Functional Verification of RTL Designs driven by Mutation Testing metrics

Youssef Serrestou, Vincent Beroulle Chantal Robach
LCIS-INPG, 50 rue Barthélémy de Laffemas, 26902 cedex, Valence
firstname.name@esisar.inpg.fr

Abstract—. **The level of confidence in a VHDL description directly depends on the quality of its verification. This quality can be evaluated by mutation-based test, but the improvement of this quality requires tremendous efforts. In this paper, we propose a new approach that both qualifies and improves the functional verification process. First, we qualify test cases thanks to the mutation testing metrics: faults are injected in the Design Under Verification (DUV) (making DUV's mutants) to check the capacity of test cases to detect theses mutants. Then, a heuristic is used to automatically improve IPs validation data. Experimental results obtained on RTL descriptions from ITC'99 benchmark show how efficient is our approach.**

Index Terms— **Automatic Test Bench Generation, Functional Verification, Mutation-based Testing.**

I. INTRODUCTION

The design methodology of complex integrated digital systems is based on assembling various soft or hard IP (Intellectual Properties) cores. A system can be seen as a network of interconnected IPs. So, the confidence in the global system depends on the quality of those embedded components. In addition, the level of confidence in these IPs directly depends on the quality of their functional verification. Nowadays, this verification has become crucial because of the reuse of the same IPs in many different SoCs or applications. In fact, if there is one (or more) design fault in one IP core, this fault is very likely to appear at least in one application. Hence, all possible IP functionalities should be carefully verified. In addition, IPs are provided from different companies which do not use the same quality metrics and validation processes. So, a standard methodology to evaluate and improve IP qualities should be defined.

The verification process generally represents an effort of 70% of the whole design effort [1, 2]. Completing the verification process on time is a challenge for many verification teams. These teams must generate efficient test cases and know when they have done enough verification efforts.

Our work targets the following problems:

- To measure the design verification quality; this is called *verification qualification*.
- To help verification engineers deciding when to stop the verification procedure.
- To show verification engineers which parts in the DUV are not at all or enough verified.
- To help the verification team improving their test plan or test data in order to increase faults detection
- To automate functional verification.

The rest of the paper is organized as follows: the next section gives an overview of our approach; section 3 presents the software platform used to perform verification qualification and improvement; section 4 presents the experimental results obtained so far, discusses these, results and lays the bases for future works.

II. QUALIFICATION AND OPTIMIZATION OF VERIFICATION DATA

When formal approaches are impossible due to design complexity, simulation-based verification, associated to qualification metrics, remains the most efficient technique. Thanks to fault oriented test data evaluation, the quality of the verification process can be measured respectively to the following three metrics:

- *Fault activation (or weak mutation* [3]*)*: verification data are evaluated by their capacity to generate a mismatch between fault free description and faulty descriptions. This mismatch is directly observed after the faulty line (i.e. the line in the faulty description affected by the fault injection).
- *Fault propagation (or firm mutation* [3]*)*: verification data are evaluated by their capacities to activate a mismatch and to propagate it from its original location (the faulty line) towards an output of the faulty block; this faulty block being the process (i.e. a VHDL group of sequential instructions) concerned by the fault injection.
- *Errors detection (or hard mutation* [3]*)*: fault propagation is observable by the way of at least one primary output.

Classical coverage metrics (i.e. functional, code, branch, and expression coverages) only consider fault activation without considering fault propagation toward primary outputs. Observability-based fault models [10] have been proposed to overcome this weakness. In the *OCCOM* (Observability-based Code Coverage Metrics) approach [10], faults called tags are injected on each variable assignment to add a positive or negative offset to the correct signal value. The propagation of these tags to observable signal is analysed using a set of static propagation rules. This observability-oriented metrics encounters difficulty with complex control flow paths. Either complex control flows have not been considered or only a small subset of possible control flow paths is considered. On the contrary, our qualification approach is applicable on complex designs. Moreover, our method provides a metrics: the *Mutation Score* (MS). This metrics allows verification engineers to measure and assess functional verification quality. Once a preliminary functional verification has been done using traditional tools (for example, [*4*]), the goal of our approach is to improve the verification quality by detecting all remaining induced errors in the DUV. For each fault, both activation and propagation must be performed to achieve detection. So, the

