PSL Assertion Checking with Temporally Extended High-Level Decision Diagrams

Maksim Jenihhin, Jaan Raik, Anton Chepurov, Raimund Ubar

Department of Computer Engineering, Tallinn University of Technology
E-mail: { maksim|jaan|raiub }@pld.ttu.ee, anton.chepurov@gmail.com

***Abstract* - The paper proposes a novel method for PSL language assertions conversion to a system representation model called High-Level Decision Diagrams (HLDD). Previous works have shown that HLDDs are an efficient model for simulation and convenient for diagnosis and debug. We present a technique, where checking of PSL assertions is integrated into fast HLDD-based simulation. There are three main contributions in the paper. The first one is a methodology for direct conversion of PSL properties to HLDD. The second one is a temporal extension for the existing HLDD model. The third one is HLDD-based simulator modification to support the new type of HLDDs and assertions checking.**

Index Terms - ***dynamic verification, assertions, property specification language (PSL), high-level decision diagrams***

1. INTRODUCTION AND MOTIVATION

Verification has become a very important phase in the state-of-the-art digital systems development process. As it was estimated in International Technology Roadmap for Semiconductors report [1], verification takes roughly 70% of design time, and consequently demands lots of costly resources such as man-hours or CPU-hours making this part of complete system development often the most expensive phase. According to [1], the problem is caused by a pair of recent processes: firstly, rapid design complexity increase and secondly, the historically greater emphasis on other aspects of the design process what has produced enormous progress (automated tools for logic synthesis, place-and-route, and test pattern generation, etc.), leaving verification as the bottleneck.

Among the recently proposed solutions the Assertion-based Verification (ABV) is one of the most promising. Verification assertions can be used in both dynamic and static verification. This paper considers only the first case, when assertions play role of monitors for particular system behaviour during the simulation. Property Specification Language (PSL) is a recently accepted IEEE standard language [14] that is commonly used to express the assertions.

The research on topic of conversion of PSL assertions to design representation such as HDL is gaining its popularity. There are several approaches published in recent time [2, 3, 4, 19]. The most widely known tool for this task is FoCs by IBM [5].

Our first attempt [18] of PSL properties translation to HLDD was implying the generation of VHDL checkers by IBM's FoCs as an intermediate step. However this experience has revealed particular limitations and inefficiency for HLDD-based assertions creation. Moreover, checkers synthesis from PSL properties are efficient mainly for the case where checkers are to be used in hardware emulation. The application of the same checker constructs for simulation in software may lack efficiency due to target language concurrency and poor means for temporal expressions. Synthesis of checkers hardware for emulation is out of the scope of current paper.

In this paper, we present an approach to checking PSL assertions using High-Level Decision Diagrams (HLDD). Here, the assertions are translated to HLDD graphs and integrated into fast HLDD-based simulation. The structure of HLDD design representation with the temporal extension proposed in this paper allows straightforward and lossless translation of PSL properties. HLDDs are a convenient model for diagnosis and debug since they provide for easy identification of cause-effect relationships.

High-Level Decision Diagrams have been proposed and further developed by the authors in Tallinn University of Technology (TUT) [17]. For more than a decade this model of digital design representation has been successfully applied for design simulation and test generation research areas. Participation of TUT in recently launched European Commission research project VERTIGO [16] has encouraged HLDD usage also in verification.

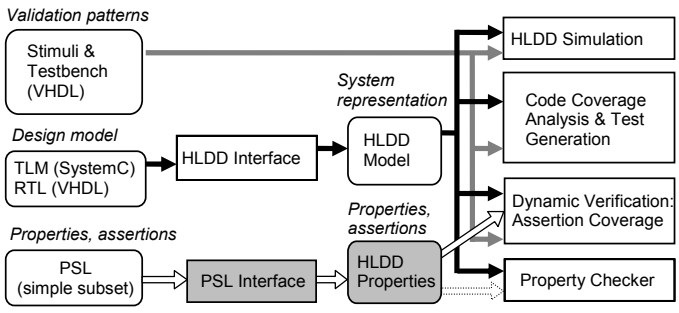

Figure 1. HLDD-based verification tools flowchart.

