testinsight

Accelerating ATE Pattern Bring Up and Debug

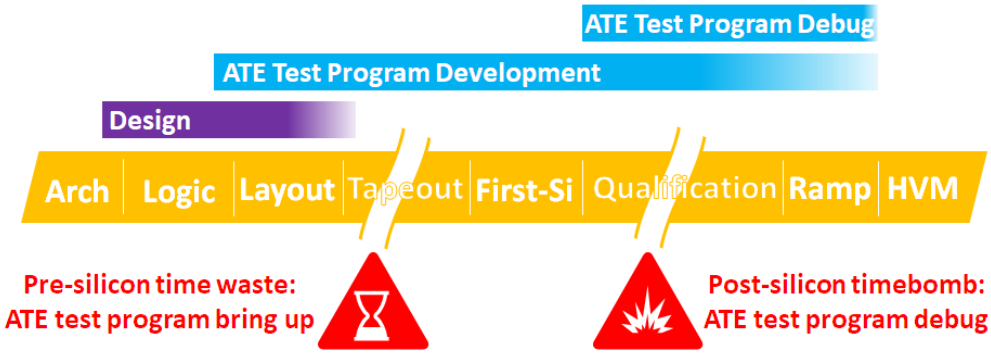
Eran Belaish,
Director of Product Marketing, TestInsight

White Paper

Both pre-silicon ATE pattern bring up and post-silicon failure debug can be significantly accelerated in many cases, preventing costly pre-silicon time waste and nasty post-silicon surprises.

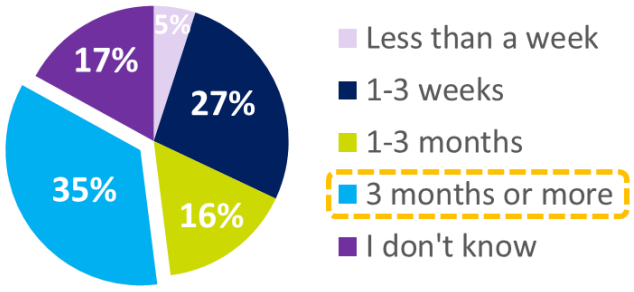
Introduction – Preventing Time Waste and Nasty Surprises

Remember that feeling when ATE patterns fail but nobody knows why? Preventing the frustration and waste of time involved in debugging such failures was the motivation for writing this whitepaper. But post-silicon failures are not the only concern of test engineers, which are also interested in making pre-silicon ATE pattern bring up quick and effective. And so, this whitepaper covers both pattern bring up and failure debug, hoping to accelerate test program development from end to end.



Pre-silicon ATE pattern bring up and post-silicon failure debug are two major concerns of test engineers

The recent poll results below show how painful these post-silicon timebombs can be, often taking several months from first silicon until shipping prototypes to customers.

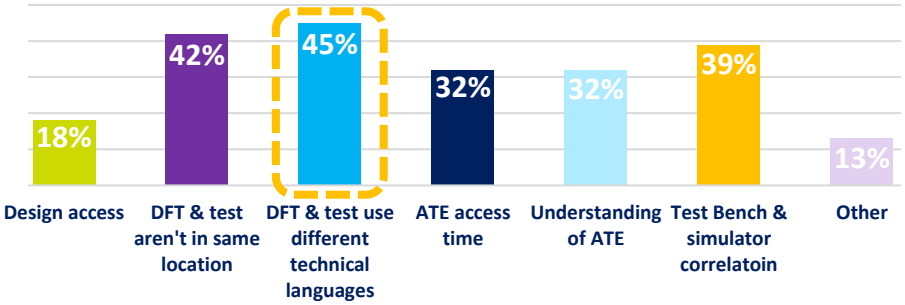


From first silicon until shipping prototypes to customers: test program dev, pattern bring up, and debug

A second poll below looked into main causes of delays in first silicon bring up and debug, which explain the long time it takes until shipping prototypes to customers. The main reason turned out to be the gap between DFT and test engineering.

It often takes several months from first silicon to shipping prototypes to customers.

One of the main causes of these painful delays is that DFT and test engineers use different technical languages.



Causes of delays in first silicon bring up and debug

To address these issues, this document begins with looking into methods for accelerating pre-silicon ATE pattern bring up and continues to surveying techniques for accelerating post-silicon failure debug. It then concludes with exploring specific tools that can be used for achieving the suggested accelerations, as some of the methods mentioned below might not be supported by all toolsets.

