Course 10

Debugging Tools

Rev. Oct 11, 2012

Applies to Active-HDL 9.2 and later

Table of Contents

Overview

Active-HDL users have access o a rich set of debugging tools that enables quick ways to detect and diagnose design issues. Many of the debugging tools have their own GUI windows that can be accessed from menu **View** as shown below:

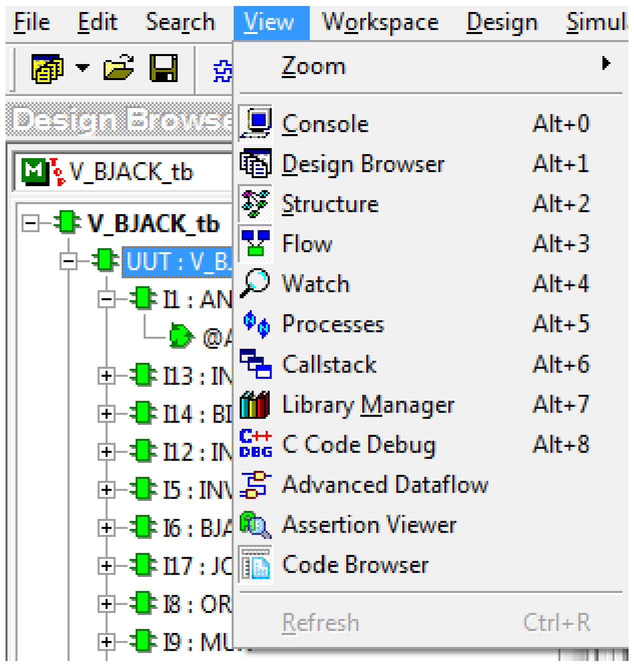


Figure 1 Debugging Tools

Design Browser

Active-HDL provides a set of dedicated windows that allow displaying, analyzing, browsing, and exploring a design hierarchy. The following windows can be used while debugging your design or viewing its structure:

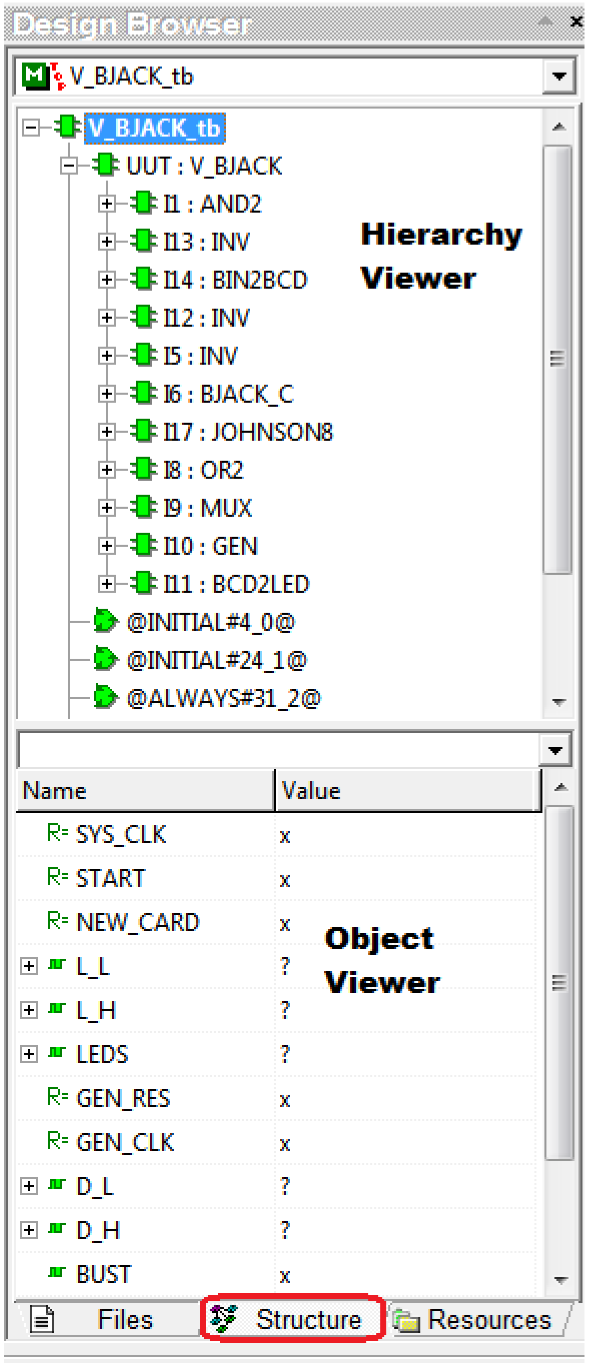
- Structure tab of design Browser
- Object Viewer

Structure tab

The **Structure** window displays a hierarchy of an elaborated design. The window presents a hierarchy of VHDL, Verilog, EDIF, or SystemC designs.

