High-Level Decision Diagrams based Coverage Metrics for Verification and Test

Maksim Jenihhin, Jaan Raik, Anton Chepurov, Uljana Reinsalu, Raimund Ubar
Department of Computer Engineering, Tallinn University of Technology
E-mail: {maksim|jaan|anchep|uljana|raiub}@pld.ttu.ee

Abstract

The paper proposes High-Level Decision Diagrams (HLDDs) model based structural coverage metrics that are applicable to, both, verification and high-level test. Previous works have shown that HLDDs are an efficient model for simulation and test generation. However, the coverage properties of HLDDs against Hardware Description Languages (HDL) have not been studied in detail before. In this paper we show that the proposed methodology allows more stringent structural coverage analysis than traditional VHDL code coverage. Furthermore, the main new contribution of the paper is a hierarchical approach for condition coverage metric analysis that is based on HLDDs with expansion graphs for conditional nodes. Experiments on ITC99 benchmarks show that up to 14% increase in coverage accuracy can be achieved by the proposed methodology.

1. Introduction

Structural coverage metric (also referred to as code coverage) has its application, both, in functional verification and high-level test. Due to the fact that it is impractical to verify exhaustively all possible inputs and states of a design, the confidence level regarding the quality of the design must be quantified to control the testing effort. The fundamental question is: when is the design simulated enough? Structural coverage is a measure of confidence and it is expressed as a percentage of items covered out of all possible items. Different definitions of items give rise to different coverage measures, or coverage metrics.

Over the years, a large variety of code coverage metrics have been proposed, including statement coverage, block coverage, path coverage, branch coverage, expression coverage, transition coverage, sequence coverage, toggle coverage etc [1],[10]. The statement coverage metric measures the ratio of code instructions exercised from the entire set of instructions by the program stimuli. Toggle coverage shows whether and how much the signals of the design toggle, i.e. how many bits change their state from 0 to 1 or vice versa. In the case of branch coverage, we measure the ratio of branches in the control flow graph of the code taken under the set of program stimuli. Condition coverage metric ([1],[6]) reports the number of Boolean sub-expressions, separated by logical operators, calculated in conditional statements causing their evaluation to one of the decisions (e.g. ‘true’ or ‘false’ values). It differs from the branch coverage, by the fact that in the latter only the final decision determining the branch is taken into account.

Structural coverage has a long history in software testing and only with the emergence of hardware description languages it has been applied to hardware verification and test. Several standards for system quality test such as DO-178B [7] for software and its analog for hardware DO-254 [8] used in airborne systems state that condition coverage along with statement and branch coverages has to be applied in the cases where system failures would cause catastrophic results.

In this paper we introduce a methodology for structural coverage analysis based on HLDD graph model proposed and developed in Tallinn University of Technology. Experiments on ITC99 benchmarks show that up to 14 % increase in coverage accuracy can be achieved by the proposed methodology compared to traditional HDL code coverage. The main new contribution of the paper is a hierarchical approach for condition coverage metric analysis that is based on HLDDs with expansion graphs for conditional nodes. The proposed methodology has significantly lower computational overhead compared to HDL-based approaches and relies on homogeneous verification flow (i.e. one model and one tool).

The rest of the paper is organized as follows. Section 2 introduces HLDD graph model and its simulation algorithm. Section 3 defines different
representation types for HLDDs based on respective rules. HLDD-based structural coverage analysis
implying a novel approach for condition coverage metric analysis is proposed in Section 4. Section 5
demonstrates advantages behind the proposed methodology by an example study and experimental
results on ITC99 benchmarks.

2. High-Level Decision Diagrams

2.1 HLDD model definition

A High-Level Decision Diagram (HLDD) is a graph representation of a discrete function. A discrete
function \( y = f(x) \) where \( y = (y_1, ..., y_n) \) and \( x = (x_1, ..., x_m) \) are vectors is defined on \( X = X_1 \times \ldots \times X_m \) with values \( y \in Y = Y_1 \times \ldots \times Y_n \) and both, the domain \( X \) and the
range \( Y \) are finite sets of values. The values of variables may be Boolean, Boolean vectors, integers.
Fig. 1 presents an example of a graphical interpretation of a HLDD.