Accelerating Pre-Silicon ATE Pattern Bring Up

Identifying the major time wasters for ATE pattern bring up is an important first step towards accelerating the development process. Once these are identified, there are several remedies that can be mixed and matched according to the specific requirements of each project.

Handling slow pattern conversion

Slow conversion of test patterns into ATE format can sometimes waste several days, and remedying it depends primarily on the performance and features of the conversion tools. The first method is to use faster conversion tools, bearing in mind that differences in performance can be as high as 200X between different tools. It is therefore recommended to conduct proper evaluation and benchmarking, including conversion performance measurement, before deciding on a toolset.

Skipping compilation to a binary format is another method that some conversion tools support as they create patterns directly in ATE binary format without compiling them. This saves up to 70% of conversion time and also prevents compilation problems.

The third method is relevant when using testers with multiple clock domains. Leveraging this ATE feature often requires running several conversion passes, one for each clock domain, and then combining them manually. Instead of this time-consuming process, some tools can handle all clock domains in a single pass saving both runtime and manual integration work.

Verifying ATE-compliance of test patterns

Design engineers, which are often unaware of ATE restrictions, might create test patterns that are not ATE-compliant, leading to lengthy iterations between designers and test engineers. To prevent it, some tools can verify that the test patterns comply

with the target ATE, covering both static compliance such as timing restrictions and dynamic hazards such as loop counter limits. Performing this check early in the project flow can save a lot of problems later on.

Avoiding unnecessary manual work

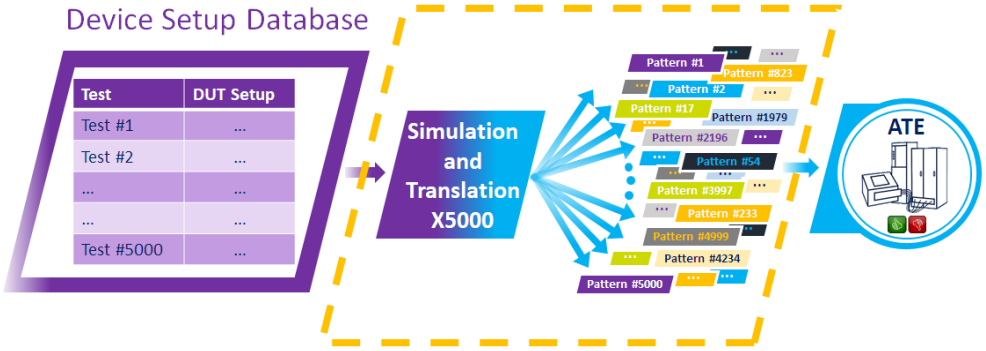
Using advanced ATE features can simplify test pattern development and save a lot of time-consuming manual work. For example, Xmodes can be used when patterns are too big for the ATE, saving the effort of modifying the tests manually. Another example is using the ATE's multiple clock domains for concurrent testing. Instead of spending considerable time during pattern development to optimize ATE runtime, simply use the tools to split single clock domains into multiple clock domains.

Advanced tools features can also be used to automate manual tasks and accelerate test development. For example, instead of manually integrating multiple patterns into a single test program, use the tools to do it automatically. Another example is a GUI-based UI that displays pattern waveforms and can facilitate faster test development.

There are several methods for accelerating ATE pattern bring up that can be mixed and matched. For example, speeding up pattern conversion using faster tools, automating development tasks using advanced features, and simplifying device setup maintenance using dedicated tools.

Preventing maintenance and repeated simulations of multiple device setups

Each production test has a specific device setup, which could differ in just a few register bits from another setup. This is how we end up with thousands of setups leading to thousands of patterns, which are time-consuming to maintain and simulate.



How to maintain thousands of patterns without harming cost, quality, and schedules?

Simplifying setup maintenance and avoiding repeated simulation of multiple setups requires both advanced tools and improved methodologies. These tools can read the setup database of the Device Under Test (DUT) and produce optimized test patterns at runtime without time-consuming simulation and translation. This allows sharing the database as a single source of truth among design, test, and silicon validation (SV) teams, leading to faster development using fewer resources.

"A Virtual ATE is a very effective tool for performing root cause analysis. Once an ATE pattern fails, a Virtual ATE allows running this pattern in simulation with far greater debug capabilities than the physical tester provides."

