DVT
Eclipse IDE

For SystemVerilog, Verilog, Verilog-AMS, e, VHDL, PSL, PSS, UPF, and CPF

The complete development environment for hardware design and verification

10 REASONS TO CHOOSE DVT Eclipse IDE

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 your 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) Eclipse IDE is an integrated development environment for SystemVerilog, Verilog, Verilog-AMS, the e language, VHDL, PSL, PSS, UPF, and CPF. It is similar to well-known programming tools such as Visual Studio®, NetBeans®, and IntelliJ®.

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

DVT IDE is a powerful tool that allows your 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 Compiler

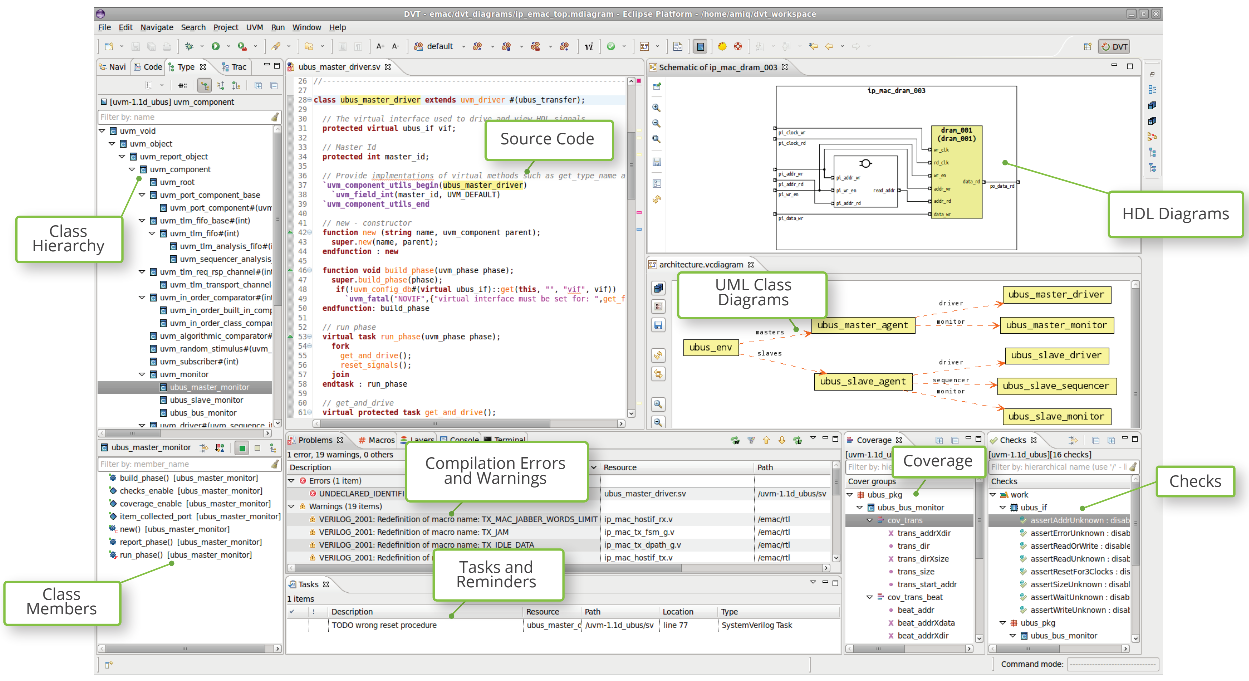
The compiler is compliant with the IEEE 1800™ SystemVerilog, IEEE 1647™ e Language, IEEE 1076™ VHDL, IEEE 1850™ PSL, and IEEE 1801™ Low Power standards. Besides fully supporting the design and verification languages, DVT IDE also flags the use of non-standard compliant language constructs, which ultimately increases simulator compatibility.