initial verification data must evolve to achieve maximal errors detection thanks to test sequences combining these two characteristics.
This problem leads us to develop the following strategy. Only test sequences with the best activation or propagation characteristics are selected. Then, these sequences are intelligently mixed up in order to build new sequences inheriting these two characteristics. This approach leads not only to increase faults activation and faults propagation but of course to achieve faults detection. The selection of sequences performed in our approach is inspired from Genetic Algorithm (GA). However, the way these sequences produce new sequences follow deterministic rules. In the following, we will explain in details which rules for mixing up sequences lead to the best results.

III. Software Platform for IP verification qualification And Improvement

Our software platform targets both functional qualification based on mutation test metrics and test data improvement. Figure 1 represents a simplified architecture of this platform: the static step corresponds to the generation of mutants through the instrumentation of the original code and the dynamic step corresponds to the test cases evaluation and optimization. Our work does not include the entire verification process which requires the definition of an Oracle and a Responses Checker, but concerns the qualification and generation of the verification test sequences only.

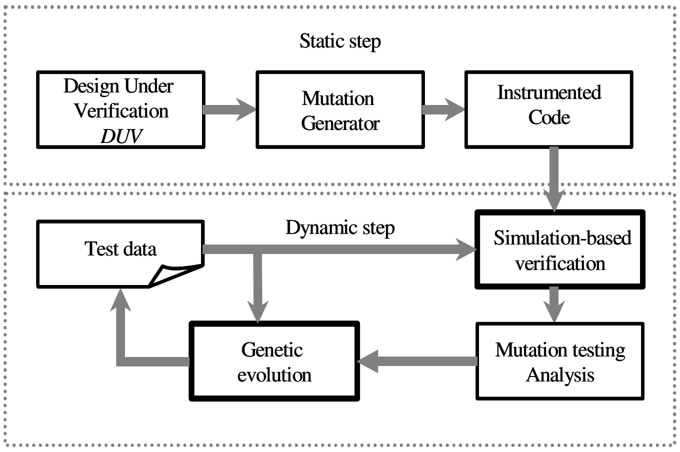
Static step
Design Under Verification *DUV*
Mutation Generator
Instrumented Code
Dynamic step
Test data
Simulation-based verification
Genetic evolution
Mutation testing Analysis

Figure 1: Simplified architecture of the Software Platform for test sequences qualification and improvement

The design Under Verification (DUV) in Fig. 1 is a VHDL RTL description. Initial test sequences associated with this description allow to simulate this description and to perform a first mutation testing analysis. The mutation testing evaluation process is described below. However, when these initial test data do not exist, we use random generator to create a first test sequences. Fault activation, propagation and detection measurements resulting from mutation testing evaluation process are used to select and to improve the initial test sequences. Then, new test sequences can again be evaluated and modified. This iterative process lasts until the Mutation Score is high enough.

