Journal of Electronic Testing (2019) 35:101–110
https://doi.org/10.1007/s10836-019-05777-0

Metric-Driven Verification Methodology with Regression Management

Marek Cieplucha[1]

Received: 11 November 2018 / Accepted: 22 January 2019 / Published online: 13 February 2019

Abstract
This paper discusses the regression testing in digital integrated circuit verification. The aim of the metric-driven verification (MDV) and its role in modern verification methodology is presented. According to recognized limitations, it is proposed to include the regression testing strategy in verification environment itself. The data collected from the verification metrics can be used to adjust the testing procedures during the simulation. This approach allows for a dynamic management of the regression testing structure and may result in a significant reduction of the simulation time. The presented solution introduces a concept of the test segments. They may be started at arbitrary simulation point related to the DUT internal state (checkpoint). Usage of such segments, in the controlled regression testing strategy, may prevent from repeating the stimuli which is not contributing to the pre-defined verification metrics.

Keywords Functional verification · Digital verification · Regression testing · Metric-driven verification · Digital IC design

1 Introduction

The aim of the functional verification, in the scope of the digital IC design, is to examine the DUT (design under test) using provided test stimuli. Its main goal is to ensure the equivalence between the hardware model and its specification. Nowadays, widely used are constrained random (CRV) and metric-driven verification (MDV) techniques. The directed testing, with fixed scenarios, has been considered not effective for complex verification tasks [15].

The CRV principle assumes that each test stimulus may contain some random elements. Such elements are *randomized*, which means their values may be different at each test execution. This randomization is often controlled by *constraints*, which allow for more precise regulation of the values that random elements may take. Running the simulation of the same test scenario with a different *seed* results in a different choice of such random items and thus a different flow of the test.

Responsible Editor: C.-W. Wu

✉ Marek Cieplucha
m.cieplucha@imio.pw.edu.pl

[1] Institute of Microelectronics and Optoelectronics, Warsaw University of Technology, Nowowiejska 15/19, Warsaw, Poland

Due to test scenario randomization, it is required to monitor whether the DUT was examined as expected. Therefore, *metrics* play a major role in CRV. Observing such metrics is necessary to make sure all expected stimuli was executed, as there is always an uncertainty whether a specific DUT operation has been tested during the actual simulation. In MDV approach, metrics are pre-defined and they constitute goals for the verification tasks. Nowadays, the frequently used metrics are related to a *coverage*. Apart from automatic coverage (such as code, toggle or FSM), the *functional coverage* has an important role in modern verification flow, as it is defined manually by engineers with an emphasis on corner cases.

The *regression testing* is a process to fully verify the DUT against a verification (or a test) plan [20]. Such plan consists of items defining the verification scope. These items may be related to the test scenario, DUT internal states or settings, logic-level sequences or high-level operations flow. They are mapped to the quantifiable metrics, so the verification progress can be observed by reaching their satisfactory values. Fulfilling the verification plan requirements is called a verification closure. A regression testing suite consist of a set of tests, which are supposed to examine the DUT according to these requirements.

Although there are methods and good practices focusing on verification environment implementation, the process of building regression testing suites is, in general, experimental. Moreover, due to the random nature of the test sequences, there is never a guarantee if all specified coverage goals will be achieved by the simulation. To improve the likelihood of hitting desirable corner cases, randomized test stimuli are often simulated repeatedly during the regression testing. This approach is however not effective, as it results in a significant extension of the total simulation time.

The main goal of work described in this paper was to present methods of building regression testing suites with a dynamic structure. That means, the regression testing strategy becomes a part of the verification environment itself, and its composition is adjusted using the metrics tracked in real time. According to this approach, it is possible to reach the planned verification goals faster than in case of running straightforward regression suites.

As it is difficult to provide a reliable data relevant to the verification of a complex design, a probabilistic model of the regression testing has been introduced. This model is used to estimate the verification effort of the MDV-based environment in terms of simulation complexity. The analysed data show advantages of the dynamic control over the regression suite structure, in comparison to the standard approach.

In this paper, the regression testing and verification efficiency topics are described in Section 2. Various improvement proposals are discussed in Section 3. The new approach of the dynamic regression management is presented in Section 4. Probabilistic model of the regression testing has been introduced in Section 5 and its evaluations were shown. Finally, a simple example verification environment implementing the discussed methodology is described in Section 6.

2 Regression Testing and Its Limitations

The methodical approach to the MDV-based verification process [20] is presented in Fig. 1. The verification plan contains elements organized in various sections. These items are extracted from the specification (functional requirements) or depend on a particular implementation (e.g. performance or system-level features). There are a number of test cases which intend to fulfil the verification plan requirements. According to a CRV approach, they are complex tests with highly randomized flow. As applying stimuli to the DUT, according to CRV principles, is a random process, there is often need for executing test sequences repeatedly, to make sure the metrics-based corner cases have been exposed. Therefore, it is often required to run the simulation of the single test scenario several times, in order to meet the coverage goals, as the same scenario may examine different design features at each test run. The coverage and other metrics are extracted from the executed tests and accumulated. They correspond to the

