Systematic Functional Coverage Metric Synthesis from Hierarchical Temporal Event Relation Graph

Young-Su Kwon
VLSI Systems Lab., KAIST
GuSeong-Dong, YuSeong-Gu
Daejeon, Korea(ROK)
yskwon@vslab.kaist.ac.kr

Young-Il Kim
VLSI Systems Lab., KAIST
GuSeong-Dong, YuSeong-Gu
Daejeon, Korea(ROK)
zerone@vslab.kaist.ac.kr

Chong-Min Kyung
VLSI Systems Lab., KAIST
GuSeong-Dong, YuSeong-Gu
Daejeon, Korea(ROK)
kyung@ee.kaist.ac.kr

ABSTRACT

Functional coverage is a technique for checking the completeness of test vectors in HDL simulation. Temporal events are used to monitor the sequence of events in the specification. In this paper, automatic generation of temporal events for functional coverage is proposed. The HiTER is the graph where nodes represent basic temporal properties or subgraph and edges represent time-shift value between two nodes. Hierarchical temporal events are generated by traversing HiTER such that invalid, or irrelevant properties are eliminated. Concurrent edge groups make it possible to generate more comprehensive temporal properties and hierarchical structure makes it easy to describe large design by combining multiple subgraphs. Automatically generated temporal events describe almost all the possible temporal properties of the design under verification.

Categories and Subject Descriptors

T2.2 [**Transaction-level, RTL and gate-level modeling and validation**]

General Terms

Functional coverage, Semi-formal verification

Keywords

Functional coverage, Temporal Event

1. INTRODUCTION

In designing SoC, functional and logic verification represents over 48% of the whole design process. Although formal verification is very efficient in verifying small designs, the state explosion problem has been the major bottleneck for the further extension of formal verification techniques to real designs. Simulation is the primary technique for functional validation of designs. Although random simulation can validate the functionality of the design in a short time, it lacks the measure of the quality of verification effort, i.e., test vectors may verify tested parts repeatedly.

The combination of simulation and formal verification provides the coverage information about the verification environment [5][8]. In this hybrid methodology, the formal properties of the design are specified using the formal language. The comprehensive set of tasks, i.e., properties are generated in a systematic fashion and checked during the simulation phase. The test vectors are selected such that they have a certain specified level of coverage with respect to those properties. The analysis of coverage directs the test vector generation to achieve 100% coverage [2][9].The *coverage metric* is a model for coverage measurement which comprises lots of *coverage tasks* that are evaluated as true when a specific event occurs. The number of occurrences of coverage tasks is accumulated and reported when a coverage report is generated. The functional coverage metric is usually a sequence of events that represents temporal properties of the design described in property specification language [1]. The definition of coverage metric for specific application is crucial for the coverage analysis to be successful in verifying the design.

In this paper, we propose a novel coverage task generation from HiTER (Hierarchical Temporal Event Relation graph) for functional coverage. We focus on the functional coverage in which coverage tasks are described by FLTL [8]. In our approach, HiTER is used as the specification for the design. In HiTER, the concurrent edge groups are defined for each node to describe multiple properties that can occur simultaneously after the occurrence of the current property. The proposed coverage metric generation algorithm generates all possible temporal events from HiTER.

This paper is organized as follows. In section 2, the related works are presented. In section 3, the HiTER and temporal event generation algorithm are presented. The experimental results are shown in section 4 with concluding remarks in section 5.

2. RELATED WORKS

Coverage metrics are categorized into two groups: code-based (program-based) coverage and functional coverage [12]. Code-based coverage comprises line coverage, condition coverage and state coverage. Line coverage is a basic type of coverage testing the syntactic properties of RTL code. Condition coverage monitors whether a certain expression evaluates to true or false during the simulation. State coverage concentrates on monitoring simulation test vectors driving the design to a specific state [11][6]. The code-based coverage is a general coverage metric easily generated through the analysis of RTL code, but they are not sufficient to represent the whole functional varieties of the design.

