Assertion Checking with PSL and 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 checking PSL language assertions using a system representation called High-Level Decision Diagrams (HLDD). Previous works have shown that HLDDs are an efficient model for simulation and test pattern generation. We present a technique, where checking of PSL assertions is integrated into fast HLDD-based simulation. Current approach applies assertion checker generation software FoCs by IBM. We show how such VHDL checkers can be mapped to HLDD constructs.**

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.

There are two challenges stated for the verification research area. The first one is verification methods for higher levels of abstraction. The second one is new Design-for-Verifiability (DFV) techniques. The approach proposed in this paper addresses the both of them.

Among the recently proposed DFVs 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. They can describe either desired or undesired behaviour and notify user about violations or occurrence of forbidden sequences consequently.

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]. The most widely known tool for this task is FoCs by IBM [5].

In this paper, we present an approach to checking PSL assertions using High-Level Decision Diagrams (HLDD). Here, assertion checking is integrated into fast HLDD-based simulation. Assertion checker generation software FoCs by IBM is applied and the resulting VHDL checkers are mapped to HLDD constructs. The work is motivated by our previous encouraging research results obtained on HLDD based simulation [6, 7] and test pattern generation [8]. This is the first attempt to use HLDD models in assertion based verification.

The paper is organized as follows. The HLDD based verification flow is explained in Section 2. Section 3 defines the HLDD graph model. Section 4 discusses assertion-based verification and PSL. Section 5 shows how HLDDs can be used for representing assertion checkers. Finally, Section 6 concludes the paper.

2. HLDD Verification Flow

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 manufacturing test generation research areas. However participation of TUT in recently launched by European Commission research project VERTIGO [16] has encouraged HLDD usage in verification.

The main areas of interest for VERTIGO research project are 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 main emphasis of this paper is put on assertion monitors (Figure 1). However, several other tools working with HLDD are under development, including dynamic verification code coverage analysis, formal methods of verification stimuli generation and model checking. The last ones reuse the engine of HLDD based ATPG known as DECIDER [8]. The following section will define HLDD model.

	Assertions	*Code coverage*
Dynamic verification	**Assertion monitors**	Code coverage analysis
Static Verification	Model checking	Input generation for code coverage

Figure 1. HLDD verification tasks distribution chart

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 fault modeling 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 2 presents an example of a graphical interpretation of a HLDD.

3.2. Digital systems simulation using HLDDs

In HLDD models representing digital systems, the non-terminal nodes correspond to conditions or to control signals, and the terminal nodes represent 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. Figure 3 presents an example of an HLDD for two variables, state and RMAX in the ITC99 benchmark b04.

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

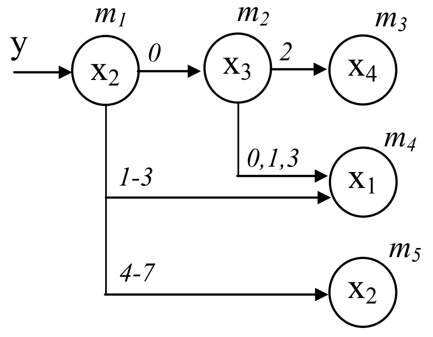

Figure 2. A HLDD for a function $y=f(x_1,x_2,x_3,x_4)$

```
...
if RESET = '1' then
      state := sA;
      RMAX := 0;
       ...
    elsif CLOCK'event and
            CLOCK='1' then
       ...
      case state is
        when sA =>
          state := SB;
        when sB =>
          RMAX := DATA_IN;
          ...
         state := sC;
        when sC =>
          ...
          if DATA_IN > RMAX then
            RMAX := DATA_IN;
              ...
          end if;
          ...
         state := sC;
      end case;
    end if;
...
```

