Symbolic Functional Vector Generation for VHDL Specifications

Fabrizio Ferrandi [#] Franco Fummi [‡] Luca Gerli [#] Donatella Sciuto [#]

[#] Politecnico di Milano
Dip. di Elettronica e Informazione
Milano, ITALY

[‡] Università di Verona
Dip. di Informatica
Verona, ITALY

Abstract

Verification of the functional correctness of VHDL specifications is one of the primary and most time consuming task of design. However, it must necessarily be an incomplete task since it is impossible to completely exercise the specification by exhaustively applying all input patterns. The paper aims at presenting a two-step strategy based on symbolic analysis of the VHDL specification, using a behavioral fault model. First, we generate a reduced number of functional test vectors for each process of the specification which allow complete code statement coverage and bit coverage, allowing the identification of possible redundancies in the VHDL process. Then, through the definition of a controllability measure, we verify if these functional test vectors can be applied to the process inputs when it is interconnected to other processes. If this is not the case, the analysis of the non-applicable inputs provides identification of possible code redundancies and design errors. Experimental results show that bit coverage provides complete statement coverage and a more detailed identification of possible design errors.

1 Introduction

Functional verification of VHDL specifications is one of the most time consuming tasks that a designer has to perform before synthesis. The aim is to verify that a description has the intended behavior: this can be accomplished by identifying a series of meaningful simulation vectors. However, there are some drawbacks to this approach. Correctness can only be guaranteed if an exhaustive application of all possible input values is performed: this task is infeasible for actual designs. Formal verification techniques are promising but, at this stage, they are not widely used in industrial environments, because they are still applicable to small designs only. Random vector generation is relatively easy, however it does not usually guarantee the verification of all functionalities of the design. Therefore, in most cases, designers perform functional vectors generation based on their knowledge of the specification; this allows the definition of a meaningful set of vectors but without any guarantee to thoroughly exercise the specification.

A different approach bases the definition of functional vectors generation on some metrics, often derived from software engineering techniques. The metrics most applied consider statements coverage and path coverage [15]. However, while statement coverage is attainable, complete path coverage is infeasible, since the path in a VHDL model grows exponentially with the size of the description.

Aim of this paper is to propose a new functional vectors generation strategy, based on *bit coverage* instead of statement or selective path coverage [13, 3]. This allows a more precise verification of the correctness of the specification, covering anyway all statements and the most significant paths of the VHDL model. Our approach can analyze:

- Behavioral VHDL specifications, that is purely algorithmic descriptions, before high-level synthesis;
- Structural VHDL specifications, described as a set of interconnected VHDL processes, i.e. RT-level descriptions of data-paths.

The approach we propose is based on test pattern generation, performed considering a VHDL-based fault model and an implicit technique to perform test vectors identification. This allows coverage of a larger number of design errors than pure statement coverage and even than tag-based approaches [8, 9]. A two-steps bottom-up procedure has been defined. First, test pattern generation is performed on each single process in isolation, by injecting behavioral faults in the VHDL code. Then, considering the structural description of the interconnected processes, a controllability analysis is performed to identify the activability of these test patterns at the inputs of the process, starting from the primary inputs. The results of this approach are the following:

- The set of test patterns that cover all bits and therefore all statements of each single process in isolation;
- The applicability of all test vectors to the single processes, through a controllability analysis of the interconnected processes;
- Redundancies in the VHDL code, if any. Two types of redundancies can be identified in a VHDL description representing the specification of a design: redundancies derived from the **over specification** of a module, e.g. behaviors of a module which can never happen; or redundancies derived from the **interconnection** of irredundant modules, e.g. behaviors of a module which cannot be activated or observed.

The proposed methodology is composed of the following steps.