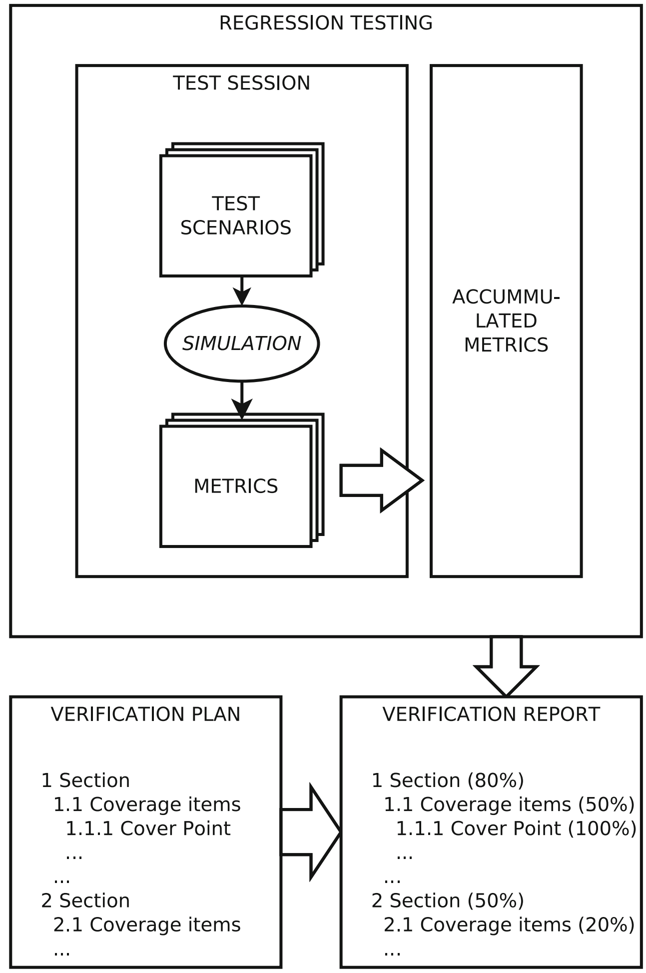

Fig. 1 Metric-driven verification flow

verification plan elements. When the testing is completed, the verification report containing metrics data mapped to the plan is generated.

The complete regression testing procedure consists of the suite of tests which are executed repeatedly. Due to the random nature of the test stimuli, we may only predict if a specific coverage goal will be finally reached. Repeating the test execution may increase the likelihood of satisfying these metrics. In modern verification methodologies, already collected metrics data is not usually reused in the testing procedure, so each test execution with a different randomization (*seed* value) may be considered as independent. This leads to the conclusion that complete CRV regression procedure can never assure the 100% coverage goal. Moreover, when we increase the test run count to hit corner cases, the already covered features are examined repeatedly and may not contribute to the metrics at all.

We may assume, that our verification goal is to satisfy the coverage metrics at a specific level (e.g. 99%) with an acceptable likelihood (e.g. 9 of 10 random regressions should achieve this value). There are always several ways of achieving such results. Some tests can better contribute to meeting specific coverage goals, so they may be preferred to be executed repeatedly. On the other hand, other tests may not improve the coverage at all – so they may be removed from the suite.

We introduce the definition of a *verification efficiency* as the ratio of achieved coverage level over a total cost of execution of a full regression testing procedure. This cost is a sum of the costs of each test in the suite. The cost of the single test may be different, depending on its scenario and randomization. It is convenient to measure the cost of the test execution in terms of a real simulated time. Due to this assumption, we assure independence from the simulation platform performance and eventual simulation parallelization. So, the goal to create an effective test suite is to satisfy given coverage metrics, assuming the overall simulation time is minimized. In the following part of the paper we will be considering optimal test suites, which mean regression structures that satisfy the coverage metrics at a specific level with minimum execution cost.

The relation between a verification closure and a regression testing is not straightforward. Fulfilling metrics defined in a verification plan can be seen as validating specific design features. However, a requirement for a successful feature validation may be ambiguous. What happens, when for example, a test is finished with and without errors for different executions? What if we allow for design failures in some cases, which may cause a test error (e.g. if we need only to satisfy DUT operability with given probability)? In general, all these assumptions must be implemented in test scenarios, which complicates their structure.

To sum up the above considerations, we can look at the process of building regression testing suite as at the part of the verification strategy. This step is not however mentioned implicitly in modern verification methodologies. There is no well-established approach to this problem and the test suite construction process is rather experimental, often supported by EDA tools. But overall, regression testing objectives are separated from the verification environment architecture.

3 Improvements of the Regression Testing Efficiency

The first step towards an improvement of the verification efficiency is to make more use of the coverage metrics in the testing procedure. It allows for an adjustment of the test constraints in order to hit missing corner cases. This idea is shown in Fig. 2 and is called a "coverage-directed test generation". It has been introduced in [7] and later extended with use of evolutionary algorithms [12]. The main principle

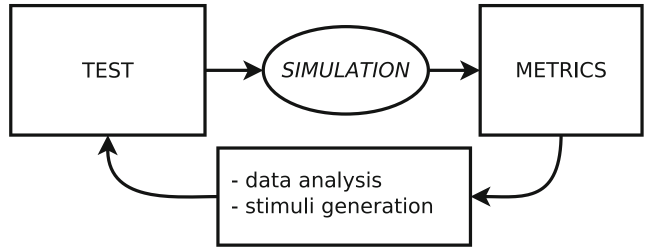

Fig. 2 Coverage-directed test generation

of this concept is to observe the impact of the stimuli on coverage. New test constraints can be generated using the feedback from these coverage results. No information about the DUT itself is processed during new tests generation. This concept may be implemented off-line, which means the data is analysed by an external application (outside the testbench) the output of which is a set of new test constraints to be applied. It is also possible to adjust the test dynamically during the simulation [4, 14]. That involves the actual test scenario modifications, i.e. implementation of the constraints adaptation inside the testbench.