The topic of the VERTIGO project is embedded systems verification and validation. It is also aimed to bridge the gap between system level modelling and verification performed at the transaction level and the traditional RTL (register transfer level) description. Tallinn University of Technology as a partner of the project contributes by developing HLDD based verification tools and making research in cooperation with the other partners in the areas of static (formal), dynamic (simulation-based) and mixed static-dynamic verification.

The HLDD-based verification flow proposed by TUT is shown in Figure 1. The main emphasis of this paper is put on assertion checking in simulation-based verification (this part is highlighted by grey colour). However, several other tools working with HLDD are under development, including dynamic verification code coverage analysis, formal methods of stimuli generation and model checking. The latter are relying on the engine of HLDD based ATPG known as DECIDER [8].

The paper is organized as follows. Assertion-based verification and PSL are discussed in Section 2. Section 3 defines the existing HLDD graph model and introduces the temporal extension to it. Section 4 presents the methodology for HLDD graphs creation from PSL specifications. The discussion of the existing HLDD-based simulator modification to support the temporal extension of HLDDs and assertion checking in simulation is provided in Section 5. Finally, Section 6 concludes the paper.

2. Assertion-based verification and PSL

Assertion-based Verification can be classified as Design-for-Verifiability (DFV) technique. The goal is to assist both formal methods and simulation-based verification and allow discovering Design under Verification (DUV) misbehaviour (causing an assertion violation) earlier and more effective. Another important advantage of ABV is its aid to debug process.

In case of dynamic verification assertions provide better *observability* on the design what allows detecting bugs earlier and closer to their origin. At the same time in case of static verification with model checking, the assertions increase the *controllability* of the design and direct verification to the area of interest. Each assertion violation discovered by model checking is reported as a counter-example.

The question of the origin of assertions can be formulated as a separate topic for research itself. An important aspect here is the problem of *completeness*. Usually assertions do not describe all the possible properties of design what would mean translation of a complete design specification to a formal assertion description language such as PSL (Property Specification Language) or SVA (System Verilog Assertions). Instead of this only design areas of concern, sometimes referred as *verification hot spots*, are targeted. In practice they are often provided by design engineer and require deep knowledge of the DUV behaviour.

Assertion-based verification popularity has encouraged a common *Property Specification Language* development by the Functional Verification Technical Committee of Accellera. After a process in which donations from a number of sources were evaluated, the Sugar language from IBM was chosen as the basis for PSL. The latest Language Reference Manual for PSL version 1.1 was released in 2004 [13]. The language became an IEEE 1850 Standard in 2005 [14].

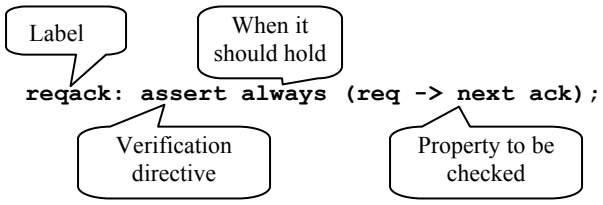

Figure 2. PSL property "reqack"

An example PSL property *reqack* structure is shown in Figure 2. Its Timing diagram is also illustrated by Figure 3a. It states that `ack` must become high next after `req` being high. A system behaviour that activates *reqack* property however obviously violating it is demonstrated in Figure 3b. Figure 3c shows the case when the property was not activated.