- **VHDL to BDD translation** (Section 2). VHDL design entities are directly converted into BDD based descriptions. This operation produces often less complex BDDs in relation to the construction of BDDs starting from the corresponding gate-level descriptions. In fact, a smaller number of registers are involved in a VHDL description with respect to its implementation, since synchronization registers are not included [6]. Moreover, the device partitioning performed by the designer usually produces unrelated functionalities which depend on few input/output variables. This is a good criterion for building small BDDs and the same operation is hard to be performed on a flat gate-level description. However, whenever the size of BDDs increases considerably (i.e. in case of circuits including large multipliers) we adopt approximate techniques, both for test generation [10] and controllability analysis.

- **Single process VHDL functional vectors identification** (Section 3). The functional vectors identification is based on test pattern generation techniques [4, 12]. A fault model based on VHDL signal/variables stuck-at is used to identify functional vectors and possibly code redundancies. The adopted fault model based on [11] has been proved to well model RT and gate-level stuck-at faults. Moreover, groups of faults are concurrently inserted and symbolically simulated instead of explicitly injecting one fault at a time (e.g., [18]). A VHDL fault is certified as redundant if there is not at least one vector distinguishing the fault-free and faulty BDD based representations.

- **Interconnected processes analysis** (Section 4). Constraints related to the interconnections of modules are taken into account by implicitly computing *controllability* sets, a concept related to controllability don't care sets [1]. Such sets capture the information on all vectors that can reach a module from the primary inputs of the circuit. This kind of sets is used to constrain test generation for each module. The approximation of this analysis concerns faults observability and the conflict between controllability and observability that are not considered in this implementation. However, this approximation allows the analysis of real size datapaths. Moreover, if the BDD description of an isolated module is unmanageable, a new approximated analysis is applied.

By using the proposed methodology, functional tests can be easily generated, guaranteeing the full code coverage for each single process, the exact identification of redundancies and difficult to test faults in the code, providing detailed information to detect design errors in the VHDL model. A byproduct of this technique is that the test vectors generated can be proved as very efficient in terms of single stuck-at fault coverage on the gate-level implementation [10].

2 VHDL and fault models

The considered VHDL description is a network of interacting processes which may include signals feedbacks. The VHDL specification is parsed into an internal description, where all VHDL templates inferencing registers are recognized and combinational and sequential behaviors are separated [6]. Each VHDL process is internally modeled as a combinational behavioral process, which may include data dependent loops, or as a sequential process representing a simple register. Registers are considered as delay elements with a transparent functionality in the controllability analysis.

Behavioral VHDL descriptions accepted represent the typical algorithmic description in which the time variable is not considered, since it is introduced by the high-level synthesis tool. Protocol descriptions and explicit controllers, usually expressed in terms of extended finite state machines are not considered here, since they implicitly assume a basic timing of the operations. Most of the VHDL behavioral constructs accepted by commercial high-level synthesis tools are supported by the VHDL analyzer. Moreover, the tool accepts also functions and procedures, which are widely used by designers in this specification phase. Main constructs excluded are the `after` statement and the inclusion of multiple `wait` instructions in the description. This kind of description models behavioral descriptions and RTL data-paths (more information can be found in [2]).

2.1 Behavioral Fault Model

The fault model adopted is associated with each single VHDL process. The behavioral fault model considers those failure modes of VHDL closely related to RT-level stuck-at faults [10]. We assume a single-fault model in the process. In particular, the fault model includes: **bit failures** (each variable, signal or port is considered as a vector of bits. Each bit can be stuck-at zero or one) and **condition failures** (each condition can be stuck-at true or stuck-at false, thus removing some execution paths in the faulty representation). The fault model excludes explicitly the incorrect behavior of the elementary operators (e.g., $+, -, *, \ldots$). Only single bit input or output faults are considered, thus including all operator's equivalent faults.

2.2 VHDL to BDD

Different techniques have been proposed in literature to translate VHDL descriptions into the corresponding BDDs [16, 17]. Such a translation is mainly required by formal verification tools (usually based on BDDs manipulation) which have to accept VHDL descriptions as input. In this work we use a similar approach to VHDL to BDD translation, aiming however at building only a partial BDD-based description of the specification, instead of the complete input/output mapping necessary for formal verification [16].

