HDL Designer 2007.1 SystemVerilog Tutorial

For use with AVM3.0 & Questa 6.3

Mentor Graphics Company

Table of Contents

1 Introduction

This document provides a self-paced tutorial with step-by-step procedures to import a simple mixed language design and use the new SystemVerilog language features in the Design Manager and DesignPad text editor in order to rapidly understand and explore a class-based testbench. The testbench uses the Questa Advanced Verification Methodology (AVM) and supports incremental compilation and launching of the Questa simulator. Further details on the AVM can be found in the Advanced Verification Methodology Cookbook Version 3.0 which can be downloaded along with examples from http://www.mentor.com/go/cookbook

A number of other tutorials can be accessed from links in the HDL Designer Series InfoHub which is opened by choosing Help and Manuals from the **Help** menu in all application windows.

All user commands in the tutorial procedures are referenced using their menu path (shown in bold text). However, many commands can also be accessed using the toolbar, shortcut bar or keyboard shortcuts. Refer to Shortcut Keys in the **Quick Reference Index** which can be accessed from the HDL Designer Series InfoHub opened through the **Help** menu.

The tutorial is divided into sections containing numbered procedures. Unnumbered paragraphs provide additional information or give advice on other ways to perform an operation. The procedure numbers are continuous through a section and restart for each new section.

The use of the term **RMB** in this document represents pressing the Right Mouse Button.

1.1 Overview of the tinycache design

**** This example is designed to be used with AVM version 3.0 ****
**** together with Questa version 6.3 ****

The “TinyCache” project is a 16 word, 8-bit, write-through cache that sits between a CPU and its memory. All writes to the memory and all reads from the memory go through the cache. If the contents of a requested memory address are not currently stored in the cache (a cache miss), the CPU must wait until the cache has loaded the word from the memory before it can receive the data.

The cache uses the 4 least significant bits (LSB) of the address to access the first 16 words of a 255 word RAM.
The cache contains 3 RAMs

- Cache RAM
  An 255x8 bit RAM which stores the cached data.
- Key RAM
  An 255x16 bit RAM which stores the addresses related to the cached data.
- Invalid
  A 255-bit register which contains the status for each entry. All entries are made invalid on reset.

The design uses two protocols. The Tinycache…

- is a master for the memory bus…
- …and a Slave for the CPU bus

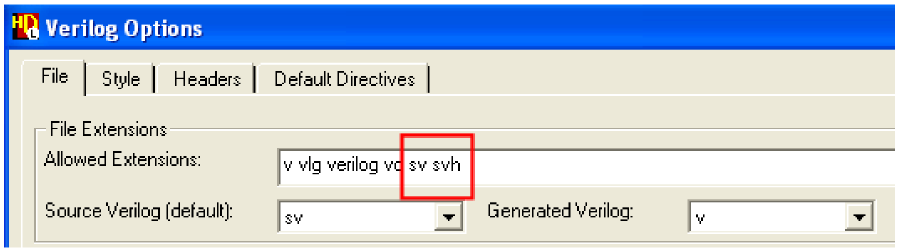
Each transaction is clearly defined and there are a number of different interactions between these transactions.

Although this is a very simple design, the approach would be similar for a complex ASIC or FPGA.

More information on the design can be found in the Functional Spec which you will be associating with the design later on in the tutorial.

2 Setup

1. **Questa 6.3**
   If you have not already installed Questa 6.3, please install it now.
2. **Invoke HDL Designer**
3. **Preference migration**
   - If you have used a previous 2007.1 beta build, your preferences will not be migrated. It is therefore recommended that you rename the HDL Designer Preferences directory to ensure you are using the most up-to-date preference settings.
     - On Windows the preferences directory is:
       C:\Documents and Settings\<user_name>\Application Data\HDL Designer Series
     - On UNIX/Linux the preferences directory is:
       <home_directory>/hdl_designer_series
   - If you have used earlier released versions of HDL Designer, a message will appear in the log window to inform you that your preferences are being migrated to a new 2007.1 set. This will not affect your existing preferences, instead a copy of your current preferences directory will be made and modified to incorporate the new 2007.1 options.
4. **General Settings**
   - If this is the first time HDL Designer has been invoked on your machine, the tool will automatically bring up the Setup Assistant.
     Otherwise please invoke it directly using **Help> Setup Assistant**
   - Step through the wizard. On the ***simulator page*** set **Questa 6.3** as your simulator. On the ***Project page*** choose **Examples**. You will be creating a new project later on.
5. **Text Editor Settings**
   - In order to review the SystemVerilog features available within the HDL Designer DesignPad text editor, please ensure DesignPad is set as the default text editor.
   - To do this, choose **Options> Main** and go to the “Text” tab.
   - Use the drop-down to select DesignPad as the Editor (and optionally also as the HDL Viewer).
6. **File Extension Settings**
   - This design uses the “.sv” extension for SystemVerilog source files and the “.svh” file extension for included class files.
   - To check that these extensions have been added to the allowed list, choose **Options> Verilog** and review the list of allowed File Extensions on the “File” tab. If the list does not include “.sv” or “.svh”, please type them in it at the end of the list.



Allowed Extensions: v vlg verilog vo sv svh
Source Verilog (default): sv
Generated Verilog: v

7. Switching back to pre-2007.1 language parsing
   - If you wish to switch back to the language parsing used in previous releases, you can do this by setting the environment variable HDS_2006_VLOG_PARSER to any value (eg 1 or TRUE).
   - Note, however, that this will restrict SystemVerilog parsing to the 3.0 Accellera standard subset which will not support the IEEE Std 1800™-2005 verification constructs used in this tutorial.

3 Projects

HDL Designer interacts either directly with your existing design files or on a copy of specified design files. You can direct HDS to analyze a design simply by specifying a directory underneath which the HDL files are located.

A library can be treated like an environment variable which maps to the directory containing the HDL files for a given design. The files can reside in any number of subdirectories and have any names.

You can associate descriptions and other data with the design by specifying an HDS mapping for the logical library name. A typical logical library name would have an “HDL” mapping to the HDL directory and an “HDS” mapping to the associated data directory. Additional mappings can be specified for directories in which downstream tool data such as a compiled simulation database or the output of synthesis should be stored. The tool will create appropriate default mappings where necessary.

When using repository-based Version Management tools, libraries will also have corresponding mappings for the HDL and HDS repository locations.

For VHDL designs, library directories correspond to VHDL library names. For Verilog designs, they represent a convenient way to refer to the design directories.

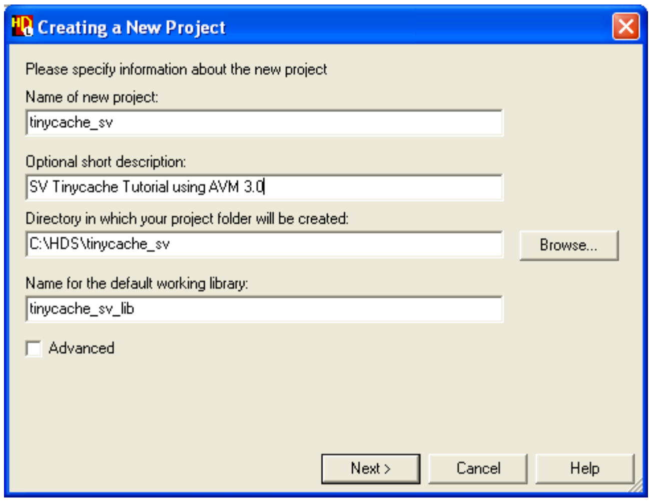
Many designs may just use a single library, although typically there would be a small number of IP directories with corresponding mappings to allow shared components to be reused without the need for copying. Complex designs may be spread across many libraries.

The mapping of logical library names to their actual location on the file system is stored in the HDS project file (*.hdp). The same libraries can be used in different projects, eg for directories of shared IP or reusable components.

3.1 Creating a new project

First you will need to create a new project in which to put the tinycache design and reference the pre-compiled AVM library.

1. Use the pulldown menu **File> New> Project…**



Please specify information about the new project

Name of new project: tinycache_sv

Optional short description: SV Tinycache Tutorial using AVM 3.0

Directory in which your project folder will be created: C:\HDS\tinycache_sv

Name for the default working library: tinycache_sv_lib

☐ Advanced

2. Enter **tinycache_sv** as the name of the project.
3. Specify a convenient location for the project folder to be created.
4. Click on **Next** to view the summary page.
5. Click on **Next** again and choose the option ***Add existing design files***
6. Click **Finish** to compete the process.
7. You will now add the tinycache files to the project…

