path π is a selected simulation session interval, having a beginning and ending times [6]. The ending time can be infinite, in such case the analysis of the path ends with simulation finish.

Event can be connected with the other events in 2 rings: right ring corresponds to the events from the same computation path, while the left ring – to the events from different computation paths, which are simultaneously transported through a certain DRTLQ element. Right ring is unordered, while the left ring is ordered by the event creation time, which is necessary for pipelined handling of activation threads.

Assertion process never has more than one activation thread with the certain beginning time. If few activation threads simultaneously exist in the process, they all correspond to the computation paths with different beginning times. The event cannot belong more than to a single activation thread. The activated event always belongs to a thread. Inactive event does not belong to any thread. Creation and activation event times are typically equal, except the case of sequential implication operator. In some cases the event might not be activated, if the handling path is auxiliary. Event activation is performed by the first element, which satisfaction allows to proceed with the path analysis. Usually, the activating elements are the first elements on the alternative paths in temporal expressions.

As all the events within the single activation thread are interconnected, the satisfaction of the thread immediately destroys all the ring of connected events no matter how deeply within the model they are located in that moment. The condition for thread satisfaction is determined with the verification goal. If the goal is to forbid a certain behavior, the first event with value 1 reaching the checkpoint resolves the thread. Only in the case when all events on the computation path finish with 0 value, the forbidding declaration is satisfied. On the other hand, if the verification goal is to prove the existence of some behavior, the conditions for thread satisfaction are strictly the opposite.

The role of those checkpoints is served with high-level model objects – assertion monitors and temporal properties.

Definition 5. The temporal property $p \in P$ in the DRTLQ model is a group of logic and temporal relations between Boolean expressions, sequential functions and other properties, describing a certain desired or forbidden system behavior.

The temporal properties are used to assert the behavior during the whole simulation cycle involving many computation paths. Basing on the results from lower temporal layers, the properties are able to check very high level declarations, like:

- existence of a certain events sequence on a selected computation path;
- a proof of a certain property on all the computation paths, on one of the future paths, a stable state within the work of another property;
- combinations of the properties with conjunction, disjunction, implication and equivalence operators.

Definition 6. The assertion monitor $m \in M$ in the DRTLQ model is top-level analysis object for a temporal definition in the assertion process, which manages the verification result propagation to the parent testbench.

Assertion monitor directly communicates with the testbench. The goal of the assertion monitor is to provide reaction on the verification result. In the simplest case such reaction can be expressed with a console simulator message, or visually demonstrated in the waveform viewer and other debugging instruments. There are other more complicated reactions possible, depending on the language and simulation environment used for verification.

The events sequences, which presence or absence is proved by the temporal properties, are always defined with the superposition of the sequential functions.

Definition 7. The sequential function $f \in F$ in the DRTLQ model is a certain computation procedure, having the sets of input, internal and output events, performing their conversion and propagation step by step on each verification clock cycle.

Looking from the single event processing point of view, sequential function is a finite state machine The event appears on the function input. Depending on the computation procedure details and the external conditions, the event is transported through the internal states to the final output state, after that it can be used by the next model elements.

The most important feature of the sequential functions is an ability for pipeline-based processing of multiple events, corresponding to different activation threads with intersecting computation paths. That's why the state of the concrete event processing within the sequential function is determined either by a storage place, or with an event extension, containing the function-specific information about the processing status. As the sequential functions can be used as the input arguments to other functions, the extensions might support nesting.

The most frequently used DRTLQ function is a queue. Queues are allocated for evaluating the time shifts between the interested events. Other frequently used sequential functions are sequence repetitions, logic operators (sequential disjunction, sequential conjunction, sequence intersection), sequence competition within the completion of another sequence, and sequential implications (an analysis of a certain events sequence after preliminary successful completion of another sequence).

The described temporal declarations are checked during the simulation with testing stimulus. Testing stimulus W are given as a sequence of words $w_{i,}$, each of them corresponds to a single clock cycle and contains a set of values for each verification variable in the process:

$$w_i = \begin{Bmatrix} b_1 \leftarrow \{0,1\} \\ b_2 \leftarrow \{0,1\} \\ \dots \\ b_n \leftarrow \{0,1\} \end{Bmatrix} \quad (3)$$

where w_i is an input stimulus vector for a clock cycle number i, n is a number of variables in the assertions process.