Another approach is to regulate constraints based just on the metrics definition, without running any simulation. It has been presented in [18] and in this paper, an EDA simulator tool can handle this task. However, this method may be useful only, when a direct relation between metrics and constraints is known.

It has been noted in [8], that for each random test it is possible to determine the probability of covering some particular metrics. This important observation will be later used in the probabilistic model of the regression testing described in Section 5. According to this remark, if we know this data, we can create an optimal regression suite in terms of the verification efficiency, as this in an optimization issue called the set cover problem. But the discussed probability data is not directly known and needs to be measured. As an extension of [8], it has been proposed to prioritize some tests using their coverage contribution estimation [21]. Further improvements involve use of a machine learning and clustering techniques, in order to select tests in the suite with minimum overall execution cost [10, 17], which corresponds to an increase of the verification efficiency.

Another important observation is that various test scenarios usually share some common sequences [13]. These particular stimuli do not improve the coverage level, but must be executed in several test simulation runs in order to set up a DUT. It can be, for example, an initialization procedure, connection configuration or register programming. It has been shown, that actual test scenario can be started from a retained state, in which a DUT is already set up. This retained state is a simulation point at a specific time moment called a *checkpoint*.

In the most advanced recent work it has been proposed to dynamically manipulate the test *seed* in order to improve the efficiency of a single test run [16]. The scenario allows for “rewinding” the test process and continuation of the simulation with a new randomized variables. The cost function, that is subject to be maximized by a *seed* manipulation engine must be defined. It may be, for example, a coverage level.

Modern EDA verification platforms provide tools for managing the regression suites and analysing the metrics data. For example, in Cadence vManager [11] there is a feature of ranking the executed runs. Generated rankings can be used to modify the regression suite by selecting appropriate runs contributing to the coverage. However, this useful method is manual and the whole process may need to be reiterated when verification environment is modified.

To conclude the related work description, the following trends can be observed for improvements of the regression testing efficiency:

- dynamic or static test constraints adjustment based on metrics data [4, 7, 12, 14, 18];
- avoiding execution of the stimuli that does not contribute to the coverage [13, 16];
- selecting appropriate scenarios or *seeds* in order to minimize the executed test count [10, 16, 17, 21].

4 Proposal of the MDV Methodology with Regression Management

The key concept of the proposed methodology is a redefinition of the regression testing process. The regression testing flow becomes dynamic and its structure is being adjusted depending on metrics data. The overall concept is shown in Fig. 3. The idea of a test segment is introduced. It is, in principle, an independent test scenario, but instead of containing a full standalone test flow, it corresponds to a specific verification subroutine, such as initialization, connection set-up, data transmission, reset, reconfiguration etc.

It is assumed that segments can be started at a specific point of the simulation (checkpoint), which means they do not need to execute all stimuli required to reach their initial state. Moreover, due to the random nature of the testing flow, each execution of the test segment results in a generation of a new checkpoint. Test segments can be executed repeatedly, starting from various checkpoints. Metrics data is used to dynamically adjust the overall testing structure.

The example verification suite execution flow with test segments and checkpoints is presented in Fig. 4. Instead of a list of tests, a new structure looks like tree, the nodes of which correspond to simulation checkpoints. In the presented example flow, segment called “INIT” is called

