



DVT IDE for Eclipse

My First Verilog/SystemVerilog Project

> The DVT Perspective



- Switch to the **DVT Perspective** from

menu Window > Perspective > Open Perspective > Other... > DVT

- The DVT Perspective presents different Views (GUI components) around the Editor



Views

Types

Compile Order

Verification Hierarchy

Design Hierarchy

...

> The Project Location



- You typically create a project in a folder that contains the source code files

*It is not mandatory to create a project where the source files are.
All “outside the project” sources will be presented under DVT Auto-Linked.*

- DVT creates **.dvt** and **.project** in the project location folder

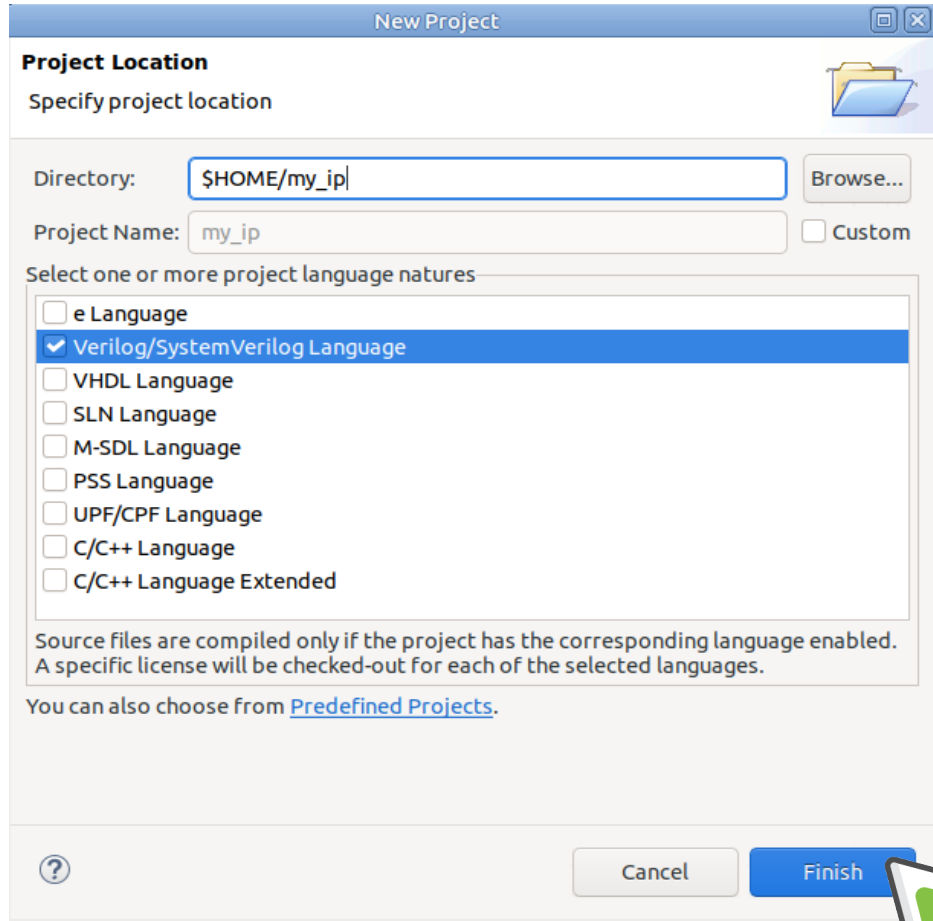
*The **.dvt** folder contains various DVT specific project settings.
The **.project** file indicates to Eclipse that a project exists in the folder.*

my_vip/
sv/
core/
examples/



my_vip/
.dvt
.project
sv/
core/
examples/

> The New Project Wizard



New Project

Project Location
Specify project location

Directory:

Project Name: ☐ Custom

Select one or more project language natures

- ☐ e Language
- ☒ Verilog/SystemVerilog Language
- ☐ VHDL Language
- ☐ SLN Language
- ☐ M-SDL Language
- ☐ PSS Language
- ☐ UPF/CPF Language
- ☐ C/C++ Language
- ☐ C/C++ Language Extended

Source files are compiled only if the project has the corresponding language enabled.
A specific license will be checked-out for each of the selected languages.

You can also choose from [Predefined Projects](#).