Block (4) demonstrates compact mathematical descriptions about the temporal properties analysis results on the given stimulus. Temporal property is said to be hold on the W (4a), if it is satisfied and all the initiated activation threads have finished. If the hold property will keep this state on any extension of the computation path, it is said to be hold tightly (4b). If there were no failed computation paths within the simulation, but there are some activation threads still under processing by the simulation finish, the property is said to be pending (4c). If there was at least one failed computation path, the property is said to be failed (4d). Finally, if W does not initiate any activation thread, the property is not activated (4e):

$$\begin{aligned} &a)\ W_1 \models p_1 \\ &b)\ W_2 \models\!\!\models p_2 \\ &c)\ W_3 \models\!\!\approx p_3 \\ &d)\ W_4 \models\!\!\neq p_4 \\ &e)\ W_5 \models\!\!? p_5 \end{aligned} \quad (4)$$

As average temporal property describes the design behavior across the several clock cycles, it is often possible, that there are unfinished activation threads

remaining in the model by the end of simulation. By the reaction principle on the unfinished threads, the properties are divided on neutral, strong (+) and weak (-). Neutral properties do not react on unfinished threads, while strong properties interpret unfinished thread as failed, and weak properties consider it as passed:

$$\begin{aligned} &a) \quad W_1 \mid\approx p_1^+ \Leftrightarrow W_1 \mid\neq p_1 \\ &b) \quad W_2 \mid\approx p_2^- \Leftrightarrow W_1 \mid= p_2 \end{aligned} \tag{5}$$

Basing on the given DRTLQ model definition it is possible to construct high-performance evaluation algorithms.

3. DRTLQ-based processing algorithms for key linear temporal logic operators

FSM-based model is able to implement any temporal operator. The FSM-based model can be synthesized into a diagnostic hardware to create the SoC with advanced built-in self-testing mechanisms. However, the assertions analysis based on the FSM model demonstrates serious performance flaws when applied to industry level ASIC designs:

- FSM-based model has a non-linear structural complexity: if the temporal formula complexity increases or parameters change, the amount of states and transitions grows exponentially;
- FSM-based model is not well-suited for the parallel analysis of the computation paths. Each computation path in the FSM-based model has to be evaluated separately. Let's assume, that the first paths do not satisfy the property after the complete analysis of all tests vectors, but the later path immediately satisfies the property. In such case, the performance resources spent on analysis of the unsuccessful paths are used senselessly;
- FSM-based model does not handle the logic sequential operators with the alternatives well. It requires either a multi-iteration processing cycle, or the superposition of constraints during the FSM generation from original formulas. Multi-iteration analysis limits the abilities of the fast analysis termination when the result is known earlier, while the constraints superposition complicates the FSM structure and its generation algorithms;
- Finally, the FSM-based representation of certain temporal operators, especially the temporal properties, covering several computation paths simultaneously, requires too large computation resources to be effective for the practical simulation.

Alternative models, f. ex. ordered binary decision diagrams [17] and tag-augmented sequences [16] solve some aspects within the listed problems. But the DRTLQ model resolves all the listed performance problems. Besides, this performance advantages do not limit the language support coverage:

- the structural complexity of the DRTLQ model is linear relatively to the formula complexity. Parameter values do not have any impact on the number of model elements and interconnections between them.
- Event storage and propagation principles and the computation procedures of the sequential functions provide a possibility for pipelined handling of several activation threads. It often happens, that the activation thread, which started later than the others, finalizes the processing of the earlier threads.
- The nested events extensions define effective principles for handling the operators with alternative paths. First event, satisfying the temporal operator on the given computation path, is able to immediately destroy all the ring of alternative events and to finalize the no longer interested handling paths no matter how deeply they are in the model.

Of course, the DRTLQ model has its own limitations. This model is oriented on software-based assertions evaluation under the control of the HDL-simulator. The possibilities for hardware synthesis for self-testing circuits production or hardware embedded simulation are very limited. This limitation is connected with the a for dynamic memory allocation and with the nontrivial computation procedures, which are very hard to be implemented in the programmable hardware.

Let's describe in details the processing algorithms for the most frequently used assertion operators – the time intervals and the repetitions. Assuming that we need to check the PSL sequence {a;[*1:3]; b}, which means waiting for the event b=1 from 2 to 4 clock cycles after the event a=1, would produce the following FSM-based checking model:

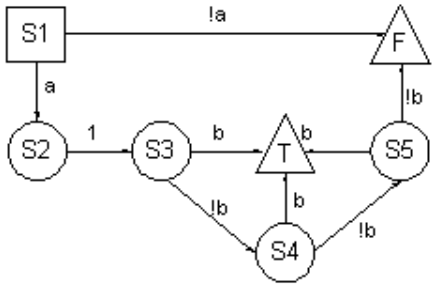

Figure 3. Checking time intervals with FSMs

Functionally, this checker is compatible with the formula we analyze. However, a terrible drawback of

this approach is a state number explosion when the parameters of the time interval change. Increasing left interval value on 1 adds a new state and a new transition to the model, while increasing the right interval adds 1 new state and 2 new transitions. Obviously, with large values of the interval parameters the FSM-based model is large and inefficient.