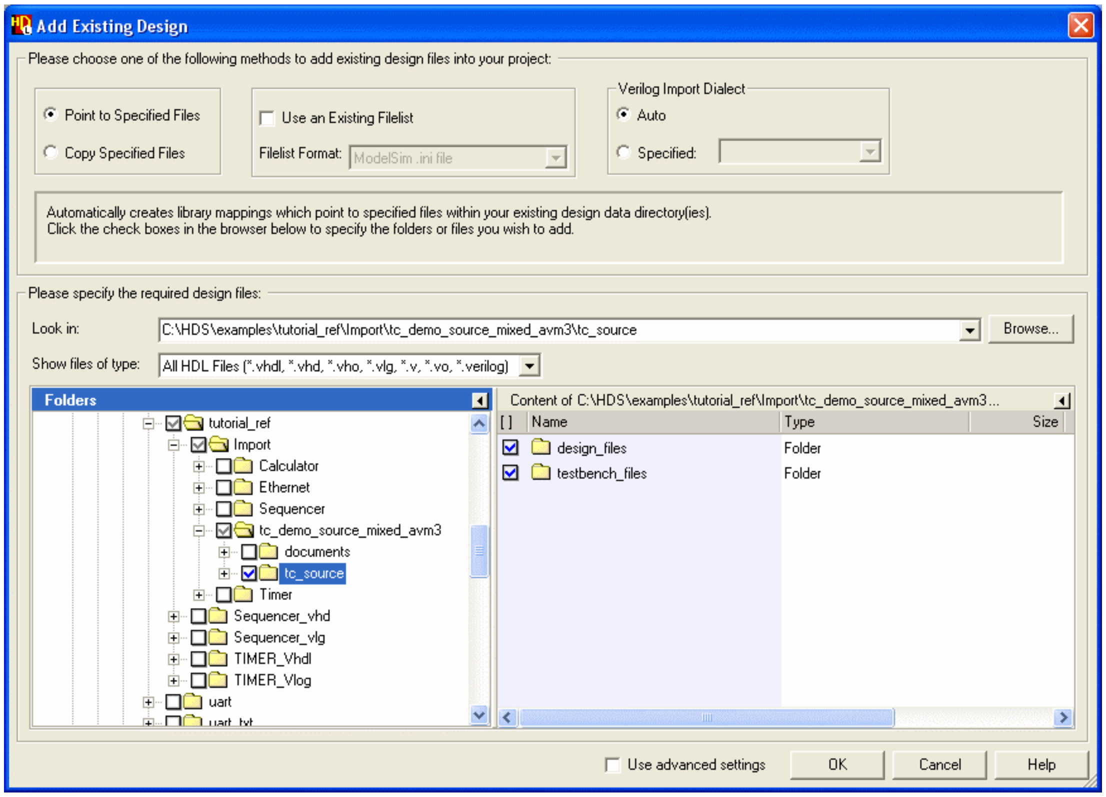
3.2 Add the tinycache files to the project

We will next point to the Tinycache design and testbench.

1. In the resulting dialog box, you can either reference (point to) or copy the existing design. Choose: ***Point to Specified Files***.
2. Set the ***Show files of type*** filter to ***All HDL Files***
3. Using the File browser, navigate to the folder containing the tinycache source files, for example:

   C:\HDS\examples\tutorial_ref\Import\tc_demo_source_mixed_avm3\tc_source

   Check the tc_source folder in the Folders pane.
   The dialog should look like this (again note the check marks and the content of the right-hand pane in the browser):



Please choose one of the following methods to add existing design files into your project:

Point to Specified Files: selected

Use an Existing Filelist: not checked; Filelist Format: ModelSim .ini file

Verilog Import Dialect: Auto

Automatically creates library mappings which point to specified files within your existing design data directory(ies).
Click the check boxes in the browser below to specify the folders or files you wish to add.

Please specify the required design files:

Look in: C:\HDS\examples\tutorial_ref\Import\tc_demo_source_mixed_avm3\tc_source

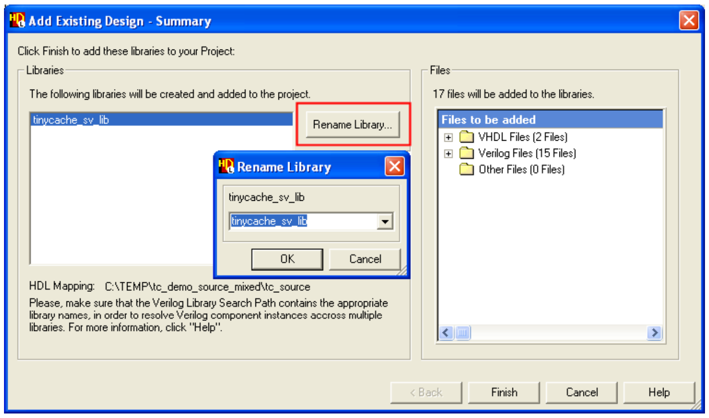
Show files of type: All HDL Files (*.vhdl, *.vhd, *.vho, *.vlg, *.v, *.vo, *.verilog)

Content of C:\HDS\examples\tutorial_ref\Import\tc_demo_source_mixed_avm3...

[]	Name	Type	Size
✓	design_files	Folder	
✓	testbench_files	Folder	

Use advanced settings: not checked

4. Click **OK.**
5. We are going to add the tinycache files to the default library, so in the ***Summary*** dialog, click on **Rename Library…**
6. In the ***Rename Library*** dialog choose **tinycache_sv_lib** from the drop-down and **OK** the dialog.



Click Finish to add these libraries to your Project:

The following libraries will be created and added to the project.

tinycache_sv_lib

17 files will be added to the libraries.

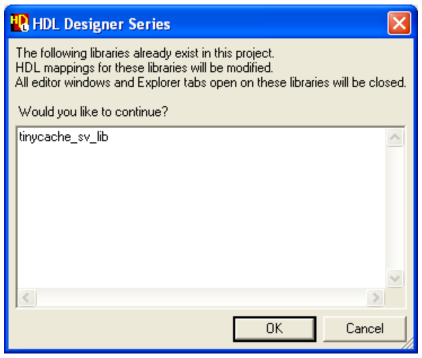
Files to be added
- VHDL Files (2 Files)
- Verilog Files (15 Files)
- Other Files (0 Files)

Rename Library: tinycache_sv_lib

HDL Mapping: C:\TEMP\tc_demo_source_mixed\tc_source

Please, make sure that the Verilog Library Search Path contains the appropriate library names, in order to resolve Verilog component instances accross multiple libraries. For more information, click "Help".

7. Then click **Finish** to add the files to the tinycache_sv_lib library within the project.
8. The following dialog maybe displayed. If so simply click **OK**.



HDL Designer Series

The following libraries already exist in this project.
HDL mappings for these libraries will be modified.
All editor windows and Explorer tabs open on these libraries will be closed.

Would you like to continue?

tinycache_sv_lib

9. The design is added to the project and the top of the design is automatically detected.
10. Note the following:
    - The tool can parse any specified HDL files irrespective of how they are named or the directory structure underneath the specified library folder.
    - Any critical syntax errors which prevent the tool from extracting the overall design structure and dependencies would be flagged, as would be any missing or unbound instances.
    - The tool also automatically detects the language and dialect of the files, allowing mixed-language designs to be added to the project in one pass.
    - In the “Design Unit Browser” you can see the cache_env_pkg SystemVerilog package file. The .svh files included by the package are all detected and shown as include files within the “Includes” virtual folder.
    - All Classes have also been identified and are shown grouped under the “Classes” virtual folder.
    - The “Files Browser” shows the actual files and folders as they appear in the file system. In this case, since we used the option “Point to Specified Files” these are the original files still in their original location.

3.3 Side Data – add functional spec & test plan

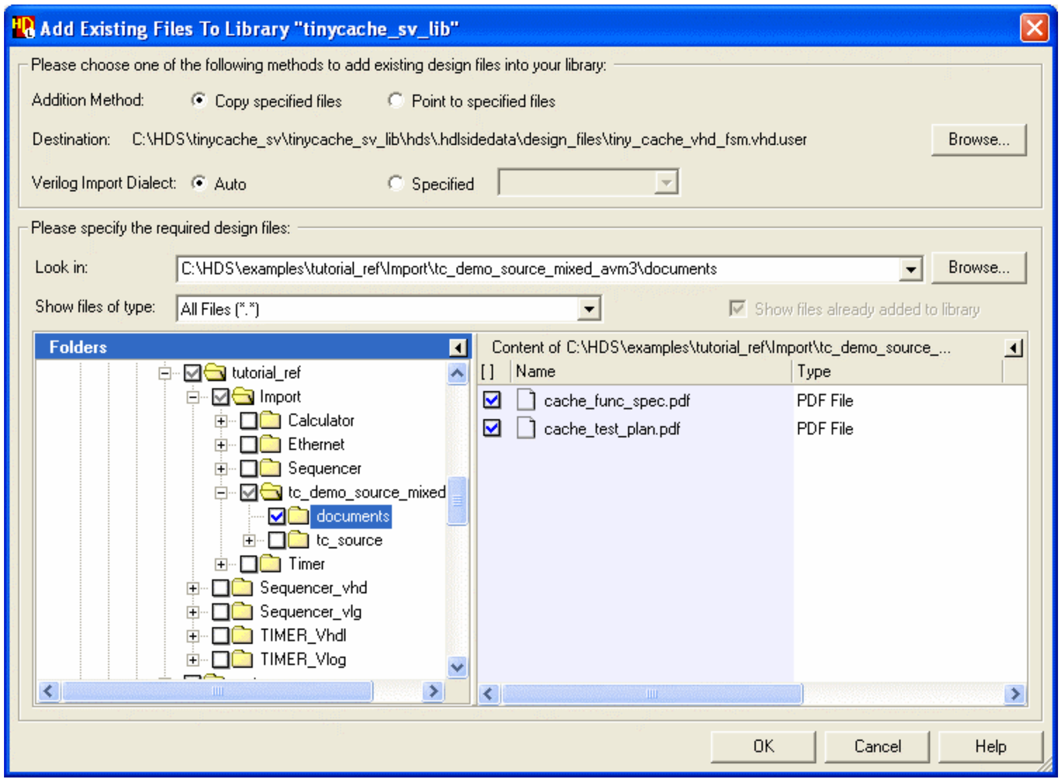
When developing, understanding or verifying a design, it is important to be able to quickly refer to the design documentation and test plan.

Just as you were able to reference or import the design and testbench, you can also create, import or reference any other files and associate them with objects in the design. The following steps show how to associate the Functional Spec and Test Plan with the top-level of the tinycache. Alternatively the documents can be placed in a sub-directory within the files browser.

