DVT
Eclipse IDE

For e, SystemVerilog, Verilog, Verilog-AMS and VHDL

The complete development environment for hardware design and verification

10 REASONS TO CHOOSE DVT

1. See the errors flagged by incremental compilation as you type.
2. Write code faster using autocomplete and error fix suggestions.
3. Quickly move around in the source code using hyperlinks.
4. Query the project database to accurately locate relevant information.
5. Easily create and reuse code and project templates.
6. Continuously improve the source code using refactoring operations.
7. Easily understand the project structure using high-level views.
8. Visualize the project architecture using UML and HDL diagrams.
9. Trace signals throughout the design.
10. Place reminders and track tasks.

BENEFITS

- Increases productivity and reduces time to market.
- Speeds up source code development.
- Enables efficient reading and understanding of complex source code.
- Simplifies debugging and legacy source code maintenance.
- Ensures higher quality source code development.
- Streamlines code review.
- Accelerates language and methodology learning.

OVERVIEW

Design and Verification Tools (DVT) is an integrated development environment (IDE) for the e language, SystemVerilog, Verilog, Verilog-AMS and VHDL. It is similar to well-known programming tools like Visual Studio®, NetBeans®, and IntelliJ®.

DVT consists of a parser, a smart code editor, an intuitive graphical user interface, and a comprehensive set of features that help with code writing, inspection, navigation, and debugging. DVT provides capabilities that are specific to the hardware design and verification domain, such as design diagrams, signal tracing, and verification methodology support.

DVT is a powerful tool that allows engineers to overcome the limitations of plain text code editors and address today’s project complexity more efficiently. It enables faster and smarter code development and simplifies legacy code maintenance for novices and experts alike.

IEEE Standard Compliant Parser

The DVT parser is compliant with the IEEE 1800 SystemVerilog, IEEE 1647 e Language, and IEEE 1076 VHDL standards. Besides fully supporting the design and verification languages, DVT also signals the use of non-standard compliant language constructs, which ultimately increases simulator compatibility.

Integration with Simulators and Other Tools

DVT integrates seamlessly with all major hardware simulators to simplify simulation analysis. It also works with revision control systems like Git, Subversion, ClearCase and bug tracking systems such as Bugzilla and ClearQuest.

Eclipse Ready

DVT is built on the powerful Eclipse Platform used by millions of engineers worldwide and inherits the best features and practices collected into the platform core. The Eclipse Platform’s extensible architecture allows DVT to integrate within a large plug-in ecosystem and work flawlessly with third-party extensions.
EFFICIENT CODE WRITING AND SIMPLIFIED MAINTENANCE

Advanced Code Editing Features
DVT incorporates advanced code editing features such as:

- On the fly syntactic and semantic checking
- Error fix suggestions
- Autocomplete and autoinstance
- Source code refactoring operations
- In-line reminders and task tracking
- Customizable code and project templates
- Macro expansion
- Dedicated wizards to generate getters and setters or override functions
- Highly configurable source code formatting
- Integration with Revision Control Systems
- Vi and Emacs emulation

DVT performs on-the-fly incremental compilation. There is no need to invoke the simulator to make sure the code compiles with no errors. DVT’s smart editor highlights the errors in real time, as you type. As a result, one can make the necessary corrections on the spot. To assist with error correction, DVT also provides fix proposals like “did you mean” when detecting a potential typo or “update instance” when module ports changed. Moreover, the developer or reviewer can quickly locate and fix various issues spread throughout the code using the Problems View where all errors and warnings are listed.

Autocomplete provides a context-sensitive list of proposals for partially entered text. This capability helps avoid typos and eliminates the need to search for definitions in other files. Autoinstance allows engineers to quickly instantiate and connect a module or entity when needed.

Refactoring allows users to perform semantic changes in code. While a plain text editor or grep/sed utility is limited to simple search and replace actions, DVT can accurately perform powerful operations like “rename method foo() of class bar”, “rename signal x of module y”. All the definitions and places where the method or signal is used are precisely updated. Users can also “extract this piece of code into a separate function”, “add a new port p to module m” or “propagate signal x from module y to module z across the design hierarchy”. Refactoring helps engineers avoid tedious and error-prone operations like scrolling through long lists of irrelevant plain text matches or repetitive copy and paste.

In-line reminders, such as TODO or FIXME, and other customized tags can be placed in code comments. These reminders are listed in the Tasks View. With a simple click, users can jump directly to the source code lines of the selected task. In-line reminders are also useful when performing code review. Since the action items are inside the source code, they are always up to date and visible to the whole team. The task list can also be used to estimate the code status and remaining development effort.

Code templates are parameterized code snippets. Combined with TODO tags, this capability enables users to easily follow the project development guidelines. Project templates generalize code templates and allow engineers to start a new project from a reference project layout.