A. *Mutation Testing*

Mutation testing was originally intended to locate and to expose weaknesses in test data [5]. It was proposed in [5] as a technique for unit software testing. In these works, the aim of mutation testing is to measure the efficiency of a test set to exercise the different functions of a program. But, this measure can also be used to generate test cases selecting only test data that are mutation adequate. The test data generated with this approach meet most of design validation criteria such as statement coverage, branch coverage, condition coverage….
A mutation generator creates a set of faulty versions (called mutants) from the original description by injecting one fault per version. These faults, i.e. "small" and syntactically correct modifications of the original instructions, are classified with the help of mutation operators. For VHDL descriptions, a set of mutation operators has been defined in [7].
Through mutant simulation, this approach leads to a metrics called the Mutation Score (*MS*). This metrics qualifies the Test Set (TS) with respect to a program P. Before defining this *MS*, let us first define *killed mutants* and *equivalent mutants*.
A killed mutant is a mutant that can be distinguished from the original program because there is at least one sequence in TS that, when applied on the inputs of the original program and the mutant, results in different outputs. An equivalent mutant cannot be distinguished from the original program whatever the simulated input data. The mutation score MS relatively to a test set TS and a program P is computed as in Eq. 1:

$$MS(TS, P) = 100 * \frac{K}{(M - E)} \qquad \textbf{Eq.1}$$

where M is the number of generated mutants, K the number of killed mutants and E the number of equivalent mutants.
In practice, the number of equivalent mutants E is generally unknown. In our following experimental results, these numbers have always been considered null. This assumption involves that in our results the MS has always been under-evaluated. Numerous works try to find a solution to equivalent mutants detection [8]. These works could be coupled to our work to achieve more accurate MS and to decrease the overall simulation time. In fact in our case, the test data generation process is repeated even for equivalent mutants for whom no detecting sequence exists.

B. *Functional Validation Data improvement*

To automate validation data generation, we adopt a strategy based on a deterministic crossover and mutation of the best sequences. As this strategy is inspired from genetic evolution (but it is functionally different), we will use Genetic Algorithm terminology to explain our approach.
GAs are generally used to solve similar problems to our validation improvement problem. For example, these algorithms have been successfully used for FPGA mapping, standard cells placement or for ATPG problems [9,11]. Initially, a population of individuals or chromosomes is randomly created. A *fitness value* is then attributed to each individual of the population. After the whole population has

been evaluated, genetic operations are performed over the individuals to create a new population. This process is repeated over a number of generations until a certain level of quality of the population or the max simulation time is reached. In our case an individual is a sequence of binary vectors applied on consecutive clock cycles starting from the reset state.

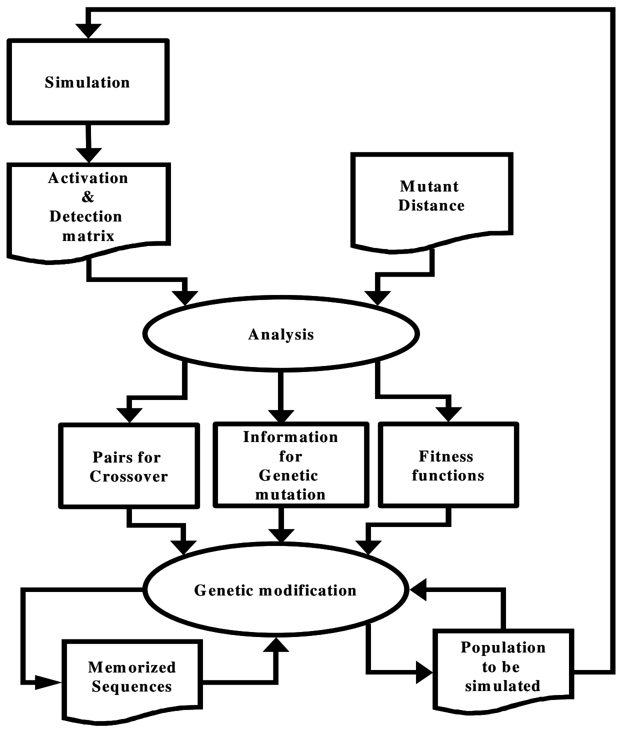

Figure 2: The flow of the developed strategy

Figure 2 shows the main operations performed to generate validation sequences. It relies on weak, firm and hard Mutation Score to compute the fitness function and to guide genetic operators. So the RTL-code of the DUV is instrumented in order to track the evolution of internal points during simulations. With this spy method, we measure activation and propagation of the injected faults. At the end of the simulation, we extract information showing relations between sequences and mutants (*Activation* and *Detection Matrix*). This information is analyzed and completed with the distances between mutants. The analysis of this information allows to evaluate each sequence and to guide the genetic modification of these sequences. The following subsections describe in details each phase of this process.