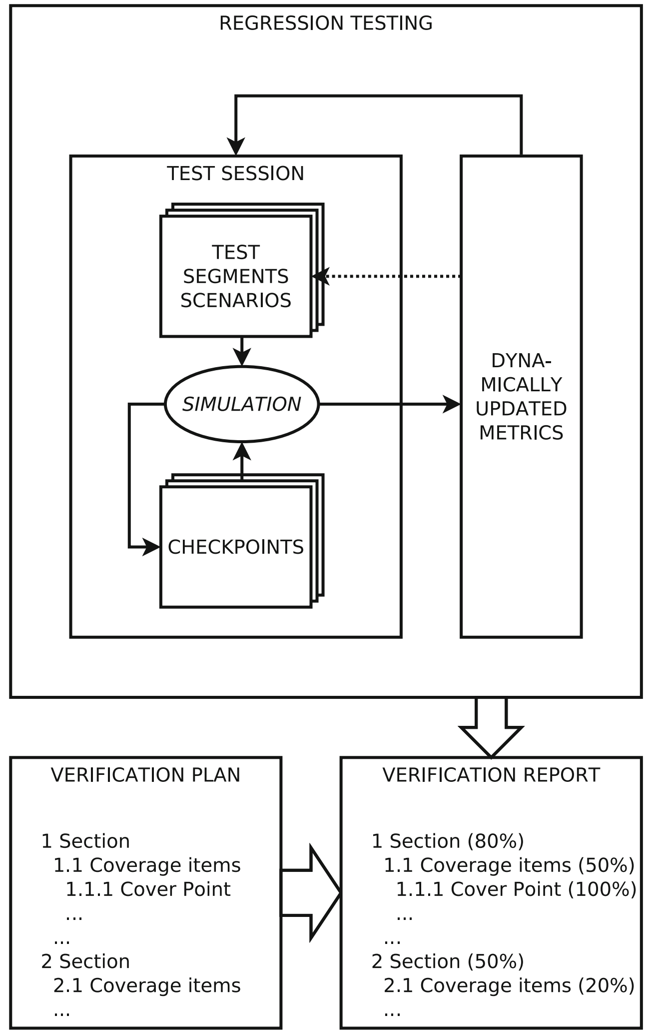

Fig. 3 Metric-driven verification with regression management

twice, what produces two new checkpoints “A1” and “A2”. Afterwards, test segments “S1 ... S6” are being executed randomly starting from different checkpoints and thus new checkpoints “B1 ... B5, C1 ... C5” are created. This structure is dynamic, so at each execution it may look slightly different. Metrics data can be also used to adjust only a particular test segment scenario, as discussed in [4, 14] (it shown by the dotted line in Fig. 4), but this case will not be analysed in this paper.

The proposed approach assumes that the verification environment must implement functions related to the regression management. So, instead of using external EDA tools or preparing regression scripts manually, it is required to include such functionality in the testbench itself.

The proposed methodology provides a general approach that could be used to resolve problems with verification efficiency, mentioned in Section 3. The methodology requires to implement new verification components responsible for the test regression suite management. However,

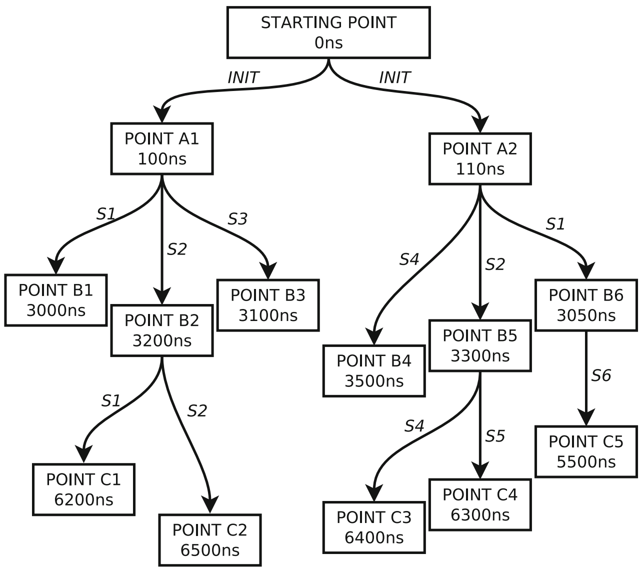

Fig. 4 Example structure of the segment-based regression suite

tools and infrastructure used nowadays for functional verification tasks, such as Universal Verification Methodology (UVM) [19] and SystemVerilog hardware verification language, do not provide a sufficient functionality to make use of presented mechanisms. The limitations of current methods have been observed [3] and there are various approaches to mitigate them. Most of existing solutions is based on external solvers or libraries [10, 12, 16, 17, 21]. There are also other proposals that make use of other verification platforms, such as enhanced simulators [18], or testbenches developed using SystemC language [14]. The recent solution is also the COCOTB platform implemented in Python [6].

For the purpose of the presented work, functions that enable checkpoints management have been implemented as an extension of the COCOTB platform. It allows for implementation of verification environments according to the presented methodology. The complete testbench based on COCOTB can be written in Python. The platform provides a VPI-based communication with an arbitrary HDL simulator. Some base verification components, such as drivers or monitors are already provided within the framework. There are also extensions of the COCOTB which make use of advanced verification mechanisms, such as functional coverage [4] and constrained randomization [5].

For the presented methodology, the arbitrary point of the simulation depends not only on the stimuli of a current test segment, but also on its starting checkpoint. In order to precisely reproduce a particular simulation event (e.g. during debugging), this checkpoint must be strictly known. For a traditional regression testing, the test case could be reproduced using its run *seed*. In case of the dynamic regression structure, the verification environment must implement a functionality facilitating rerunning the same simulation flow. The easiest way to achieve that is to store checkpoints as the simulation output files and reuse them when necessary.

Using dynamic regression suites managed by the testbench allow for major improvement of the verification efficiency. First of all, simulation time may be reduced thanks to checkpoints usage. That enables possibility of simulating common phases of the tests (or phases not improving the coverage) with lower repetition count. Moreover, if the particular stimulus requires to be repeated many times, it is more efficient to restart it from the specific checkpoint and therefore execute only the most suitable part of the stimulus. Regression management can be also implemented such way that it "learns" which segments should be executed by observing their coverage contribution. This automation allow for making the regression suites self-organising and therefore capable of achieving coverage goals much faster than during traditional testing.

5 Probabilistic Model of the MDV-Based Regression Testing

This section describes mathematical simulations of the MDV-based verification closure procedure of the regression testing. Simulations were performed in order to estimate the verification efficiency of this process. Two approaches are described: a traditional one, where tests are executed repeatedly and a new one, implemented according to the new methodology assumptions.

The standard approach to the regression testing assumes that test runs are executed independently. It will be called a sequential regression testing. According to the new proposed methodology, test segments can be started at arbitrary, already existing checkpoints. This approach will be called a tree regression testing.