In the functional coverage analysis, the coverage metric focuses on the functionality of the design that is specific to the implementation and application where coverage metric is described using temporal property specification language such as LTL [8]. In the cross-product functional coverage model, the list of coverage tasks comprises all possible combinations of values for a given set of attributes [3][13]. The cross-product model is a promising model for the functional coverage, but the number of cross-product states where several state machines interact with each other can easily become too large where there are many invalid states that are not reached with any combination of input sequences. Another problem is that the manual description requires the user to specify all possible temporal properties. Boolean expression in user-specified constraints for interface protocol is used as coverage metric in [9]. In [4], the witness graph that is abstracted CDFG (Control Data Flow Graph) is used as coverage metric. The witness graph is the extracted control data flow graph similar to the ECFM in [7]. The difference is that any part of the design that does not affect the user-defined property is removed in the witness graph.

In this paper, we focus on the automatic generation of the functional coverage metrics from HiTER specified by the user. The algorithm for temporal property generation makes all possible coverage tasks by traversing the graph while removing the invalid properties.

3. HIERARCHICAL TEMPORAL EVENT RELATION GRAPH

HiTER (Hierarchical Temporal Event Relation graph) is the graph where nodes represent a basic property or subgraph and edges represent the time-shift value between two nodes. The basic property of the node is described as a Boolean logic or temporal logic defined in OpenVera [10]. In the *temporal property generation algorithm*, BFS traverses HiTER to combine the temporal expressions of the nodes to form longer or more complex expressions hierarchically. Finally, the hierarchically constructed temporal expressions are defined as assertion properties. The generated properties are FLTL (Finite Linear Temporal Logic) that is a variant of LTL (Linear time Temporal Logic) [8].

A node in HiTER is basic property or subgraph. The subgraph is composed of basic properties and subgraphs. BFS (Breadth-First-Search) algorithm traverses HiTER to generate all possible temporal properties. If BFS algorithm reaches a subgraph, the subgraph is recursively expanded to generate properties. The hierarchical structure of HiTER makes it easy to describe large HiTER where the design comprises many state machines. The user describes the HiTER for each state machine and combines graphs to generate larger HiTER.

The property generation procedure for a graph or subgraph is explained as follows. Nodes and its outgoing edges are shown in Figure 1. The temporal properties for a PCI state machine are explained below. The state *FRAME* can be followed by only one of *TTwD*, *TTwoD* and *RETRY* according to the PCI specification. After *FRAME*, the DMA controller state machine may save the PCI data in local storage or check the flag from PCI data. *FRAME* can be followed by only one of *SAVE* or *CHK*. The outgoing edges

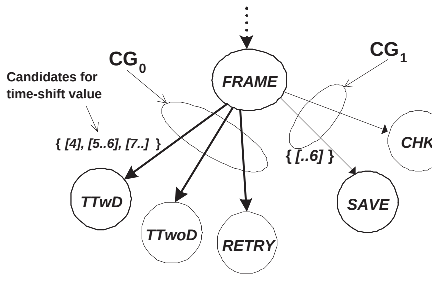

Figure 1: Concurrent edge groups of the node and time-shift value of the edge. Each edge has several candidates for time-shift value.

of *FRAME* can be grouped into a set of groups called CG (Concurrent edge Group)'s. In Figure 1, the parent node, *FRAME* has two CGs, i.e., CG_0 and CG_1. A node in HiTER may have one or more CGs so that several nodes may be child nodes of the current node. CG allows several events to occur concurrently after one event has occurred. CG has two distinct advantages over state machine or cross-product states. The first is that more than two events can occur concurrently after one event has occurred while only one state is allowed as the next state in the state machine and thus CG enables the user to describe the interaction between different state machines closely interacting with each other. The user may specify an edge between two nodes in different state machines so that the temporal relation between two state machines can be specified. Secondly, only valid cross-product states are explored in HiTER. The user specifies temporal relations between different state machines and then only possible, i.e., valid sequences of temporal properties according to the specification are generated. Figure 1 also shows the time-shift value for edge edge. If *TTwD* is selected for CG_0 while *SAVE* is selected for CG_1, the algorithm selects one of the candidates for time-shift value to make the temporal property for the child nodes. If *[5..6]* is selected, the generated property for *TTwD* becomes $P(\text{TTwD}) = P(\text{FRAME})$ *[5..6]* *TTwD*; where $P(n)$ is the property of node n. The property of *TTwD* is constructed hierarchically from that of *FRAME*.