1) Initial population

Experiments show that a preliminary functional verification reaches more than 50% of the Mutation Score. But improving this test quality implies a large supplementary testing effort. Our approach provides a pragmatic way to automatically improve the first verification sequences in order to achieve better quality with reduced efforts. We compare, as it is shown in Fig. 3, the results achieved by our approach when the initial population is randomly generated and when this population is supplied by verification engineers. Obviously, this last contains more information. This information can be exploited to achieve high mutation score. Thus, numerous mutants detected by way up this population are not detected with a random initial population.

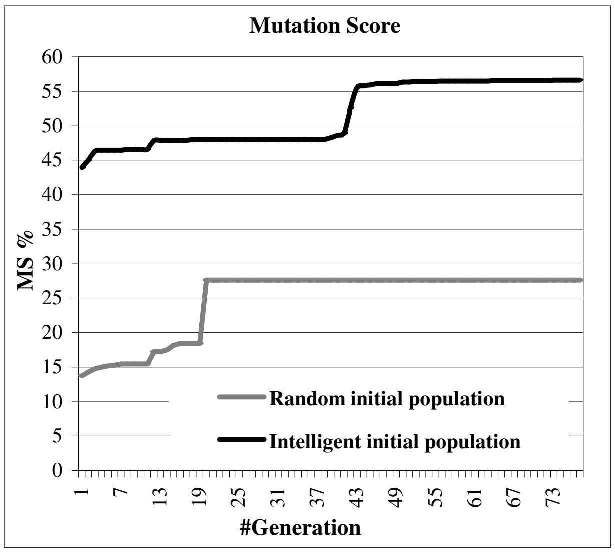

Figure 3: Comparison between Intelligent and Random initial populations

2) Fitness calculation

Sequences are evaluated by their capacities to distinguish the behaviour of faulty descriptions from fault-free one. This differentiation can be observed from internal signals (firm mutation) or directly from primary outputs (hard mutation). So the fitness function is divided in two parts:

$$f_{firm}(S_i) = \frac{1}{l_i}\sum_{j=1}^{N}\frac{a_{ij}}{A_j} \qquad (2)$$

$$f_{hard}(S_i) = \frac{1}{l_i}\sum_{j=1}^{N}\frac{d_{ij}}{D_j}$$

Where a_{ij} is the number of times that a sequence S_i has involved internal differences (activations) between DUV behavior and mutant M_j. A_j is the number of times that internal signals or variables of mutant M_j are differentiated from the corresponding signals or variables in the fault-free description. Thus, A_j indicates the degree of facility to activate/propagate mutant M_j. Where d_{ij} is the number of times that a sequence S_i has propagated to a primary output a difference between DUV behaviour and mutant M_j. D_j is the number of times that primary outputs of mutant M_j are differentiated from their corresponding outputs in the fault-free description. Thus, D_j indicates the degree of facility to detect mutant M_j. N is the number of mutants.

The population size and the individual size (maximum number of clock cycles between two re-initialisations) have great impacts on the quality and rapidity of the approach. With many sequences or/and long sequences the probability to detect mutants growths, unfortunately the simulation time growths also. These parameters depend on the complexity of the description under verification. In our approach, fitness calculation and genetic operations take into account the sequence size l_i in order to generate optimal verification sequence. Moreover, for this reason, the initial population consists of sequences with different lengths; the average length depends on the DUV. For optimization, we take into account the sequence length l_i, in order to differentiate the sequence that have the same activation or detection fitness.