The **Structure** window allows you to view:

- Hierarchy of the currently simulated design.
- Structure of any project in the off-line simulation mode.

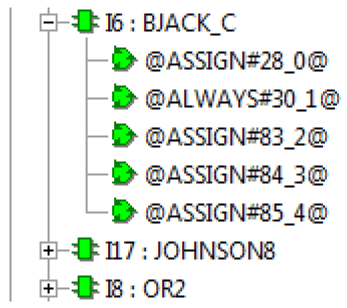


```
V_BJACK_tb
  UUT : V_BJACK
    I1 : AND2
    I13 : INV
    I14 : BIN2BCD
    I12 : INV
    I5 : INV
    I6 : BJACK_C
    I17 : JOHNSON8
    I8 : OR2
    I9 : MUX
    I10 : GEN
    I11 : BCD2LED
  @INITIAL#4_0@
  @INITIAL#24_1@
  @ALWAYS#31_2@
```

Name	Value
SYS_CLK	x
START	x
NEW_CARD	x
L_L	?
L_H	?
LEDS	?
GEN_RES	x
GEN_CLK	x
D_L	?
D_H	?
BUST	x

Figure 2 Design Browser

If a statement has no label in source code, the Structure tab displays a substitute label automatically assigned by the simulator, for example `@ASSIGN#28_0@`, as shown below.



```
I6 : BJACK_C
    @ASSIGN#28_0@
    @ALWAYS#30_1@
    @ASSIGN#83_2@
    @ASSIGN#84_3@
    @ASSIGN#85_4@
I17 : JOHNSON8
I8 : OR2
```

Figure 3 Labels in Structure Tab

Object Viewer

The **Object Viewer** window shows design objects (ports, signals, nets, generics, parameters, constants, variables, etc.) coming from a design region specified in the Hierarchy Viewer.
The object types along with additional information are presented in a tabular format.
The following columns can be enabled in the Object Viewer window:

- Name

  It displays the name and attribute (type) of the object.

- Value

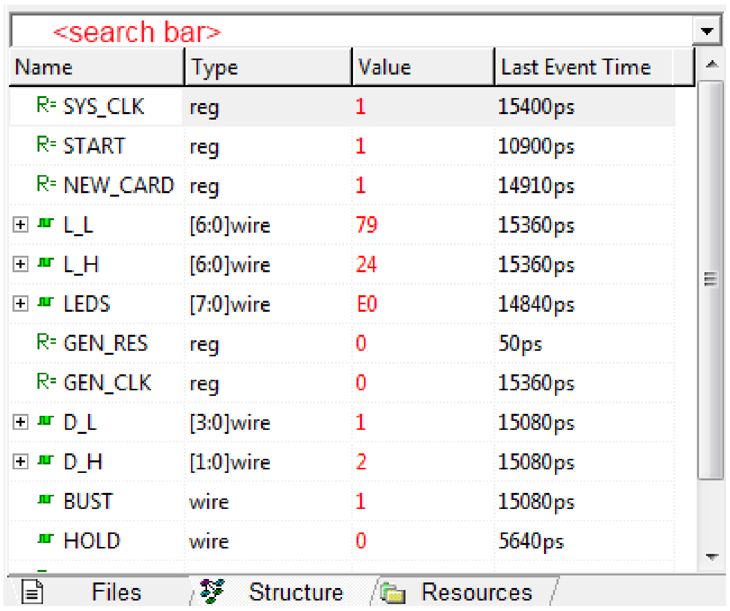
  It displays the current value of the object.

- Type

  This Column displays the type of the object.

- Last Event Time

  It displays the time of the last event. This column is used only for signals, nets, and registers.



