Risk-based testing reduces time to market of dynamic EDA tools

Mohamed Bahnasawi

Siemens EDA
Siemens
Cairo, Egypt
Mohamed.bahnasawi@siemens.com

Ahmed Khater

Siemens EDA
Siemens
Cairo, Egypt
Ahmed.khater@siemens.com

Reem ElAdawi

Siemens EDA
Siemens
Cairo, Egypt
reem.eladawi@siemens.com

Abstract

The integrated circuit (IC) industry uses electronic design automation (EDA) tools in its iterative cycle of analyzing, designing, and verifying ICs. Industry growth creates a need for improved and expanded functionality that performs processes with fewer resources in a shorter timeline with a very low risk of failures. The functional complexity of EDA tools continually increases as we strive to provide the most efficient and most accurate utilities.

When tool quality is critical, the risk of failure must be minimized. Risk-based testing (RBT) reevaluates the risks to steer test efforts in relation to customer priorities. With limited resources and time constraints, prioritizing test efforts is a must to catch critical bugs in early stages to fix them. Risk identification and analysis are critical components of targeted effort allocation.

In this paper, RBT is applied to “Create Random Array” (CRA) and “VIA Random Placement” (VRP), which are highly dynamic tools. We demonstrate that applying the RBT technique reveals critical bugs as early as possible, which can significantly reduce time to market while ensuring tool quality. The results include metrics for measuring and controlling the progress, efforts, and cost of test activities. We also show that RBT is applicable to any tool operating in a dynamic environment with limited resources.

Biography

Mohamed Bahnasawi is a Senior Software QA Engineer at Siemens EDA with 5 years of experience in automation and testing.

Ahmed Khater is a QA Team Lead at Siemens EDA with 12+ years of experience in Software Developments and Testing Process.

Reem ElAdawi is a Test Engineer Director at Siemens EDA with 25+ years of experience in Software Development and Testing Process.

1 Introduction

Electronic design automation (EDA) tools have a long history of vigorous innovation, driven by the exponential expansion of the integrated circuit (IC) industry, which relies heavily on EDA tools during its long cycle of designing, analyzing, and validating results. To satisfy realistic design schedules and budgets, bigger designs require significantly higher designer productivity, creating a constantly evolving role for EDA tools. As a result of the increased functional complexity of EDA tools, particularly commercial ones, the necessity of quality process rises.

EDA development teams must find a balance between supporting new technology nodes and expanding the tools' capabilities when introducing new and expanded features to meet the expectations of EDA customers. Because of the intense competition among customers to support new technology nodes, EDA development teams aim for aggressive deadlines while maintaining staff skill levels. As new capabilities are introduced, the tools become more dynamic in dealing with more complex scenarios. At the same time, the customer's primary need is zero defects in the fabricated chips, which cannot be met without EDA tools with zero bugs. Even a few bugs in an EDA tool might result in irreparable and costly hardware defects. Some hardware defects cost hundreds of millions of dollars in the replacement process of defected chips (Thomas 2011).

Thus, high quality is not an option in EDA tools, as customers need qualified tools with almost zero bugs, within tight time restrictions. However, EDA tools have frequent upgrades. EDA software quality is also particularly difficult to evaluate because of its long runtimes, and the fact that the tools' outputs often contain tough iterated and numerical issues that are not always instantly verifiable. The quality process has also become more difficult because it must be completed, and results evaluated under time restrictions. However, regression testing is a must for each tool. Regression runs are standard for both hardware designs and EDA tools. Because very few modifications may occur in a single day, several bugs were discovered by using these runs. (Ng 2005)

Testing should unveil software defects that risk the product's mission-critical operations. However, software testing takes a significant amount of time. Testing can cost up to 40 percent of the overall of the total original cost of software development (Pressman 1995). Especially for EDA, due to its demanding time schedule, the testing process is done under extreme time pressure. In this environment, it is necessary to develop a strategy for prioritizing efforts and allocating resources to the software components that must be extensively tested to ensure that the software requirements are met.