Errors inside macros are difficult to investigate. With the DVT’s macro expansion feature, engineers can examine and debug macro code fragments in context with the source code.
Code and Project Navigation Features

Maintaining tens of thousands of lines of code can be challenging. DVT simplifies maintenance by providing capabilities like:

- Hyperlinks
- Project database queries
- Semantic search for usage
- Structural views
- Signal tracing
- UML and HDL diagrams
- Comments and macro or parameter values in tooltips

These features enable users to navigate easily through complex code, locate the relevant information faster, and understand the source code quickly. They also contribute to reduce project costs, by allowing users to avoid locking a simulator license just to inspect the design hierarchy or the verification environment architecture.

Hyperlinks help navigate faster through multiple project files. It practically eliminates the need of using the grep command or memorizing various details such as file names and locations. To look up the definition of an element, one can simply hover the mouse over the element name to turn it into a hyperlink. In this way, users save time by jumping directly to the element definition instead of searching for it.

Project database queries allow users to quickly locate specific elements. For example by typing a few letters in a search bar, one can look for a specific module, entity, class, macro, assertion or coverage item.

Semantic search for usage lets users quickly find out who is calling “method foo”, who is using “signal clk of module fifo” or “what are all the constraints on packet size”.

Unlike plain text grep/sed searches, the semantic search results are accurate. In other words, a search for “who is calling method foo of class a” will not match calls to “method foo of some other class b”.

In order to help understanding the project architecture, DVT offers structural views for examining class hierarchies, function call hierarchies, design and verification hierarchies, and aspect-oriented programming (AOP) layers.

Using the signal tracing functionality, designers can effortlessly locate the signal source, an operation called “trace drive”, or the signal destination, an operation called “trace load”. The signal trace is presented in the design hierarchy tree and can also be visualized as a diagram.

DVT enables engineers to inspect a project through diagrams. Designers can use HDL diagrams like schematic, state machine or flow diagrams. Verification engineers can use UML diagrams like inheritance or collaboration diagrams. Diagrams are hyperlinked and synchronized with the source code and can be saved for documentation purposes.
Users can easily browse through UVM-based classes like agents, monitors or sequences, examine component and sequence trees, visualize architecture diagrams, search for factory related constructs that may influence the testbench behavior and generate code using UVM specific code templates and wizards.

DVT includes an automated UVM compliance-checking capability to help verification engineers automate the process of checking their VIP against the UVM Compliance Checklist. With this capability, they can effortlessly and consistently apply the UVM rules and ensure compliance to the official UVM User Guide.

One can invoke the simulator and browse its output in a dedicated Console View. The view highlights simulation errors and warnings and provides hyperlinks that take the user directly to the source code. To simplify reading, different colors are assigned to the log in accordance with the message source and severity. By providing simulator log recognition, DVT significantly simplifies simulation analysis and debugging.

In addition, the external builder integration enables engineers to use any code analysis tool directly in DVT. Errors and warnings are back annotated to the source code, which speeds up debugging.

DVT automatically generates HTML documentation from code comments. The documentation is always synchronized with the source code and easily available to be shared with other team members or customers. The generated documentation also includes structural information like UML and HDL diagrams and design hierarchies. Moreover, any additional documentation can be easily linked with the generated documentation.

The cross-language capabilities allow users to work with source code written in multiple languages and easily understand the whole design.

Features like hyperlinks, design hierarchy browsing, HDL diagrams, and signal tracing work across SystemVerilog/Verilog, VHDL and C/C++ sources. For example one can click on an instance in Verilog and jump to its VHDL definition or click on a SystemVerilog DPI call and jump to its C definition.

Faster and Smarter Code Development
DVT was developed with maximizing the design and verification productivity in mind. Users do not need to switch from the editor to the simulator, browser or console and thus, they can focus on code writing or review. Moreover, DVT reduces substantially the time spent performing repetitive tasks such as locating a class or module definition, finding all places where a function is called, renaming a variable, and searching for relevant information in large source code files or documentation.

By using hyperlinks, autocomplete, in-line documentation, semantic search, task-tracking, and smart log view features users can find what they need through a single click or shortcut. As a result, the speed and quality of code development increase significantly.

Efficient Project Management
DVT helps manage design and verification projects more efficiently. The ability to easily review the source code using features like hyperlinks, project database queries, structural views and HDL or UML diagrams enables both managers and engineers to see the whole picture clearly and control a project from a higher perspective. Action item tracking using in-line reminders listed in the Tasks View allows the team lead or manager to better organize and control the development effort.

Lower Language Learning Curve
Beginners feel comfortable with the DVT friendly user interface. In addition, the combination of features such as compilation errors highlighted as you type, error fix proposals, autocomplete, and code templates together with the access to integrated documentation speed up the learning process.

The technical support team is available to promptly answer your questions, provide you with training, and work with you to determine your needs. Your requirements and feedback are important. Whether you are looking for technical support or new features to improve your design and verification flow, AMIQ’s technical support team strives to answer your requests in a timely manner.

Copyright 2017 AMIQ EDA S.R.L. All rights reserved.
Eclipse and Eclipse Ready are trademarks of Eclipse Foundation, Inc.
The information contained herein is subject to change without notice.
DVT-0617-A4