![Graphical interpretation of HLDD](image)

\( G = (M, E, Z, \Gamma) \),
\( M = \{m_0, m_1, m_2, m_3, m_4\} \);
\( E = \{e_1, e_2, e_3, e_4\} \), \( I = (m_0, m_1) \), \( e_2 = (m_0, m_2) \), \( e_3 = (m_0, m_3) \), \( e_4 = (m_0, m_4) \);
\( Z(m_0) = \{x_0\} \), \( Z(m_1) = \{x_1\} \), \( Z(m_2) = \{x_2\} \), \( Z(m_3) = \{x_3\} \), \( Z(m_4) = \{x_4\} \);
\( \Gamma(e_1) = \{0\} \), \( \Gamma(e_2) = \{1, 2, 3\} \), \( \Gamma(e_3) = \{4, 5, 6, 7\} \), \( \Gamma(e_4) = \{2, 1, 3\} \).

Fig. 1. A high-level decision diagram representing a function \( y = f(x_1, x_2, x_3, x_4) \)

Definition 1: A high-level decision diagram is a directed non-cyclic labeled graph that can be defined
as a quadruple \( G = (M, E, Z, \Gamma) \), where \( M \) is a finite set of vertices (referred to as nodes), \( E \) is a finite set of
edges, \( Z \) is a function which defines the variables labeling the nodes, and \( \Gamma \) is a function on \( E \).

The function \( Z(m_i) \) returns the variable \( x_i \), which is labeling node \( m_i \). 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 \( e = (m_{pc}, m_{sc}) \in E \), where \( E \) is the set of all the possible ordered pairs in set \( E \). Graphical interpretation of \( e \) is an edge leading from node \( m_{pc} \) to node \( m_{sc} \).

It is said that \( m_{pc} \) is a predecessor node of \( m_{sc} \), and \( m_{sc} \) is a successor node of the node \( m_{pc} \), respectively. \( \Gamma \)
is a function on \( E \) representing the activating conditions of the edges for the simulating procedures. The value of \( \Gamma(e) \) is a subset of the domain \( X \) of the variable \( x_i \), where \( e = (m_{pc}, m_{sc}) \) and \( Z(m_{sc}) = x_i \). It is
required that \( Pm_i = \{ \Gamma(e) \mid e = (m_{pc}, m_{sc}) \in E \} \) is a partition of the set \( X \).

Fig. 1 presents a HLDD for a discrete function \( y = f(x_1, x_2, x_3, x_4) \). HLDD has only one starting node (root
node) \( m_0 \), for which there are no preceding nodes. The nodes that have no successor nodes are referred to as
terminal nodes \( M^{term} \in M \) (nodes \( m_3 \), \( m_4 \), and \( m_4 \) in Fig. 1). Design representation by high-level decision
diagrams, in general case, is a system of HLDDs rather than a single HLDD. 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.

2.1 Systems simulation using HLDDs

In our earlier works [4], we have implemented an algorithm supporting, both, Register-Transfer Level
(RTL) and behavioral design abstraction levels. Algorithm 1 (Fig. 2) presents the HLDD based
simulation engine for RTL, behavioral and mixed HDL description styles.

```
SimulateHLDD()
For each diagram G in the model
mCurrent = m0
Let xCurrent be the variable labeling mCurrent
While mCurrent is not a terminal node
If xCurrent is clocked or its DD is ranked after G then
Value = previous time-step value of xCurrent
Else
Value = present time-step value of xCurrent
End if
For \( \Gamma(e) \mid Value \in \Gamma(e) \), \( e = (m_{pc}, m_{sc}) \), \( m_{pc} = m_{next} \)
End if
End while
Assign xG = xCurrent
End for
End SimulateHLDD
```

Fig. 2. Algorithm 1. RTL/behavioural simulation on HLDDs

In the RTL style, the algorithm takes the previous time step value of variable \( x_j \) labeling a node \( m_j \) 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
calculating \( x_j \) is ranked after current decision diagram. Otherwise, the present time step value will be used.
3. Representation types of HLDD model

The methodology for HLDD-based coverage analysis proposed in Section 5 implies manipulation techniques on HLDD model aimed to different aspects of the proposed analysis.

In the proposed methodology we distinguish three types of HLDD representation according to their compactness, and with consideration of the HLDD reduction rules. These rules are similar to the reduction rules for BDDs presented in [5] and can be generalized as follows:

- **HLDD reduction rule1:** Eliminate all the redundant nodes whose all edges point to an equivalent subgraph.

- **HLDD reduction rule2:** Share all the equivalent subgraphs.

The three representation types in the increasing order of compactness are:

- **Full tree HLDD** contains all control flow branches of the design. This type of representation includes a lot of redundancy. They introduce large space requirements and relatively slow simulation times.