Integration with Simulators and Other Tools

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

Eclipse Ready

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


DVT - emac/dvt_diagrams/ip_emac_top.mdiagram - Eclipse Platform - /home/amiq/dvt_workspace
Source Code
Class Hierarchy
HDL Diagrams
UML Class Diagrams
Compilation Errors and Warnings
Coverage
Checks
Tasks and Reminders
Class Members

ubus_master_driver.sv

```
//-----------------------------------------------------------------

class ubus_master_driver extends uvm_driver #(ubus_transfer);

  // The virtual interface used to drive and view HDL signals
  protected virtual ubus_if vif;

  // Master Id
  protected int master_id;

  // Provide implmentations of virtual methods such as get_type_name a
  `uvm_component_utils_begin(ubus_master_driver)
    `uvm_field_int(master_id, UVM_DEFAULT)
  `uvm_component_utils_end

  // new - constructor
  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction : new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
     if(!uvm_config_db#(virtual ubus_if)::get(this, "", "vif", vif))
       `uvm_fatal("NOVIF",{"virtual interface must be set for: ",get_f
  endfunction: build_phase

  // run phase
  virtual task run_phase(uvm_phase phase);
    fork
      get_and_drive();
      reset_signals();
    join
  endtask : run_phase

  // get_and_drive
  virtual protected task get_and_drive();
```

Problems

1 error, 19 warnings, 0 others

Description	Resource	Path
Errors (1 item)		
UNDECLARED_IDENTIFI	ubus_master_driver.sv	/uvm-1.1d_ubus/sv
Warnings (19 items)		
VERILOG_2001: Redefinition of macro name: TX_MAC_JABBER_WORDS_LIMIT	ip_mac_hostif_rx.v	/emac/rtl
VERILOG_2001: Redefinition of macro name: TX_JAM	ip_mac_tx_fsm_g.v	/emac/rtl
VERILOG_2001: Redefinition of macro name: TX_IDLE_DATA	ip_mac_tx_dpath_g.v	/emac/rtl
VERILOG_2001: Redefinition of	ip_mac_hostif_tx.v	/emac/rtl

Tasks

1 items

✓	!	Description	Resource	Path	Location	Type
		TODO wrong reset procedure	ubus_master_c	/uvm-1.1d_ubus/sv	line 77	SystemVerilog Task

DVT Eclipse IDE Overview

EFFICIENT CODE WRITING AND SIMPLIFIED MAINTENANCE

Advanced Code Editing Features

DVT IDE 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 IDE performs on-the-fly incremental compilation. There is no need to invoke the simulator to make sure the code compiles with no errors. Its smart editor highlights the errors in real time, as you type. As a result, users can make the necessary corrections on the spot. To assist with error correction, DVT IDE also provides fix proposals such as "did you mean" when detecting a potential typo and "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 IDE can accurately perform powerful operations like "rename method foo) of class bar" and "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," and "propagate signal x from module y to module z across the design hierarchy." Refactoring helps engineers avoid tedious and error-prone operations such as scrolling through long lists of irrelevant plain text matches or repetitive copy and paste.

In-line reminders, such as TODO, FIXME, and 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.