- Invoke the wizard

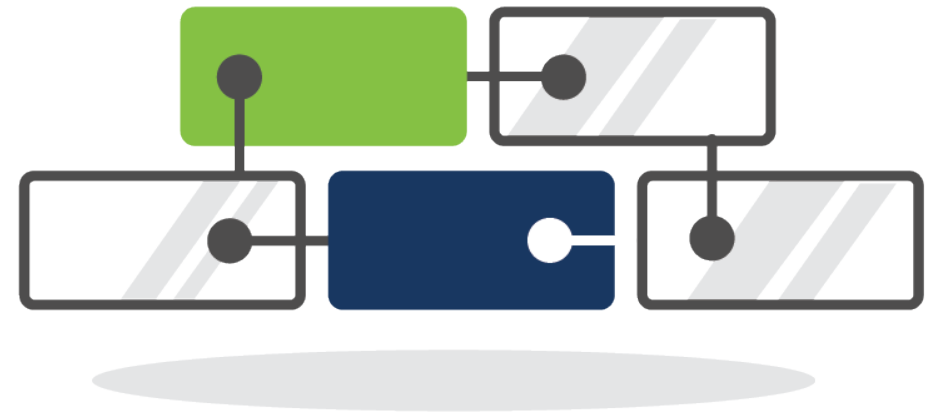
From menu File > New > DVT Project

- Specify the project location
- Specify the project nature. If a project was already configured, that is if .dvt and .project already exist, the wizard recognizes the existing project.

*You can open one of the **Predefined Projects**, if you want to see how an example project is configured.*

> Build Configurations [1]

- In order to provide advanced functionality (like error signaling, hyperlinks, autocomplete, design and UVM components hierarchy, etc.) DVT analyzes the source code files in your project. This analysis process is called **build**.
- In order to build, DVT uses the compilation arguments that you specify in a build file. The default build file is **.dvt/default.build**.
- By default, DVT scans the project folder and automatically detects how to compile the source code files. This is specified by the **+dvt_init_auto** directive used by default in the build file.

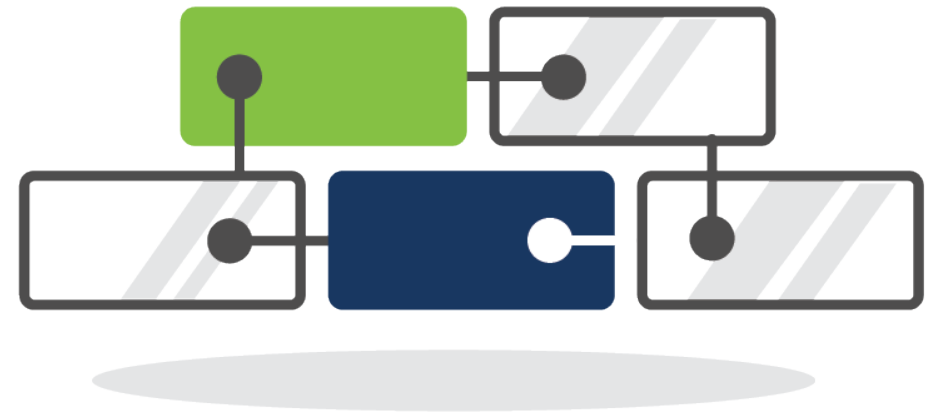


> Build Configurations [2]

- A quick way to set up the build configuration is to start from **a simulation log**. Simply add **+dvt_init_from_simlog+simulation.log** to your build file.

For a robust and scalable flow integration consider reusing the simulator arguments, for example via a dedicated `dvt_ide` Makefile target.

FPGA Support: create a DVT project in the same location as an existing Quartus/Xilinx ISE/Vivado project. All source files and settings defined in the Quartus/Xilinx ISE/Vivado project configuration files will be automatically recognized.



> The .build File Syntax



- In a **.build file** you can specify:

Absolute paths or project root relative paths

System variables like \$var, \${var} or %var%

+incdir+<path> directives to indicate search directories for files included with `include "filename"

+define+<DEFINE>=<replacement> or -define <DEFINE>=<replacement> directives

-v <file> or -y <dir> to specify a Verilog source library file/directory

-f <path> or -F <path> to include a file containing more arguments

- For more options see: <https://eda.amiq.com/documentation/eclipse/sv/toc/build-config/index.html>
- In order to reuse existing argument files that you pass to a simulator, DVT supports several compatibility modes like **vcs**, **ius**, **xcelium** or **questa**