Risk-based testing (RBT) uses risk assessment to drive all phases of testing process while reducing testing costs, such as time and resources, without affecting the quality of the tested tools. By focusing testing activities according to the risk assessment of the tool, which assumes that new, unclear, or undiscovered scenarios, and complex flows are more likely to fail (Felderer 2014). By identifying the risk factors in the tool's functional specifications and prioritizing the requirements based on these identified risks, the design and execution of tests become more efficient at covering critical situations in early stages, giving the development team time to fix detected defects without delaying the tool's delivery to customers.

This paper discusses the application of RBT in testing two EDA dynamic tools that have many testing scenarios to be covered due to their randomized flows. The paper will show the effect of using RBT in covering important flows and report their defects in early stages. The paper is organized as follows: Section 2 describes the approach of RBT and its main concepts and activities. Section 3 is a brief for the EDA tools that were tested using RBT approach. Section 4 presents the results and outcomes of the experiment. And finally, Section 5 summarizes the conclusions.

2 Risk-Based Testing Approach

Risk-based testing (RBT) is a testing technique that uses the risks of the software product as a guiding element to assist choices throughout the testing process. The tool's stakeholders should first identify the risks that will most probably impact the quality of the tested tool. And to be aligned with the concept of risk, the risk is a failure which hasn't occurred yet, but it may or may not occur in the future (Allam 2013).

The identified risks will be assessed, prioritized, and used in the guidance of the testing activities to be more powerful in reporting bugs and taking decisions early on, giving the development team enough time to fix any reported issues and then stakeholders reevaluate the risk of the tool to reassess used testing approaches and so on.

As shown in Figure 1 (Amland 1999), the RBT process has five key activities which are bounded in rectangles and they will be explained in this section.

2.1 Risk Identification

Risk identification is the activity used to detect any risks that can affect the project's ability to achieve its objective. This activity is a very early one as it is for reviewing the functional specification and product requirement documents. QA team performs this activity in parallel with the code implementation phase performed by the development team.

In this activity, it is necessary to review the functional specifications as developed by the development team to assess technical risks, detect any flaws, or unclear functionality, and to check whether there are any illogically supported flows.

Simultaneously, the product requirement documents should be evaluated to detect requirement risks such as illogical requirements, flows that conflict with the supported feature in the tool, and whether the functional specifications fulfill all the requirements.