3) Selection strategy

During each generation, we memorize the new sequences that detect the remaining mutants. This selection is deterministically performed by using detection fitness function. The memorized sequences form the final population that will be used for final verification when the DUV is compared to the reference model. In addition, this growing population participates, the whole generation process, to genetic operations in order to generate sequences able to detect remaining non-detected mutants.

A random selection, based on "roulette wheel" [9], may be used to create diversity in the next population. This random selection is only performed when the crossover and mutation strategies, explained in the following, do not produce enough individuals. These randomly selected sequences participate only to non-deterministic crossover and mutation.

4) Crossover strategy

To generate a new sequence targeting specific mutant detection, a pair of individuals to be crossed, is chosen following deterministic rules. Deterministic rules exploit our knowledge of the test data generation problem. When the targeted mutant M is already activated, the first sequence is an activating sequence for M, which has the best firm fitness value. The second one is a sequence propagating numerous mutants: a sequence which has a high hard fitness value. The crossover sites are chosen close to the vectors activating M and vectors detecting the mutants. Figure 4 describes this crossover scheme. The first vectors of the first test sequence and the final vectors of the second test sequence are associated to build a new sequence.

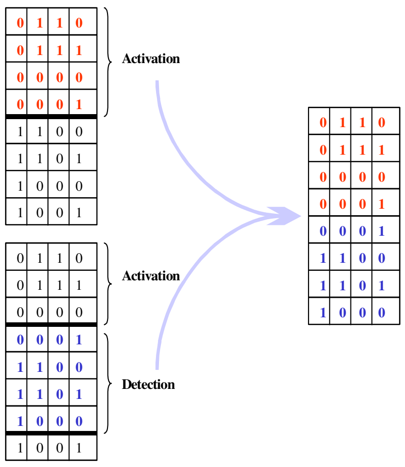

Figure 4: Crossover scheme

5) Genetic Mutation strategy

The mutation strategy aims at generating population diversity. It modifies one or several consecutive test vectors in the original sequence. In this paper, we call it “genetic mutation” and not only mutation to distinguish it to the mutation injection of the mutation testing approach. When the targeted mutant M is not activated, the test sequences activating mutants which have the best fitness are selected to be genetically mutated. Then, for each sequence the genetic mutation site is chosen close to the activating vector. Figure 5 illustrates this genetic operator.

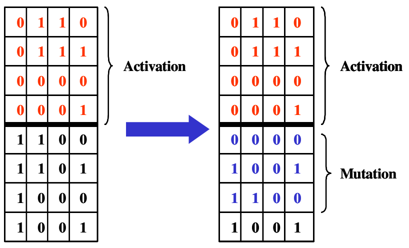

Figure 5: Mutation scheme

6) Impacts of activation and detection fitness on MS

The following table illustrates with an example the impacts of activation and detection fitness functions on MS. This example has been extracted from b15 of ITC'99 benchmark [12]. In Tab. 1, the activation of mutant M_1 and the propagation of mutant M_3 are easily achieved but the propagation of M_1 and the activation of mutant M_2 are more difficult. These difficulties come from the condition "count = 0" that must be reached. To target the activation of mutant M_2, sequences that activate many times M_1 or M_3 are selected to be modified. The part of the sequences that activate M_1 is conserved and the useless part is replaced.

Generations after generations, M_1 is more and more activated until we reach the condition "count = 0" in the original program that allows to activate M_2. To target the propagation of mutant M_1 the best sequences that activate this mutant are selected to be crossed with the sequences that propagate M_3.