To learn more about this unique tool, download the Virtual ATE whitepaper here.

Accelerating Post-Silicon ATE Pattern Debug

To quickly handle failing patterns during production testing it is essential first to detect the root cause of each failure out of several different potential reasons:

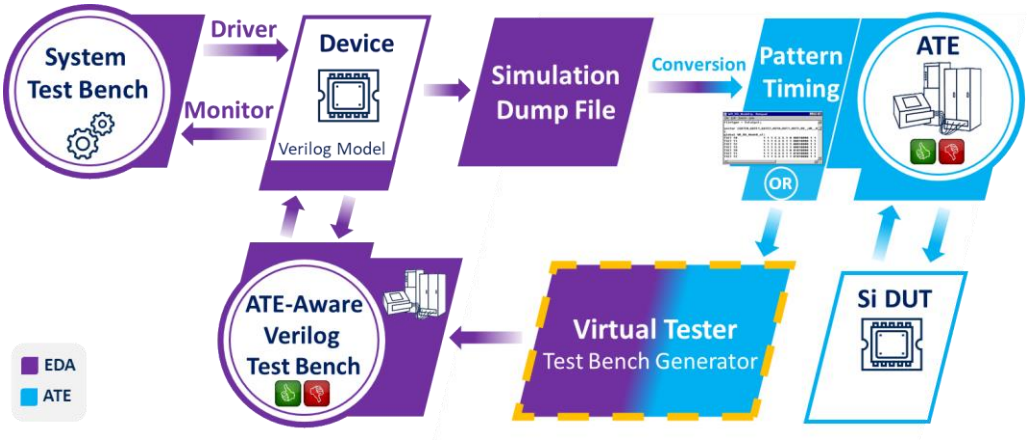
- **Design issues** such as forgetting forced values in a BIST
- **Test engineering issues** such as a faulty pattern conversion from EDA to ATE
- **Production issues** such as actual product defects

An effective root cause analysis saves both valuable post-silicon debug time and blamestorming, where the different teams blame each other for the failures.

Quick root cause analysis using a Virtual ATE

A Virtual ATE is a very effective tool for performing root cause analysis. It converts test program files into an ATE-aware Verilog model that emulates how ATE drives the DUT. Once an ATE pattern fails, a Virtual ATE allows running this pattern in simulation with far greater debug capabilities than the physical tester provides. It can verify that tests are ATE-compatible and identify pattern conversion flaws, quickly indicating if failures are caused by design issues, test engineering issues, or actual production issues.

A virtual ATE can also bridge the gap between design and test engineers, which is one of the main reasons for delays in first silicon bring up and debug, as explained above. Test engineers can use the tool to generate an ATE-aware Verilog test bench used for ATE emulation by the design team, allowing the different teams to use a common language.



A virtual ATE tester closes the open loop between design and test, simplifying production test debug

To learn more about this unique tool, download the Virtual ATE whitepaper here.

"When a pattern fails on ATE, a simple yet effective check is to compare the pattern with its source EDA format using a cross-format comparison tool. Such a tool can track down the root cause of any difference and immediately tell if anything went wrong."

Compare failing ATE patterns with their source EDA format

When a pattern fails on ATE, a simple yet effective check is to compare the pattern with its source EDA format using a cross-format comparison tool. Such a tool can track down the root cause of any difference and immediately tell if anything went wrong when converting the original pattern into ATE format.

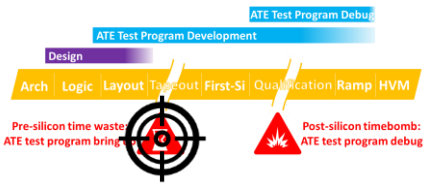
Drowning in formats – A tool that compares different test formats simplifies and accelerates debug flow

Prevent Failures in the First Place – Debug Before Silicon

Why wait for silicon to find out? It is possible not to be surprised again by ATE failures using pre-silicon validation of the test program, which saves valuable time and resources during production testing. Many of the tools and methodologies mentioned above can be used for pre-silicon pattern validation, simplifying and accelerating post-silicon debug as a result. For example, checking for tester compliance, comparing ATE patterns with their source EDA format, and using a Virtual ATE can validate the test program long before silicon is available.