1. Click on the **Sub Browser** tab which is located down the right hand edge of the tool. This shows you the *Side Data* and *Downstream* browser panes if they are not already visible.
2. In the ***tinycache_sv_lib*** library, select the ***tiny_cache_vhd*** design unit by clicking on it.
3. In the Side Data browser, RMB on ***User Data*** & choose **Add> Existing Files…**
4. In the resulting dialog box, choose **Copy Specified Files**.
5. Set the ***Show files of type*** filter to ***All Files***
6. Using the File browser, navigate to the ***documents*** folder, for example:

   C:\HDS\examples\tutorial_ref\Import\tc_demo_source_mixed_avm3\documents

   Check the ***documents*** Folder inside ***tc_demo_source_mixed_avm3***
   The dialog should look like this (again note the check marks and the content of the right-hand pane in the browser):



Please choose one of the following methods to add existing design files into your library:

Addition Method: [✓ Copy specified files] Point to specified files

Destination: C:\HDS\tinycache_sv\tinycache_sv_lib\hds\.hdlsidedata\design_files\tiny_cache_vhd_fsm.vhd.user

Verilog Import Dialect: [✓ Auto] Specified

Please specify the required design files:

Look in: C:\HDS\examples\tutorial_ref\Import\tc_demo_source_mixed_avm3\documents

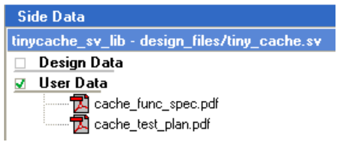
Show files of type: All Files (*.*)

Show files already added to library

Content of C:\HDS\examples\tutorial_ref\Import\tc_demo_source_...

[ ]	Name	Type
✓	cache_func_spec.pdf	PDF File
✓	cache_test_plan.pdf	PDF File

7. **OK** the dialog.
8. You can now open the documents at any time by selecting the “tiny_cache_vhd” design unit, expanding the “User Data” folder and double-clicking on the document within the Side Data browser.

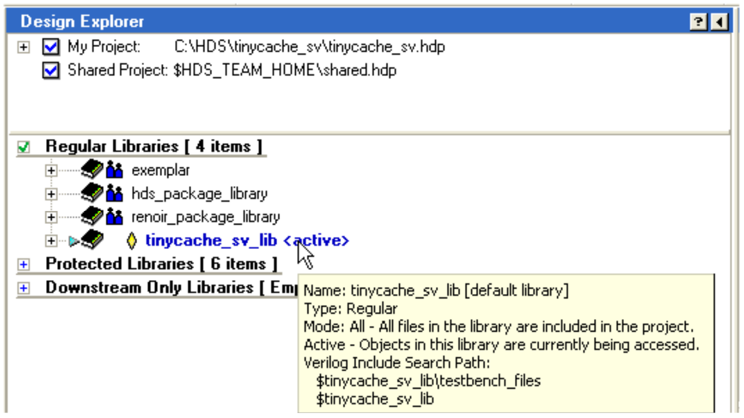


3.4 Check the Verilog `include search path for the library

As we have seen, the library contains a number of include files in subfolders. In order for these include files to be found, the appropriate search path must be defined.

1. To check the Verilog Include search path for the ***tinycache_sv_lib***, click on the ***Project*** tab at the bottom left of the tool.

2. Ensure the ***Regular Libraries*** group is expanded.
3. Hover the cursor over the ***tinycache_sv_lib*** and check the entries for “Verilog Include Search Path”


Design Explorer
Regular Libraries [4 items]
exemplar
hds_package_library
renoir_package_library
tinycache_sv_lib <active>
Protected Libraries [6 items]
Downstream Only Libraries [Em

My Project: C:\HDS\tinycache_sv\tinycache_sv.hdp
Shared Project: $HDS_TEAM_HOME\shared.hdp

```
Name: tinycache_sv_lib [default library]
Type: Regular
Mode: All - All files in the library are included in the project.
Active - Objects in this library are currently being accessed.
Verilog Include Search Path:
  $tinycache_sv_lib\testbench_files
  $tinycache_sv_lib
```

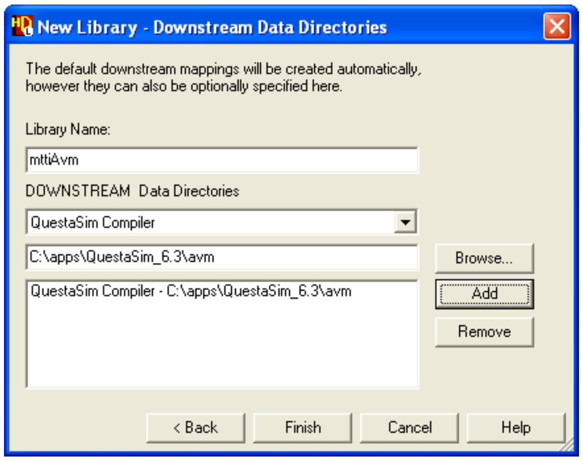
4. Ensure the list contains **$tinycache_sv_lib \testbench_files** and **$tinycache_sv_lib**
   Note that the "$" in front of a library name acts as a environment variable which refers to the HDL source mapping directory for the library.
5. If not, you can edit the search path using RMB over the ***tinycache_sv_lib*** library and choosing **Edit Verilog Include Search Path**

3.5 Add the pre-compiled mtiAvm library reference to the project

The AVM library is available pre-compiled in the Questa installation. Since the AVM library will potentially be used by a number of libraries in the project, we need to add it to the project library search path so all other libraries in the project can find the compilation units within it (eg the package).

In a real development environment, the AVM library reference would typically be added to a shared project so that it can be accessed by many different projects across many different design teams.

1. Staying in the Project Tab, use File> New> Library…
2. Click library type **Downstream Only** then click **Next>.**
3. Enter library name **mtiAvm**
4. From the drop-down list choose **QuestaSim Compiler** and browse to the pre-compiled AVM library in your Questa installation eg C:\apps\QuestaSim_6.3\avm
5. Click the **Add** button then click **Finish** in the wizard.


New Library - Downstream Data Directories

The default downstream mappings will be created automatically, however they can also be optionally specified here.

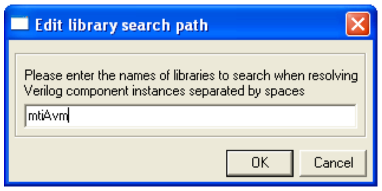
Library Name: mttiAvm

DOWNSTREAM Data Directories: QuestaSim Compiler

C:\apps\QuestaSim_6.3\avm

QuestaSim Compiler - C:\apps\QuestaSim_6.3\avm

6. Now add the library to the search path.
   RMB over ***My Project*** and choose **Edit Verilog Library Search Path**
7. Enter **mtiAvm** and **OK** the dialog.
   (Note there is no "$" – the Library Search Path contains logical library names rather than directory names)


Edit library search path

Please enter the names of libraries to search when resolving Verilog component instances separated by spaces

mtiAvm

Note that, if you wish to analyze the AVM source, you could add it to the project in the same way you added the tinycache design. See Appendix A – Using the AVM source library on page 21 .

4 Design Browsers

The Design Manager provides 3 main views of the HDL design data:

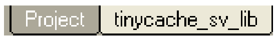
- Design Unit Browser
  This is a “virtual” representation of the main declarative objects within the design.
- Design Files Browser
  This shows the actual file structure of the files and folders which are part of the project, HDL source files can also be expanded to see the top-level objects within them.
- Design Hierarchy Browser
  This shows the static instance hierarchy from a given level. Modules, classes and Program Blocks can be dragged into this browser (or the menu item Edit> Show Hierarchy used) in order to explore the design hierarchy underneath that level.
- Object List
  The Design Unit browser can be toggled to an object list where you can group, sort and filter all objects (including instances)

You can control which objects are visible, which columns of additional data are shown (including user-defined property information) and save these settings in alternative “viewpoints”.

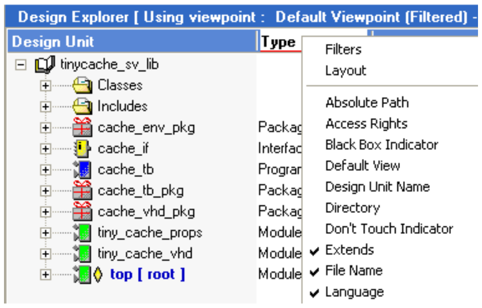
4.1 Columns

For this tutorial, we will modify the column information shown within each browser:

1. **Click** on the ***tinycache_sv_lib*** tab at the bottom left of the tool.



2. **Click** on the button in the top-right corner of the Sub Browser pane to collapse it. This gives more space for the other browser panes.
3. In the Design Unit browser, **RMB** over the ***header row***.

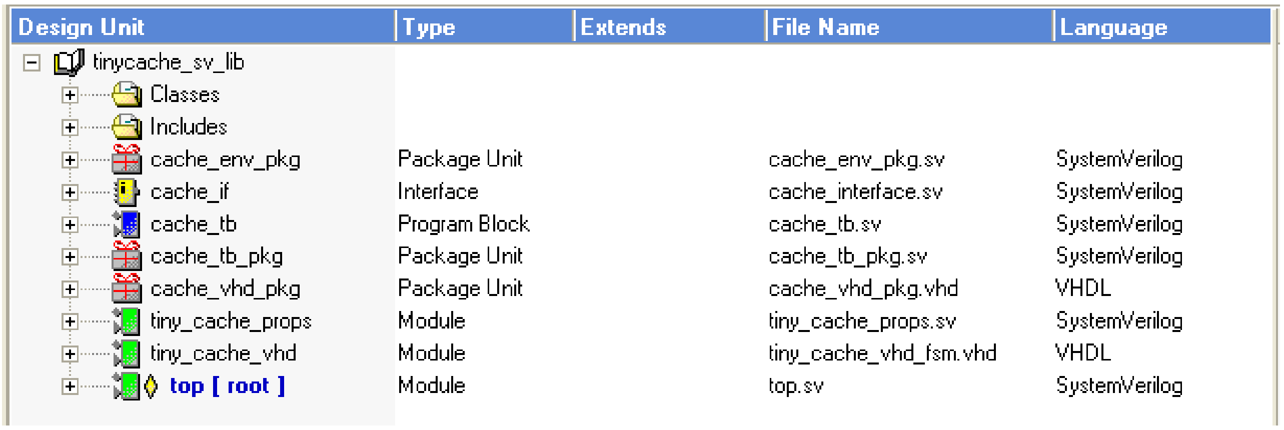


4. In the drop-down list, ensure the following fields have check marks
   - **Type, Extends, File Name , Language**

Note: Instead of changing each one individually, you can choose the option **more…** at the end of the list and change the settings in the Viewpoint Manager.