DUV	Mutant 1 (M_1)	Mutant 2 (M_2)	Mutant 3 (M_3)

case gamma is
.
.
.
when G6 =>
if count = 0 then
play<=PLAY_OFF;
count := timebase;
gamma := G7;
else
count := count - 1;
gamma := G6;
end if;
	case **gamma** is
	.
	.
	.
	when G6 =>
	if count = 0 then
	play <=PLAY_OFF;
	count := timebase;
	gamma := G7;
	else
	count := count + 1;
	gamma := G6;
	end if;
		case **gamma** is
		.
		.
		.
		when G6 =>
		if count = 0 then
		play <= PLAY_OFF;
		count := timebase;
		gamma := G8;
		else
		count := count -1;
		gamma := G6;
		end if;
			case **gamma** is
			.
			.
			.
			when G6 =>
			if count = 0 then
			play <=PLAY_OFF;
			count := timebase;
			gamma := G7;
			else
			count := count - 1;
			gamma := G7;
			end if;

Table 1: Example of DUV and mutants

Figure 6 shows correlation between mutant activation and detection. The lowest curve represents the detection score and the highest curve represents the activation score. We observe that activated and not detected mutants are more and more exercized, in order to propagate their effect to primary outputs or to cause other mutant activation. Of course, each time one mutant is detected then both activation and detection scores fall down because this mutant is removed from the list of mutants.

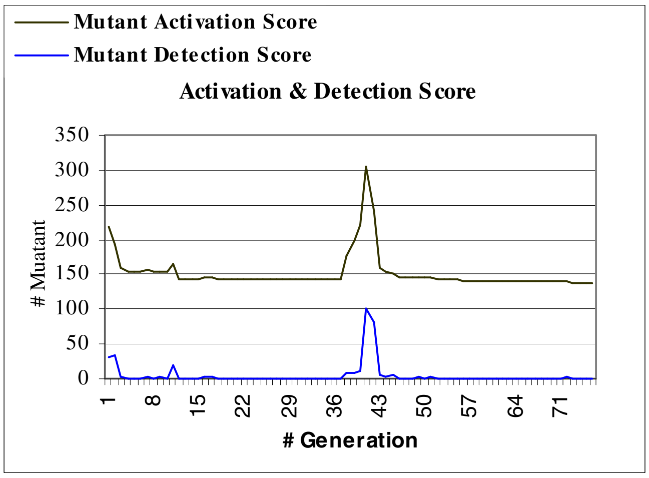

Figure 6: Mutant Activation and Detection Scores

7) Use of distances between mutants

We generate, during the static phase and after the generation of faulty descriptions, a matrix showing correlation between mutants. We call this matrix *Distance Matrix*. This matrix indicates the cost for exercizing (or detecting) a mutant M_i if M_j is activated (or detected). It is used to choose sequences to be crossed and sequences to be mutated. Currently the mutants' closeness is simply determined using the mutation line number. Even if this metrics is simplistic, it allows improving the efficiency of the genetic operations. In the following, we describe in detail this strategy and we present the results.

Thanks to crossover operator, we try to propagate activated mutants. So, when a mutant M is already activated, first we select the best sequences that activate this mutant; this selection is performed relatively to the activation fitness value. To complete pairs for crossover, we look for detected mutants in M's neighbourhood, if it contains detected mutants, we select sequences that detected neighbour mutants with the best detection fitness. When no neighbour mutant is detected, we select for each activation sequence, a sequence which has good detection fitness. Finally, the first sequence in each formed pair is an activating sequence for M, which has good activation fitness and the second one is a sequence with high detection fitness. The crossover sites are chosen close to the vectors activating M and the vectors detecting neighbour mutants.

In addition, genetic mutation allows creating a new sequence targeting a specific remaining mutant activation. Individuals, to be genetically mutated, are chosen following deterministic rules or randomly. When a targeted mutant M is not activated, the best test sequences activating the nearest mutants are selected to be genetically mutated. If there is no activated mutant in M's neighbourhood, we select with the roulette wheel technique the sequences to be mutated. Then, for each sequence the genetic mutation site is chosen close to the vector activating M. The following tables summarize these processes.