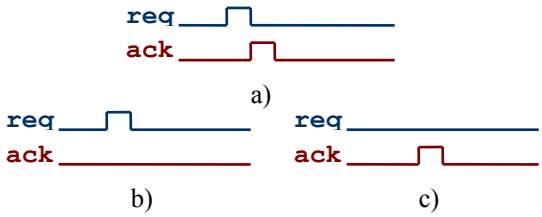

Figure 3. Timing diagrams for the property "reqack"

For the convenience of verification engineers PSL is a multi-flavoured language, which means that it supports common constructs of VHDL, Verilog, IBM's GDL, SystemVerilog and SystemC [15]. PSL is also a multi-layered language [13]. The layers include:

- *Boolean layer* – the lowest one, consists of boolean expressions in HLD (e.g. `a &&(b || c)`)
- *Temporal later* – sequences of boolean expressions over multiple clock cycles, also supports Sequential Extended Regular Expressions (SERE) (e.g. `{A[*3];B} |-> {C}`)
- *Verification layer* - it provides directives that tell a verification tool what to do with specified sequences and properties.
- *Modelling layer* - additional helper code to model auxiliary combinational signals, state machines etc. that are not part of the actual design but are required to express the property.

The temporal layer of PSL language has two constituents:

- Foundation Language (FL), that is Linear Temporal Logic (LTL) with embedded SERE
- Optional Branching Extension (OBE), that is Computational Tree Logic (CTL)

The second one considers multiple execution paths and models design behaviour as execution trees. CTL can only be used in formal verification. Therefore this part of PSL is left

$G_y=(M,E,X,D)$,
$M=\{m_1, m_2, m_3, m_4, m_5\}$;
$E=\{e_1, e_2, e_3, e_4, e_5\}$, $e_1=(m_1, m_2)$, $e_2=(m_1, m_4)$,
$e_3=(m_1, m_5)$, $e_4=(m_2, m_3)$, $e_5=(m_2, m_4)$;
$X(m_1)=X(m_5)=(x_2,\{0,1,2,\ldots,7\})$, $X(m_2)=(x_3,\{0,1,2,3\})$,
$X(m_3)=(x_4,\ldots)$, $X(m_4)=(x_1,\ldots)$;
$D(e_1)=\{0\}$, $D(e_2)=\{1,2,3\}$, $D(e_3)=\{4,5,6,7\}$,
$D(e_4)=\{2\}$, $D(e_5)=\{0,1,3\}$.

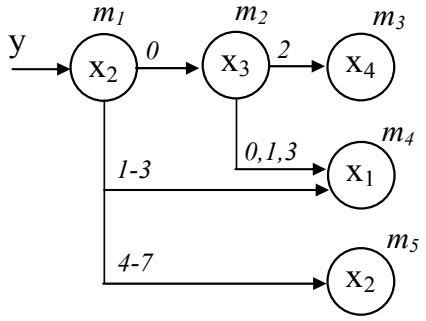

Figure 4. A HLDD for a function $y=f(x_1,x_2,x_3,x_4)$

for future work related to HLDD-based model checking implementation (the dashed arrow in Figure 1). This paper we will consider only FL part of PSL. However, a subset of FL is applicable for translation to HLDD assertions.

Our initial goal, and also the VERTIGO project requirement, is to support FL subset known as PSL Simple Subset. This subset is gaining its popularity and is supported by many verification and simulation tools. It is explicitly defined in [13] and loosely speaking it has two requirements for time: to advance monotonically and be finite and restrictions on types of operands for several operators.

3. High-Level Decision Diagrams

Decision Diagrams (DD) have been used in verification for about two decades. Reduced Ordered Binary Decision Diagrams (BDD) [9] as canonical forms of Boolean functions have their application in equivalence checking and in symbolic model checking. Recently, a higher abstraction level DD representation, called Assignment Decision Diagrams (ADD) [10], have been successfully applied to, both, register-transfer level (RTL) verification and test [11, 12].

The main issue with the BDDs and assignment decision diagrams is the fact that they allow logic or RTL modeling, respectively. In this paper we consider a different decision diagram representation, High-Level Decision Diagrams (HLDD) that, unlike ADDs can be viewed as a generalization of BDD. HLDDs can be used for representing different abstraction levels from RTL to TLM (Transaction Level Modeling) and behavioral. HLDDs have proven to be an efficient model for simulation and diagnosis since they provide for a fast evaluation by graph traversal and for easy identification of cause-effect relationships [6, 7].