5. Similarly, in the File browser ensure the following fields have check marks
   - **Type, Extends, Language**
6. Finally, in the Hierarchy browser ensure the following fields have check marks:
   - **Design Unit Name, Extends, File Name, Library**
7. You can click-drag a column header to move the column or drag the dividers to change the size of the column.

4.2 Design Unit Browser Objects



Design Unit	Type	Extends	File Name	Language
tinycache_sv_lib				
Classes				
Includes				
cache_env_pkg	Package Unit		cache_env_pkg.sv	SystemVerilog
cache_if	Interface		cache_interface.sv	SystemVerilog
cache_tb	Program Block		cache_tb.sv	SystemVerilog
cache_tb_pkg	Package Unit		cache_tb_pkg.sv	SystemVerilog
cache_vhd_pkg	Package Unit		cache_vhd_pkg.vhd	VHDL
tiny_cache_props	Module		tiny_cache_props.sv	SystemVerilog
tiny_cache_vhd	Module		tiny_cache_vhd_fsm.vhd	VHDL
top [root]	Module		top.sv	SystemVerilog

The following objects can be seen in the tinycache_sv_lib library:

- VHDL and Verilog Modules
- SV Program Block
- SV Interface
- SV and VHDL Packages
- Classes virtual folder
- Includes virtual folder

1. **Expand** each object in turn.
2. **Double-click** on the package cache_env_pkg
   The package is opened in the DesignPad text editor. You will look at the editor in more detail later on.

This package `includes all the tinycache testbench classes.

3. **Drag select** the first included file **trans.svh** (take care not to select the quotes) and use the menu item **File> Open Selected**.

This opens the transaction class file in DesignPad. This file contains the classes for the main tinycache transactions.

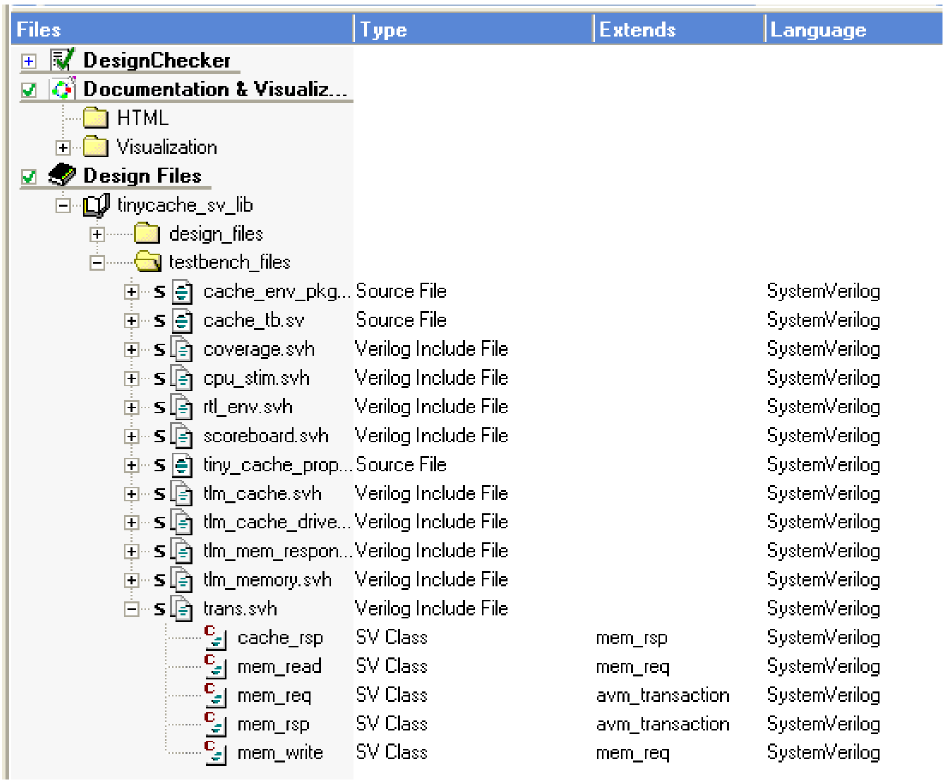
4. Close **trans.svh** using **File> Close**.
5. The file **cache_env_pkg** should still be open. **Double-click** the second package name **cache_tb_pkg** and use the menu item **File> Open package**.

This opens the cache_tb_pkg file in DesignPad which contains the parameters and user-defined types.

6. **Close** DesignPad using **File> Exit**.

4.3 File Browser

1. You can also expand the files themselves to see the top-level objects contained within them.
   For example, **expand** folder ***testbench_files*** and then expand file ***trans.svh*** to see the transaction classes defined within it which you saw earlier.



Files	Type	Extends	Language
DesignChecker			
Documentation & Visualiz...			
HTML			
Visualization			
Design Files			
tinycache_sv_lib			
design_files			
testbench_files			
cache_env_pkg...	Source File		SystemVerilog
cache_tb.sv	Source File		SystemVerilog
coverage.svh	Verilog Include File		SystemVerilog
cpu_stim.svh	Verilog Include File		SystemVerilog
rtl_env.svh	Verilog Include File		SystemVerilog
scoreboard.svh	Verilog Include File		SystemVerilog
tiny_cache_prop...	Source File		SystemVerilog
tlm_cache.svh	Verilog Include File		SystemVerilog
tlm_cache_drive...	Verilog Include File		SystemVerilog
tlm_mem_respon...	Verilog Include File		SystemVerilog
tlm_memory.svh	Verilog Include File		SystemVerilog
trans.svh	Verilog Include File		SystemVerilog
cache_rsp	SV Class	mem_rsp	SystemVerilog
mem_read	SV Class	mem_req	SystemVerilog
mem_req	SV Class	avm_transaction	SystemVerilog
mem_rsp	SV Class	avm_transaction	SystemVerilog
mem_write	SV Class	mem_req	SystemVerilog

4.4 Hierarchy Browser

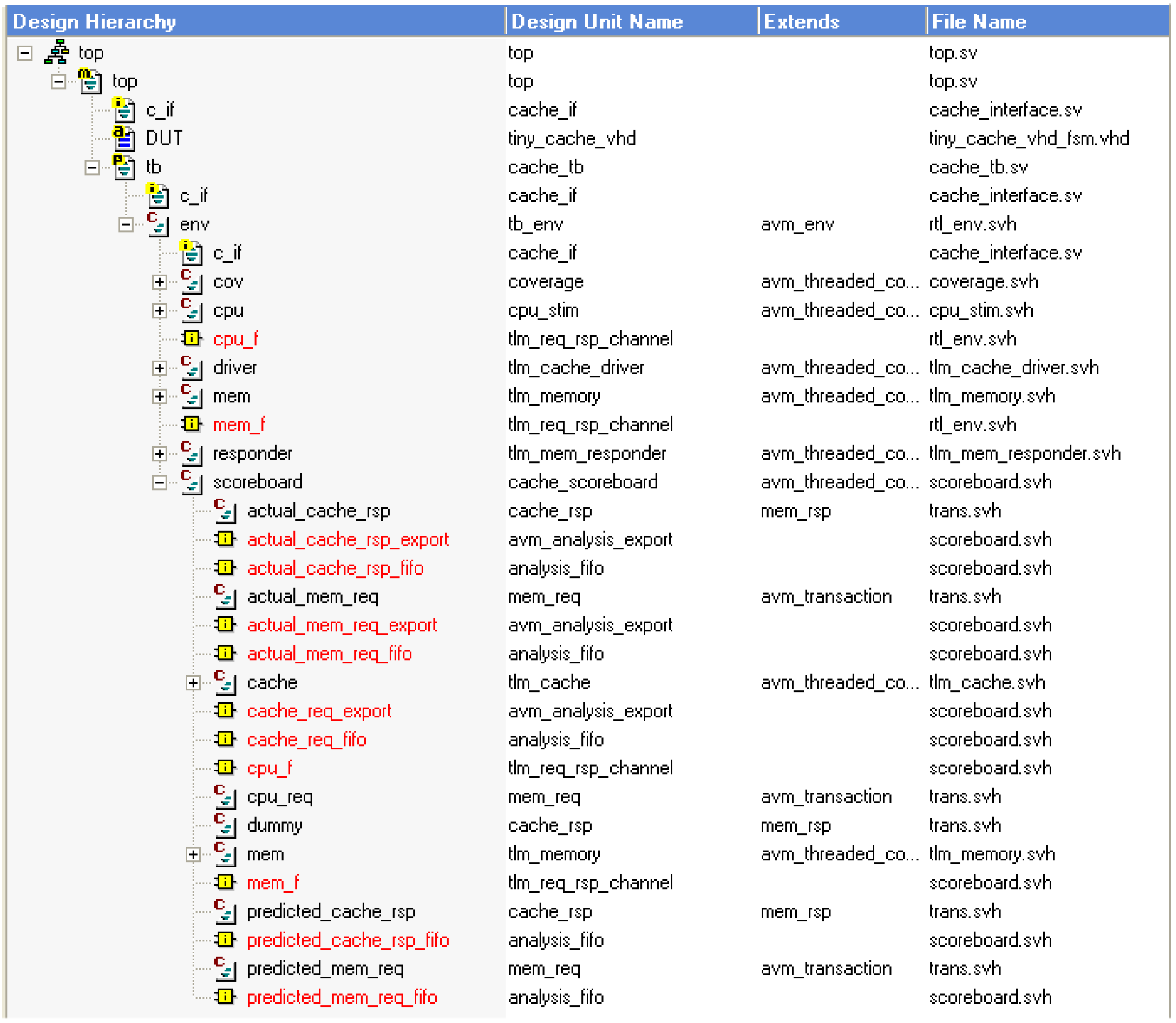
1. If not already shown, **RMB** over the Design Unit ***top*** and choose **Show Hierarchy**.
   This shows you the static instance hierarchy of the design.
2. **Expand *top*** to see the following:
   - The tiny_cache_vhd VHDL RTL device under test (DUT)
   - The cache_tb Program block (tb) which contains the testbench
   - The cache_if interface which connects the DUT to the testbench program block (c_if)
3. Now **expand** the Program Block ***tb*** to see the top-level verification environment class (env).
4. **Expand** the Class ***env*** which will show the top-level instances within the testbench.