- **Reduced HLDD** is obtained by application of the HLDD reduction rule 1 to the full tree representation. This HLDD representation is still a tree-graph. This type of representation combines the advantages of two full-tree representation and minimized representation. The HLDDs of this type are reasonably compact and they allow more stringent coverage measurement than the minimized representation. Furthermore, the average HLDD path length, and therefore the simulation speed, is exactly equal to the more compact minimized representation!

- **Minimized HLDD** is obtained by application of both HLDD reduction rules 1 and 2 to the full tree representation. This representation is the most compact of the three. However, the minimization step may cause loss of coverage measurement accuracy.

Let us consider an example typical design latw09_ex1. The functional segment of its RTL VHDL representation is provided in Fig. 3. The parts of the variables’ names have the following notations {V- an output variable; cS - a conditional statement; D- a decision; T- a terminal node; C- a condition; W- a value}. The emphasized by bold keywords determine if a line has a statement, a branch or conditions. Figures 4, 5 and 6 present a reduced, minimized, full-tree and HLDD model representations correspondingly.

4. HLDD-based coverage analysis

4.1 Statement coverage mapping

The initial idea of statement coverage representation on HLDDs was proposed in [2]. This paper demonstrates the details. The statement coverage metric has a straightforward mapping to HLDD-based coverage. It maps directly to the ratio of nodes mCurrent traversed during the HLDD simulation presented in Algorithm 1 (Subsection 2.2) to the total number of the HLDD nodes in the DUV’s representation. The
appropriate type of HLDD representation for the analysis of, both, statement and branch coverage metrics is the reduced one. The variations in the analysis caused by different HLDD representation types are discussed in Subsection 5.1.

4.2 Branch coverage mapping

Similar to the statement coverage, branch coverage also has very clear representation in HLDD model. (Also initially proposed in [2], the details are demonstrated in this paper). It is the ratio of every edge $e_{\text{active}}$ activated in the simulation process presented by Algorithm 1 (Subsection 2.2) to the total number of edges in the corresponding HLDD representation of the DUV.

The numbers (underlined) from the second column ($Dcn$) in Fig. 3 correspond to the lines with 7 branches (i.e. decisions). The 12 HLDD nodes of the two graphs in Fig. 4 correspond to these decisions. Covering all edges in a HLDD model (i.e. full HLDD edge coverage) corresponds to covering all branches in the respective HDL. However, similar to the previously discussed statement coverage mapping, here the opposite is also not true and HLDD edge coverage is slightly more stringent that HDL branch coverage.

The VHDL branches (Fig. 3) 1 and 2 are represented in Fig. 4 by respective edges 1a, 1b and 2a, 2b. The duplicated edges are also emphasized by subscript indexes, (e.g. $Dcn_1$, $Dcn_2$, $Dcn_3$).

4.3 A hierarchical approach for condition coverage analysis

Condition coverage metric reports all cases each Boolean sub-expression, separated by logical operators or and and, in a conditional statement causes the complete conditional statement to evaluate to one of the decisions (e.g. ‘true’ or ‘false’ values) under the given set of stimuli. It differs from the branch coverage, by the fact that in the branch coverage only the final decision determining the branch is taken into account. In case, if we have $n$ conditions joined by logical and operators in a logical expression of a conditional statement, it means that the probability of evaluating the statement to the decision ‘true’ is $1/2^n$ (considering pure random stimuli for the condition values). Calculation of the condition coverage based on HDL representation is a sophisticated multi-step process. However, the condition coverage metric
allows discovering information about many corner cases of the DUV.

In this section we present a methodology for condition coverage metric HLDD-based. The approach is based on a hierarchical DUV representation where the conditional statements with complex logical expressions (normally represented by single nodes in HLDD graphs) are expanded into separate HLDD graphs.

Let us consider the example design latw09_ex1 provided in Figures 3 and 4. It contains the following 3 conditional statements:

```vhdls
cs1: if (c1 and c2)
    case cS2_C1 is
      when cS2_C1_W1 => cS2_D1;
      when cS2_C1_W2 => cS2_D2;
      when cS2_C1_W3 => cS2_D3;
    end case;
    cs2: if (c1 and (not c2) or c3)
    case:
      when cs2_C1_W1 => cs2_D1;
      when cs2_C1_W2 => cs2_D2;
      when cs2_C1_W3 => cs2_D3;
    end case;
    cs3: if (c1 and c2) case
      when cS3_C1 => cS3_D1;
      when cS3_C2 => cS3_D2;
      when cS3_C3 => cS3_D3;
    end case;
end if;
```

The HLDD expansion graphs for these conditional statements are provided in Fig. 7. Here the terminal nodes are marked by background colors according to different decisions for better readability.