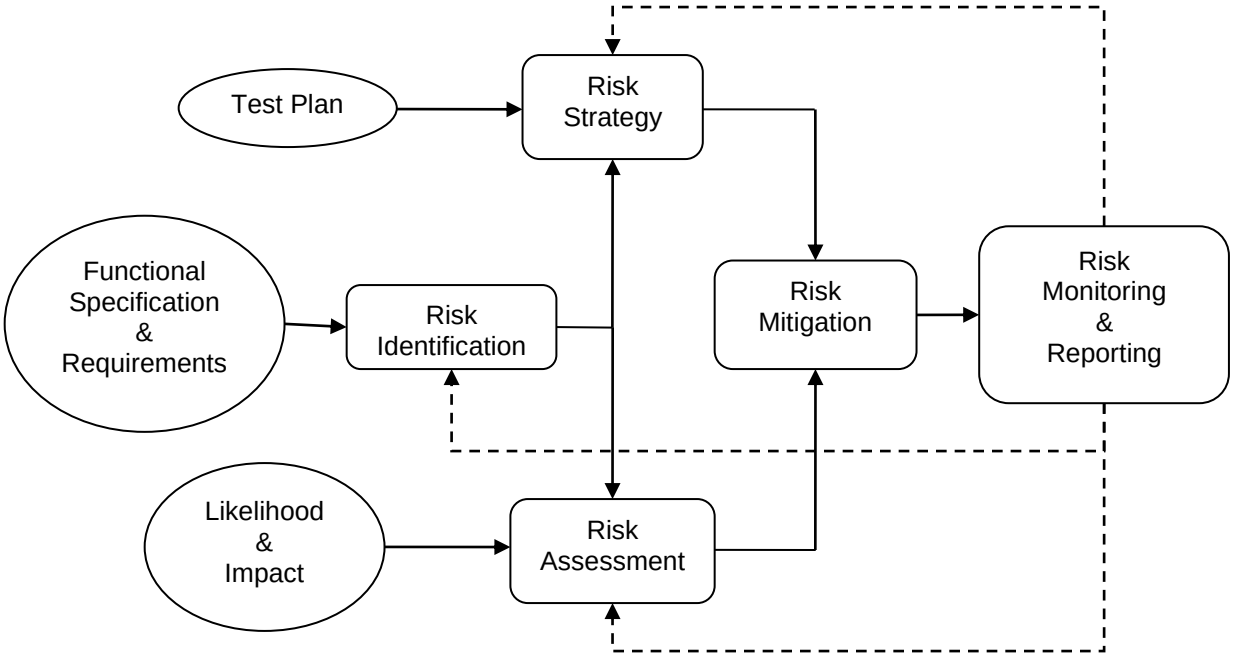

Figure 1: Risk Management Process

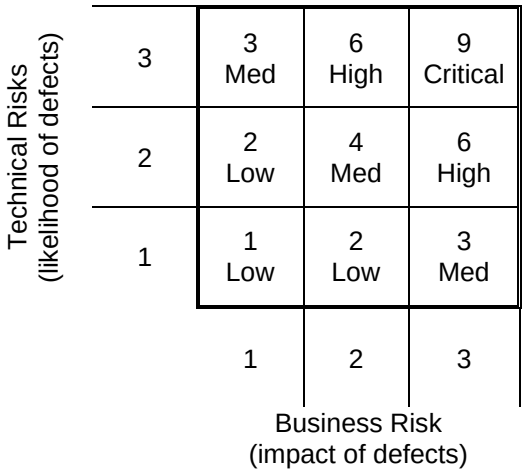

Figure 2: Risk Assessment by using grid 3x3

2.2 Risk Strategy

Before software testing begins, it is essential to think strategically about risks and understand what risk strategies are most likely to be successful, given the project context. Strategizing risks is used to develop testing procedures and alternatives for any contingent event. These plans will be used to guide risk management throughout software testing operations. During testing operations, four key risk strategies are employed: (i) Risk Avoidance is a technique of avoiding engaging in a feature or product because of the significant risk associated with it. It is also possible to disable or protect the code of this feature until all testing operations are completed. (ii) Risk Reduction which means taking steps to mitigate risks. This is accomplished by implementing tests provided in the test plan to expand the covered scenarios for the tested functionality. (iii) Risk Transfer, which is accomplished by outsourcing testing to a third party. This is because there is a lack of experience or resources. (iv) Risk Acceptance entails deciding not to deal with the risk or taking no action in response to it.

2.3 Risk Assessment

The identified risks will be assessed by the stakeholders of the tool in a brainstorming meeting. The testing team should describe the identified risks in detail with exampled scenarios. In this activity, the team provides a score for each risk from the perspective of (i) Likelihood of occurrence “Probability” (ii) the impact upon this occurrence “Consequence”.

The likelihood or the probability of the existence of a risk is focused on the technical risks. It comprises assessing the risk based on criteria such as software complexity, frequency of usage, potential defect locations and integration between legacy and new features.

The impact or severity of the risk is emphasized on business risk. As a result, the impact score is determined based on the criticality of the risk to the customer, the absence of solutions, and the customer's reliance on such a flow.

The levels or available scores for Likelihood and Impact differ from one study to another but in this case study, the grid of 3x3 (Figure *2*) is employed to represent 3 levels for each of the two factors. By multiplying the two scores, the identified risks will be analyzed for prioritization.

2.4 Risk Mitigation

The testing team will define the testing strategy, the number of testing cycles for each risk level, the scenarios that will be tested for each risk, and the number of tests for each scenario at this phase. These

judgments are made depending on the available time frame and testing resources. The testing team will then begin the design and implementation phase.

2.5 Risk Monitoring and Reporting

These activities are for tracking and evaluating the tested risk levels with the stakeholders. Because the risk process has become an integral element of the development process, it is critical to assess its effectiveness. If there is any vulnerability because of the risk management process that is not monitored and corrected, it will serve as a risk. The results of risk monitoring methods can be utilized in the development of new tactics and update current techniques that have proven unproductive (Blancher 2013).