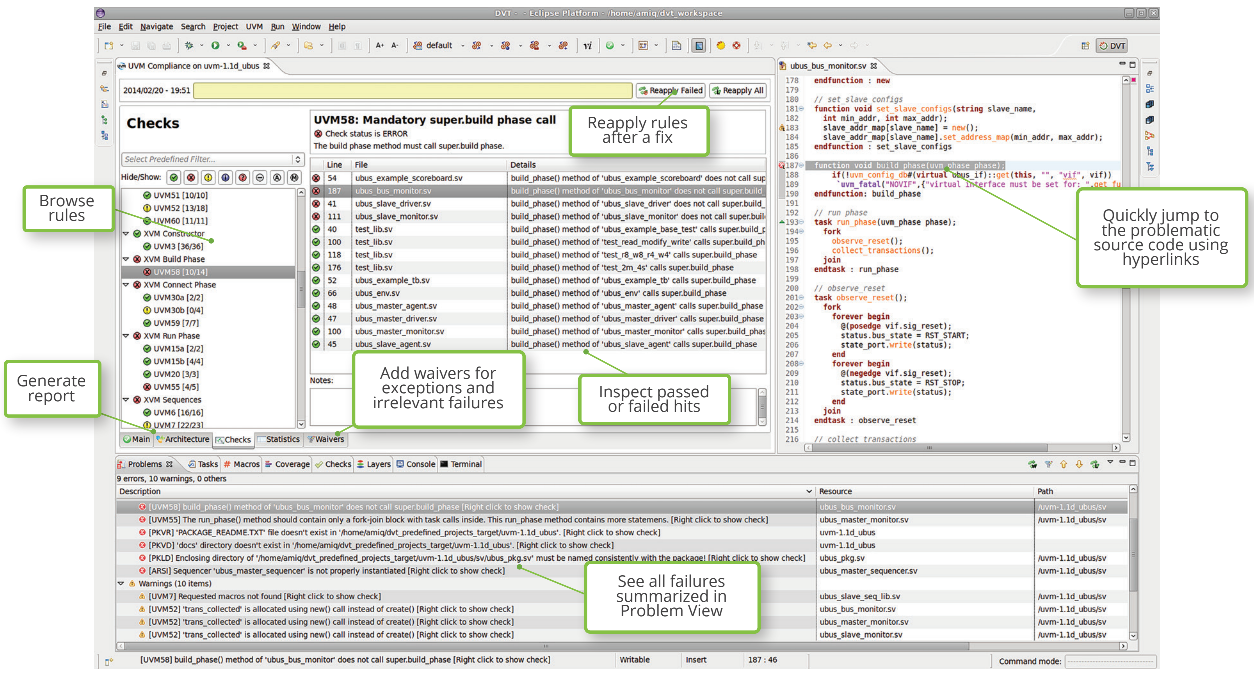
Code templates are parameterized code snippets. Combined with TODO tags, this capability enables your users to easily follow the project development guidelines. Project templates generalize code templates and allow engineers to start a new project from a reference project layout.

Errors inside macros are difficult to investigate. With DVT IDE'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 IDE simplifies maintenance by providing capabilities such as:

- Hyperlinks
- Breadcrumb navigation bar
- Project database queries
- Semantic search for usage, writers, or readers
- Structural views
- Signal tracing
- UML and HDL diagrams
- Comments and macro or parameter values in tooltips
- Semantic source code coloring


Reapply rules after a fix
Browse rules
Quickly jump to the problematic source code using hyperlinks
Generate report
Add waivers for exceptions and irrelevant failures
Inspect passed or failed hits
See all failures summarized in Problem View

DVT - - Eclipse Platform - /home/amiq/dvt_workspace

UVM Compliance on uvm-1.1d_ubus

2014/02/20 - 19:51

Checks

- UVM51 [10/10]
- UVM52 [13/18]
- UVM60 [11/11]
- XVM Constructor
  - UVM3 [36/36]
- XVM Build Phase
  - UVM58 [10/14]
- XVM Connect Phase
  - UVM30a [2/2]
  - UVM30b [0/4]
  - UVM59 [7/7]
- XVM Run Phase
  - UVM15a [2/2]
  - UVM15b [4/4]
  - UVM20 [3/3]
  - UVM55 [4/5]
- XVM Sequences
  - UVM6 [16/16]
  - UVM7 [22/23]

UVM58: Mandatory super.build phase call

Check status is ERROR

The build phase method must call super.build phase.

	Line	File	Details
✗	54	ubus_example_scoreboard.sv	build_phase() method of 'ubus_example_scoreboard' does not call sup
✗	187	ubus_bus_monitor.sv	build_phase() method of 'ubus_bus_monitor' does not call super.build_
✗	41	ubus_slave_driver.sv	build_phase() method of 'ubus_slave_driver' does not call super.build_
✗	111	ubus_slave_monitor.sv	build_phase() method of 'ubus_slave_monitor' does not call super.buil
✓	40	test_lib.sv	build_phase() method of 'ubus_example_base_test' calls super.build_p
✓	100	test_lib.sv	build_phase() method of 'test_read_modify_write' calls super.build_ph
✓	118	test_lib.sv	build_phase() method of 'test_r8_w8_r4_w4' calls super.build_phase
✓	176	test_lib.sv	build_phase() method of 'test_2m_4s' calls super.build_phase
✓	52	ubus_example_tb.sv	build_phase() method of 'ubus_example_tb' calls super.build_phase
✓	66	ubus_env.sv	build_phase() method of 'ubus_env' calls super.build_phase
✓	48	ubus_master_agent.sv	build_phase() method of 'ubus_master_agent' calls super.build_phase
✓	47	ubus_master_driver.sv	build_phase() method of 'ubus_master_driver' calls super.build_phase
✓	100	ubus_master_monitor.sv	build_phase() method of 'ubus_master_monitor' calls super.build_phas
✓	45	ubus_slave_agent.sv	build_phase() method of 'ubus_slave_agent' calls super.build_phase