The probabilistic model is based on numerical data describing the relations between the cost of the stimulus and the probability of covering the particular metrics.

5.1 Model Definitions

The fundamental assumption of the presented model is that regression testing session is a random process and planned verification items are random variables. The probability of covering such particular element is known [8]. For the simplification of the model, it is also assumed that metrics cannot drive the stimuli of a standalone test (or segment), so the probability of covering the particular metric is constant during the whole session.

The main goal of the regression testing is to achieve a given coverage level, which means satisfying metrics with given probability. Let's define a set of N metrics $M_0...M_{N-1}$ and assume that each probability of covering the particular metric is independent of all others and their weights are equal. This approach is good approximation only when a single metric corresponds to some group of coverage primitives, i.e. a subsection of the verification plan. The probability of covering metrics is known and will be given as a set defined by function of a multiple arguments $P(M_i, ...)$ and $0 \leq P(M_i, ...) \leq 1$. In case of sequential regression testing, it depends only on a test case ($P(M_i, T_j)$) and in case of tree regression testing, it depends on a test segment and a starting checkpoint of this segment ($P(M_i, S_j, C_k)$).

The function $f(P(M_i, ...))$ defines the probability of covering given metric during the full regression testing. The final verification goal is to satisfy all defined metrics with overall probability exceeding an assumed level, which is given by Eq. 1.

$$G = \frac{1}{N}\sum_{i=0}^{N}[f\left(P(M_i, ...)\right)] \tag{1}$$

Let's define a total cost of a regression testing session $Cost_{total}$ as a sum of costs of all test runs. During the model simulations, we are looking for a regression testing suite structure of minimal cost, which corresponds to the highest verification efficiency. This assumption can be described by Eq. 2.

$$\begin{cases} G \geq G_{assumed} \\ Cost_{total} = \min \end{cases} \tag{2}$$

In a sequential regression testing, a test session is a list of test scenarios that may be executed arbitrary number of times. So, for a list of tests $T_0...T_{L-1}$ let's define a function describing a session structure, which associates a number of repetitions with the particular test: $R(T_i) = n$. Additionally, also the (average) cost of a test run is known: $Cost(T_i)$. The overall cost of the regression testing may be therefore given by Eq. 3.

$$Cost_{total} = \sum_{i=0}^{L} Cost(T_i)R(T_i) \tag{3}$$

The probability of covering a given metric during the full regression testing can be calculated according to the multiplication rule for independent events. If we also consider test repetitions, the overall goal function is given by Eq. 4.

$$G = \frac{1}{N}\sum_{i=0}^{N}\left\{1 - \prod_{j=0}^{L}\left[1 - P(M_i, T_j)\right]^{R(T_j)}\right\} \tag{4}$$

For the tree regression testing let's define a set of test segments $S_0...S_{H-1}$ and checkpoints $C_0...C_{Z-1}$. The checkpoint C_0 is a root of the tree – the starting point of the simulation. Test segments can be executed repeatedly and start from different checkpoints. In case of representing a tree regression testing structure, it is required to define a two-argument function describing number of repetitions of a segment in a specific checkpoint: $U(S_i, C_j) = n$. Additionally, this function must meet additional requirements. First of all, in order to run a segment starting in a specific point, this point must be already created, which means the simulation must reach this state beforehand. An emerging checkpoint depends on a segment and its starting point, which means, each segment run creates a new checkpoint. We will however introduce a simplification, assuming that for each segment started in a particular point, only one new checkpoint may be created.

Let's introduce a parameter representing a regression tree depth by q. For $q = 1$ all segments start from a single checkpoint, so a tree structure corresponds to the sequential regression testing, thus a function $U(S_i, C_j)$ is non-zero only for C_0. For $q = 2$ segments may also start at checkpoints emerged after running segments started in C_0. It means, that for H segments started in a particular checkpoint, there may be created maximum $H + 1$ new checkpoints. In general, for arbitrary q, maximum number of checkpoints Z is given by Eq. 5.

$$Z = \sum_{i=1}^{q} H^{(i-1)} \tag{5}$$

The overall cost of the tree regression suite corresponds to the similar function in sequential regression testing and is given by Eq. 6.

$$Cost_{total} = \sum_{i=0}^{H} Cost(S_i)\left[\sum_{j=0}^{Z} U(S_i, C_j)\right] \tag{6}$$

The probability of covering a given metric during the full regression testing can also be calculated in a similar way, what is represented by Eq. 7.

$$G = \frac{1}{N}\sum_{i=0}^{N}\left\{1 - \prod_{j=0}^{H}\prod_{k=0}^{Z}\left[1 - P(M_i, S_j, C_k)\right]^{U(S_j, C_k)}\right\} \tag{7}$$

5.2 Simulation Methodology

In order to estimate the verification efficiency, it was required to evaluate the cost of the full regression testing procedure which reaches a specific overall coverage level. To perform such simulations the following data is needed:

- set of tests (or segments) and their costs – $T_0...T_{L-1}$ and $Cost(T_i)$ (or $S_0...S_{H-1}$ and $Cost(S_i)$),
- the function defining the probability of covering all metrics by the test – $P(M_i, T_j)$; or by the segment in the specific checkpoint – $P(M_i, S_j, C_k)$,

– target overall coverage level – $G_{assumed}$.