3 Case Study

The tools examined using the RBT technique will be presented in this section. "Create Random Array" (CRA) and "VIA Random Placement" (VRP) are highly dynamic tools. Each of them has its own functionality but both are generating random output and have a high level of complexity

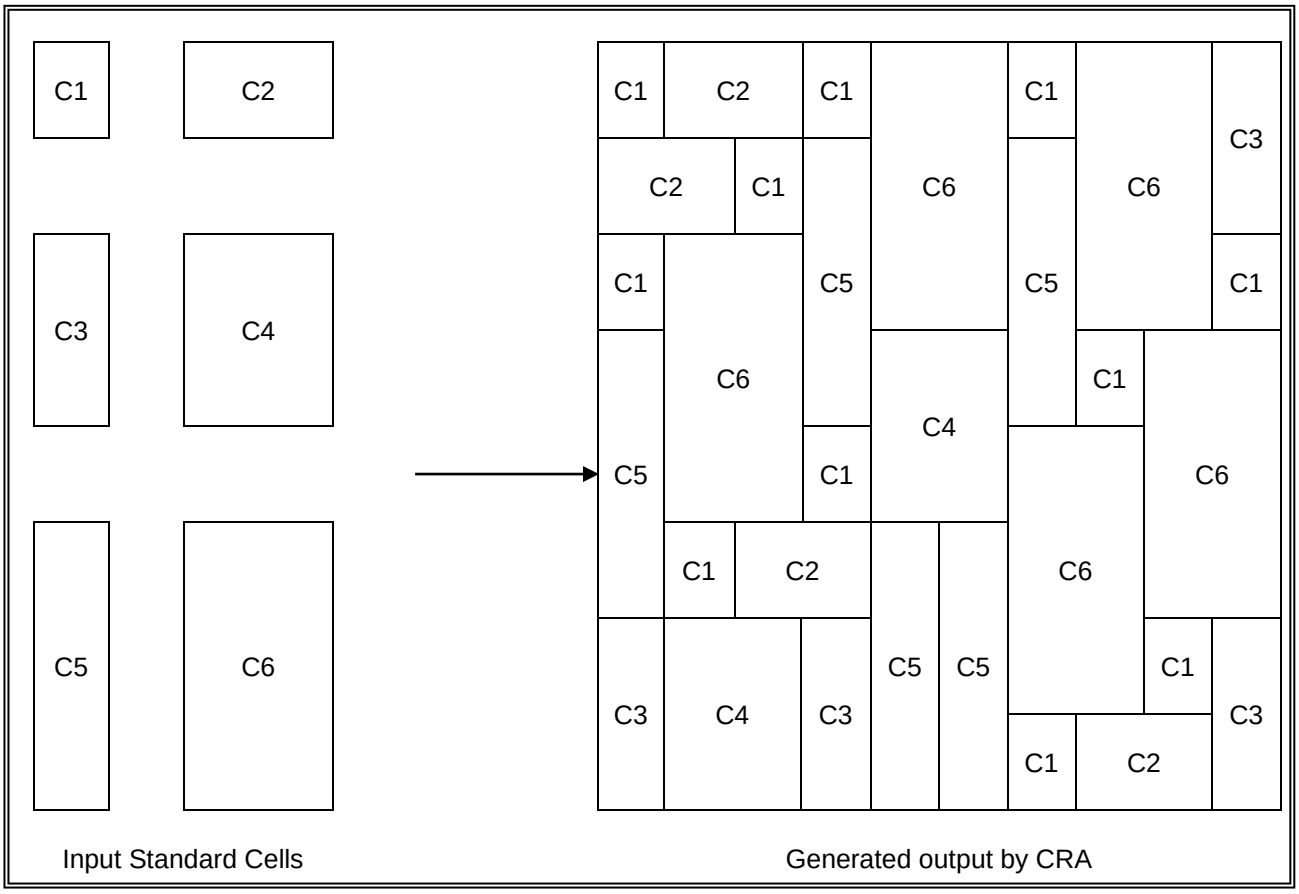

Figure 3: Example of input standard cells and generated output by CRA

3.1 Create Random Array (CRA)

For developing new technology nodes, it is necessary to have several layouts for process calibration. Specially for standard cell development teams as they need to make sure that any legal placement of their cells, with any combination of nearby placement or abutments, should produce a robust layout. Hence, they need to test a large variety of possible placements.

However, no realistic designs are accessible in the early stages. This highlights the significance of having synthetic layout generators. CRA is one generator that generates random arrays from provided cells. Where the user provides a tool list of standard cells and the quantity of random arrays to be created.