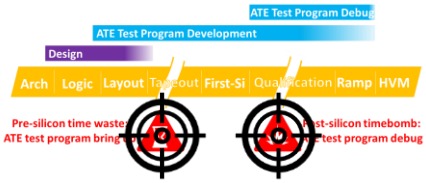
Conclusion

ATE pattern bring up and debug can be significantly accelerated using the right tools and methodologies, saving valuable resources. As some of the methods mentioned above are not supported by all toolsets, here are a few examples of specific tools that can be used to achieve such acceleration:



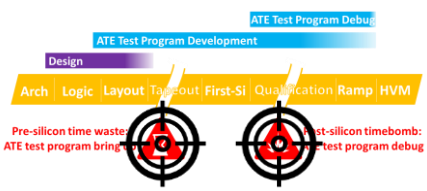
TDL - **Converting design vectors into ATE programs**

The fastest conversion tool available combines ease of use and GUI with effective features facilitating faster test development, such as skipping compilation to a binary format and supporting multiple clock domains.



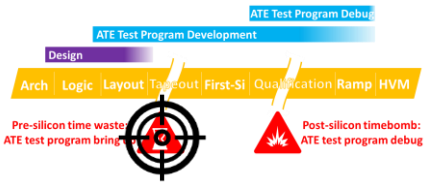
Virtual Tester (VT) - **Pre-silicon test program validation using a virtual ATE**

VT Creates an ATE-aware Verilog model, which allows pre-silicon testing with the same results as post-silicon ATE, making production test debug shorter and more predictable. VT can be used both during post-silicon debug for a quick root cause analysis and for pre-silicon test program validation to prevent failures in the first place.



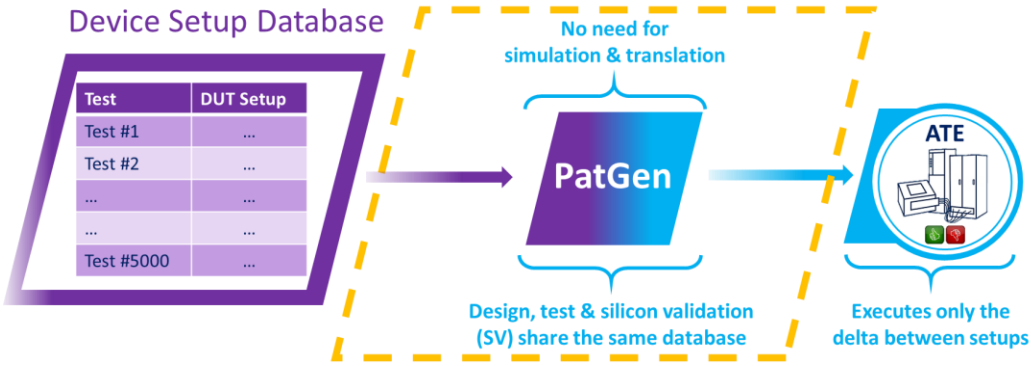
TestDiff – **Cross-format test comparison**

TestDiff compares test vectors of different EDA and ATE formats at waveform level for improved quality and debug flow. This allows comparing failing ATE patterns with their source EDA format and track down the root cause of any difference. Same as VT, TestDiff can also be used both during post-silicon debug and for pre-silicon test program validation.

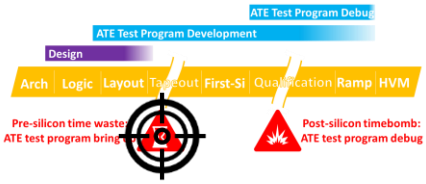


PatGen - Runtime pattern generation with no simulation or translation

PatGen simplifies DUT setup maintenance and saves valuable resources by producing optimized test patterns at runtime with no need to simulate and translate them. This allows using a single database shared by design, test, and SV teams, which accelerates development and requires fewer resources to handle. PatGen also provides transaction-level test creation, update, and debug instead of bit-level ones, which makes debug faster and more intuitive thanks to a higher level of abstraction.