Crossover process

```
Cross ← Ø
For each activated mutant $M_j$
1- Select sequence $S_i$ that realized:
      a) - $S_i$ activates $M_j$ i.e. $a_{ij} \neq 0$.
      b) - $f_{firm}(S_i)$ is maximal.
2- If there are detected mutants in $M_j$ neighbourhood, select the nearest one $M_k$, then find sequence $S_l$ that realizes:
      a) - $S_l$ detects $M_k$ i.e. $d_{lk} \neq 0$.
      b) - $f_{hard}(S_l)$ is maximal
3- If $M_j$ neighbourhood does not contain any detected mutant, select with roulette wheel sequence $S_l$.
3- Add the new pair to Cross set : Cross ← $Cross \cup \{(S_i, S_l)\}$
4- Perform crossover on Cross set to generate two children
```

Mutation process

```
Mut ← Ø
For each not activated mutant $M_j$
1- If there are activated mutants in $M_j$ neighbourhood, select the nearest one $M_k$, then find sequence $S_l$ that realizes:
      a) - $S_l$ activates $M_k$ i.e. $a_{lk} \neq 0$.
      b) – $f_{firm}(S_l)$ is maximal.
2- If $M_j$ neighbourhood do not contain any detected mutant, select with roulette wheel sequence $S_l$.
3- Add the new sequence to Mut set : Mut ← $Mut \cup \{S_l\}$
4- Mutate the sequences in Mut set.
```

The figure 7 shows the effectiveness of using mutants' correlations to guide genetic operators: we compare the mutation scores achieved in the cases of using or not this information. The first curve, named *Basic GA* presents the MS obtained with a basic genetic algorithm (e.g. random genetic operators are applied to perform sequence evolution). The MS is used as a fitness function. The second curve *Generation without mutants correlations* does not use the Distance Matrix but uses two fitness function, activation and detection functions, and deterministic sequences generation as explained in previous section. The last curve Generation with mutant's correlations shows the results of the algorithm which uses the Distance Matrix to guide genetic operators.

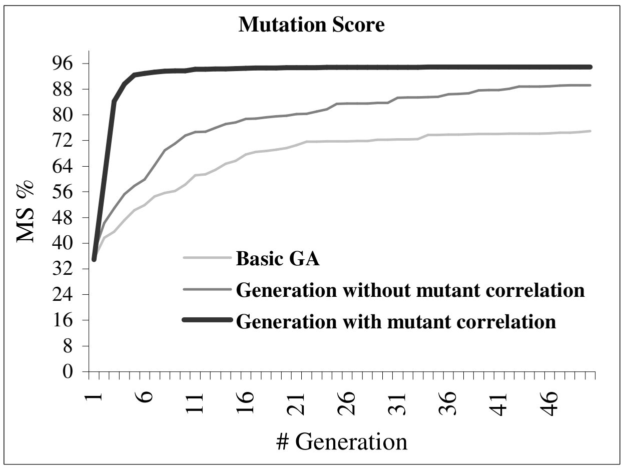

Figure 7: Impact of mutant's distances on our approach

IV. Experimental results

We applied our approach to 14 ITC'99 [12] benchmark circuits, on a 1.6 GHz Pentium-4, with 1 GB RAM, running Linux Operating System. We use *ModelSim* from Mentor Graphics to perform VHDL simulations.
No comparison with other works [10, 11, 12] can be easily done. In fact, either these works target manufacturing test using RTL descriptions, or they use formal approaches without giving any results on complex circuits.
The table 2 gives the number of lines in each VHDL RTL analyzed description. The number of injected mutants is also reported. In addition, Tab. 2 shows Mutation Score (*MS*) and CPU time for each circuit. The reported CPU time includes both simulation and generation durations to reach indicated Mutation Score. The generation of new sequences thanks to crossover and mutation deterministic strategies does not exceed 1% of the simulation time.
Our approach is a mutation-directed generation and not a coverage-directed generation, i.e. the vectors generated target mutants detection and not code coverage. So, no significant comparisons can be done between the reached MS and the coverage measurement obtained by our vectors. However, we observe for the majority of the circuits analyzed that the generated vectors also perform a high coverage score.