> Build the Project



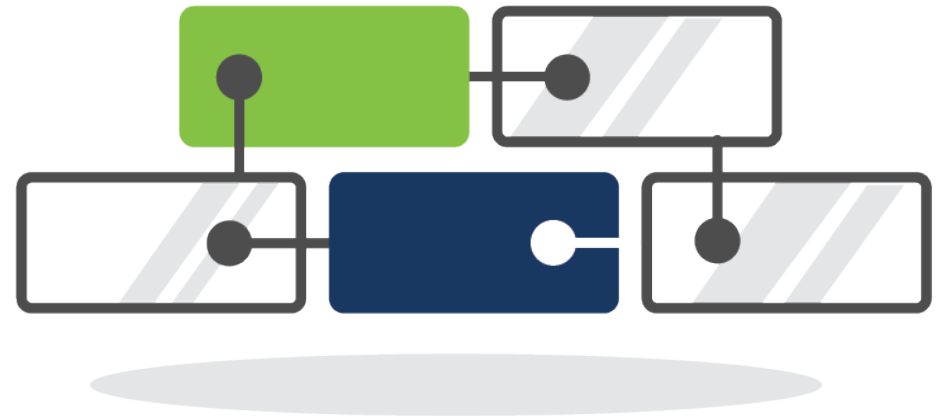
- Building a project means compiling and indexing all the source files in order to provide hyperlinks, autocomplete, class browsing ...

Full Build or Rebuild = compile all using directives from the current build file.

Incremental Build = compile changes; as you edit files, DVT incrementally builds the project.



- **After changing .dvt/*.build, you have to Rebuild the project from the toolbar button**



> Check the Build



- The **Compile Order View** shows all the compiled files

*You can open the Compile Order View from menu Window > Show View > Compile Order.
It is typically located on the right side of the Editor, next to the Outline View.*

- The **Problems View** shows all the errors and warnings in your project
- It is recommended to walk through the errors in the following order:

Build Config Errors: file not found, incdir not found, included file not found ...

Syntax Errors: unexpected token “vitual” instead of “virtual” ...

Semantic Errors: non existing port, wrong number of arguments when calling a function ...

You can open the Problems View from menu Window > Show View > Problems.

It is typically located below the Editor.

> Features Overview [1]



- **Hyperlinks:** in the editor `press Ctrl and hover` class names, method names, and in general any identifier. A hyperlink appears. Click it to go to the definition. In addition to this hyperlink, a list presents more options, for example Show Usages.
- **Show Usages/Readers/Writers:** in the editor `press Ctrl and hover` an identifier. From the hyperlinks list chose Show Usages/Readers/Writers to see all places where a variable, signal, function, class, macro etc. is used/read/written.
- **Autocomplete:** in the editor `Ctrl + Space` triggers autocomplete. For example driver. `<Ctrl+Space here>` will show driver API.
- **Quick Fixes:** in the editor on a line with errors `Ctrl +1` pops-up quick fix proposals to correct typos, to declare missing variables etc.
- **Rename Refactoring:** place the cursor over an identifier and `right click > Refactor > Rename` or `Shift+Alt+R` to rename and update all usages across the entire project.

> Features Overview [2]



- **Type Hierarchy:** place the cursor over a class name and press F4 to see the OOP inheritance. Ctrl + T is a quick way to navigate the OOP hierarchy – it works for classes and functions/tasks
- **Design Hierarchy:** place the cursor over the module name and press Shift + F4 to see the design structure
- **Verification Hierarchy:** place the cursor over an UVM test class and press Shift + F6 to see the verification environment topology
- **All Classes/Modules/Interfaces/...:** menu Window > Show View > Types
- **All Macros:** menu Window > Show View > Macros
- **All TODOs/FIXMEs:** menu Window > Show View > Tasks
- **Code Templates:** menu Window > Show View > Code Templates
- **To quickly find a class, module, macro or compiled file:** Ctrl + Alt + Q
- **To quickly open a file:** Ctrl + Shift + R

> Features Overview [3]



- **Diagrams:**
 - right click on a module / class / variable to get schematics / UML / state machine diagrams
 - other diagrams available from dedicated contexts: UVM Components / Bitfield for UVM regs & packed data types / Wavedrom
- **Semantic Search:** Ctrl + H to search for a class, method... both definitions and/or where it is used
- **Code Formatting:** press Ctrl+Shift+F to format the whole editor or selection
- **Toggle Comment:** Ctrl + / for current line or selection
- **Matching Begin - End:** double-click on the begin, end, function, endfunction ...
- **Column Selection:** Shift+Ctrl+A or from the main toolbar button
- **All Shortcuts:** press Ctrl + Shift + L to pop-up a list of all shortcuts



And many more, please contact support@amiq.com for a demo.