3 Single Process Analysis

Each process of the network is analyzed in isolation to identify the functional test vectors allowing to deeply exercise the specification. This analysis is based on the following steps.

The construction of the fault-free BDD and faulty BDD descriptions is performed concurrently, by injecting behavioral faults during the VHDL to BDD translation, by applying the algorithm introduced in [10].

For each fault $\mathcal{F}$, the VHDL to BDD translation performs a statement by statement symbolic execution and injects the considered fault during the symbolic execution (or efficiency reasons, a group of faults is injected at each step). After injection, the fault is traced during the translation of the following statements, thus allowing the verification of its propagation to the primary outputs.

For all those faults for which the VHDL to BDD translation is completed, the functional test vectors identification procedure must verify if it is possible to generate a test vector for such a fault. Therefore, for each output bit, we compute the fault-free f and the faulty BDD $f_{\mathcal{F}}$. If the function $TV = f\ bdd\_xor\ f_{\mathcal{F}}$ is different from the boolean function zero for at least one bit of one output port of the specification, then the fault $\mathcal{F}$ is testable and the test vector corresponds to an assignment of the input values which satisfies the function TV. Such an assignment allows the activation of the signal, variable or condition bit in which the fault is injected and then the propagation of its effects to the outputs. This therefore guarantees functional verification of a path from inputs to outputs. Otherwise, the fault is defined to be *behaviorally* redundant [10].

If the BDD associated with the output function is manageable, the test generation algorithm is always able to classify a fault as redundant or testable. Unfortunately, the size of BDDs for complex specifications can become too large to allow the successful termination of the redundancy identification algorithm. To solve the input domain problem we propose to handle inputs by partitioning the entire inputs domain into subsets which can be easily managed. In fact, we randomly decompose the behavioral specification in several BDD-based descriptions with reduced size. Hence, in this case the problem of redundancy identification consists now in a multiple application of the algorithm to sub-problems of affordable size, whose solution can be found very efficiently. The fault is classified as redundant if and only if no test vectors can be found for all sub-problems [10].

Whenever all these strategies fail to provide a manageable BDD description of a process, faults that remain untested are classified as difficult to test. Our approach provides the exact code location of the fault, thus allowing the designer to inspect that portion of code in order to identify a potential design error.

4 Interacting Processes Analysis

To take into account the interaction among processes, we compute a new type of controllability set for each process. The concept of controllability don't care set has been defined in logic synthesis [1] as the set of primary input combinations that never occur. Two sets can be identified: *external* controllability don't care set (CDC^{ext}) and *internal* controllability don't care set (CDC). The first one is the set containing the input patterns never produced by the environment at the network's inputs, while the second one refers to internal nodes. To compute the internal CDC set the network is traversed by considering different cuts moving from the inputs to the outputs. A CDC set is defined for each cut, corresponding to the bit vectors never applied to the nets traversed by the cut. Similar considerations can be made for the observability don't care (ODC) set. CDC and ODC sets are used for logic optimization and for the synthesis of testable circuits at gate level [1]. The concept of CDC set can be exploited in the analysis of interacting VHDL processes, for functional verification, since it provides which bit vectors are applicable to a module.

4.1 Controllability Set Computation

Let us define the *controllability set* (CS) of a process as the set of all bit vectors that can be applied to the process itself, given the interconnection topology and the functions of the other processes in the specification. The controllability set is the complement of the CDC set.

Computation of the controllability set of a process is based on the *image* computation procedure adopted to solve the implicit state enumeration problem for finite state machines [7]. The CS computation of each process is based on the SIAM algorithm proposed by Coudert et al. in [7], coupled with the use of subtree recombination [5] and extended with the possibility of saving on file intermediate BDDs.