3.1. HLDD data structure

Definition: A HLDD representing a discrete function y=f(x) is a directed non-cyclic labeled graph that can be defined as a quadruple G=(M,E,X,D), where M is a finite set of vertices (referred to as *nodes*), E is a finite set of *edges*, X is a function which defines the *variables labeling the nodes* and the variable domains, and D is a function on E. The function $X(m_i)$ returns a pair (x_i,X_i), where x_i is the variable letter, which is labeling node m_i and X_i is the domain of x_i. Each node of a HLDD is labeled by a variable. In special cases, nodes can be labeled by constants or algebraic expressions. An edge $e\in E$ of a HLDD is an ordered pair $e=(m_1,m_2)\in E^2$, where E^2 is the set of all the possible ordered pairs in set E. D is a function on E representing the activating conditions of the edges for the simulating procedures. The value of D(e) is a subset of X_i, where $e=(m_i,m_j)$ and $X(m_i)=(x_i,X_i)$. It is required that $Pm_i=\{D(e) \mid e=(m_i,m_j)\in E\}$ is a partition of the set X_i. HLDD has only one starting node (*root node*), for which there are no preceding nodes. The nodes, for which successor nodes are missing, are referred to as *terminal nodes*.

Figure 4 presents an example of a graphical interpretation of a HLDD.

```
IF reset = 1 THEN state:=s0;
ELSE
  CASE state
  WHEN s0:
     a:=in1; b:=in2; ready:=0;
     state:=s1;
  WHEN s1:
     IF a≠b THEN state:=s2;
     ELSE state:=s5; ENDIF;
  WHEN s2:
     IF a>b THEN state:=s3;
     ELSE state:=s4; ENDIF;
  WHEN s3:
     a:=a-b; state:=s1;
  WHEN s4:
     b:=b-a; state:=s1;
  WHEN s5:
     ready:=1;
     state:=s5;
  END CASE
END IF
```

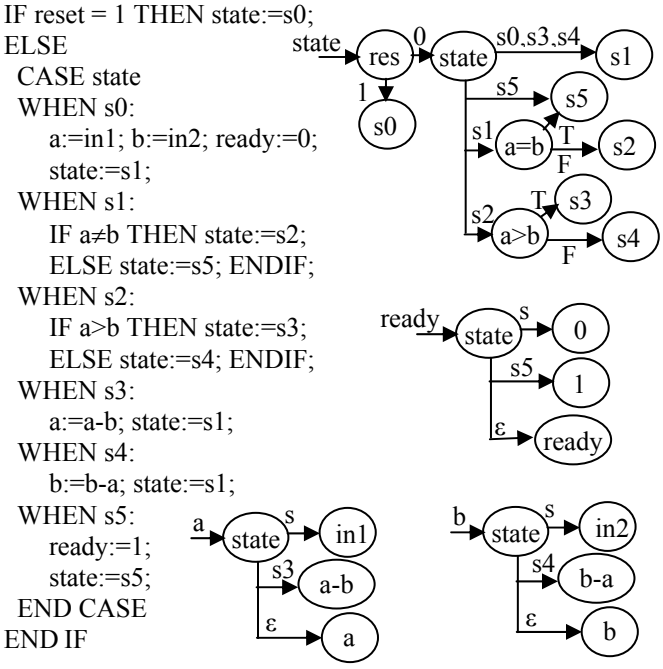

Figure 5. a) RTL VHDL and b) the corresponding HLDD

3.2. HLDDs for digital systems

HLDD models can be used for representing digital systems. In such models, the non-terminal nodes correspond to conditions or to control signals, and the terminal nodes represent data operations (functional units). Register transfers

and constant assignments are treated as special cases of operations. When representing systems by decision diagram models, in general case, a network of HLDDs rather than a single HLDD is required. During the simulation in HLDD systems, the values of some variables labeling the nodes of a HLDD are calculated by other HLDDs of the system.