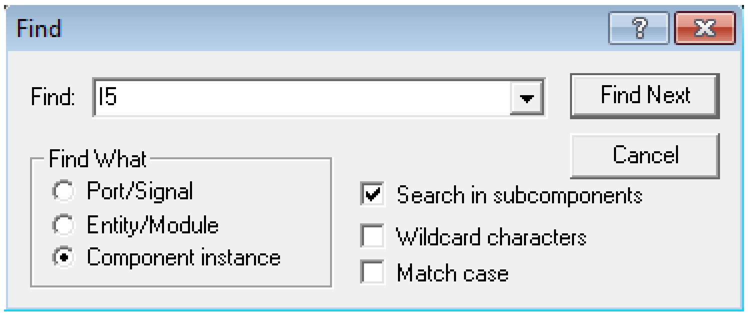
<search bar>

Name	Type	Value	Last Event Time
SYS_CLK	reg	1	15400ps
START	reg	1	10900ps
NEW_CARD	reg	1	14910ps
L_L	[6:0]wire	79	15360ps
L_H	[6:0]wire	24	15360ps
LEDS	[7:0]wire	E0	14840ps
GEN_RES	reg	0	50ps
GEN_CLK	reg	0	15360ps
D_L	[3:0]wire	1	15080ps
D_H	[1:0]wire	2	15080ps
BUST	wire	1	15080ps
HOLD	wire	0	5640ps

Figure 4 Object viewer

Structure Find

The **Structure Find** is very useful when searching for design units (nodes) and objects through a project hierarchy selected in the Hierarchy Viewer. The shortcut to bring the tool up is to set the focus in the Structure tab and right click on any instance and click Find.



Find: I5

Find What:
- Port/Signal [ ]
- Entity/Module [ ]
- Component instance [✓]

- [✓] Search in subcomponents
- [ ] Wildcard characters
- [ ] Match case

Figure 5 Find in Structure Tab

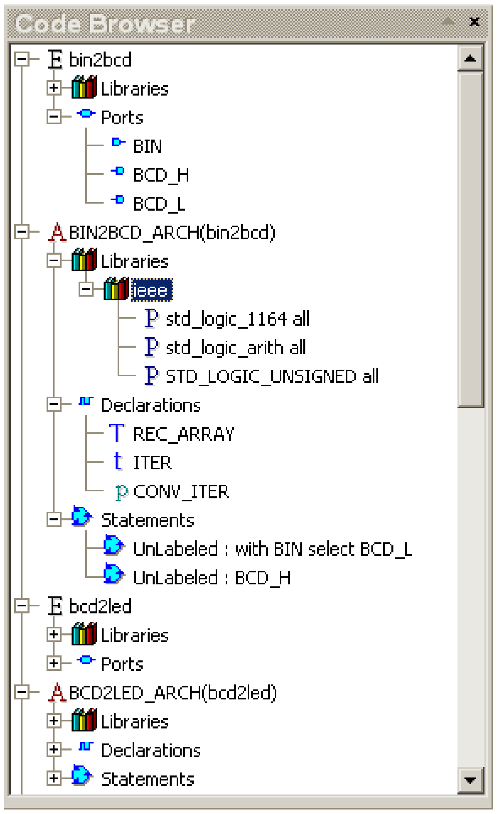
You can search Port/Signal, Entity/Module and Component instance.

Code Browser

The Code Browser is a tool that presents the structure of source code being currently open in the **HDL Editor** window and allows conveniently navigating through source code. Double-clicking an item in the Code Browser scrolls the HDL Editor window and places the insertion point in the corresponding line of code. This feature is especially useful for source files containing hundreds of lines of code. On the other hand, when you edit a section of source code, the edited item can be automatically expanded and highlighted bold in the Code Browser. To enable this functionality, select the **Synchronize with HDL Editor in the Editors | HDL Editor | Code Browser** section of the **Preferences** dialog box.

The Code Browser also performs an on-the-fly source code analysis and provides you with preliminary information on correctness of your code. Refer to the Source Code Analysis section of Code Browser in help file for more information.

To open the Code Browser, click the Code Browser button on the toolbar.



```
Code Browser
E bin2bcd
  Libraries
  Ports
    BIN
    BCD_H
    BCD_L
A BIN2BCD_ARCH(bin2bcd)
  Libraries
    ieee
      P std_logic_1164 all
      P std_logic_arith all
      P STD_LOGIC_UNSIGNED all
  Declarations
    T REC_ARRAY
    t ITER
    p CONV_ITER
  Statements
    UnLabeled : with BIN select BCD_L
    UnLabeled : BCD_H
E bcd2led
  Libraries
  Ports
A BCD2LED_ARCH(bcd2led)
  Libraries
  Declarations
  Statements
```

Figure 6 Code Browser

As shown in the figure above, the Code Browser presents **design units** along with their contents in a **structured manner**. The items presented in the window contain **design entities, architectures, included libraries, declarations of types, functions, signals, ports**, etc. Each item is nested under the entity or architecture it belongs to. You can expand or collapse selected nodes by clicking the expand/collapse boxes displayed at the nodes.