If two nodes are joined to a single node, the validity of the time-shift value for the destination node is checked. In Figure 2, *TTwD* and *SAVE* are joined to *S_DMA*. *INIT* is the *root node*. *TTwD* and *SAVE* are dependent on *INIT*. The minimum time-shift value for *S_DMA* becomes 29 for the edge from *TTwD* to *S_DMA* but the maximum time-shift value for the edge from *SAVE* to *S_DMA* becomes 15. In this case, as there is no overlap between the two time-shift values, i.e., no valid time-shift value for *S_DMA* can exist. Thus, the property for *S_DMA* becomes empty.

Figure 2: Validity check of time-shift values. ***S_DMA*** **is the invalid node because there is no overlap between two time-shift values.**

The temporal property generation algorithm is shown in Figure 3. In the temporal property generation algorithm, BFS traverses HiTER and generates all possible properties. k is index number of a graph in specification of the design. In the main loop, FcmGen(k) generates all possible properties for graph k where an initial value of k is the index of top graph. $Omega_k$ is a set of properties generated in FcmGen(k). The generated properties at each iteration are saved in $Omega_k$. The iteration loop inside of "do" statement assigns the property to all nodes and selects one of the possible time-shift values for each edge. The iteration is repeated until no more different properties are generated. Initially, Π, the generated properties in the current iteration, is set to be empty. $P(n_i)$ which is the generated property of node n_i is also set to be empty for all nodes. The algorithm performs BFS on HiTER where n_s is the node visited during BFS. If n_s is a subgraph k', FcmGen(k') is run recursively to generate properties for graph k'. If subgraph k' is explored before, i.e., FcmGen(k') was executed at previous iteration, the generated properties for subgraph k' is reused. The return value of FcmGen(k'), generated properties of subgraph k', is assigned as concurrent edge group of n_s. In other words, the concurrent edge group of n_s becomes the set of generated properties from FcmGen(k') where the number of concurrent edge group is one. If n_s is a leaf node, its property of n_s becomes the element of Π. G denotes the concurrent edge group(CG) of node n_s. The algorithm selects one of the edges, e for G that is the next to the edge selected during the previous execution of FcmGen(). t is the next time-shift value of e and n_d is the destination node of n_s. If n_d was already explored, i.e., n_d is the destination of some other node, the validity of the time-shift value need to be checked. $P'(n_d)$ is the property that is assigned to n_d by the current selected edge and $P(n_d)$ is the property that is already assigned to n_d. In this case, the time-shift value of temporary property $P'(n_d)$ is compared with $P(n_d)$. If two time-shift values overlap with each other, then $P(n_d)$ becomes the logical ANDing of the two properties. The AND operation on two properties means that two properties should be satisfied. In this case, the end time of the composite property is the end time of the property that completes later. If there no valid time-shift value, the property of n_d becomes empty. After traversing all nodes, Π contains all the generated properties of leaf nodes. As explained before, the main loop executes FcmGen() iteratively to generate all possible properties of leaf nodes.

```
$k$ = index of top graph;
FcmGen(k) {
 $\Omega_k = \emptyset$;
 do {
  $\Pi = \emptyset$;
  foreach node $n_i$, $P(n_i) = \emptyset$;
  foreach node $n_s$ visited in BFS {
   if $n_s$ is unexplored subgraph $k'$, $WG$=FcmGen($k'$)
and continue;
   if $n_s$ is leaf node, $\Pi \leftarrow P(n_s)$ and continue;
   foreach CG $G$ in $WG$ of $n_s$ {
    $e \leftarrow$ select next edge in $G$;
    $t \leftarrow$ select next time-shift of $e$;
    $n_d =$ dest. node of $n_s$;
    if $n_d$ was already explored {
     $P'(n_d) = P(n_s)[t]\, n_d$;
     if valid, $P(n_d) = P(n_d) \& P'(n_d)$;
     else $P(n_d) = \emptyset$;
    } else $P(n_d) = P(n_s)[t]\, n_d$;
   }
  }
 $\Omega_k \leftarrow \Pi$;
 } while($\Pi \neq \emptyset$);
 return $\Omega_k$;
};
```

Figure 3: Temporal property generation algorithm. BFS algorithm traverses HiTER and generates valid properties.