Implementing the same formula with DRTLQ models will result in the following data structures:

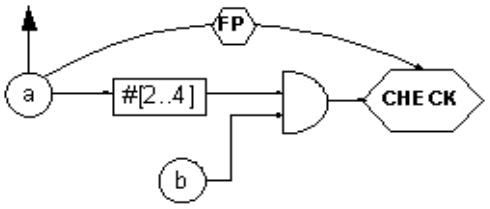

Figure 4. Time interalvs in DRTLQ model

The circles represent verification variables, the rectangle is a queue, modeling the time shift, AND gate implements sequence matching logic, a large hexagon is an assertion monitor, the small hexagon is a direct connection between the verification variable and the assertion monitor, allowing a fast termination of the analysis thread, if a = 0. The queue, implementing the time shift, is isolated from handling the event b = 1, which is implemented with the AND-gate. Such processing flow preserves the constant structural complexity without dependency on the actual parameter values.

The performance of the queues processing is determined with the internal events storage organization. The events are stored in the ordered single-linked list. The first event is the one, which was registered in the queue first. The queue reacts on 2 queries: the clock signal and the data fetch query from the next element. The reaction on the clock signal is to register the new event from the input at the end of the queue. The request of the next element extracts all the events, which are ready for further processing. If the pending time has reached the maximum parameter, the ring is propagated to the output. If the minimal pending time has already passed, but the maximum time is not yet reached, the event is copied and this copy is propagated to the output. If is is known, that the next element will destroy the event (f.ex, b = 0 for the PSL definition shown above), the copying might be omitted. If the minimal pending time was not yet reached, the event stays within the queue.

The experiments have shown a significant performance advantage of the DRTLQ model in comparison with the FSMs. Fig 5a shows the measurements for small intervals (from 1 to 10 cycles), and fig. 5B – for large (from 50 to 100 cycles):

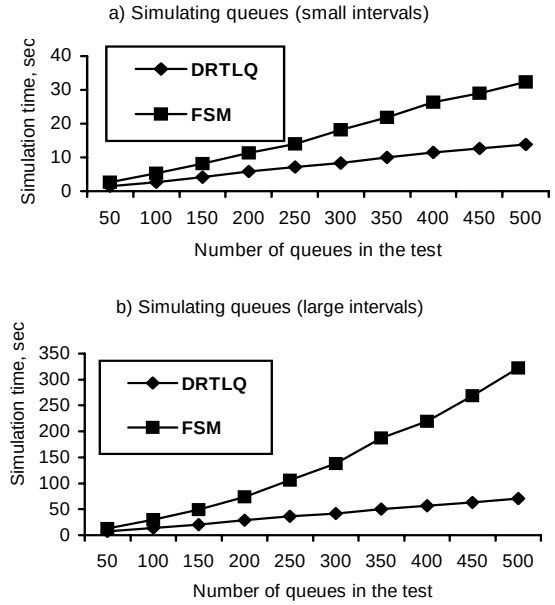

Figure 5. Performance comparison of the queues evaluation between the DRTLQ and FSM-based models

Even better are the performance results of the DRLTQ model on the repetition operators. Let's consider the checking of the {a;b[*3:∞]; c} PSL sequence. Formally the semantics of this formula is represented with the following relation:

$$\left\{\begin{array}{l} w_0 \models a \\ w_{1,2,3} \models b \\ \exists i = 1..\infty, w_{3+i} \models c, w_{3+1..(i-1)} \models b \end{array}\right\} \Rightarrow (W \models p) \quad (6)$$

The FSM-based checking model will look like (Fig. 6):

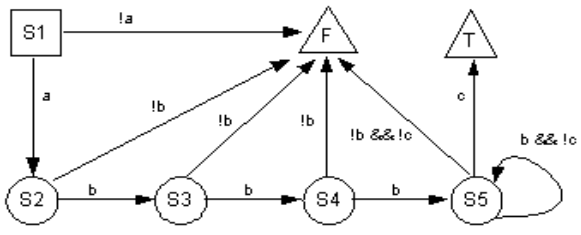

Figure 6. Checking (6) with FSM-based model

Similarly to the queues, this repetitions implementation has a nonlinear structural complexity: the number of model elements depends on the repetition parameters.