NOTE: The source code analysis may impact the performance of opening designs and files. If you are not going to use the Code Browser during your daily work, use the default values of options in **Preferences | Editors | HDL Editor | Code Browser** as shown in above image. Otherwise, when using code browser constantly, in order to minimize the performance impact on design and file operations, set the Background compilation option to Always. It slows down opening the design, but you save time on opening subsequent files. As of now Code Browser is for VHDL only.

Watch Window

The Watch window is a tool for displaying values of both HDL/SystemC objects and C/C++ variables. Values of HDL/SystemC objects such as ports, signals, nets, or registers are displayed when the debugger is in the HDL scope. When you step from HDL to C/C++/SystemC code, the debugger and the Watch window switch to the C/C++ scope and show C/C++ variables.

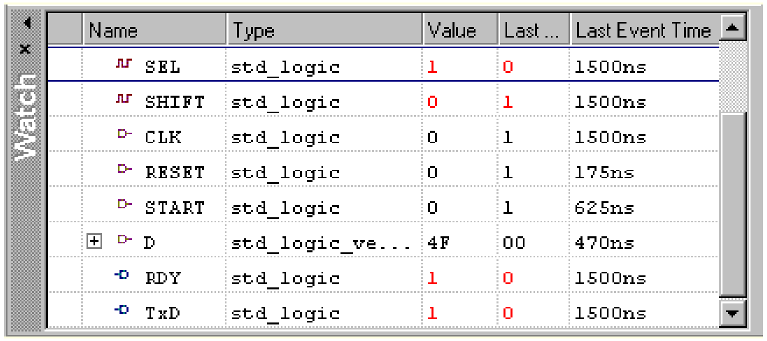
The Watch Window can also be used when no simulation is running and simulation results have been loaded from *.asdb* files. The functionality of the Watch window is then limited to showing values of signals recorded in the simulation database (*.asdb*).

The Watch window is empty when it is first displayed. There are several ways to add objects to the Watch window:

- Drag a signal from Object Viewer and Structure tab.

- Use the Add to | Watch command from the pop-up menu.
- Use the **`add watch`** macro command followed by a signal name. Wild cards are allowed.

The Watch window uses a grid consisting of seven columns: **Name**, **Value**, **Last Value**, **Type**, and **Last Event Time**.



Name	Type	Value	Last ...	Last Event Time
SEL	std_logic	1	0	1500ns
SHIFT	std_logic	0	1	1500ns
CLK	std_logic	0	1	1500ns
RESET	std_logic	0	1	175ns
START	std_logic	0	1	625ns
D	std_logic_ve...	4F	00	470ns
RDY	std_logic	1	0	1500ns
TxD	std_logic	1	0	1500ns

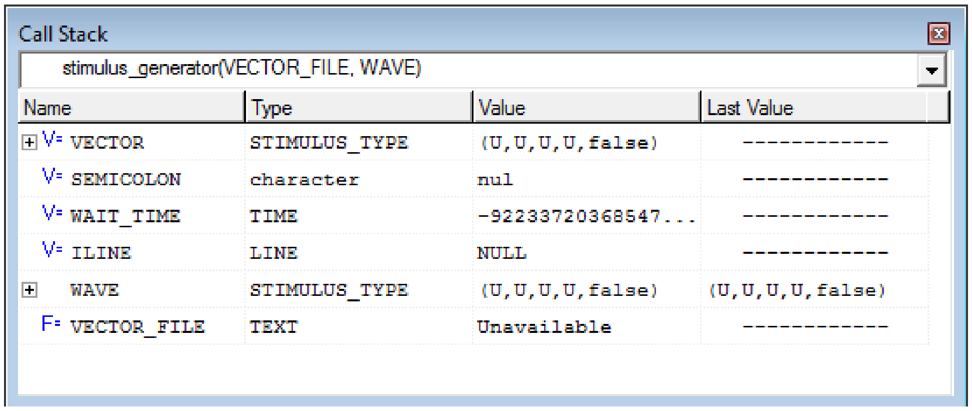
Figure 7 Watch Window

Call Stack Window

The Call Stack window in Active-HDL can show:

- A call stack of HDL subprograms including processes, VHDL procedures and functions, and Verilog tasks and functions.
- A call stack of C/C++ function calls.