You can also expand into the other hierarchical blocks - the ***coverage*** block and the ***scoreboard***

Since we are using the pre-compiled AVM library, objects within the AVM library are not displayed (eg cpu_f, mem_f)

5. If necessary, use the **F5** key to refresh the browser contents.

Note that objects from the AVM library are shown in red since the source is not available.



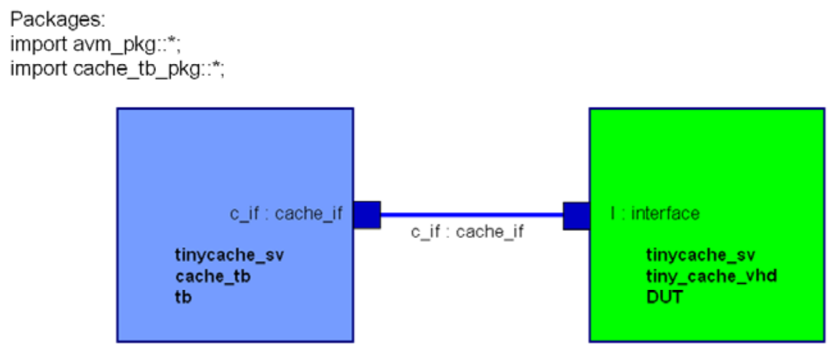
Design Hierarchy	Design Unit Name	Extends	File Name
top	top		top.sv
top	top		top.sv
c_if	cache_if		cache_interface.sv
DUT	tiny_cache_vhd		tiny_cache_vhd_fsm.vhd
tb	cache_tb		cache_tb.sv
c_if	cache_if		cache_interface.sv
env	tb_env	avm_env	rtl_env.svh
c_if	cache_if		cache_interface.sv
cov	coverage	avm_threaded_co...	coverage.svh
cpu	cpu_stim	avm_threaded_co...	cpu_stim.svh
cpu_f	tlm_req_rsp_channel		rtl_env.svh
driver	tlm_cache_driver	avm_threaded_co...	tlm_cache_driver.svh
mem	tlm_memory	avm_threaded_co...	tlm_memory.svh
mem_f	tlm_req_rsp_channel		rtl_env.svh
responder	tlm_mem_responder	avm_threaded_co...	tlm_mem_responder.svh
scoreboard	cache_scoreboard	avm_threaded_co...	scoreboard.svh
actual_cache_rsp	cache_rsp	mem_rsp	trans.svh
actual_cache_rsp_export	avm_analysis_export		scoreboard.svh
actual_cache_rsp_fifo	analysis_fifo		scoreboard.svh
actual_mem_req	mem_req	avm_transaction	trans.svh
actual_mem_req_export	avm_analysis_export		scoreboard.svh
actual_mem_req_fifo	analysis_fifo		scoreboard.svh
cache	tlm_cache	avm_threaded_co...	tlm_cache.svh
cache_req_export	avm_analysis_export		scoreboard.svh
cache_req_fifo	analysis_fifo		scoreboard.svh
cpu_f	tlm_req_rsp_channel		scoreboard.svh
cpu_req	mem_req	avm_transaction	trans.svh
dummy	cache_rsp	mem_rsp	trans.svh
mem	tlm_memory	avm_threaded_co...	tlm_memory.svh
mem_f	tlm_req_rsp_channel		scoreboard.svh
predicted_cache_rsp	cache_rsp	mem_rsp	trans.svh
predicted_cache_rsp_fifo	analysis_fifo		scoreboard.svh
predicted_mem_req	mem_req	avm_transaction	trans.svh
predicted_mem_req_fifo	analysis_fifo		scoreboard.svh

4.5 Testbench Environment

4.6 Top-level

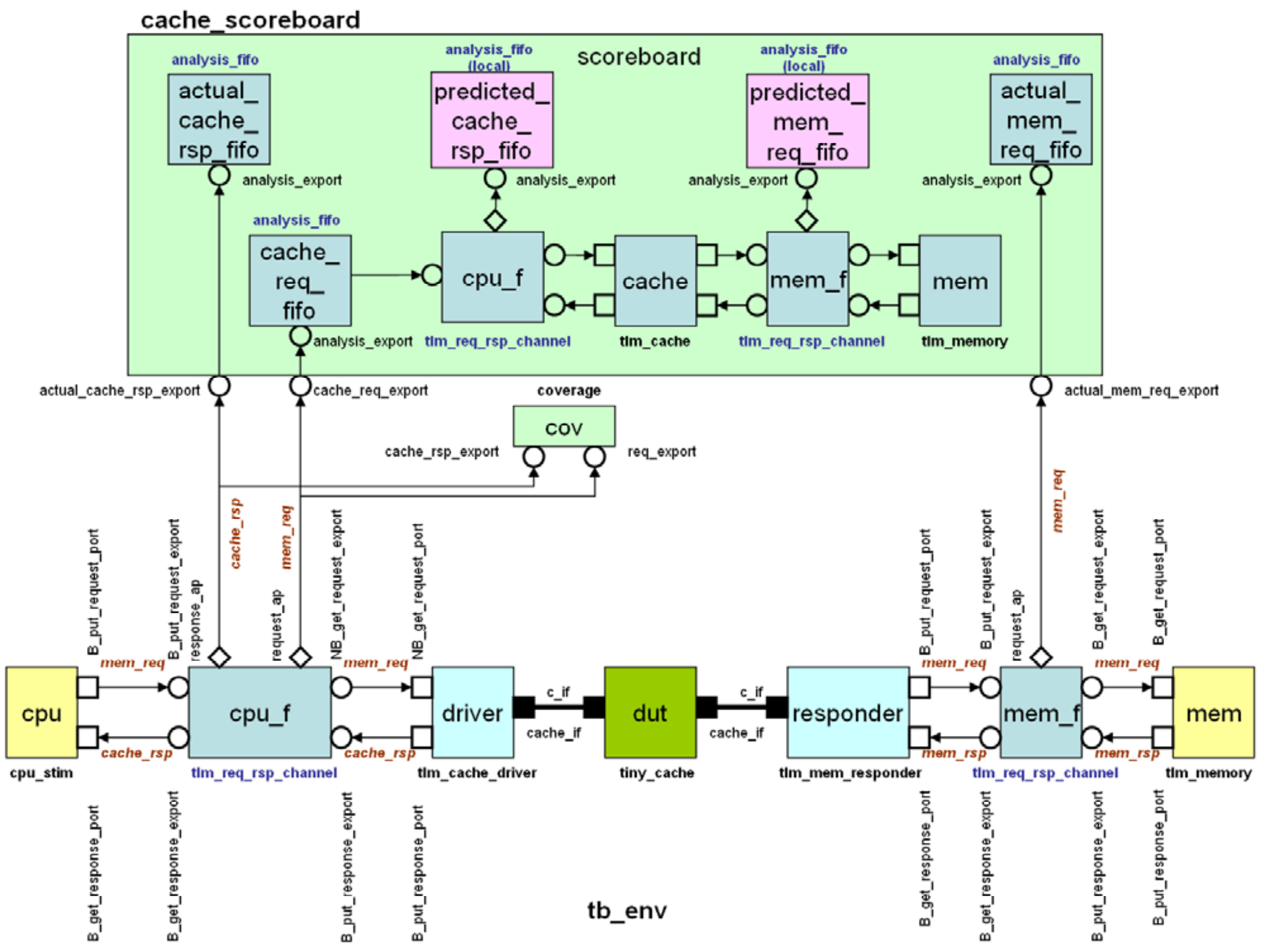
The diagram below shows the top-level with the Tinycache Device Under Test connected to the testbench Program Block via a single SystemVerilog interface

A Module containing the SVA properties is also “Bound” to the DUT at this level.



4.7 Testbench

The following diagram shows a representation of the overall testbench



As mentioned earlier, the Tinycache DUT sits between the CPU and the main memory.

Across the bottom is the master/slave testbench structure, from left to right this comprises:

- ***cpu*** - In the testbench the CPU block emulates the behaviour of the CPU by generating stimulus - ie read, write and reset operations via mem_req (memory request) transactions.
- ***cpu_f*** - This contains 2 fifos, one in each direction (depth of 1). Since the transaction layer is untimed, each fifo holds the next transaction until it can be processed.
- ***driver*** - This converts between transactions and "pin wiggles" in the interface connected to the DUT
- ***dut*** - this is the RTL-level cache
- ***responder*** – This does the same as the ***driver*** but on the memory side
- ***mem_f*** – This does the same as ***cpu_f*** but on the memory side
- ***mem*** - This is a high-level model of the memory.

The other two main blocks are the ***coverage*** block and the ***scoreboard***.
To maximise reuse, these would typically be attached to the interface via a monitor (similar to a one-way driver/responder). In this case, since it is a custom interface, they are attached to the driver/responder themselves via analysis ports.

- ***cov*** - This is a hierarchical block and contains the covergroups to gather functional coverage information on the operations performed, addresses accessed etc.
- ***scoreboard*** - This contains a replica of the main testbench, but this time with a high-level model of the ***tinycache***.
  The same information goes to the scoreboard (via analysis ports) and the predicted and actual requests and responses are "stored" in fifos and compared. Messages then write out the information to the log.