When using PatGen there is no need to handle patterns anymore

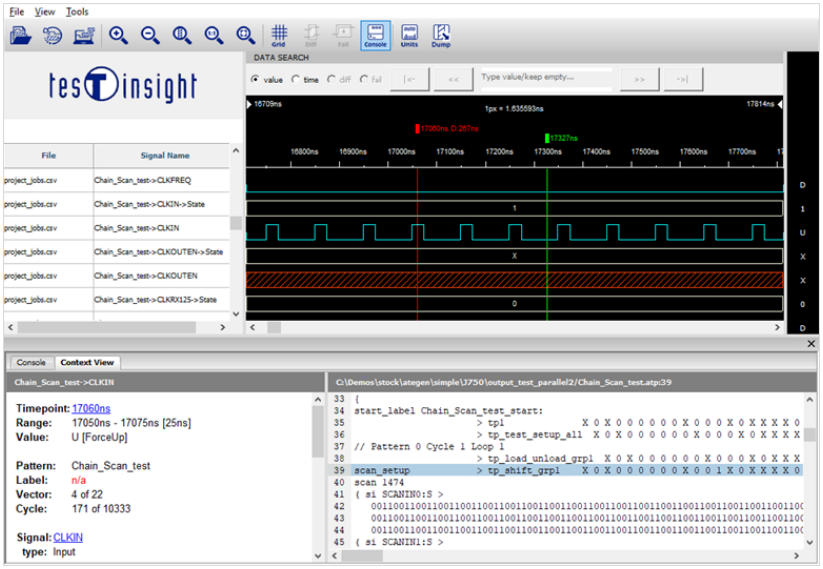


Test Vector Studio – GUI-based test pattern development

Test Vector Studio combines a user-friendly GUI with powerful pattern development tools such as Tester Rule Check (TRC) and Test Vector Editor. The GUI allows to view and analyze ATPG and ATE patterns at both waveform level and source code one, simplifying test development. TRC provides early vector verification, checking both static compliance and dynamic hazards with the specific target ATE. Test Vector Editor supports vector manipulation, parallelization, and compression, optimizing tester utilization and allowing greater ATE selection.

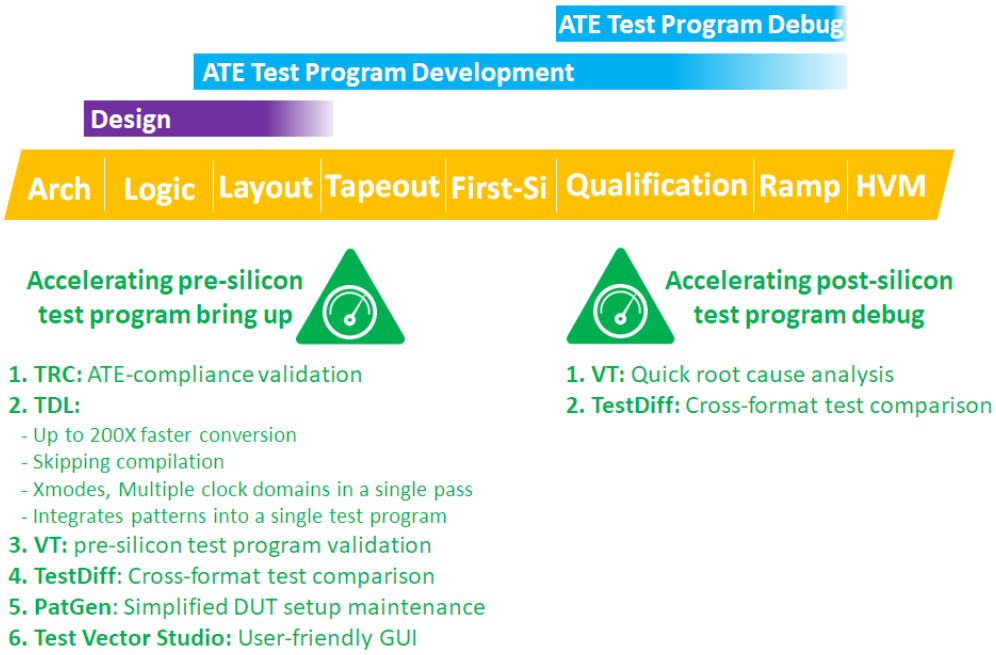
End-to-end acceleration can be achieved using tools such as these:

- TDL: the fastest conversion tool available
- Virtual Tester: a virtual ATE
- TestDiff: cross-format test comparison
- PatGen: simplifies DUT setup maintenance
- Test Vector Studio: GUI-based test development



Test Vector Studio - Using a GUI simplifies and accelerates both pattern development and failure debug

As depicted below, putting it all together in a single flow can provide the desired end-to-end acceleration, prevent nasty post-silicon surprises, and simplify test program debug.



End-to-end acceleration helps to deliver on time and prevent nasty post-silicon surprises

To learn more about useful tools for your test program, please contact info@testinsight.com