4. EXPERIMENTS

We have applied HiTER property generation algorithm to the "8-channel PCI DMA controller (PDC8)". PDC8 is DMA controller that transfers data between main memory in PC and eight FIFOs. The verification environment and internal structure of PDC8 is shown Figure 4. C code was written to read and write data to the main memory through PLI interface connected to the bridge model. The bridge model receives commands from C program and communicates with main memory model and PCI bus. The bus arbitration logic inside of the bridge model selects either PLI interface or PCI bus as the master of the memory bus. The arbiter in PDC8 reads the status of all FIFOs, selects one of them and sends the index of selected FIFO to data transfer controller. The data transfer controller then computes the burst transfer length which length can be determined as the size of space in the main memory that can be accessed in burst mode or the number of available data in FIFO.

We have applied software simulation using VCS in verifying PDC8. We manually described the temporal properties as coverage metrics in VCS before using HiTER. The generated properties are written in OpenVera assertion properties and checked on VCS [10]. The randomly generated 100M PCI transaction were applied as test stimuli and the properties were checked. When we have prototyped PDC8

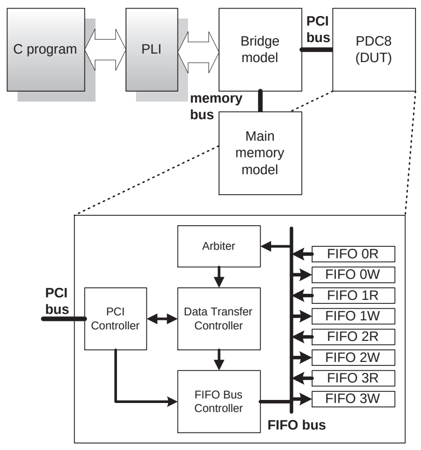

Figure 4: A design example of PDC8 which is an 8-channel PCI DMA controller that transfers data between the main memory and eight FIFOs.

using PCI board with Xilinx FPGA connected to PCI bus of Sun Blade 2000 workstation, there were still bugs related to the state machines of data transfer controller and FIFO bus controller. We have found that we omitted the temporal properties that are closely related to those two state machines. The manually written temporal properties are not sufficient to disclose all the bugs that are closely related to the temporal properties of several state machines. If we describe all cross-product states, the number of all cross-product states is 222300 which includes a large number of unreachable states where arbiter, data transfer controller, PCI controller and FIFO bus controller has 18, 25, 26, and 19 states, respectively.

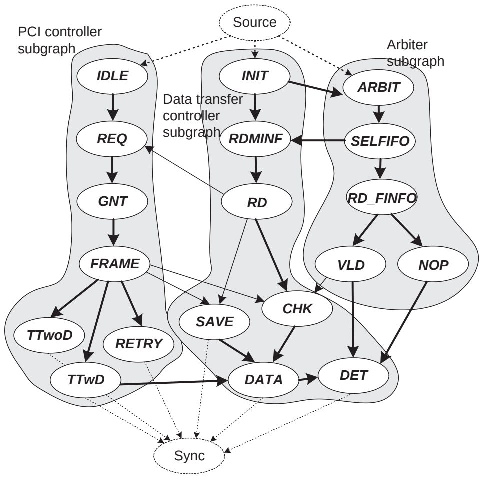

Figure 5: The HiTER for read operation of PDC8, which consists of three subgraphs, 19 nodes and 24 edges.