In addition, the assertions/coverpoints module ***tiny_cache_props*** is "bound" to the VHDL DUT so that assertions and temporal coverage can monitor activity of the RTL cache.

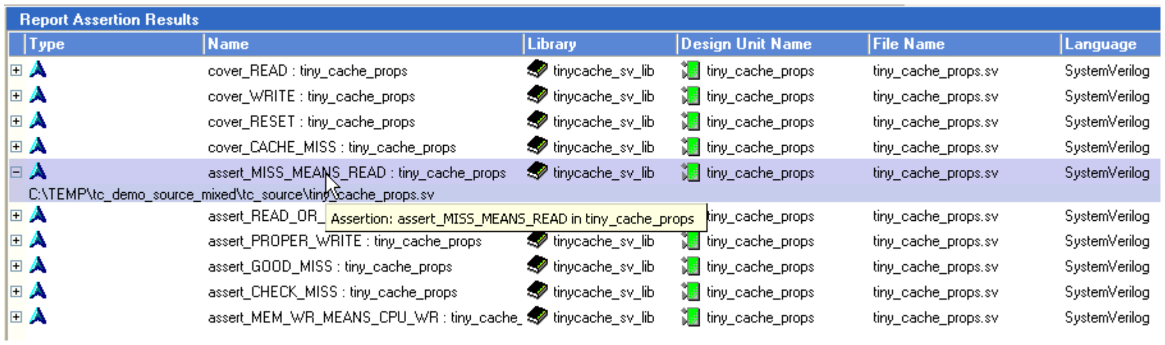
1. **Double-click** on the top-level testbench environment instance ***env***.
   This opens the ***tb_env*** class in the text editor.
   - Note the code browser to the left of the editing area which groups the main objects within the file
   - You can control the amount of information shown here using **Options> Preferences** and choosing ***Code Browser***.
   - The Code Browser provides two-way cross referencing.
2. **Expand** the ***tb_env*** class in the ***Code Browser*** and then the ***declarations*** group.
3. **Click** on items in the ***Code Browser*** and see them highlighted in the ***editor*** pane. Clicking in constructs in the ***editing*** pane will highlight them in the ***Code Browser***.
4. **Close** DesignPad using **File> Exit.**

4.8 Assertions and Covergroups

Assertions and coverage are vital aspects of an Advanced Verification methodology. In addition to the advanced Find capabilities, HDL Designer provides specific reports to summarise and locate assertions, property-based temporal coverage and SVA functional covergroups within the design. You can report assertions/coverage within a single object, its hierarchy or within the entire library

1. In the ***Design Manager***, **RMB** on the Design Unit ***top*** and choose **Reports> Assertions(SV)> For Library**

A report tab is opened. You can sort, filter and group these entries (RMB on the header) or even add/remove additional columns of information.



Report Assertion Results

Type	Name	Library	Design Unit Name	File Name	Language
⊞	cover_READ : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊞	cover_WRITE : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊞	cover_RESET : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊞	cover_CACHE_MISS : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊟	assert_MISS_MEANS_READ : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
C:\TEMP\tc_demo_source_mixed\tc_source\tiny_cache_props.sv					
⊞	assert_READ_OR_		tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊞	assert_PROPER_WRITE : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊞	assert_GOOD_MISS : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊞	assert_CHECK_MISS : tiny_cache_props	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog
⊞	assert_MEM_WR_MEANS_CPU_WR : tiny_cache_	tinycache_sv_lib	tiny_cache_props	tiny_cache_props.sv	SystemVerilog

Assertion: assert_MISS_MEANS_READ in tiny_cache_props

2. **Double-click** on any entry (eg ***assert_MISS_MEANS_READ***).
   DesignPad opens and highlights the assertion.

3. Go back to the Assertions report and **close** it (RMB over the tab at the bottom or click on the small ☒ at the top-right of the report tab.

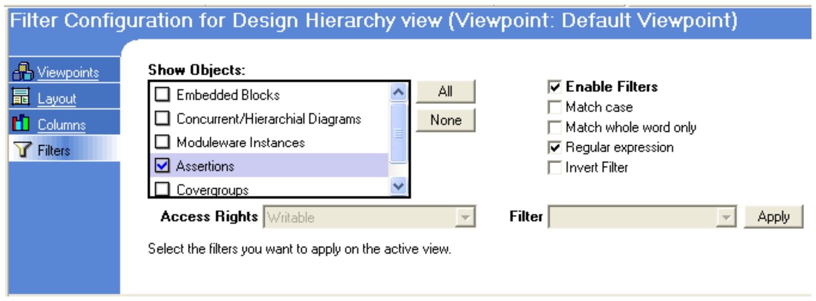
4. You can optionally view assertions in the hierarchy browser using the Viewpoint filter mechanism. There are Additional Viewpoint filters for

   - Assertions, covergroups for the Hierarchy browser
   - Interfaces and Program Blocks for the Design Unit browser

5. Choose **Tools> Viewpoint manager**

6. **Click** on the ***filters*** tab (ensure the ***hierarchy window*** is active, if not click in the ***hierarchy window*** to activate it)

7. **Scroll** in the ***Show Objects*** window and ensure the ***Assertions*** entry is checked.



Filter Configuration for Design Hierarchy view (Viewpoint: Default Viewpoint)

Show Objects:
- ☐ Embedded Blocks
- ☐ Concurrent/Hierarchial Diagrams
- ☐ Moduleware Instances
- ☑ Assertions
- ☐ Covergroups

- ☑ Enable Filters
- ☐ Match case
- ☐ Match whole word only
- ☑ Regular expression
- ☐ Invert Filter

Access Rights: Writable

Filter:

Select the filters you want to apply on the active view.

8. Now **close** the ***Viewpoint Manager*** using the ☒ in the top right of its window.

9. RMB on the Design Unit ***tiny_cache_props*** and choose
   Show Hierarchy

10. If necessary, **expand** this and you will see the assertions within ***tiny_cache_props***.

11. Similarly for Covergroups, RMB on the Design Unit **top** and choose
    Reports> Covergroups (SV)> For Library

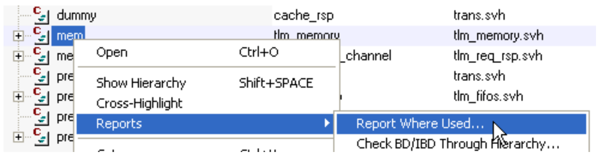
12. **Double-click** on any entry (eg ***address_cover***).
    DesignPad opens and highlights the construction for the new ***address_cover*** object.

13. To locate the declaration, **RMB** on ***address_cover*** and choose **Go To Declaration**.
14. Close DesignPad and then close the CoverGroup report (**RMB** over the tab at the bottom or click on the small at the top-right of the report tab.

4.9 Where Used

When making changes to a design object, it is important to be able to find out where and how that object is used. The ***Where Used*** report lists the instances of a selected Interface, Program Block, Package or Class.

1. **Expand** the ***Scoreboard*** instance in the hierarchy browser.
2. **RMB** on ***mem*** and choose **Reports> Report Where Used…**



3. A dialog appears to allow you to change the scope of the report. Leave the scope as ***Libraries in this Design Explorer tab only*** and **OK** the dialog.

Another report tab appears (similar to those for assertions and covergroups earlier).

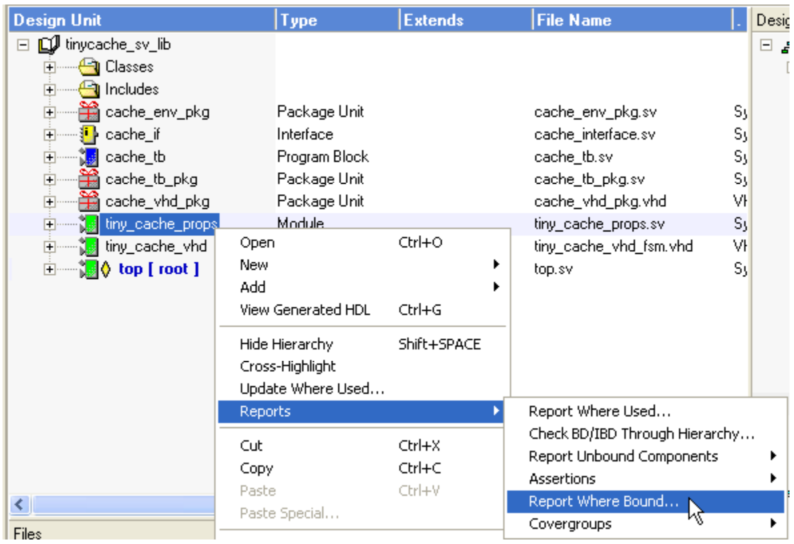
You can see that the ***mem*** class (which is the memory model) is used in 2 places – in the top level of the testbench where it acts as a model of the main memory and also in the scoreboard. This is because the scoreboard replicates the testbench using a high-level model of the DUT in order to predict the response to a given stimulus and to compare the predicted value with the actual values.

4. To check – **double click** on each entry in turn which will open the containing class in DesignPad.
5. **Close** the report.

4.10 Where Bound

The SystemVerilog "Bind" construct allows you to associate verification components with a DUT without editing the source. This is especially useful when the DUT is of another language as is the case in the tutorial example where the SVA Properties, Assertions and Cover directives have been placed in a Module and associated with the VHDL DUT in the top-level.

1. In the Design Unit Browser, **RMB** over ***tiny_cache_props*** and choose **Reports> Where Bound (SV)**



Design Unit	Type	Extends	File Name
tinycache_sv_lib			
Classes			
Includes			
cache_env_pkg	Package Unit		cache_env_pkg.sv
cache_if	Interface		cache_interface.sv
cache_tb	Program Block		cache_tb.sv
cache_tb_pkg	Package Unit		cache_tb_pkg.sv
cache_vhd_pkg	Package Unit		cache_vhd_pkg.vhd
tiny_cache_props	Module		tiny_cache_props.sv
tiny_cache_vhd			tiny_cache_vhd_fsm.vhd
top [root]			top.sv

Another report tab appears (similar to those for assertions and covergroups earlier).