Circuits	**Characteristics**		**Results**		
	#Lines	#Mutants	**#VECT**	**MS [%]**	**CPU [s]**
B02	111	79	79	***100***	***21***
B03	141	314	399	***92***	***26***
B04	103	480	168	***68.6***	***74***
B05	319	516	664	***68.4***	***194***
B06	129	188	126	***100***	***25***
B07	92	303	501	***71.7***	***171***
B08	89	276	346	***86.9***	***23***
B09	103	381	295	***100***	***22***
B10	168	1011	1414	***86.4***	***766***
B11	110	881	838	***87.5***	***655***
B12	560	1500	4220	***57.5***	***5565***
B13	296	810	3100	***92.2***	***1873***
B14	509	3732	3150	***94.6***	***3129***
B15	672	4336	3600	***89.8***	***4448***

Table 2: Mutation Score compared to Classical metrics

V. Conclusion and Future Work

We have presented a framework guided by mutation-based test for functional qualification and improvement. The core of our framework uses an evolution strategy to build sequences targeting mutant's detection.. Experimental results show that our approach is able to generate high quality validation sequences even on complex VHDL RTL descriptions,. These results are obtained with realistic computational cost.
The use of mutant correlations to guide genetic operators has provided good results. However, the method used to extract this correlation is too simple. In our future work, we will propose a new way to compute these correlations.

References

[1] P. Rashinkar , P. Paterson, L. Singh, "System-on-a-chip Verification" , *Kluwer Academic Publishers, ISNB 0-7923-7279-4, 2003.*

[2] P. Wilcox, Cadence Design Systems, Inc. "Professional Verification A Guide to Advanced Functional Verification*", ISNB 1-4020-7875-7, Kluwer Academic Publishers, 2004.*

[3] A.J. Offutt, R.H. Untch, "Mutation 2000: Uniting the Orthogonal", Mutation 2000: Mutation Testing in the Twentieth and the Twenty First Centuries, pages 45--55, San Jose, CA, October 2000.

[4] Verisity Design, Inc, http://www.cadence.com/verisity/, http://www.verisity.com/products/specman.html

[5] R. De Millo and A. Offutt, "Constraint-based Automatic Test Data Generation", IEEE Transactions on computers, Vol. 17, No. 9, pp. 900-910, 1991.

[6] Y. Serrestou, V. Beroulle, C. Robach, "IP validation using genetic algorithm guided by mutation testing", *DCIS'06, XXI Conference on Design of Circuits and Integrated Systems, Barcelona, November 2006.*

[7] Y. Serrestou, V. Beroulle, C. Robach, "How to improve a set of design validation data", DDECS'06", 9th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, Prague, April 2006.

[8] A.J. Offut and J. Pan "Detecting equivalent mutants and the feasible path problem". The Journal of Software Testing, verification. and Reliability, vol. 7, pp. 165-192Septembre 1997.

[9] P. Mazumder, E. Rudnick, "Genetic Algorithms for VLSI design, layout & Test Automation". Prentice-Hall Inc. A simon&Schunster Company, 1999.

[10] F. Fallah, S. Devadas, and K. Keutzer. Occom, "Efficient computation of observability-based code coverage metrics for functional verification. *In Design Automation Conference, pages 152–157, June 1998.*

[11] F. Corno, M. S. Reorda, G. Squillero, A. Manzone, and A. Pincetti. Automatic test bench generation for validation of RT-level descriptions: an industrial experience. *In Design Automation and Test in Europe, pages 385–389, 2000.*

[12] F. Corno, M. S. Reorda, and G. Squillero. "RT-level ITC'99 benchmarks and first ATPG results", *IEEE Design & Test of Computers*, 17(3): *44–53, July-September 2000.*