We described HiTER for PDC8 shown in Figure 5, which is composed of subgraphs, 19 nodes and 24 edges. There are three subgraphs in HiTER for PDC8, “PCI controller subgraph”, “Data transfer controller subgraph” and “FIFO bus controller subgraph”. The “PCI controller subgraph” describes the sequence of DMA operation of PCI controller. The “Data transfer controller subgraph” describes sequence of events for data transfer controller. The arbiter selects one of FIFOs and the data transfer controller initially reads the index of selected FIFO from arbiter in “RDMINF” node. It reads data from FIFO and transfer read data to PCI controller. After reading all data, it saves the incremented index to the main memory. The “arbiter subgraph” describes the sequence of event for arbiter. The arbiter selects one of FIFOs and reads the number of available spaces in FIFO from FIFO bus controller in “RD” node. “FRAME”, “RD”, “INIT” and so on have multiple concurrent edge groups. The time-shift values for edges are not shown here.

The statistics of properties for PDC8 is shown in Table 1. The number of generated properties is 3423 where all the properties are valid properties according to the specification. This amounts to only 1.5% of 222300 states in the cross-product model. In the experiment using HiTER, we discovered that automatic generation of temporal properties from HiTER makes all possible temporal events. The temporal properties related to bugs caused by the close operation among several state machines were also generated from the proposed coverage metric generation algorithm.

5. CONCLUSIONS

In this paper, the novel coverage metric generation methodology is presented based on the proposed HiTER which can describe the temporal relation between basic properties hierarchically. The proposed temporal property generation algorithm traverses HiTER and generates valid temporal properties automatically. The hierarchical structure of HiTER makes it easy to describe large HiTER where the design comprises many state machines. The generated properties have hierarchical structure to save the memory consumption by the property checking.

Table 1: The statistics of properties for PDC8.

	# States
# of graphs	4
# of nodes for PCI controller	6
# of nodes for data transfer controller	7
# of nodes for arbiter	5
Total # of nodes	18
# of edges	25
# of time-shift values	82
# of generated properties	3423

6. REFERENCES

[1] E. Emerson. *Handbook of Theoretical Computer Science*. Elsevier Science Publishers, Amsterdam, 1990.

[2] S. Fine and A. Ziv. Coverage directed test generation for functional verification using Bayesian networks. In *IEEE/ACM Design Automation Conference*, pages 286–291, 2003.

[3] R. Grinwald, E. Harel, M. Orgad, S. Ur, and A. Ziv. User defined coverage - a tool supported methodology for design validation. In *IEEE/ACM Design Automation Conference*, pages 158–163, 1998.

[4] A. Gupta, A. E. Casavant, P. Ashar, and X. G. Liu. Property-specific testbench generation for guided simulation. In *Asia and South Pacific Design Automation Conference*, pages 524–531, 2002.

[5] P.-H. Ho and et al. Smart simulation using collaborative formal and simulation engine. In *IEEE/ACM International Conference on CAD*, pages 120–126, 2000.

[6] Y. Hoskote, D. Moundanos, and J. A. Abraham. Automatic extraction of the control flow machine and application to evaluating coverage of verificaiton vectors. In *IEEE International Conference on Computer Design*, pages 532–537, 1995.

[7] D. Moundanos, J. A. Abraham, and Y. V. Hoskote. Abstraction techniques for validation coverage analysis and test generation. *IEEE Transactions on Computer*, 47(1):2–14, Jan. 1998.

[8] J. Ruf, D. W. Hoffmann, T. Kropf, and W. Rosenstiel. Simulation-guided property checking based on multi-valued AR-automata. In *IEEE Design Automation and Test in Europe*, pages 742–748, 2001.

[9] K. Shimizu and D. L. Dill. Deriving a simulation input generator and a coverage metric from a formal specification. In *IEEE/ACM Design Automation Conference*, pages 801–806, 2002.

[10] Synopsys. *OpenVera Language Reference Manual*, 2.2 edition, Apr. 2002.

[11] S. Ur and Y. Yadin. Micro architecture coverage directed generation of test programs. In *IEEE/ACM Design Automation Conference*, pages 175–180, 1999.

[12] S. Ur and A. Ziv. Off-the-shelf vs. custom made coverage models, which is the one for you? In *International Symposium on Software Testing and Analysis Review*, 1998.

[13] A. Ziv. Cross-Product functional coverage measurement with temporal properties-based assertions. In *IEEE Design Automation and Test in Europe*, pages 834–839, 2003.