The heuristic procedure has been implemented, the aim of which was to find the regression suite structure ($R(T_i)$ or $U(S_i, C_j)$) with minimum $Cost_{total}$.

In order to make a reliable comparison between the sequential and the tree regression testing, it is required to generate a consistent data for probability and cost functions. When the regression tree depth $q = 1$, the segments are in principle independent tests, so $Cost(T_i) = Cost(S_i)$ and $P(M_i, T_j) = P(M_i, S_j, C_0)$ as checkpoints other than C_0 do not exist. For $q > 1$ it may be assumed that segments are created the way that they are phases of the test from sequential regression testing suite. For example, as shown in Fig. 4, the set of consecutive segments [„INIT", „S2" started in „A2" and „S4" started in „B5"] corresponds to such test. Therefore the cost and probability functions should follow the general assumption that the arbitrary complete tree branch data is consistent with the same data for a single sequential test.

5.3 Simulation Results

The example results comparing the sequential and the tree structure for consistent costs and metrics data are shown in Fig. 5. The graph shows the cost of the regression suite (simulation time) for an assumed achieved overall coverage level. The data is normalized such way that the lowest cost of the regression suite is equal to 1. For $q = 1$ the results correspond to the sequential test suite, so no segments and checkpoints are used at all.

It can be seen that the overall cost of the regression suite is lower when the tree structure is used for all overall coverage levels above 80%. Moreover, the cost increase is lower for high coverage level targets. These results are intuitively expected, as in the tree structure the stimuli responsible for hitting corner cases do not require to rerun the preceding phases (e.g. initialization).

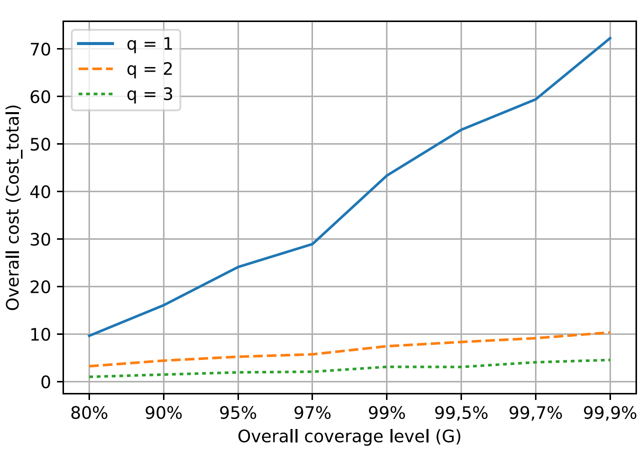

Fig. 5 Cost of the regression suite in function of the overall coverage

Similar results were achieved for different sets of generated data. It has been also observed, that a higher segmentation (higher values of q) can improve the overall cost even further. This is also an expected observation, as more checkpoints means that simulation time is reduced due to possibility of repeating segments that are shorter. However, in a real environment, the structure of the tree will not be balanced and is unlikely to be optimal due to a random nature of the test stimuli, so the expected simulation time reduction is smaller.

6 Example of the Verification Environment Implementation

In order to demonstrate a working example of the discussed MDV methodology with regression management, the comprehensive verification environment has been developed for the APBI2C controller [1]. This controller is a simple example of a multi-protocol DUT and contains such typical building blocks as configuration registers, FIFO and bus-specific logic.

The simplicity of the chosen deign-under-verification may be considered as not sufficient for verification methodology assessment. On the other hand, implementing an environment for a complex IP block is quite a challenge and its effectiveness may be impacted by many other aspects, such as implementation flow, poor planning and even team-working quality. In other words, even following the given methodology, the end result may be different. For simple environments this problem is expected to be reduced.

Despite the design and its test environment simplicity, the implemented testbench structure contains all typical verification elements, such as monitors, drivers, simple scorebaording and coverage. The structure of the verified module together with the verification environment is shown in Fig. 6. The testbnech was implemented in Python using the COCOTB platform and is available as an open source [2].

The verification environment consists of:

– APB protocol agent, responsible for data transfer via APB bus (register configuration and FIFO operations),
– I^2C protocol agent, responsible for data transfer via I^2C bus (which corresponds to an external I^2C device),
– objects defining I^2C transfers (data and protocol-specific settings),
– test segments: read I^2C transmission, write I^2C transmission and APB-only register operations,
– main test scenario which is responsible for the test data operation and segments execution, so in principle the regression management,
– functional coverage.