The contents of the Call Stack window depend on the current execution context. If the simulation has been halted in HDL code, an HDL call stack will be shown. If you step through C/C++ code or a C/C++ code breakpoint has been hit, the Call Stack window will show a call stack of C/C++ function calls.



stimulus_generator(VECTOR_FILE, WAVE)

Name	Type	Value	Last Value
VECTOR	STIMULUS_TYPE	(U,U,U,U,false)	------------
SEMICOLON	character	nul	------------
WAIT_TIME	TIME	-92233720368547...	------------
ILINE	LINE	NULL	------------
WAVE	STIMULUS_TYPE	(U,U,U,U,false)	(U,U,U,U,false)
VECTOR_FILE	TEXT	Unavailable	------------

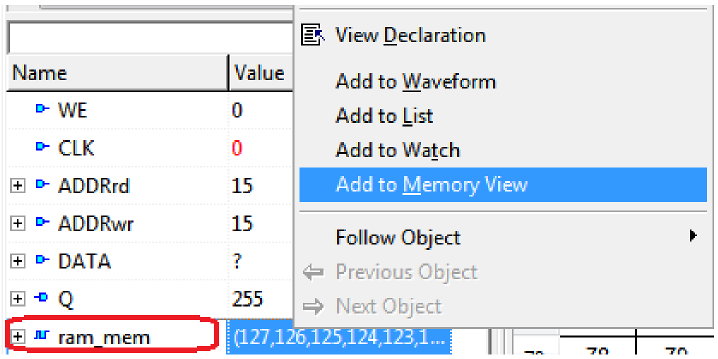
Figure 8 Call Stack Window

Memory Viewer

The Memory Viewer is a tool for viewing and manipulating contents of memories. The Memory Viewer shows memories in a spreadsheet-like grid where each cell in the grid corresponds to one memory cell.

The Memory Viewer can display contents of memories defined in the currently simulated design as well as contents of files with memory initialization data. Both types of memories (design objects and files) can be edited in the viewer. A memory file can be used for setting values of memory cells in the simulated model.

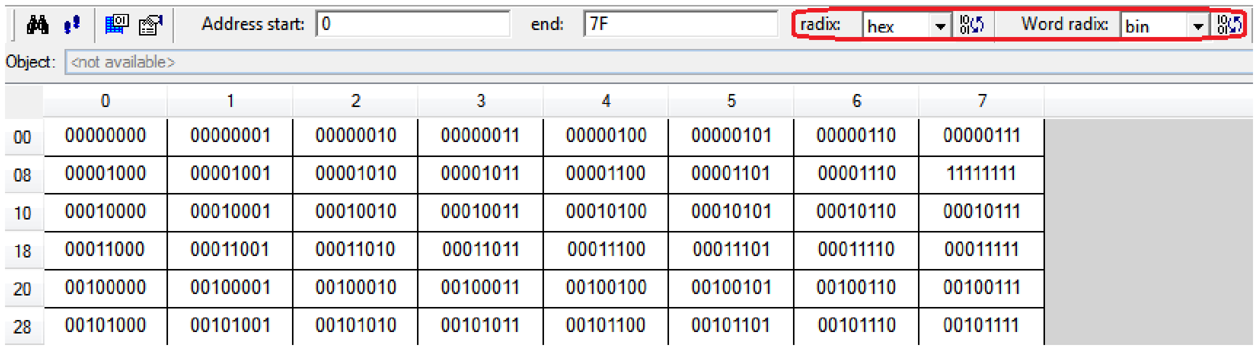
An easy way to add objects into Memory Viewer is to right click on the item in Object Viewer and select to Add to Memory View.



Name	Value
WE	0
CLK	0
ADDRrd	15
ADDRwr	15
DATA	?
Q	255
ram_mem	(127,126,125,124,123,1...

Figure 9 Adding objects to Memory Viewer

The display mode (radix, notation, etc.) for values displayed in memory cells can be adjusted. Likewise, you can modify the number of memory cells displayed in each row in the Memory Viewer window. Memories displayed in the Memory Viewer can be searched. You can search for specific values or use wildcards or regular expressions for more advanced searches. You can also go to a specific address in the memory.