Fig. 5b presents the HLDD system for the RTL VHDL code shown in Fig. 5a implementing the greatest common divisor algorithm. In the Figure, T and F stand for true and false, respectively. The ε character denotes default edges.

4. T-HLDD Assertions Construction

The idea of the proposed method relies on the principle of 'divide and conquer'. The method is based on partitioning PSL properties into elementary entities containing only one operator. There are two main stages in the approach. The first one is preparatory and consists of *Primitive Property Graphs Library* creation for elementary operators. The second stage is recursive *hierarchical construction* of the Temporally extended HLDD (T-HLDD) for a complex property using the PPG Library elements.

4.1. PPG Library creation

Prior to the T-HLDD construction procedure a *Primitive Property Graph* (PPG) should be created for every PSL operator supported by the proposed approach. All the created PPGs are combined into one *PPG Library*. The library is extensible and should be created only once. It implicitly determines the supported PSL subset. The method currently supports only weak versions of PSL operators. However, by means of the supported operators a large set of properties expressed in PSL can be derived.

Primitive Property Graph is a special type of HLDD graph. Compared to the basic HLDD model used for representing the design (defined in Section 3), these graphs have two distinctions. The first distinction is the requirement for all the PPGs to have a standard interface. The second distinction is usage of the HLDD model with a temporal extension. In the following the distinctions will be discussed in detail.

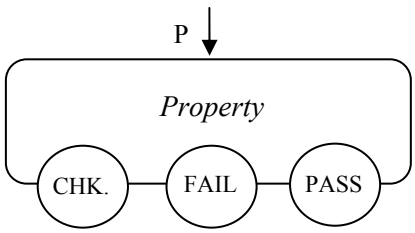

Figure 6. Standard PPG interface

The *standard interface* for all PPGs was introduced in order to support the hierarchy in a recursive complex property construction described in the next subsection. PPG has one root node and exactly 3 terminal nodes (CHECKING, FAIL and PASS, respectively), as opposed to an arbitrary number of terminal nodes in usual HLDD graph. The standard PPG interface is shown in Figure 6.

The terminal nodes in PPG have the following meaning:

- FAIL – assertion *P* has been simulated and does not hold;
- PASS – the assertion has been simulated and holds;
- CHECKING – *P* has been simulated and it does not fail, nor does it pass non-vacuously

In this paper we support only weak operators. In order to extend the subset to support strong operators a third output PENDING would be needed. Its addition would influence the proposed modification of the Simulator explained in Section 5. Example PPGs created for 3 PSL operators are shown in Figure 7. Note, that the logic implication operator '->' in Fig. 7b exits to the terminal node 'CHECKING' when the precondition P_a fails. This is due to the fact that in assertion checking the designer is not interested in non-vacuous passes of the property. Also, consider the PPG for operator 'until' shown in Fig. 7c. It is the SERE and 'until' operators that create cyclic THLDDs. The proposed assertion checking procedure is capable of handling such cycles.