You can see that the ***tiny_cache_props*** module (which contains the Properties, Assertions and Cover directives) is bound as an instance ***tc_bind*** in file ***top.sv***.

2. **Double-click** on the entry
   DesignPad opens and highlights the bind instance which binds the properties module to the VHDL tiny_cache_vhd component.

3. Close DesignPad and then close the Where Bound report (**RMB** over the tab at the bottom or click on the small ☒ at the top-right of the report tab.

5 The DesignPad Text Editor

DesignPad is a fully-featured, design-aware text editor. It provides syntax parsing and language templates for VHDL, Verilog and SystemVerilog, plus customisable syntax highlighting for these and a range of other languages including PSL, XML, C/++, Tcl, XML etc.

The editor is tightly integrated with HDL Designer to provide Drag 'n drop Component instantiation, Design navigation (open up/down), Cross-referencing and Task and Version Management integration.

In addition to common text editor features, DesignPad also provides Outlining/code folding, Column editing, bracket matching, HTML Export, link traversal, file compare and powerful Search functions including "find in files".

A Code Browser provides an interactive abstraction of the key constructs within the file.

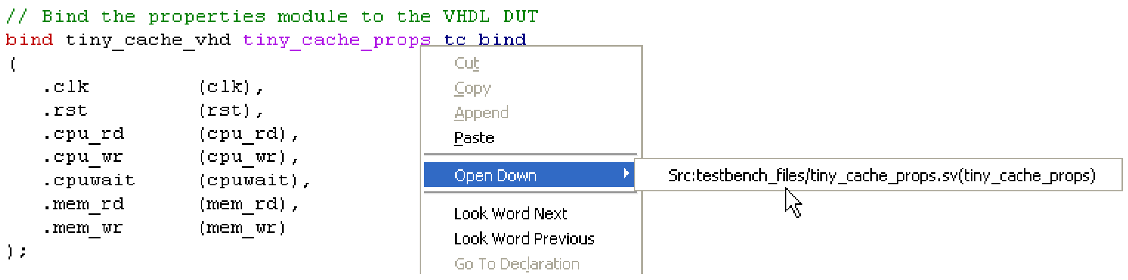
With customizable, menus and shortcuts, the editor also supports Emacs and Vi keystroke emulation.

For more information on DesignPad please see **Help> User Guide** from within DesignPad; or from the Design Manager see **Help> Help and Manuals** to open the HDS InfoHub and choose DesignPad User Manual from the **Help and Manuals** tab.

5.1.1 Design Traversal

1. In the ***Design Unit browser***, **double-click** on ***top***.
   This will open the module "top" in DesignPad.

2. Click the cursor on the ***tiny_cache_props*** instance and **RMB** to choose **Open Down > <file path>**
   This opens the properties module.

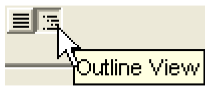


```
// Bind the properties module to the VHDL DUT
bind tiny_cache_vhd tiny_cache_props tc_bind
(
   .clk          (clk),
   .rst          (rst),
   .cpu_rd       (cpu_rd),
   .cpu_wr       (cpu_wr),
   .cpuwait      (cpuwait),
   .mem_rd       (mem_rd),
   .mem_wr       (mem_wr)
);
```

5.1.2 Outline mode

In addition to the code browser view, you can also perform "code folding" by using "Outline mode".

1. **Click** on the right-hand icon underneath the line number column or choose **Document> Outline Mode.**



The code constructs are collapsed allowing you to choose which parts of the file you wish to view or work on.

2. Experiment with **expanding** and **collapsing** some of the constructs.

3. To turn Outline mode off either click on the left-hand icon underneath the line number column or choose **Document> Outline Mode.**

5.1.3 Syntax checks

DesignPad performs syntax checking when the file is saved. You can also check the syntax at any time simply by clicking on the syntax check toolbar button or **Document> Check Syntax**.

1. Remove the trailing parenthesis ")" from the module declaration line and check the syntax using **Document> Check Syntax.**

A report pane appears and flags any warnings or errors. You can click on the entries to take you to the problem area of the code. This report can also be saved or printed for future reference.

2. **Close** the file **without saving** either by clicking on the ✖ in the top-right corner of the editing pane or by using **File> Close**.
   Ensure you say ***No*** when asked to save the changes.

6 Questa Verification Flow

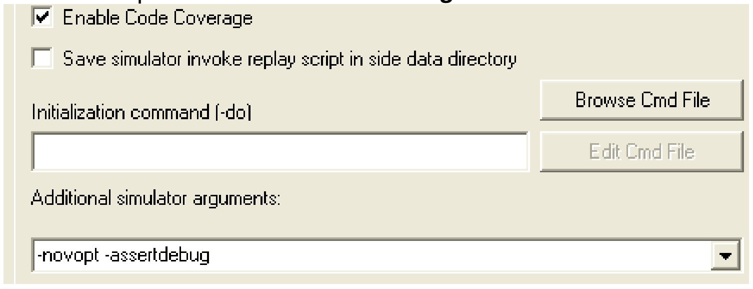
After initial analysis of the project files, HDL Designer incrementally parses files when they change and identifies the impact on any dependencies. This allows the tool to automatically determine which files need to be recompiled at any given point.

1. **Select** Design Unit ***top*** in the ***Design Unit Browser***
2. To run the Questa compile and simulation flow either **RMB** on the ***Simulate*** icon in the main Shortcut Bar and choose **Run through components** or use the pulldown menu **Tasks> QuestaSimFlow> Run through components.**

Compilation works across multiple libraries.
During compilation, you may see a number of warning messages. These can be suppressed in future compilations by adding the Questa suppress command to the compilation task (see Appendix B - Changing Questa compilation settings in advance on page 23).

The Simulation dialog will appear.

3. Check the option ***Enable Code Coverage***…


Enable Code Coverage
Save simulator invoke replay script in side data directory
Initialization command (-do)
Browse Cmd File
Edit Cmd File
Additional simulator arguments:
☑ Enable Code Coverage
☐ Save simulator invoke replay script in side data directory
Additional simulator arguments: -novopt -assertdebug

4. In the ***Additional simulator arguments*** field **type**
   -novopt -assertdebug
   then **OK** the dialog

QuestSim invokes

5. Enable the ***Analysis windows*** using **View> Coverage> Assertions, View> Coverage> Cover Directives, View> Coverage> CoverGroups**
6. Setup the ***wave window*** by running the wave.do script e.g.
   do C:/HDS/examples/tutorial_ref/Import/tc_demo_source_mixed_avm3/wave.do
7. **Run** simulation for a short time e.g.
   run 1000
8. Examine the simulation waveforms and the assertions in the waveform.
9. Also refer to the messages in the log window – The ***configure*** function within the ***tb_env*** class currently has the message verbosity level set to the maximum of 3. This can be changed to a lower level of 1 or 2 or turned off altogether by setting the value to 0.
10. Ensure ***top*** is selected in the ***Sim tab*** of the ***Workspace window*** and View the ***assertions*** and ***cover directives*** results in the ***Analysis window***.

11. Now **click** on the ***cache_env_pkg*** in the ***Sim tab*** of the ***Workspace window*** and View the ***covergroup*** results in the ***Analysis window***.
12. **Close** Questa using **File> Quit**.

7 Conclusion

This concludes the overview tutorial.

You should now feel confident to follow similar steps on your own existing design or to create a new design using the file template, language templates and drag and drop instantiation capabilities.

For information on the many other HDL Designer features which were not covered in this tutorial (such as Design Management, Version Management, etc.) please see **Help> Help and Manuals** to open the HDS InfoHub and choose the HDS User Manual from the **Help and Manuals** tab.

8 Appendix A – Using the AVM source library

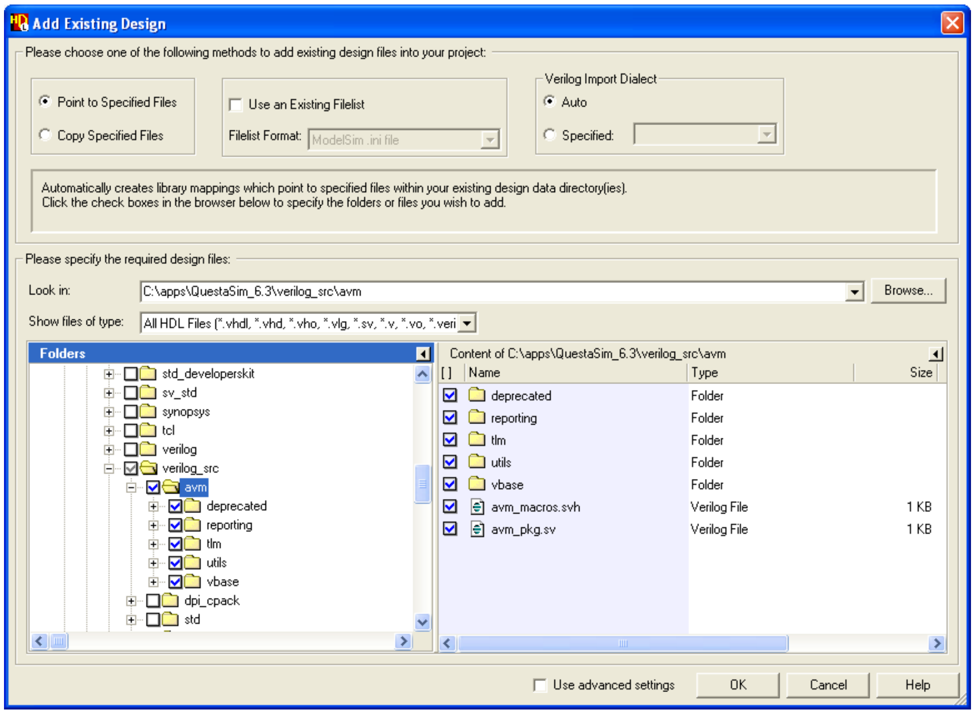
If you would like to explore the AVM source within HDL Designer rather than using the pre-compiled version, you simply follow similar steps to those for “importing” the tinycache design.