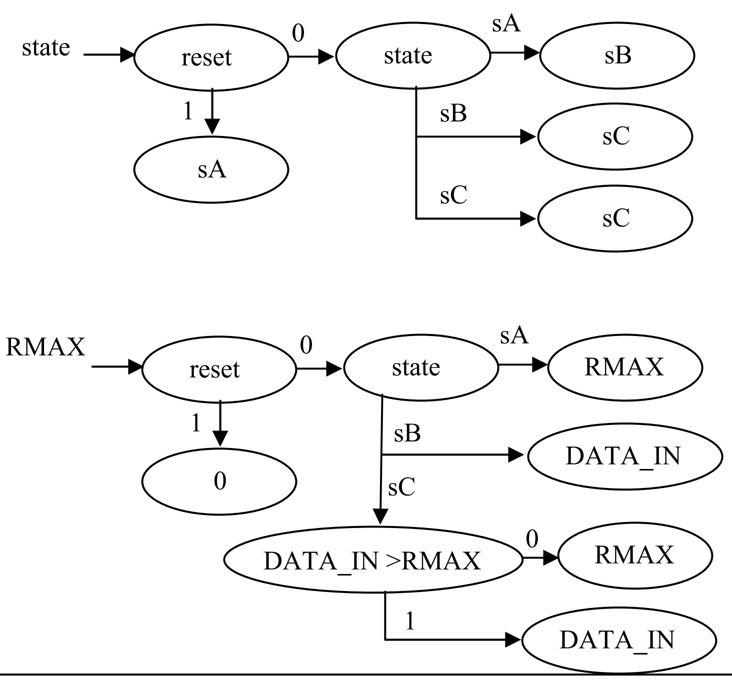

Figure 3. b04 example: HLDDs for variables *state* and *RMAX*

The basis for assertion checking approach presented in this paper is a simulator engine based on HLDDs. We have implemented an algorithm supporting, both, Register-Transfer Level (RTL) and behavioral style Hardware Description Language (HDL) styles. 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 special order. 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 (See Section 3.1 for definitions!):

***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
```

The following section will discuss the idea of assertion-based verification and PSL language. It will also mention the commercial tool FoCs from IBM for VHDL checkers generation from PSL properties. Section 4 explains how such checkers described in VHDL language can be converted into HLDD models.

4. Assertion-based verification and PSL

As it was already noticed in the first section ABV 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 the 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), SVA (System Verilog Assertions) or CTL (Computation Tree Logic). 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].

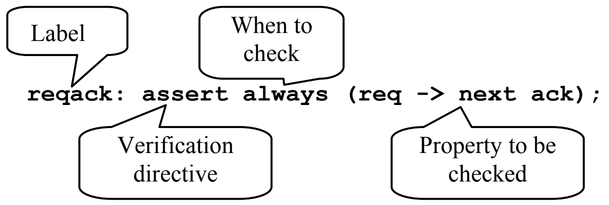

Figure 4. PSL property "reqack"

An example PSL property *reqack* structure is shown in Figure 4. Its Timing diagram is also illustrated by Figure 5-a. 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 5-b. Figure 5-c shows the case when the property was not activated.

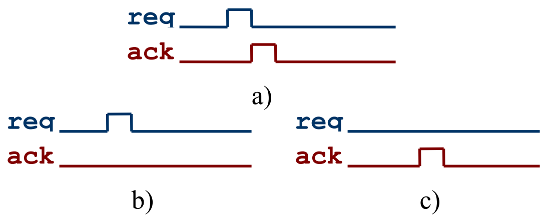

Figure 5. Timing diagrams for the property "reqack"

For the convenience of verification engineers PSL is multi-flavoured language, what means that it supports common constructs of VHDL, Verilog, IBM's GDL, SystemVerilog and SystemC [15].

PSL is also a multi-layered language. The layers are [13]:

- *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 (the main one) has two constituents [13]:

- 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 for future work related to HLDD-based model checking implementation. This paper we will consider only FL part of PSL. However not even the whole FL is applicable for translation to HLDD monitors. Only its subset also known as PSL Simple Subset is suitable for this purpose.

PSL Simple 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.

A verification tool may support PSL or its Simple Subset directly or require translation of PSL assertions to checkers in its own format. The most widely known tool for this automatic translation is FoCs by IBM. The input properties for this translation can be expressed both in Sugar and PSL (GDL or Verilog flavour). The target language of the checkers to be generated can be chosen from VHDL, Verilog or C++. In frames of the current approach we exploit FoCs to generate VHDL checkers. Another advantage of FoCs is ability of its usage without GUI what allows easy translation step integration to HLDD verification flow.

The next section will describe the conversion process of a VHDL checker generated from PSL property to HLDD monitor.

5. PSL Assertions Integration to HLDDs

Let us consider an example of a PSL assertion *p* as given bellow:

```
p: assert always ({a; [*2] ;b} |=> {c})
```

The precondition of this assertion is the sequence of system behaviour when at the beginning *a* becomes high, followed by a whatever-sequence 2 clock cycles long and then *b* becoming high. This precondition activates the assertion and requires *c* to become high just after it (non-overlapping implication) in order for assertion to be satisfied.