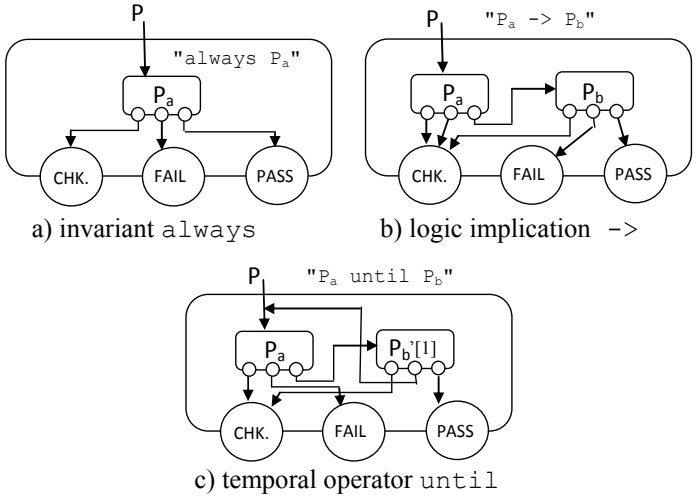

Figure 7. PPGs for a set of PSL operators

One of the main motivations for PSL introduction was poor ability of standard HDL languages to express temporal relations between expressions in assertions. The main instruments for this purpose used in PSL are repetition operators of its own and of Sequentially Extended Regular Expressions (SERE). A powerful part of the repetition operators are their auxiliary suffixes (e.g for next* family they are next_*a*, next_*e!*, next_*event* etc). In current paper we propose a temporal extension for HLDD model that supports the following 3 PSL constructs (See Table 1).

TABLE 1. TEMPORAL EXTENSION FOR HLDD

PSL construct	Explanation	Equivalent HLDD extension for a *Variable*
next[n]	property holds at time step *n*	*Variable'[n]*
next_a[j:k]	property holds at all time steps within *j* to *k* range	*Variable'[j..k]_a*
next_e[j:k]	property holds at least once within *j* to *k* time steps range	*Variable'[j..k]_e*

Additionally we introduce the notion of *END* as a special case of value of *k* in the expression *Variable'[j..k]_sfx* (where *_sfx* is one of *_a* or *_e*). The maximum bound of the time steps sequence may take value END if it is not explicitly determined. The time point END occurs at the end of simulation and implicitly determined by:

- Number of test vectors
- The amount of time provided for simulation
- Simulation interruption

The main purpose of the proposed temporal extension is transferring additional information and directives to the HLDD Simulator that will check assertions. Let us refer to the HLDD graphs with the described extension as *Temporally Extended HLDDs* (T-HLDD).

4.2. Recursive hierarchical construction of properties

Complex properties are hierarchically constructed from elementary graphs in PPGs Library in the following way. At first, the property should be parsed. During the parsing phase the PSL property is partitioned into entities containing one operator only. The hierarchy of operators is determined by the PSL operators precedence specified by IEEE1850 Standard. Hierarchical construction is performed in the top-down manner. It starts for the operators with lowest precedence where the sub-operations are then recursively substituted with the operators having higher precedence. For example, *always* and *never* operators have the lowest level of precedence and consequently their corresponding PPGs have the highest level in the hierarchy. The sub-properties (operands) are step-by-step substituted by lower level PPGs until the lowest level where sub-properties are pure signals or HLD operations.

Let us consider an example PSL property for GCD implementation given in Figure 5.

P1: assert always((! ready) and (a=b) ->next_e[1:3](ready))

The resulting T-HLDD graph describing this property is shown in Figure 8.

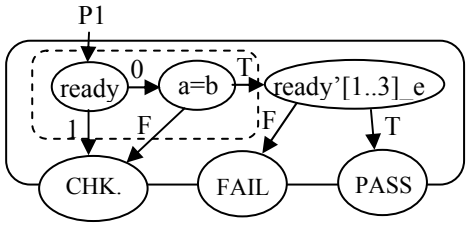

Fig. 8. T-HLDD for property P1

5. HLDD-BASED SIMULATOR EXTENSION

The basis for assertion coverage analysis in this paper is a simulator engine relying on HLDD models. In our earlier works [7], we have implemented an algorithm supporting, both, Register-Transfer Level (RTL) and behavioral design abstraction levels. This algorithm is briefly explained below and it will be used for simulating the system model.

In the RTL style, the algorithm takes the previous time step value of variable x_j labeling a node m_i if x_j represents a clocked variable in the corresponding HDL. Otherwise, the present value of x_j will be used. In the case of behavioral HDL coding style HLDDs are generated and ranked in a specific order to ensure causality. For variables x_j labeling HLDD nodes the previous time step value is used if the HLDD diagram calculating x_j is ranked after current decision diagram. Otherwise, the present time step value will be used.