Address start: 0 end: 7F radix: hex Word radix: bin

Object: <not available>

	0	1	2	3	4	5	6	7
00	00000000	00000001	00000010	00000011	00000100	00000101	00000110	00000111
08	00001000	00001001	00001010	00001011	00001100	00001101	00001110	11111111
10	00010000	00010001	00010010	00010011	00010100	00010101	00010110	00010111
18	00011000	00011001	00011010	00011011	00011100	00011101	00011110	00011111
20	00100000	00100001	00100010	00100011	00100100	00100101	00100110	00100111
28	00101000	00101001	00101010	00101011	00101100	00101101	00101110	00101111

Figure 10 Memory Viewer

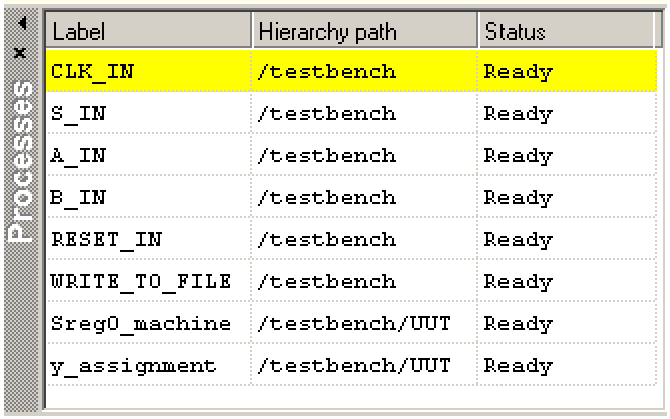
Process Viewer

The **Process Viewer** is a tool that displays a flat list of all processes in the simulated model. The term process is used here to mean:

- In VHDL: process statements, concurrent signal assignment statements, concurrent assertion statements and concurrent procedure call statements
- In Verilog: continuous assignment statements, initial and always procedural blocks, and primitives

The Process Viewer can only be used after simulation has been initialized.

The processes are executed in the order they appear in the window, starting from the top-most one. The yellow bar and the Active label indicate the process currently being executed. You can manually reorder the list of pending processes and change their execution order.
The list of processes is divided into three columns, which show the label, hierarchical path and status of each process. For processes without explicit labels, the compiler generates pseudo-labels, which include the number of the source file line in which a process is located (e.g., line__12). In the case of processes declared within generate statements, their labels are modified by a suffix showing the instance number.



Label	Hierarchy path	Status
CLK_IN	/testbench	Ready
S_IN	/testbench	Ready
A_IN	/testbench	Ready
B_IN	/testbench	Ready
RESET_IN	/testbench	Ready
WRITE_TO_FILE	/testbench	Ready
Sreg0_machine	/testbench/UUT	Ready
y_assignment	/testbench/UUT	Ready

Figure 11 Processes Viewer

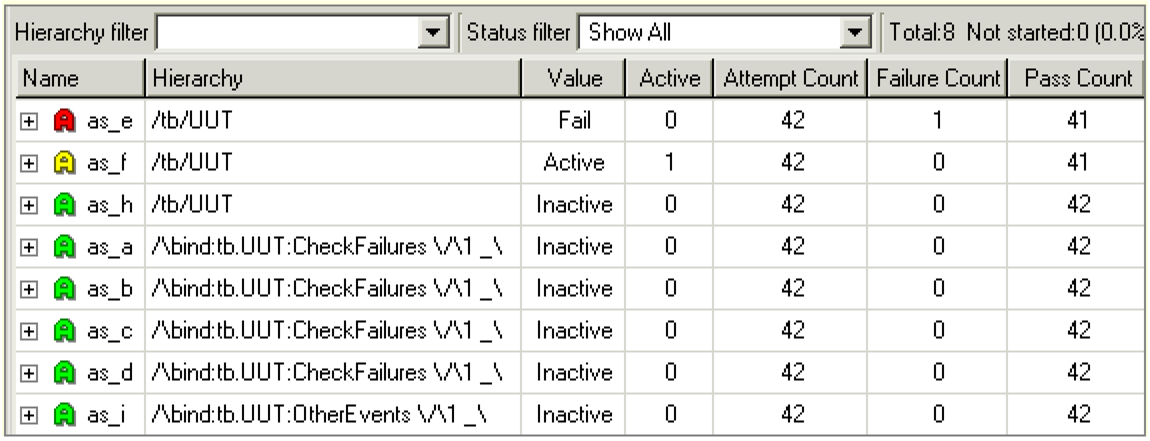
Assertion Viewer

The **Assertion Viewer** window shows two types of OVA, PSL, and SystemVerilog objects:

- assertions
- cover statements (covers for short)

Statistics gathered for these objects during simulation are presented on separate tabs. The **Assertions** tab displays statistics for assertions while **Covers** for the cover statements used in a design. The information provided by the viewer includes names, signals used in each assertion/cover, values, execution counts etc.