So as shown in Figure *3*, CRA is a utility that creates random placements of standard cells. By tackling the difficulty of multi-height standard cell placement, this tool aids quality assurance of standard cell libraries including hundreds or thousands of cells. The CRA utility may be used by library developers to confirm that all placement possibilities of surrounding abutted cells are allowed.

CRA takes an input layout with certain standard cells to randomly place the multi-height standard cells into arrays. The adjacent standard cell arrays are "Design Rule Check" (DRC) compatible. This application generates a layout file for each standard cell array placed, which may be used to design a placement solution to enhance the standard cell library.

3.2 VIA Random Placement (VRP)

The “VIA Random Placement” VRP flow generates and places vias in the design layout at random using a set of input criteria. Using the VRP flow allows "computer-aided design" CAD engineers and designers to do early pattern analysis for "lithographic" LITHO simulation on designs with metal and through layers to identify probable sources of LITHO hotspots.

Running VRP flow necessitates the application of DRC criteria, such as minimum enclosure and spacing requirements, as well as specifications for the types and size of the created vias. The flow randomly inserts and positions vias based on the input, while adhering to the rule checks and via requirements. According to the regulations, through placement happens at places of overlap and intersections between two layers.

The VRP requires an input layout database along with files for the via spacing and enclosure rules. The output is a layout database with the generated random vias with respect to the spacing and enclosure rules. This layout can be merged with the original one which is used as input to VRP and then it will be used in LITHO simulation activities.

4 Results

In this section, the metrics of applying RBT on each of CRA and VRP will be shown in detail. The metrics includes the time spent for each activity in the RBT process, the number of identified risks, the type of testing for each risk and the result of prioritizing them.

4.1 Create Random Array (CRA)

By studying the CRA feature under test, the testing team identified 36 risks and Table 1 shows the results of Risk Assessment activity.

Table 2 shows the planned time for the testing activities of CRA and as indicated in the table, the overhead of using RBT in testing the feature is roughly 90 minutes, with the balance of the time planned for regular testing activities that will be used in RBT method or various approaches. So, the overhead time is less than 4% of total time.

Risk Assessment	Testing Categories				Total
	Functional	Integration	Performance	Negative	
Critical	4	5	3	3	15
High	5	0	1	0	6
Med	7	1	0	3	11
Low	3	0	0	1	4
Total	19	6	4	7	36

Table 1: CRA Metrics | Risk Assessment

Testing Activity	Time in mins
Risk Identification	~60
Risk Assessment	~30
Prepare the testing environment & input data	~480 (1 working day)
Testing scenarios for each risk including automation	~53-70 (Depending on scenario's complexity)
Total	~2370 (5 working days)

Table 2: CRA Metrics | Time planning of testing activities

RBT produced a priority ranking of potential bugs having the greatest impact to customers During the Critical Risks testing, 5 bugs were discovered, all of them are found through testing the tool with the data provided by the customer and interrupt the main flow for customer scenarios. As a result of the early detection of these bugs, the development team had more time to fix them to meet the release cycle and improve the tool's quality for all usage scenarios that the customer had specified. Thus, during the critical risk testing phase, stakeholders desired to continue with the Risk Reduction Strategy as it was very essential to deliver these flows with high quality to the customer.

Due to time constraints, some lower priority features received only partial testing as some scenarios were broken by the discovered bugs. Since these were of lower importance and there was insufficient time to fully testing them, the stakeholders preferred not to delay the release and to change the testing strategy for them from Risk Reduction Strategy to Risk Avoidance Strategy by protecting these flows under beta variable until all testing operations were completed.

Finally, in most testing approaches, performance testing is conducted in the latter stages of the testing process after functional testing is completed. Because performance was critical to the customer in this feature, there are some risks which are categorized under performance testing ranked as critical or high in Risk Assessment. So, these scenarios were tested before some scenarios under functional testing.

4.2 VIA Random Placement (VRP)

Regarding VRP feature under test, the testing team identified 28 risks and the same as CRA, Table 3 shows the results of Risk Assessment activity.

Table 4 shows the planned time for the testing activities of VRP and as indicated in the table, the overhead of using RBT in testing the feature is around 80 minutes which is less than 6% of total time.