Algorithm 1 presents the HLDD based simulation engine for RTL, behavioral and mixed HDL description styles. (Refer to Section 3.1 for HLDD data structure definition).

Algorithm 1. RTL/behavioral simulation on HLDDs

For each diagram G in the model
  $m_{Current} = m_0$
  Let $x_{Current}$ be the variable labeling $m_{Current}$
  While $m_{Current}$ is not a terminal node
    If is $x_{Current}$ clocked or its DD is ranked after G then
      Value = previous time-step value of $x_{Current}$
    Else
      Value = present time-step value of $x_{Current}$
    End if
    If Value $\in D(e_{active})$, $e_{active} = (m_{Current}, m_{Next})$ then
      $m_{Current} = m_{Next}$
    End if
  End while
  Assign $x_{Current}$ to the DD variable x_G
End for

In order to understand assertion checking on T-HLDDs consider the PSL assertion example P1 provided in Figure 8. The assertion represents a property to be checked against the GCD implementation given in Figure 5. It states that always when *ready* is low and *a* is equal to *b* then after 1 to 3 time-steps (clock cycles) *ready* will be asserted.

Let us introduce the concept of *time-window* of an assertion. We say that a time window is the maximum number of time steps from current step to the end step when assertion has to be evaluated. Assertion's time-window is denoted by w_{max}. For example, the assertion shown in Fig. 8 has a time window of $w_{max} = 4$ because its evaluation starts at current time moment and ends 3 time-steps later.

The procedure of assertion checking should be preceeded by executing Algorithm 1 which calculates the simulation trace that is a starting point for assertion checking. The procedure is an extension of HLDD simulation as it takes into account temporal information at the nodes and has an exit condition in order to avoid eternal loops that are due to the cyclic nature of the general case of THLDDs. The main task of the procedure is assertions diagrams traversal till terminal nodes with strict consideration of time windows which gives the evaluation result.

However, event-driven evaluation of assertions to speed up the checking process has not been considered in current work and there lies the focus of our future research. While there already exists commercial assertion checking tools taking advantage of events the core benefits of developing an event-driven HLDD based solution lies in fast evaluation by graph traversal and for easy identification of cause-effect relationships provided by the model that is especially useful in debug and diagnosis.

TABLE 2. DD AND HDL-BASED SIMULATION COMPARISON

Circuit	Simulation time [s]		Ratio HDL/HLDD
	HLDD simulation	HDL cycle-based	
gcd	0.20	0.51	2.55
mult8x8	0.32	1.00	3.13
diffeq	0.25	1.26	5.04
huff_enc	0.34	1.40	4.12
circ1	0.14	2.05	14.64

Currently the T-HLDD based assertion checking is not yet implemented. However, we have compared HLDD model simulation (Algorithm 1) to a state-of-the-art HDL simulator. Table 2 presents run-times of the HLDD models and corresponding VHDL models simulation. The simulators compared include the HLDD-based simulator [7] and an efficient commercial cycle-based HDL simulation tool Cyclone (Synopsys). The experiment was run on a 366 MHz SUN UltraSPARC 60 workstation with 512 MB RAM under Solaris 2.5.1 operating system. During the experiments, real test stimuli generated by test generator DECIDER [8] were used in order to activate all possible states of the circuit behavior (in contrast to random simulation vectors, which in reality do not allow to simulate all possible behaviors). In order to achieve a better timing resolution all the test sets were multiplied by ten. For optimal performance, Synopsys tools *cylab* and *cysim* were run with *-perf* and *-2state* options. The decision diagram event-driven cycle-based simulation tool implementation offers the gain in simulation time between 2.5 and more than 14 times in comparison to the cycle-based HDL simulator.