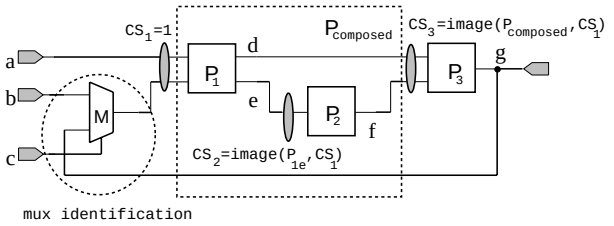

Figure 1. Example of CS computation.

The CS computation is performed by considering one process P_i at a time and starting the analysis first from processes depending on primary inputs only, followed by processes depending on already analyzed processes. In case non reconvergent paths exist for the process P_i, its controllability set is given by the Cartesian product between the set of codomains associated with the preceding processes. Otherwise, the controllability set of P_i is given by the codomain associated with the composition of the processes belonging to the reconvergent paths.

In case the network of interacting processes has cycles, all signals associated with the loops are identified. Then, components behaving as multiplexers are identified and signals connected to such components are transformed into primary inputs in order to open cycles. In fact, if multiplexers or equivalent processes can be identified, cycles can

be virtually broken by driving the control inputs of the multiplexer in the proper configuration. This operation allows the computation of controllability sets for all processes, so that such controllability sets do not depend on values propagating inside the cycles. An approximate computation is performed in case no multiplexers or equivalent processes can be identified: a cycle is cut and the controllability set related to the cut is set to *full* controllability. This implies that the resulting controllability set contains the exact one.

Figure 1 shows an example of controllability set computation: the exact formula for the computation of the controllability set $CS(P_i)$ is reported near each process P_i.

4.2 Design Errors Identification

The combined analysis of the test vectors identified considering the behavioral fault model and the controllability sets computed for each process allows us to identify those faults caused by the interconnections among processes. These faults (either redundant or difficult to test) can occur either on the input ports or on internal process signals or variables. In the first case, they are related to a possible incorrect connection specification. In the second case they show a possible over specification of the process, when interconnected.

A fault in a process P_i is classified as redundant if no test vector exists belonging to $CS(P_i)$. This analysis is performed by building the fault-free BDD representation and the faulty BDD of process P_i considering only the subdomain corresponding to $CS(P_i)$. The fault-free and faulty BDDs thus obtained are used as described in Section 3 to perform functional tests identification. If a functional test does not belong to the input sub-domain corresponding to the controllability set, this vector cannot be propagated from the primary inputs to the process, and therefore the associated faults cannot be activated. Redundant faults can be inserted in the code, in order to correct the specification.

The computation of controllability sets is performed before identification and insertion of redundant faults and it does not require a re-computation after the removal of each redundant fault. In fact, the insertion of redundant faults into a process P_i does not modify the controllability sets of the other processes.

Lemma 1 *Let P_j be a process depending on process P_i so that $CS(P_j) = image(P_i, CS(P_i))$. The insertion of a redundant fault in P_i does not alter $CS(P_j)$.*

Let P_i' be process P_i after the insertion of a redundant fault. A fault has been defined as redundant if no test vectors exist which differentiate P_i' from P_i. This fact implies that $image(P_i', CS(P_i')) \equiv image(P_i, CS(P_i))$ and, from the definition of controllability set, $CS(P_j) = image(P_i', CS(P_i'))$. Thus, the controllability set of P_j is not altered by the injection of a redundant fault. Equivalent reasonings can be applied to the general case of CS computation in presence of reconvergent paths and cycles.

5 Experimental Results

The proposed methodology has been implemented by using the LEDA LVS Libraries for VHDL parsing, the CUDD binary decision diagrams library for BDD manipulation and the Berkeley VIS environment for interface and utilities. The current implementation is composed of more than 120K C code lines. Experiments have been run on a SunUltra 30/248 with 1Gbyte RAM.