The DRTLQ model implementation will look like:

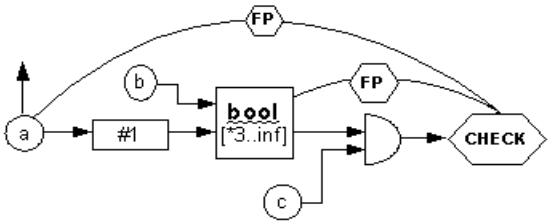

Figure 7. Checking (6) with DRTLQ model

Despite that this representation does not look compact from the first view, the model will keep this structure when parameter values are changed, while FSM-based model will grow. Similarly to the queues, the repetitions handling is isolated in DRTLQ from the next element. Processing is invoked only after a query. Depending on the variable c value, the repetition algorithm will make the optimal decision. Additionally to the checks of the registration time, repetitions check the event b=1. If b=0, all the events within the repetition are propagated to the monitor via the direct connection element as failed. The implementation of the goto-repetitions from PSL language differs only with the additional internal ring on the b-input, making a copy of the last successful event on the each clock cycle, until the new event will appear. Non-consequent repetitions additionally have the buffered output ring. The copies of the output events are created until one of the generated events will not resolve all the computation path. The repetitions processing speed shown below. The fig. 8A shows the performance difference with small repetition parameters, while 8b – with the larger values.

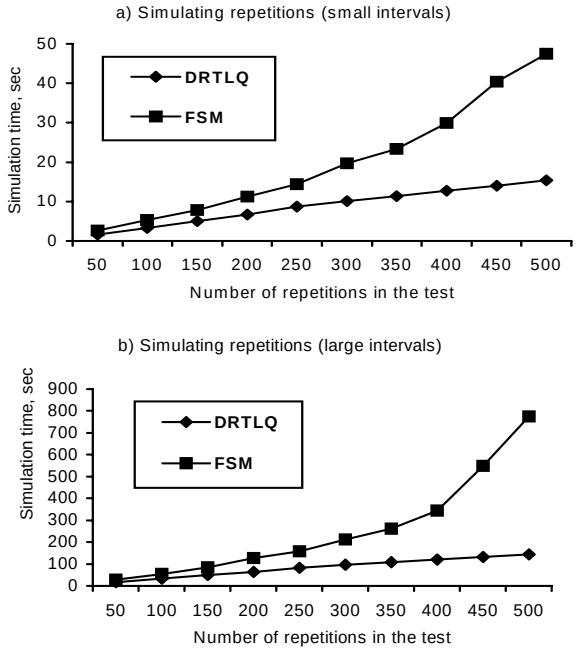

Figure 8. Performance comparison of the repetitions evaluation between the DRTLQ and FSM-based models

Also, a very important role in the DRTLQ model performance belongs to the resolving algorithms of the logic sequence operators with multiple alternatives. The combination of such typical sequential operators, like conjunction, disjunction, intersection, generate really large amount of analysis alternatives. The FSM-based model for checking such relatively simple PSL formula {{a;b} || {c;d}} && {!e[*2]} will be very complex (Fig. 9).

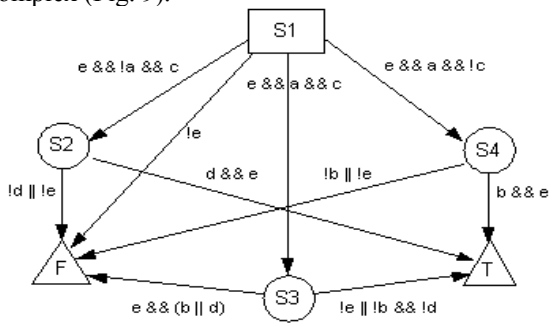

Figure 9. Handling sequential logic operators with FSMs

Despite that the handling here is parallel, the combination of logic operators complicates the transition expressions. Also, here all processing paths must be represented explicitly.