![Expansion graphs for conditional statements of latw09_ex1](image)

These 3 expansion graphs can be considered as sub-graphs representing “virtual” variables (because they are not real variables of the latw09_ex1 VHDL representation) cS1, cS2, cS3. Thus, together with the two HLDD graphs for variables V1 and V2 from Fig. 4 these sub-graphs compose design’s hierarchical HLDD representation.

The full condition coverage metric maps to the full coverage of terminal nodes of the conditional statements expansion graphs during the complete hierarchical DD system simulation with the given stimuli. The size of the items’ list $I_C$ for this coverage metric is:

$$I_C = \sum_{i=1}^{n_{iC}} 2^{n_{iC}} + \sum_{k=1}^{n_{case}} n_{c_k}$$

Here $n_{iC}$ is the number of $i$-type conditional statements, $n_{ci}$ is the number of conditions in the $i$-th conditional statement and $n_{case}$ is the number of case-type conditional statements and $n_{ck}$ is the number of conditions in the $k$-th conditional statement. (E.g. condition coverage items’ list for latw09_ex1 is $I_C = (2^2 + 2^3) + (3) = 15$)

The main advantage of the proposed approach is low computational overhead. Once the hierarchical HLDD is constructed, the analysis for an every given stimuli set is evaluated in a straightforward manner during HLDD simulation (Algorithm 1, Subsection 2.2).

The size of the hierarchical HLDDs with the expanded conditional statements grows with respect to $I_C$ and therefore there is a significant increase of the memory consumption. However, the length of the average sub-path from the root to terminal nodes grows linear to the number of the conditions. Therefore, since the simulation time of a HLDD has a linear dependency to the average sub-path from the root to terminal nodes, it will grow only linearly with respect to the number of conditions.

5. Experiments

5.1 Example study

This subsection provides an example study for the proposed HLDD-based coverage analysis methodology based on the latw09_ex1 design. This design has 2 outputs and 6 control input signals (i.e. cS1_C1, cS1_C2, etc). The first output ($V1$) can be assigned to 2 possible values while the second one ($V2$) has 3 possibilities (i.e. $\{V1_T1, V1_T2\}$ and $\{V2_T1, V2_T2, V2_T3\}$). The values labeling the terminal nodes in this example are symbolic and their source is neglected.

Let us assume for a simulation experiment a set of stimuli Stim_1 with the following test vectors presented in the upper part of Table 1. The resulting values for the outputs $V1$ and $V2$ are also shown in the table. The lower part of the table demonstrates the items of the considered coverage metrics that are covered by a particular vector. For VHDL metrics the items are numbered statements and branches from Fig. 3. The HLDD metrics’ items are numbered nodes and
edges of the reduced HLDD design representation (Fig. 4). The last row of the table shows covered items of the condition coverage metric that are terminal nodes of the conditional nodes expansion graphs shown in Fig. 7. A subscript suffix (e.g. ‘T4’) points to the respective expansion graph’s terminal number counting from top to bottom. The framed VHDL items for vectors 3 and 4 may be considered covered by some HDL simulators, but their execution does not influence the final output of the process.

Table 1. A set of stimuli Stim_1 for latw09_ex1

<table>
<thead>
<tr>
<th>Input / Output</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>cS1_C1</td>
<td>1</td>
</tr>
<tr>
<td>cS1_C1</td>
<td>1</td>
</tr>
<tr>
<td>cS2_C</td>
<td>W1</td>
</tr>
<tr>
<td>cS3_C1</td>
<td>-</td>
</tr>
<tr>
<td>cS3_C2</td>
<td>-</td>
</tr>
<tr>
<td>cS3_C3</td>
<td>V1</td>
</tr>
<tr>
<td>V2</td>
<td>V2_T1</td>
</tr>
<tr>
<td>Metric</td>
<td>Items covered</td>
</tr>
<tr>
<td>VHDL stm.</td>
<td>1,2,4,5 1,3,4,6</td>
</tr>
<tr>
<td>VHDL brnc.</td>
<td>1,3     2,4</td>
</tr>
<tr>
<td>HLDD nodes</td>
<td>4,1a,2a,4,2,5</td>
</tr>
<tr>
<td>HLDD edges</td>
<td>3,2a_3  2a_3</td>
</tr>
<tr>
<td>cond. exp. grp.</td>
<td>cS2_T1, cS1</td>
</tr>
</tbody>
</table>

Table 2 presents a resulting coverage values obtained for Stim_1. Here we consider covered duplicated items as one, i.e. if node 4_1 is covered then node 4_2 is also covered. That is the reason why the values for full-tree and reduced HLDDs are equal. Please consider the fact that HLDD-based nodes and edges coverage values are lower and therefore more stringent than the respective VHDL metrics’ values. The remained uncovered HLDD items are edges 2a_1b and nodes 3a_2b. In the minimized HLDD (Fig. 5) this benefit is lost with minimization. However, even with full statement and branch coverages achieved the full condition coverage would require additional test vectors. It adds orthogonal dimension of confidence in terms of stringency and verification/test accuracy.