The Assertion Viewer can be started after the initialization of simulation by using the **Assertion Viewer** option from the **View** menu or the `view assert` macro command. The **Assertion Viewer** window is shown below.



Hierarchy filter | Status filter: Show All | Total:8 Not started:0 (0.0%

Name	Hierarchy	Value	Active	Attempt Count	Failure Count	Pass Count
as_e	/tb/UUT	Fail	0	42	1	41
as_f	/tb/UUT	Active	1	42	0	41
as_h	/tb/UUT	Inactive	0	42	0	42
as_a	/\bind:tb.UUT:CheckFailures \/\1 _\	Inactive	0	42	0	42
as_b	/\bind:tb.UUT:CheckFailures \/\1 _\	Inactive	0	42	0	42
as_c	/\bind:tb.UUT:CheckFailures \/\1 _\	Inactive	0	42	0	42
as_d	/\bind:tb.UUT:CheckFailures \/\1 _\	Inactive	0	42	0	42
as_i	/\bind:tb.UUT:OtherEvents \/\1 _\	Inactive	0	42	0	42

Figure 12 Assertion Viewer

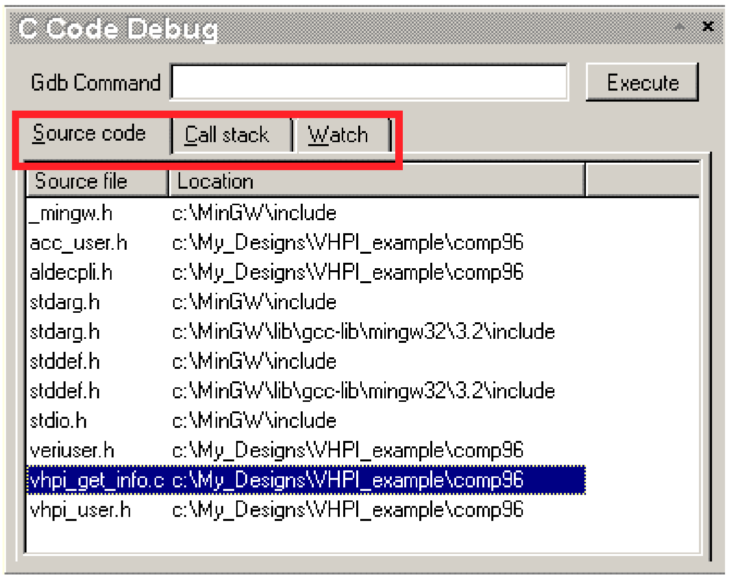
C Code Debug

The **C Code Debug** option is a feature that allows designers to debug PLI, VHPI, DPI, SystemC, or C/C++ source code with the open-source **gdb** debugger. During the debugging session, the simulator running in the batch mode (VSimSA) is loaded to the debugger.

The simulation of the design along with the VHPI/PLI/DPI application using the DLL file is performed within the VSimSA simulator through the built-in VHPI/PLI interface. The installation of Active-HDL provides the complete set of tools needed to compile and debug C/C++ code. Since the VSimSA simulator is loaded to the **gdb** environment, the VHPI/PLI application can be debugged by using debugging features built-in to the **HDL Editor** and **C Code Debug** windows while the simulator's output can be observed in the VSimSA window. The simulation results also can be observed in the Accelerated Waveform window

NOTES: In order to use the **C Code Debug** option, the license for the batch mode simulation (VSimSA) is required. This feature may be unavailable in some editions of Active-HDL. To enable the debug, -g3 should be added to the compilation options in the **C/C++ Configuration** dialog box.

The **C Code Debug** option is available in the **View** menu or the main toolbar.