DRTLQ model representation for the same formula combines the performance and linear structural complexity:

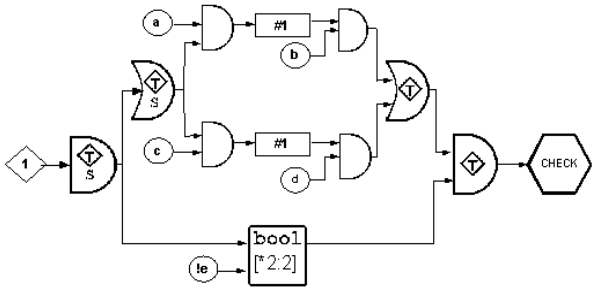

Figure 10. Handling sequential logic operators with DRTLQ

It is obvious from the picture, that the logic operators are implemented with the pair of elements: a split and a join. If an event goes through the split-element, it enters the scope of the sequential function, and the corresponding extension is attached. The extension contains a special context and status information. If the evaluation result is known in the middle of the path between the split and join elements, the extension is used for immediate transporting to the join element. Depending on the event value and the operator logic, the join element might kill the remaining alternatives or might wait until they will finish. The join element detaches the extension, attached in the split element. If the operators are nested, only the last extension is detached, so the

further last extension will correspond to the next level operator.

So, the combination of structural minimalism and parallel alternative threads handling make the DRLTQ model a high-performance assertions evaluation engine for the most frequently used verification constructs.

4. Conclusions

This paper presented a new algorithmic DRTLQ model for temporal assertions checking within the SoC simulation cycle with the testing stimulus, which demonstrates an effective detection and localization of the design problems, high-performance computations and maximum possible coverage the assertions description languages.

The most important scientific and practical results include:

- the definition of the DRLTQ model, oriented on solving the SoC verification efficiency problems;
- the detailed algorithms description for key temporal logic operators, including the most frequently used constructs like queues, repetitions, sequential functions;
- the comparison between suggested model and existing equivalents, which demonstrated a significant advantage of the DRTLQ model over classic FSM model by means of the computation performance;
- the linear complexity between the model size and it's performance relatively to the analyzed formula complexity was shown on the theoretical conclusions and experimental results; linear complexity is definitely much more effective than the exponential complexity of the classic FSM models.

Practical implementation of the suggested model is integrated to the Riviera verification environment from one of the EDA industry leading companies – Aldec Inc. The practical role of this research is in the dramatic decrease of the verification cost in the SoC design cycles.

5. References

[1] Smith M.J.S "Application-Specific Integrated Circuits", VLSI Systems Series, 1997, 1040p.

[2] Bergeron J. "Writing testbenches: functional verification of HDL models", Springer, 2003, 512p.

[3] Fazzari S. "The role of Verification IP in a Complex core Design", Cadence Design and Reuse electronic magazine, 2005, http://www.us.design-reuse.com/articles/article7493.html

[4] Bening L., Foster H. "Principles of Verifiable RTL Design", Kluwer Academic Publishers, 2001, 281p.

[5] Drechsler R. "Advanced Formal Verification", Kluwer Academic Publishers, 2004, 277p.

[6] Clarke E.M.Jr, Grubmerg O., Peled D.A. "Model Checking", The MIT Process, 2002, 314p.

[7] Bormann J., Fedeli A., Frank R., Winkellman K. "Combined Static and Dynamic Verification", Research report, PROSYD forum, 2005.

[8] Foster H., Krolnik A., Lacey D., "Assertions-based Design", Kluwer Academic Publishers, 2003, 392p.

[9] Yeung P. "The Four Pillars of Assertions-Based Verification", Mentor Graphics, EuroDesignCon 2004, 21p.

[10] IEEE-1850 "IEEE Standard for Property Specification Language (PSL)", 2005, 143p.

[11] IEEE-1800 "IEEE Standard for System Verilog Language", 2005, 586p.

[12] "OpenVera Language Reference Manual: Assertions", version 1.4.1, Synopsys Inc., 2004, 136p.

[13] Rashinkar P., Paterson P., Singh L. "System-on-chip Verification: Methodology and Techniques", Kluwer Academic Publishers, 2002, 393p.

[14] Meyer A.S. "Principles of Functional Verification", Elsevier Science, 2004, 206p.

[15] Canfield W., Emerson E.A., Saha A. "Checking formal specifications under simulations", International Conference on Computer Design, 1997, pp. 455-460.

[16] Chang K.-H., Tu W.-T., Yeh. Y.-J., and Kuo S.-Y. "A tag-augmented temporal logic checker"

[17] Große D., Drechsler R. “CheckSyC: An Efficient Property Checker for RTL SystemC Designs”, IEEE International Symposium on Circuits and Systems, 2005, pp. 4167-4170.

[18] Forczek M., Zaychenko S. "Assertions-Based Verification for SystemC", East-West Design & Test Workshop, 2005, pp. 54-61.

AUTHORS INDEX