6. CONCLUSIONS

The paper proposed a novel method for Property Specification Language (PSL) assertions simulation-based checking. The method uses a digital design representation called Temporally extended High-Level Decision Diagrams (T-HLDDs). Previous works have shown that HLDDs are an efficient model for simulation and diagnosis since they provide for a fast evaluation by graph traversal and for easy identification of cause-effect relationships. In this paper, the model was extended to support temporal operations inherent in PSL properties and also to directly support assertion checking. We presented a hierarchical approach to generate T-HLDDs based on a library of Primitive Property Graphs (PPG). Basic algorithms for T-HLDD based assertion checking were discussed.

As a future work we see development of event-driven assertion checking methods on T-HLDDs and their integration to design error diagnosis and debug solutions.

ACKNOWLEDGMENTS

The work has been supported partly by EC FP 6 research project VERTIGO FP6-2005-IST-5-033709 [16], Enterprise Estonia funded ELIKO Development Centre, Estonian SF grants 7068 and 7483, Estonian Information Technology Foundation (EITSA) and Nations Support Program for the ICT in Higher Education "Tiger University".

REFERENCES

[1] International Technology Roadmap for Semiconductors 2006 report [*www.itrs.net*]

[2] S. Gheorghita and R. Grigore, "Constructing Checkers from PSL Properties," 15th International Conference on Control Systems and Computer Science (CSCS15), vol. 2, pp. 757–762, 2005.

[3] Bustan D., Fisman D., and Havlicek J. Automata Construction for PSL. The Weizmann Institute of Science, Technical Report MCS05-04, May 2005

[4] Marc Boulé and Zeljko Zilic. Efficient Automata-Based Assertion-Checker Synthesis of PSL Properties. In Proceedings of the 2006 IEEE International High Level Design Validation and Test Workshop (HLDVT'06), pages 69–76, 2006.

[5] IBM AlphaWorks, "FoCs Property Checkers Generator ver. 2.04," [*www.alphaworks.ibm.com/tech/FoCs*], 2007.

[6] R. Ubar, J. Raik, A. Morawiec, Back-tracing and Event-driven Techniques in High-level Simulation with Decision Diagrams. *ISCAS 2000*, Vol. 1, pp. 208-211.

[7] Raimund Ubar, Adam Morawiec, Jaan Raik. Cycle-based Simulation with Decision Diagrams, Proceedings of the DATE Conference, pp. 454-458, 1999.

[8] J. Raik, R. Ubar, Fast Test Generation for Sequential Circuits Using Decision Diagrams Representations. *Journal of Electronic Testing: Theory and Applications* 16, Kluwer Academic Publisher, 2000, pp. 213-226.

[9] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35, 8:677-691, 1986

[10] V. Chayakul, D. D. Gajski, L. Ramachandran, "High-Level Transformations for Minimizing Syntactic Variances", *Proc. of ACM/IEEE DAC*, pp. 413-418, June 1993.

[11] I. Ghosh, M. Fujita, "Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams", *Proc. of ACM/IEEE DAC*, pp. 43-48, 2000.

[12] L. Zhang, I. Ghosh, M. Hsiao, "Efficient Sequential ATPG for Functional RTL Circuits", *Int. Test Conf.*, pp.290-298, 2003.

[13] Accellera, "Property Specification Language Reference Manual", v1.1, June 9, 2004.

[14] IEEE-Commission, "IEEE standard for Property Specification Language (PSL)," 2005, IEEE Std 1850-2005.

[15] Cindy Eisner, Dana Fisman, "A Practical Introduction to PSL", Springer Science, 2006.

[16] EU's 6th Framework Programme research project VERTIGO web page [*www.vertigo-project.eu*], 2007.

[17] R. Ubar. "Test Synthesis with Alternative Graphs", *In IEEE Design and Test of Computers*, pp. 48–57. 1996.

[18] Maksim Jenihhin, Jaan Raik, Anton Chepurov, Raimund Ubar. Assertion Checking with PSL and High-Level Decision Diagrams. *IEEE 8th Workshop on RTL and High Level Testing (WRTLT'07)*, October 12-13, 2007.

[19] K. Morin-Allory, D. Borrione. Proven correct monitors from PSL specifications, *Proc. DATE*, 2006.