During the Risk Identification process, there is a scenario that is not well stated in the requirements documents and functional specifications. This case revealed a discrepancy between the intended output of the code in implementation and the expected result by the customer. As a result, this is considered a bug, and it is detected at a very early stage of testing. As a result, the development team was given the opportunity to resolve this problem as soon as feasible.

There were no further bugs introduced throughout the testing scenarios for this functionality. As a result, the testing strategy utilized to test this feature is the Risk Reduction Strategy. This is accomplished by testing prioritized scenarios that have been validated through the Risk Assessment process to increase the tool's coverage and satisfaction.

Risk Assessment	Testing Categories				Total
	Functional	Integration	Performance	Negative	
Critical	6	5	0	2	13
High	3	0	0	6	9
Med	0	1	0	0	1
Low	1	0	2	2	5
Total	10	6	2	10	28

Table 3: VRP Metrics | Risk Assessment

Testing Activity	Time in mins
Risk Identification	~60
Risk Assessment	~20
Prepare the testing environment & input data	~240 (0.5 working day)
Testing scenarios for each risk including automation	~20-60 (Depending on scenario's complexity)
Total	~1440 (3 working days)

Table 4: VRP Metrics | Time planning of testing activities

It appears that RBT did not introduce any advantages over traditional testing techniques as the discrepancy between functional specifications and requirements documents is always detected in a very early stage in the traditional testing techniques, but in fact, the goal of testing is to increase the quality and satisfaction with the tool, not just to catch bugs. RBT was reporting at an early stage the satisfaction of the testing team with the tool quality, and it can be used in reallocating resources for testing other low-quality tools.

5 Conclusion

Using the RBT approach in these case studies demonstrated that the technique focuses on flows that are more likely to fail or are more vital to the customers. Consequently, it allows stakeholders to make early decisions to lower the risk of the delivered feature or product, hence shortening their time to market.

These case studies illustrate two distinct situations. A low-quality feature that RBT highlights concerns early on, and it gave the development team the chance to fix them and regarding some lower scenarios which there is not enough time to test them, the stakeholders preferred to use a different Risk Strategy since these scenarios are not important to the customer and not fully qualified. The second situation had only one issue in the provided documents and it was of good quality, thus the Risk Strategy adopted was to lower the risk by making early fixes to detected bugs and increasing satisfaction with the feature before going live.

The overhead in adding RBT to the testing process is around 4-6% of the total time planned for the full testing process. By applying RBT technique, major defects were usually reported at an early stage. Also, instead of blindly following traditional testing process, such as applying performance testing at the end, RBT raised awareness of the critical need for high performance over other features in this release and the order of testing became dependent on the customer's needs and the risky flows.

References

Nicely, Thomas. 2011. "Pentium FDIV flaw FAQ". trnicely.net. Archived from the original on June 18, 2019. Retrieved June 18, 2019. http://www.trnicely.net/pentbug/pentbug.html (Internet archive)

Ng A, Markov IL. 2005. "Toward quality EDA tools and tool flows through high-performance computing". *Sixth international symposium on quality electronic design (isqed'05) 2005 Mar 21 (pp. 22-27).*

Redmill, Felix. 2004. "Exploring Risk-based Testing and Its Implications" *Software Testing, Verification and Reliability, Vol. 14, No. 1, March 2004*

Pressman, R. 1995, *Engenharia de Software*. 1st ed. Makron Books, São Paulo, Brazil.

Felderer, M. and Schieferdecker, I., 2014. "A taxonomy of risk-based testing". *International Journal on Software Tools for Technology Transfer*, 16(5), pp.559-568

Alam, Md. Mottahir. 2013. "Risk-based Testing Techniques: A Perspective Study" *International Journal of Computer Applications (0975 – 8887) Volume 65– No.1, 2013*

Amland, Stale. 1999. "Risk Analysis Fundamentals and Metrics for software testing including a Financial Application case study" *5th International Conference EuroSTAR '99, November 8 - 12, 1999, Barcelona, Spain*

Blancher, Nicolas. 2013. "Systemic Risk Monitoring (—SysMo‖) Toolkit— A User Guide" *IMF Working Paper (International Monetary Fund).*