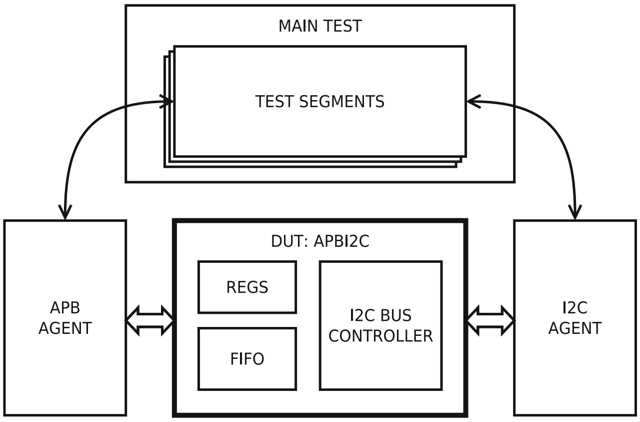

Fig. 6 Schematic diagram of the APBI2C controller and testbench

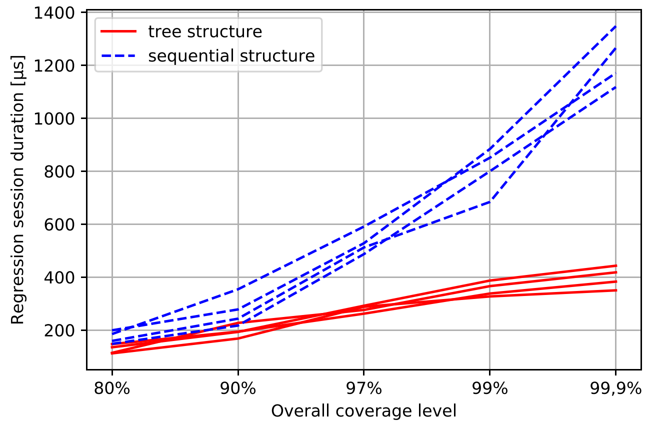

Fig. 8 Cost of the regression suite in function of the overall coverage for APBI2C controller

The functional coverage implementation contains items related to a protocol-specific logic and various I^2C transfer properties. It can be divided into 3 main sections: APB protocol coverage, I^2C protocol coverage and end-to-end data transfer settings. In particular, the cross coverage point containing 64 bins has been defined the aim of which was to monitor executed end-to-end transmission. The regression completion requirement is to satisfy a given coverage level which includes the defined elements from all sections.

The main test generates random data transfer settings and executes test segments. There is a constraint function implemented which prevents from generating already tested and covered settings. In principle, this feature drives the segments execution and therefore applies the regression management, so the main test implements the verification flow shown in Fig. 3. If this constraint is disabled, all segments are executed independently, so they begin at the starting point of the simulation (no checkpoints are used). It is therefore possible to make a comparison of two regression strategies: the sequential one with independent tests and the default one, using a tree structure. The segments execution algorithm is shown in Fig. 7. A new checkpoint is created when a test segment finished successfully. For more implementation details of the presented environment please refer to the open source project [2].

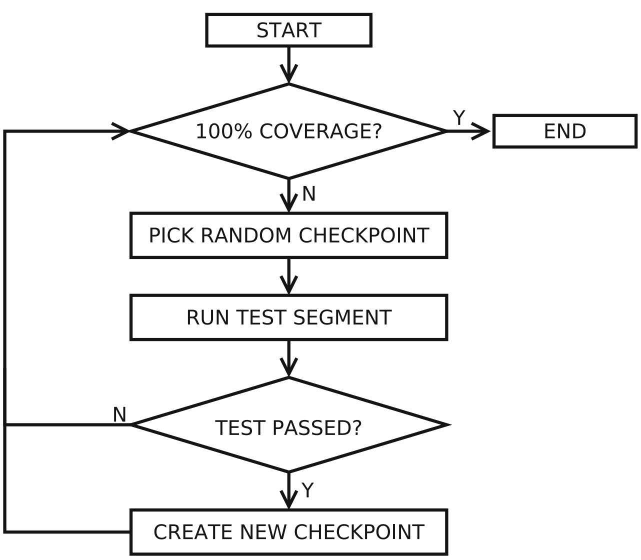

Fig. 7 Segments execution algorithm

The cost of the regression (total simulation time) in function of the overall coverage level is shown in Fig. 8. Several runs of the regression have been executed with a sequential (blue) and a tree regression suite structure (red). These results match pretty well the data provided by the model, presented in Fig. 5.

It can be seen that the tree structure efficiency advantages arise when target coverage level is going up to 100%, which is consistent with the probabilistic model data. For the 99.9% target coverage the overall regression simulation time was approximately 3 times shorter compared to a traditional, sequential regression testing structure.

7 Methodology Evaluation and Conclusion

In this paper a new verification methodology with regression management has been presented. It has been shown that building the regression suite based on real-time metrics can significantly reduce the overall cost of the simulation and therefore improve the overall efficiency of this process. Moreover, this approach gives better results for high overall coverage targets, what is in particular important for hitting functional corner cases.

The discussed proposal takes advantage of the idea of a test segmentation, which is dividing test scenarios into smaller parts and starting them from stored states of the simulation (checkpoints). When regression suite is build this way, its structure resembles a tree, which is shown in Fig. 4.

The regression testing procedure according to CRV principles is a random process, the goal of which is to satisfy given coverage metrics. This process has been described by a mathematical model and simulations of the coverage closure have been performed. They showed the advantage of the tree structure regression testing over the traditional approach, where tests are executed repeatedly and independently on each other.

The currently used verification techniques (SystemVerilog/UVM and commercial simulators) are not intended to design a verification environment that includes the regression management procedure. The proposed methodology principles have been implemented in Python using the COCOTB platform and have been proven to work for advanced functional verification tasks. The provided verification environment for APBI2C controller uses this methodology and it has been shown, that (for a given target coverage level) the simulation time is reduced compared to a traditional regression testing strategy.

In the mathematical model described in Section 5 it has been shown that splitting tests into smaller segments (and using checkpoints) enables building regression testing suites with significantly reduced simulation time. It is possible as simulation scope may be changed to emphasize corner cases. The described model is purely theoretical, so provided data will not be valid for real environments, as much more factors may impact their performance and efficiency. However, an important relation between segmentation factor and the simulation time have been shown as well as rapidly rising regression testing complexity for very high coverage goals (and its possible reduction). The data produced by the mathematical model matches real environment performance analysis, as described in Section 6.