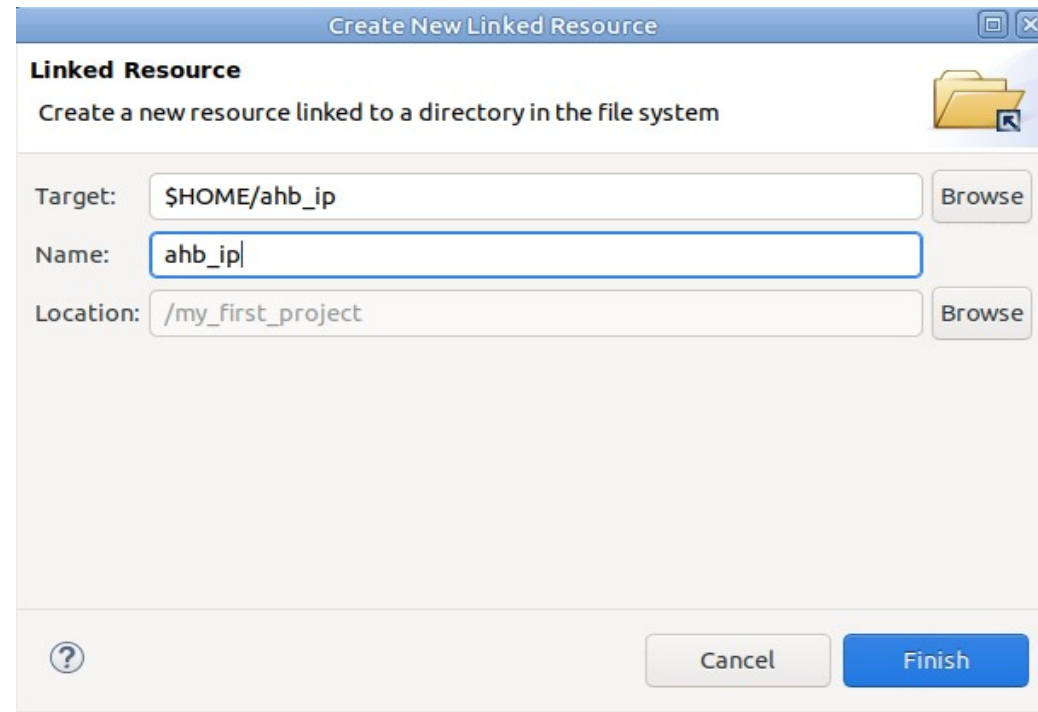
> Advanced: Linked Resources [1]



- Sometimes your source code is spread on the disk, not everything is under one “source code root” folder
- Files outside the Project Location folder (“external files”) are by default automatically brought during a full build under the **DVT Auto-Linked virtual folder**
- What you see there are the actual files, any change will be visible on the disk and the other way around
- The hierarchy under DVT Auto-Linked may be too deep. To bring the files “closer” you can:
 - Use **+dvt_auto+link_root+alias=/path/to/folder** directives in **default.build**.
 - Use **Linked Resources**. Linked Resources are logical entities, no additional files are created on your disk. Their definition is stored in the **.project** file.

> Advanced: Linked Resources [2]

Right click on Project > **New** > **DVT Linked Resource**, specify the target location and linked resource name.



The dialog box is titled "Create New Linked Resource". It contains the following fields and buttons:

- Linked Resource**: Create a new resource linked to a directory in the file system (with a folder icon and a link icon).
- Target:** Text field containing "\$HOME/ahb_ip" and a "Browse" button.
- Name:** Text field containing "ahb_ip".
- Location:** Text field containing "/my_first_project" and a "Browse" button.
- Buttons at the bottom: "?", "Cancel", and "Finish".

Rebuild the project in order to avoid duplicate files in the newly created folder and DVT Auto-Linked.

> More Information



- Demo Movies: <https://eda.amiq.com/tutorials>
- Cheatsheet for commonly used keyboard shortcuts:
https://eda.amiq.com/cheatsheets/DVT_Eclipse_IDE_Keyboard_Shortcuts.pdf
- Step by step basic tutorial:
https://eda.amiq.com/getting-started/My_First_SystemVerilog_Project_with_the_DVT_Eclipse_IDE.pdf
Please contact us for more training materials
- Features with snapshots:
<https://eda.amiq.com/documentation/eclipse/sv/toc/tips-and-tricks/index.html>
- User Guide: <https://eda.amiq.com/documentation/eclipse/sv/index.html>
- Datasheet: https://eda.amiq.com/datasheets/amiq_dvt_ide_datasheet.pdf



Mail to support@amiq.com