```
PROCESS (clk)
BEGIN
  IF ( ( clk = '1' ) ) THEN
    focs_ok <=
    ( focs_vout(4) OR NOT( c ) ) ;
  ELSE
    focs_ok <= '1' ;
  END IF;
END PROCESS;

PROCESS
...
VARIABLE focs_vout : std_logic_vector(4 DOWNTO 0);
BEGIN
  WAIT UNTIL (clk'EVENT AND clk = '1');
  ...
  focs_vout(4 DOWNTO 0) := reverse( ( ( ( ( ( (
    focs_v(0) AND a ) ) & ( ( focs_v(1) AND '1' )
    ) ) & ( ( focs_v(2) AND '1' ) ) ) & ( (
    focs_v(3) AND b ) ) ) & ( ( focs_v(4) AND
    NOT( c ) ) ) ) );
  ...
END PROCESS;
```

Figure 6. VHDL describing the checker for assertion p

The VHDL of the checker generated by FoCs has the form shown in Figure 6.

The resulting VHDL code can be converted to HLDD graphs and added on top of the design under verification (DUV) as shown in Figure 7. In the Figure we used a notation where trailing quote character after diagram variable denotes one clock cycle delay. The HLDD variables corresponding to the inputs and outputs of the checker (i.e. reset, a, b, c and p) are shown by bold font. Generation of respective HLDDs from similar checkers described in VHDL can be easily automated.

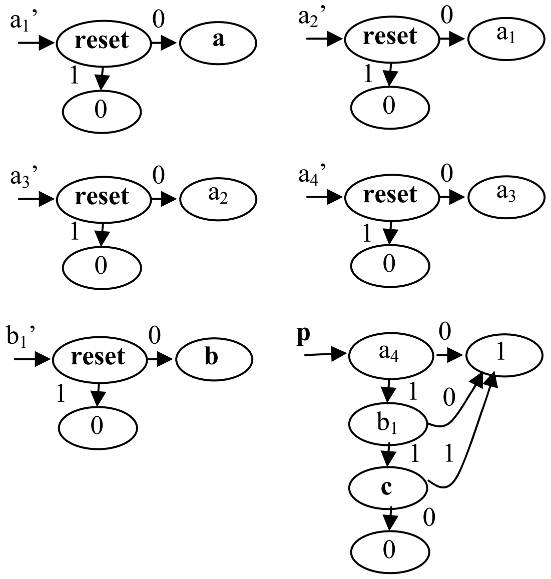

Figure 7. HLDD for the assertion checker VHDL in Figure 6

The HLDD of the checker would be seamlessly integrated and added on top of the design under verification (See Figure 8). This allows uniform model representation for, both, the DUV and the assertion checker. Simulation of such integrated HLDD model speeds up the assertion checking process considerably. Previous experiments comparing HLDD simulation times to those of state-of-the-art commercial simulators have shown that it outperforms the commercial event-driven tools by a factor of 10 [6] and the cycle-based counterparts by a factor of 3-4 [7].

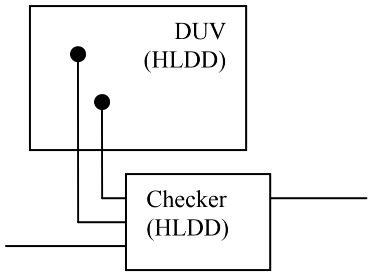

Figure 8. PSL assertion integration to HLDD model

6. Conclusions

The paper has proposed a novel method for checking PSL language assertions using a digital design representation called High-Level Decision Diagrams (HLDD).

Previous works have shown that HLDDs are an efficient model for simulation and test pattern generation. The speedup in simulation which is equal to factors of time and some other their advantages has encouraged HLDDs application for verification. At the same time Assertion-based Verification approach provides obvious benefits for both static and dynamic verification. This fact and the recent rapid spread of PSL assertions application and support have motivated PSL assertions integration into HLDD-based verification flow.

The paper has described in details PSL properties conversion to HLDD simulation monitors, implying FoCs by IBM for an intermediate step of VHDL checkers generation. The approach has been demonstrated on example of a real life type PSL temporal property.

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 5910, Estonian Association of Information Technology and Telecommunications (ITL), 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.