Notes:

ubus_bus_monitor.sv

```
endfunction : new

// set_slave_configs
function void set_slave_configs(string slave_name,
  int min_addr, int max_addr);
  slave_addr_map[slave_name] = new();
  slave_addr_map[slave_name].set_address_map(min_addr, max_addr);
endfunction : set_slave_configs

function void build_phase(uvm_phase phase);
    if(!uvm_config_db#(virtual ubus_if)::get(this, "", "vif", vif))
     `uvm_fatal("NOVIF",{"virtual interface must be set for: ",get_fu
endfunction: build_phase

// run_phase
task run_phase(uvm_phase phase);
  fork
    observe_reset();
    collect_transactions();
  join
endtask : run_phase

// observe_reset
task observe_reset();
  fork
    forever begin
      @(posedge vif.sig_reset);
      status.bus_state = RST_START;
      state_port.write(status);
    end
    forever begin
      @(negedge vif.sig_reset);
      status.bus_state = RST_STOP;
      state_port.write(status);
    end
  join
endtask : observe_reset

// collect_transactions
```

9 errors, 10 warnings, 0 others

Description	Resource	Path
[UVM58] build_phase() method of 'ubus_bus_monitor' does not call super.build_phase [Right click to show check]	ubus_bus_monitor.sv	/uvm-1.1d_ubus/sv
[UVM55] The run_phase() method should contain only a fork-join block with task calls inside. This run_phase method contains more statemens. [Right click to show check]	ubus_master_monitor.sv	/uvm-1.1d_ubus/sv
[PKVR] 'PACKAGE_README.TXT' file doesn't exist in '/home/amiq/dvt_predefined_projects_target/uvm-1.1d_ubus'. [Right click to show check]	uvm-1.1d_ubus	
[PKVD] 'docs' directory doesn't exist in '/home/amiq/dvt_predefined_projects_target/uvm-1.1d_ubus'. [Right click to show check]	uvm-1.1d_ubus	
[PKLD] Enclosing directory of '/home/amiq/dvt_predefined_projects_target/uvm-1.1d_ubus/sv/ubus_pkg.sv' must be named consistently with the package! [Right click to show check]	ubus_pkg.sv	/uvm-1.1d_ubus/sv
[ARSI] Sequencer 'ubus_master_sequencer' is not properly instantiated [Right click to show check]	ubus_master_sequencer.sv	/uvm-1.1d_ubus/sv
Warnings (10 items)		
[UVM7] Requested macros not found [Right click to show check]	ubus_slave_seq_lib.sv	/uvm-1.1d_ubus/sv
[UVM52] 'trans_collected' is allocated using new() call instead of create() [Right click to show check]	ubus_bus_monitor.sv	/uvm-1.1d_ubus/sv
[UVM52] 'trans_collected' is allocated using new() call instead of create() [Right click to show check]	ubus_master_monitor.sv	/uvm-1.1d_ubus/sv
[UVM52] 'trans_collected' is allocated using new() call instead of create() [Right click to show check]	ubus_slave_monitor.sv	/uvm-1.1d_ubus/sv

[UVM58] build_phase() method of 'ubus_bus_monitor' does not call super.build_phase [Right click to show check]   Writable   Insert   187 : 46

UVM Compliance Checking

These features enable users to navigate easily through complex code, locate the relevant information faster, and understand the source code quickly. They also 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. This practically eliminates the need for using the grep command or memorizing details such as file names and locations. To look up the definition of an element, users can simply hover the mouse over the element name to turn it into a hyperlink. This saves time by jumping directly to the element definition instead of having to search for it.

The editor and diagrams show a breadcrumb navigation bar that clearly indicates the current location in the design hierarchy. It enables engineers to quickly find their way and easily move up and down in the design as needed.

Project database queries allow users to quickly locate specific elements. For example, typing a few letters in a search bar locates 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.". It is also very easy to quickly locate all places where a variable, signal or parameter is written or read, as well as where events are emitted.