8.1 Add the AVM files to the project

1. Choose **File> Add> Existing Design**. This will bring up the dialog you used earlier which allows you to create new design objects or add existing design files.
2. In the resulting dialog box, you can either reference (point to) or copy the existing design. Choose: ***Point to Specified Files***.
3. Set the ***Show files of type*** filter to ***All HDL Files***
4. Using the File browser, navigate to the folder containing the AVM source files in the Questa installation, for example:

   C:\apps\QuestaSim_6.3\verilog_src

   Check the avm folder in the Folders pane.
   The dialog should look like this (note the check marks and the content of the right-hand pane in the browser):



Add Existing Design

Please choose one of the following methods to add existing design files into your project:

[✓ Point to Specified Files] Copy Specified Files

Use an Existing Filelist

Filelist Format: ModelSim .ini file

Verilog Import Dialect: [✓ Auto] Specified:

Automatically creates library mappings which point to specified files within your existing design data directory(ies).
Click the check boxes in the browser below to specify the folders or files you wish to add.

Please specify the required design files:

Look in: C:\apps\QuestaSim_6.3\verilog_src\avm

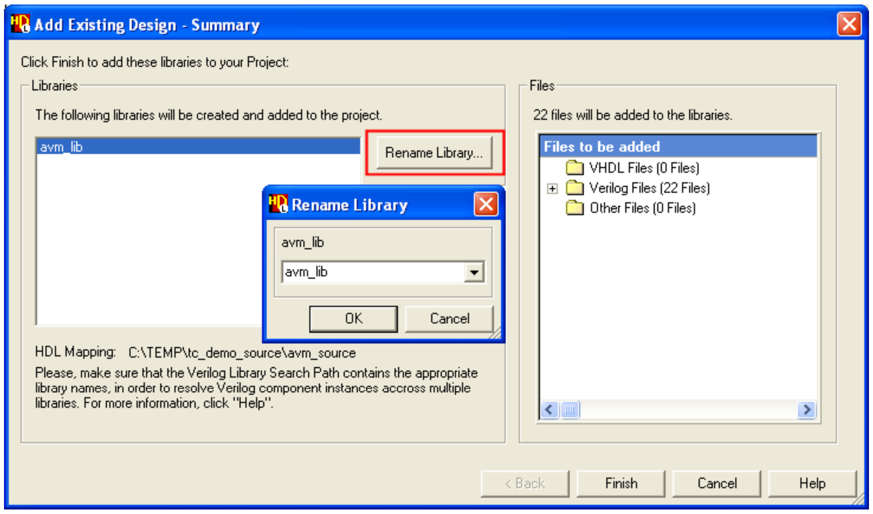
Show files of type: All HDL Files (*.vhdl, *.vhd, *.vho, *.vlg, *.sv, *.v, *.vo, *.veri

Folders:
- std_developerskit
- sv_std
- synopsys
- tcl
- verilog
- ✓ verilog_src
  - ✓ avm
    - ✓ deprecated
    - ✓ reporting
    - ✓ tlm
    - ✓ utils
    - ✓ vbase
  - dpi_cpack
  - std

Content of C:\apps\QuestaSim_6.3\verilog_src\avm

[ ]	Name	Type	Size
✓	deprecated	Folder	
✓	reporting	Folder	
✓	tlm	Folder	
✓	utils	Folder	
✓	vbase	Folder	
✓	avm_macros.svh	Verilog File	1 KB
✓	avm_pkg.sv	Verilog File	1 KB

5. Click **OK**
6. We are going to have a separate library for the AVM, so in the ***Summary*** dialog, click on **Rename Library…**
7. In the ***Rename Library*** dialog type in **avm_lib** and **OK** the dialog.


Add Existing Design - Summary
Click Finish to add these libraries to your Project:
Libraries
The following libraries will be created and added to the project.
avm_lib
Rename Library...
Rename Library
avm_lib
avm_lib
OK
Cancel
Files
22 files will be added to the libraries.
Files to be added
VHDL Files (0 Files)
Verilog Files (22 Files)
Other Files (0 Files)
< Back
Finish
Cancel
Help

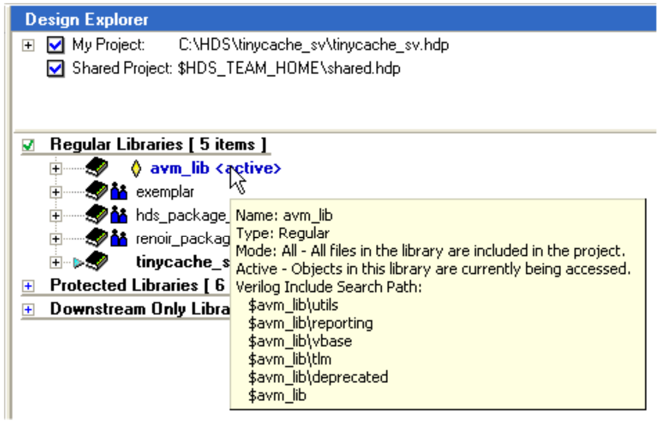
HDL Mapping: C:\TEMP\tc_demo_source\avm_source
Please, make sure that the Verilog Library Search Path contains the appropriate library names, in order to resolve Verilog component instances accross multiple libraries. For more information, click "Help".

8.2 AVM_LIB – check the Verilog `include search path for the library

As we have seen, the library contains a number of include files in subfolders. In order for these include files to be found, the appropriate search path must be defined.

6. To check the Verilog Include search path for the ***avm_lib***, click on the ***Project*** tab at the bottom left of the tool.

7. Ensure the ***Regular Libraries*** group is expanded.
8. Hover the cursor over the ***avm_lib*** and check the entries for “Verilog Include Search Path”

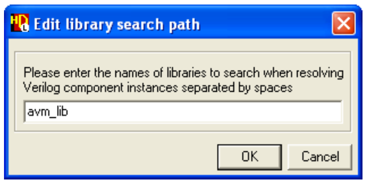


Name: avm_lib
Type: Regular
Mode: All - All files in the library are included in the project.
Active - Objects in this library are currently being accessed.
Verilog Include Search Path:
$avm_lib\utils
$avm_lib\reporting
$avm_lib\vbase
$avm_lib\tlm
$avm_lib\deprecated
$avm_lib

9. Ensure the list contains at least **$avm_lib**
10. If not, you can edit the search path using RMB over the ***avm_lib*** library and choosing **Edit Verilog Include Search Path**

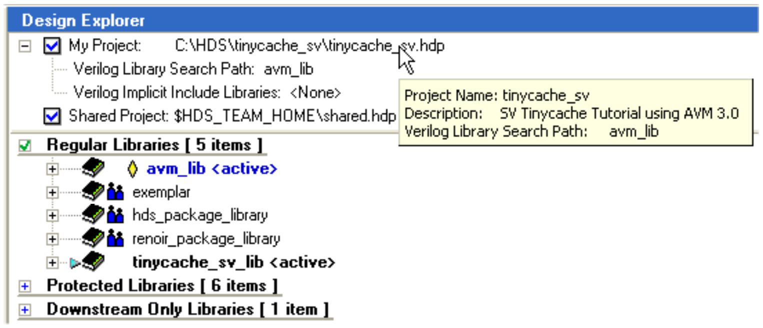
8.3 Change the Library search path for the project to use the new AVM_LIB

1. Staying in the Project Tab, RMB over ***My Project*** and choose **Edit Verilog Library Search Path**
2. Delete the pre-compiled library eg mtiAvm (if it is there) and enter **avm_lib** and **OK** the dialog. (Note there is no “$” – the Library Search Path contains logical library names rather than directory names)


Edit library search path

Please enter the names of libraries to search when resolving Verilog component instances separated by spaces

avm_lib


Design Explorer

My Project: C:\HDS\tinycache_sv\tinycache_sv.hdp
Verilog Library Search Path: avm_lib
Verilog Implicit Include Libraries: <None>
Shared Project: $HDS_TEAM_HOME\shared.hdp

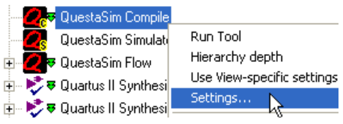
Project Name: tinycache_sv
Description: SV Tinycache Tutorial using AVM 3.0
Verilog Library Search Path: avm_lib

Regular Libraries [5 items]
avm_lib <active>
exemplar
hds_package_library
renoir_package_library
tinycache_sv_lib <active>
Protected Libraries [6 items]
Downstream Only Libraries [1 item]

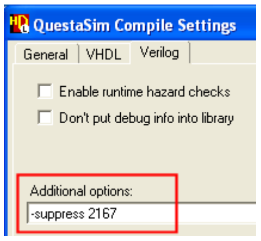
9 Appendix B - Changing Questa compilation settings in advance

The following steps explain how to modify the Questa Compile task setting. In addition to the options supplied in the dialog, you can also enter any vcom or vlog options. In this example, you will add the “-suppress” option to suppress a specific warning message.

1. **Click** on the ***Tasks/Templates tab*** which is located down the right hand edge of the tool.
2. Ensure the ***Tasks tab*** is active.
3. **RMB** on the ***Questa Compile task*** and choose **Settings…**



4. In the resulting dialog, **click** on the ***Verilog tab***
5. **Enter *-suppress 2167*** (including the minus sign) in the ***Additional Options field*** and **OK** the dialog.


QuestaSim Compile Settings

General | VHDL | Verilog

☐ Enable runtime hazard checks
☐ Don't put debug info into library

Additional options:
-suppress 2167