The first set of experiments concerns single-process VHDL descriptions. We selected well known high-level synthesis examples [14] to verify the effectiveness of the proposed *bit coverage* model with respect to the statements coverage methodology. Functional test patterns have been generated by adopting the proposed error model (stuck-at on each VHDL signal and condition) and by injecting only one fault for each VHDL instruction. This second method guarantees the activation of all instructions and it can be associated with the statements coverage methodology.

Name	Inputs	Outputs	Data size	Modules	VHDL lines
diffeq	162	96	32	27	613
ellipf	258	256	32	37	866
fir	768	32	32	58	859
gcd	66	32	32	14	322

Table 1. Characteristics of single-process VHDL circuits.

Table 1 shows the characteristics of the analyzed benchmarks in terms of input and output bits number, data size of the signals involved, number of modules and VHDL lines of the VHDL description of the benchmark at the RT level. RT level descriptions are considered the starting point of the synthesis process, while the single-process behavioral descriptions represent the specification.

		Statement Coverage			Bit Coverage		
Name	Err.	Vec.	CPU	%Co.	Vec.	CPU	%Co.
diffeq	24196	326	503	92.4	585	3256	99.3
ellipf	20764	196	74	98.7	265	1624	99.2
fir	23672	313	2930	93.8	490	4803	99.0
gcd	4714	290	2562	75.6	468	2264	97.3
average	18337	281	1517	90.1	452	2987	98.7

Table 2. Comparison on error coverage.

We assume that the designer generates functional test patterns at the behavioral level (specification) to verify the correctness of the RTL description (implementation) that has been manually designed. This method is commonly used in industrial environments since behavioral synthesis tools are not widely adopted yet due to the low predictability of their results. Random design errors have been inserted in the RT level descriptions and functional vectors have been simulated by using a commercial RTL fault simulator. Table 2 shows the results of this comparison. Random application of test vectors achieves on average 10% coverage. The statement coverage method does not guarantee to cover an acceptable number of design errors (90.1% on average). On the contrary, the proposed VHDL fault model covers the majority of errors (98.7% on average) with some cases higher than 99%. On the other hand, *bit coverage* requires on average twice the execution time (in seconds) than statement coverage. Concerning interacting processes, we investigated the ability of the pro-

posed methodology to identify specification redundancies by using controllability sets. VHDL descriptions, designed at the RT level, have been selected.

Name	I.	O.	Mod.	V.Lin.	CPU	Err.	Red.Fau.
add-sub_fp	33	16	26	1499	4969	3776	241
am2910	22	16	33	617	4965	4385	589

Table 3. Specification redundancies identification.

The first five columns of Table 3 show the characteristics of the analyzed benchmarks: `add-sub_fp` is a 16-bit floating point adder/subtractor and `am2910` is the well known processor. The time for the computation of the controllability sets and for redundancies identification is then reported. The last two columns report the number of examined errors and the number of redundant faults identified by applying the controllability analysis.

The analysis of each redundant fault identified two important design errors. Concerning the `add-sub_fp` description we found that the most significant bit of the `mantissa` normalization function was always stuck-at, thus avoiding its correct functionality. Concerning the `am2910` description, the `POP` operation was incorrectly described, thus behaving as a `PUSH` operation. This made redundant the faults on the instruction controlling the increment/decrement of the stack pointer.

6 Concluding Remarks

This paper presented a new approach for functional vector generation based on a very precise behavioral error model and a controllability metric for coverage evaluation. The error model defined allows the generation of vectors which aim at covering all bits of signals, variables and conditions inside a VHDL process, while the controllability metric provides a verification of the interactions among processes. The result of this generation, in addition to the set of functional vectors, is also the identification of those parts of the code that may include design errors. In fact, those errors which are not covered by the test generation approach can be either behaviorally redundant faults, or hard to test faults, which my identify design errors.

Further work will be directed in the extension of the VHDL descriptions analyzed and to the combined analysis of controllability and observability measures. Observability metrics have been already presented in [9], however our approach aims at defining observability sets in a similar manner as the controllability sets defined here.