Using checkpoints requires to store the DUT state in the memory. This is an additional complexity that will impact the testbench performance, as this data must be saved and read during the simulation. It may be however noticed that this issue will depend on simulator engine and is methodology-independent. Even in the worst case, dumping all DUT registers data and restoring it is expected to be very quick compared to test stimulus complexity.

In general, it is difficult to clearly evaluate the described methodology and confirm that it produces better results in any case. There are many factors that may impact the verification environment implementation and therefore its performance. Also, the idea of segmentation may not be fully applicable for all verification tasks. The goal of this paper was however to show a general rule for building verification environments. The main concept of the proposed methodology is making use of the coverage metrics in real time and building regression suites according to this data. The verification complexity is therefore reduced as the simulation effort is better adjusted for meeting the pre-defined requirements.

In the presented methodology it is assumed that the regression management functionality is implemented inside the testbench itself. There are two verification platform requirements that need to be satisfied in order to make use of this feature. The first one is an access to the coverage metrics data in real time. This is roughly possible in SystemVerilog, but limited to a single test scope. The second requirement is an access to checkpoints from the testbench code, so the functionality of saving and restoring a DUT state. The Verilog Programming Interface [9] provides dedicated functions for these operations, but not all commercial simulators support it. If these two requirements are satisfied, the new methodology can be implemented using a different platform than presented in this paper.

References

1. APB to I2C controller, OpenCores.org, 2016. [Online]. Available: https://opencores.org/project,apbi2c
2. APBI2C COCOTB TESTBENCH EXAMPLE, 2018. [Online]. Available: https://github.com/mciepluc/apbi2c_cocotb_example
3. Bromley J (2013) If SystemVerilog is so good, why do we need the UVM? Sharing responsibilities between libraries and the core language. In: Proceedings of the 2013 forum on specification and design languages (FDL), pp 1–7
4. Cieplucha M, Pleskacz W (2016) New architecture of the object-oriented functional coverage mechanism for digital verification. In: Proceedings of the international verification and security workshop (IVSW). IEEE, pp 1–6
5. Cieplucha M, Pleskacz W (2017) New constrained random and metric-driven verification methodology using python. In: Proceedings of the design and verification conference and exhibition US (DVCon)
6. COCOTB 1.0 documentation, Potential Ventures, 2014–2019. [Online]. Available: http://cocotb.readthedocs.org/en/latest/introduction.html
7. Fine S, Ziv A (2003) Coverage directed test generation for functional verification using Bayesian networks. In: Proceedings of the design automation conference (DAC), pp 286–291
8. Fine S, Ur S, Ziv A (2004) Probabilistic regression suites for functional verification. In: Proceedings of the design automation conference (DAC). IEEE, pp 49–54
9. IEEE Standard for Verilog Hardware Description Language, IEEE Std., 2006
10. Ikram S, Ellis J (2017) Dynamic regression suite generation using coverage-based clustering. In: Proceedings of the design and verification conference and exhibition US (DVCon)

11. Incisive vManager User Guide, Cadence Design Systems, Inc., 2016, product Version 15.20
12. Ioannides C, Barrett G, Eder K (2011) Introducing XCS to coverage directed test generation. In: Proceedings of the IEEE international high level design validation and test workshop (HLDVT), pp 57–64
13. Jain RK, Sudhakar S (2015) Want a boost in your regression throughput? Simulate common setup phase only once. In: Proceedings of the design and verification conference and exhibition US (DVCon)
14. Kuznik C, Mueller W (2011) Aspect enhanced functional coverage driven verification in the SystemC HDVL. In: Proceedings of the international SoC design conference (ISOCC). IEEE, pp 154–157
15. Meyer A (2003) Principles of functional verification, 1st edn. Newnes
16. Nelson E (2017) Improving constrained random testing by achieving simulation verification goals through objective functions, rewinding and dynamic seed manipulation. In: Proceedings of the design and verification conference and exhibition US (DVCon)
17. Sokorac S (2017) Optimizing random test constraints using machine learning algorithms. In: Proceedings of the design and verification conference and exhibition US (DVCon)
18. Teplitsky M, Metodi A, Azaria R (2015) Coverage driven distribution of constrained random stimuli. In: Proceedings of the design and verification conference and exhibition US (DVCon)
19. Universal verification methodology (UVM) 1.2 user's guide. Accellera, 2015. [Online]. Available: http://www.accellera.org/images//downloads/standards/uvm/uvm_users_guide_1.2.pdf
20. Vasudevan S (2006) Effective functional verification—principles and processes, 1st edn. Springer US
21. Wang S, Huang K (2016) Improving the efficiency of functional verification based on test prioritization. Microprocess Microsyst 41:1–11. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933115002008

Marek Cieplucha holds a PhD in Microelectronics from Warsaw University of Technology. He is specialized in digital IC design and verification with main focus on functional verification methodologies and their effectiveness. He is interested in strengthening relations between science and industry in this field, looking for easily applicable solutions for improving the verification process. He has published his research results in several conferences worldwide and contributes the open source verification activities. He also cooperates with EDA business and deploys his solutions in the industry projects.