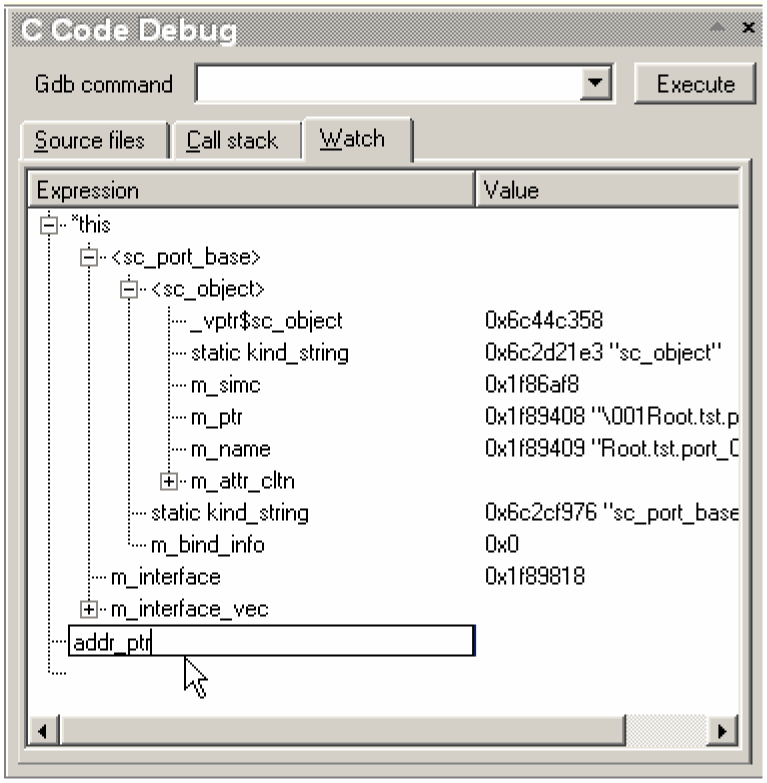


Source file	Location
_mingw.h	c:\MinGW\include
acc_user.h	c:\My_Designs\VHPI_example\comp96
aldecpli.h	c:\My_Designs\VHPI_example\comp96
stdarg.h	c:\MinGW\include
stdarg.h	c:\MinGW\lib\gcc-lib\mingw32\3.2\include
stddef.h	c:\MinGW\include
stddef.h	c:\MinGW\lib\gcc-lib\mingw32\3.2\include
stdio.h	c:\MinGW\include
veriuser.h	c:\My_Designs\VHPI_example\comp96
vhpi_get_info.c	c:\My_Designs\VHPI_example\comp96
vhpi_user.h	c:\My_Designs\VHPI_example\comp96

Figure 13 C Code Debug

The **C Code Debug** window provides the graphical interface that allows you to perform basic debugging operations. Before you start debugging, you need to specify settings for the debug session (To do so, go to the **Design** menu and choose the **Settings** option. Next, select the **C Code Debug Settings** category in the **Design Settings** dialog box). After you have specified all settings, you can initialize the debugging session by choosing the **Initialize C Code Debug** option from the **Simulation** menu or issue the cdebug command in the **Console** window. This starts the debugging session, which allows displaying a list of source code files in the **Source Code** tab, loading them into the HDL Editor window by double-clicking the selected file, inserting breakpoints (F9), and tracing the C/C++ source code by using the **Trace Into**, **Trace Over**, or **Trace Out** options. The **Run For** (F5) option allows you to continue the debugging session after a breakpoint has been reached. You can also issue the gdb command with parameters from the **Console** window to control the debugging session. The **Console** window also allows you to pass GDB commands to the debugger e.g. set a code breakpoint in a C source code (for example: gdb break 86). If you set the breakpoint by using the gdb command, the list of code breakpoints in the **Breakpoints** dialog box will be also updated.

Additionally, the **Call Stack** tab allows you to view detailed information on subprograms and selected call stack objects declared in the C/C++ code along with their values. In the **C Code Debug** window, you can also switch to the **Watch** tab that allows you to insert, edit, and remove variables being watched during the debugging session.



Expression	Value
*this	
<sc_port_base>	
<sc_object>	
_vptr$sc_object	0x6c44c358
static kind_string	0x6c2d21e3 "sc_object"
m_simc	0x1f86af8
m_ptr	0x1f89408 "\001Root.tst.p
m_name	0x1f89409 "Root.tst.port_C
m_attr_cltn	
static kind_string	0x6c2cf976 "sc_port_base
m_bind_info	0x0
m_interface	0x1f89818
m_interface_vec	
addr_ptr	

Figure 14 C Code Debug – Watch window