Acknowledgments

We would like to sincerely acknowledge Dr.Rajski and Dr.Kassab of Mentor Graphics for their contribution to this work.

References

[1] K. Barlett, R. Brayton, G. Hachtel, R. Jacoby, R. Rudell, and Sangiovanni-Vincentelli. Don't care set specification in combinational and synchronous logic circuits. *IEEE Trans. on CAD/ICAS*, 7:723–739, June 1988.

[2] C. Bolchini, G. Buonanno, F. Ferrandi, F. Fummi, D. Sciuto, M. Bombana, P. Cavalloro, and P. Borrego. Definition of methodology for testability analysis at the RTL and CDFG levels requirement specs for functional pattern quality evaluator. *Technical Report of Deliverable 2.3.A, Esprit project n.20616 - REQUEST*, 1996.

[3] A. Chandra and V. Iyengar. Constraint solving for test generation: A technique for high-level design verification. *Proc. IEEE ICCD*, pages 245–248, 1992.

[4] K. Cheng and A. Krishnakumar. Automatic generation of functional vectors using the extended finite state machine model. *ACM Trans. on design Automation of Electronic Systems*, 1(1):57–59, Jan. 1996.

[5] H. Cho, G. Hachtel, S. Jeong, B. Plessier, E. Schwarz, and F. Somenzi. ATPG aspects of FSM verification. *Proc. IEEE ICCAD*, 1990.

[6] D. Corvino, I. Epicoco, F. Ferrandi, F. Fummi, and D. Sciuto. Automatic VHDL restructuring for RTL synthesis optimization and testability improvement. *Proc. IEEE ICCD*, pages 587–596, 1998.

[7] O. Coudert, C. Berthet, and J. Madre. Verification of sequential machines using boolean functions vectors. *Proc. Workshop on Applied Formal Methods for Correct VLSI Design*, 1989.

[8] F. Fallah, S. Devadas, and K. Keutzer. Functional vector generation for HDL models using linear programming and 3-satisfaiability. *Proc. ACM/IEEE DAC*, pages 528–533, 1998.

[9] F. Fallah, S. Devadas, and K. Keutzer. OCCOM: Efficient computation of observability-based code coverage metrics for functional verification. *Proc. ACM/IEEE DAC*, pages 152–157, 1998.

[10] F. Ferrandi, F. Fummi, and D. Sciuto. Implicit test generation for behavioral VHDL models. *Proc. IEEE ITC*, pages 436–441, 1998.

[11] S. Ghosh and T. Chakraborty. On behavior fault modeling for digital designs. *Journal of Electronic Testing: Theory and Applications*, 2:135–151, 1991.

[12] N. Giambiasi, J. Santucci, A. Courbis, and V. Pla. Test pattern generation for behavioral descriptions in VHDL. *Proc. EuroVHDL*, pages 228–235, 1991.

[13] R. Grinwald, E. Harel, M. Orgad, S. Ur, and A. Ziv. User defined coverage - a tool supported methodology for design verification. *Proc. ACM/IEEE DAC*, pages 158–163, 1998.

[14] 1991 and 1992 high level synthesis benchmarks. *ftp://mcnc.mcnc.org/pub/benchmark/HLSynth91[92].*

[15] B. Marick. The craft of software testing. *Prentice-Hall, Englewood Cliffs*, pages 221–229, 1995.

[16] S. Minato. Generation of BDDs from hardware algorithm descriptions. *Proc. IEEE ICCAD*, nov 1996.

[17] R. Herrmann and H. Pargmann. Computing Binary Decision Diagrams for VHDL Data Types. *Proc. European Design Automation Conference (EURO-DAC94)*, pages 578–585, September 1994.

[18] T. Riesgo, Y. Torroja, E. De La Torre, and J. Uceda. Quality estimation of test vectors and functional validation procedures based on fault and error models. *Proc. IEEE DATE*, pages 955–956, 1998.