Table 2. Coverage metrics accuracy comparison for latw09_ex1 example study

<table>
<thead>
<tr>
<th>Coverage metric</th>
<th>VHDL, %</th>
<th>HLDD, %</th>
<th>Statement coverage, (%)</th>
<th>Branch coverage, (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>minimized</td>
<td>reduced</td>
<td>full-tree</td>
<td>red.</td>
</tr>
<tr>
<td>Statement</td>
<td>10/10 = 100</td>
<td>8/8 = 100</td>
<td>11/13 = 85.6</td>
<td>11/13 = 85.6</td>
</tr>
<tr>
<td>Branch</td>
<td>7/7 = 100</td>
<td>7/7 = 100</td>
<td>7/9 = 77.8</td>
<td>7/9 = 77.8</td>
</tr>
<tr>
<td>Condition</td>
<td>n/a</td>
<td>7/15 = 46.7</td>
<td>7/15 = 46.7</td>
<td>7/15 = 46.7</td>
</tr>
</tbody>
</table>

Table 3. Characteristics of different HLDD representations

<table>
<thead>
<tr>
<th>Design</th>
<th>Number of nodes</th>
<th>Number of edges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Red.</td>
</tr>
<tr>
<td>b01</td>
<td>30</td>
<td>57</td>
</tr>
<tr>
<td>b02</td>
<td>16</td>
<td>26</td>
</tr>
<tr>
<td>b06</td>
<td>47</td>
<td>116</td>
</tr>
<tr>
<td>b09</td>
<td>44</td>
<td>69</td>
</tr>
</tbody>
</table>

Table 4 shows comparison results of the proposed methodology based on different HLDD representations and coverage analysis achieved by a commercial state-of-the-art HDL simulation tool from a major CAD vendor using the same sets of stimuli.

As it can be seen, the reduced HLDDs with expanded conditional nodes allow equal or more stringent coverage evaluation in comparison to the commercial coverage analysis software. For three designs (b01, b06 and b09) more stringent analysis is achieved using HLDDs. The HLDD model allows increasing the coverage accuracy up to 13 % more exact statement measurement and 14 % branch measurement (b09 design). In our previous work [2] it was shown that HLDD-based coverage analysis has significantly lower (tens of times) computation (i.e. measurement) time overhead compared to the same commercial simulator.

Table 4. Comparison of code coverage analysis results

<table>
<thead>
<tr>
<th>Design</th>
<th>Stimuli (vectors)</th>
<th>Statement coverage, (%)</th>
<th>Branch coverage, (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>red.</td>
<td>min.</td>
<td>VHDL</td>
</tr>
<tr>
<td>b01</td>
<td>14</td>
<td>86.0</td>
<td>100</td>
</tr>
<tr>
<td>b02</td>
<td>10</td>
<td>92.3</td>
<td>100</td>
</tr>
<tr>
<td>b06</td>
<td>11</td>
<td>80.2</td>
<td>100</td>
</tr>
<tr>
<td>b09</td>
<td>52</td>
<td>98.3</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>53</td>
<td>87.0</td>
<td>100</td>
</tr>
</tbody>
</table>

6. Conclusions

It is important to emphasize that all coverage metrics (i.e. statement, branch, condition or a combination of them) in the proposed methodology are analyzed by a single HLDD simulation tool which relies on HLDD design representation model. Different levels of coverages are distinguished by simply generating a different level of HLDD (i.e. minimized, reduced, or hierarchical with expanded conditional nodes). Experimental results demonstrate feasibility and efficiency of the proposed methodology.

5.2 Experimental results

This subsection presents experimental results for four ITC99 benchmarks [9] that evaluate the proposed HLDD-based structural coverage analysis methodology. Table 3 presents the characteristics of the different HLDD representations introduced in Section 3.
Acknowledgments The work has been supported in part by EC FP 6 research project VERTIGO FP6-2005-IST-5-033709, Enterprise Estonia funded ELIKO Development Centre, Estonian SF grants 7068 and 7483, and Estonian Information Technology Foundation (EITSA).

References