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

To help in understanding the project architecture, DVT IDE 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 IDE enables your engineers to inspect a project through diagrams. Designers can use HDL diagrams such as schematic, state machine, and flow diagrams. Verification engineers can use UML diagrams such as inheritance and collaboration diagrams. Bit field diagrams for packed data structures and UVM registers are also automatically generated. Diagrams are hyperlinked and synchronized with the source code and can be saved for documentation purposes. Users can easily search and filter diagrams as needed, for example, visualizing only the clock and reset signals in a schematic diagram.

Semantic source code coloring simplifies reading. For example, inactive pre-processing regions are grayed-out, input ports are visibly distinct from output ports, and local variables and class variables have different colors.

In the design or verification source code, users can include waveform specifications using the popular open-source WaveDrom format which are automatically rendered as waveforms.

PREPROCESSED FILES SUPPORT

DVT IDE provides full IDE capabilities when working with files that contain “preprocessor” statements in other languages such as Perl and Python’s Jinja2 library, or in proprietary languages. Even if the actual code for simulation is generated in subsequent steps from such files, all the features like navigational hyperlinks, autocomplete, on-the-fly error detection, quick fixes, refactoring, etc. work smoothly as if using files written using only a standard language.

VERIFICATION METHODOLOGY SUPPORT

DVT IDE supports the Universal Verification Methodology (UVM). Its powerful UVM oriented features help users learn UVM faster, accelerate adoption, and build UVM verification environments with ease.

Users can easily browse through UVM-based classes such as agents, monitors, and sequences, examine component and sequence trees, visualize architecture diagrams including TLM port connections, browse register maps, search for factory related constructs or config db getters and setters that may influence the testbench behavior, and generate code using UVM specific code templates and wizards.

DVT IDE users can interactively run the Verissimo SystemVerilog Testbench Linter product, including its checks for SystemVerilog and UVM compliance. Verissimo provides hundreds of customizable rules and advanced capabilities for a thorough audit of testbenches.

DVT IDE also features an OVM to UVM migration wizard, which provides advanced automated transition capabilities using refactoring scripts.

LOW POWER SUPPORT

DVT IDE reads UPF or CPF power format files and presents power domains in diagrams, design hierarchies, tooltips, and the breadcrumb navigation bars. Any changes to UPF or CPF files are incrementally analyzed, and power domains updated on the fly. Such capabilities simplify power domain design and debugging.

SIMULATOR INTEGRATION

Users can invoke the simulator and browse its output in a dedicated Console View. This 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 IDE significantly simplifies simulation analysis and debugging.

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

CROSS-LANGUAGE CAPABILITIES FOR MIXED-LANGUAGE PROJECTS

The cross-language capabilities allow users to work with source code written in multiple languages and easily understand the whole design. Features such as hyperlinks, design hierarchy browsing, HDL diagrams, and signal tracing work across SystemVerilog/Verilog, VHDL and C/C++ sources.

For example, a user 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.

PORTABLE STIMULUS SUPPORT

The Portable Stimulus Standard (PSS) allows the creation of higher-level verification tests that are portable from IP to system and from simulation to hardware. A large number of test scenarios can be generated from a single PSS description. DVT IDE provides support for creating and editing PSS code, solves the behavior expressed in the model, and displays examples of valid scenarios. Visualization of detailed scenarios makes it easier for users to determine whether the abstract PSS model is correct.

INCREASING DESIGN AND VERIFICATION PRODUCTIVITY AND QUALITY

Faster and Smarter Code Development

DVT IDE 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 therefore they can focus on code writing and review. Moreover, DVT IDE substantially reduces 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 IDE helps manage your design and verification projects more efficiently. The ability to easily review the source code using features such as 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 IDE 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.

INTEGRATED SOLUTION

DVT Eclipse IDE is closely integrated with the other design and verification products available from AMIQ EDA, including **DVT IDE for VS Code, DVT Debugger, Verissimo SystemVerilog Linter, and Specador Documentation Generator.**

TECHNICAL SUPPORT

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.

CONTACT AMIQ

SUPPORT & EVALUATION
support@amiq.com

SALES
sales@amiq.com

WEBSITES
www.dvteclipse.com / www.amiq.com