Course 9

HDE Based Debugging

Table of Contents

HDL Code Breakpoints

Introduction

An HDL code breakpoint can be set in HDL source files that are VHDL, Verilog, and SystemVerilog. A breakpoint can also be set in OVA and PSL code, for example in lines that contain assert or cover statements. The simulator also allows you to set a breakpoint on a design object, such as a VHDL signal, Verilog and SystemVerilog register or net, SystemC signal, or an EDIF net. This section describes using both **Source Code breakpoints** and **Signal breakpoints**.

Source Code Breakpoints

A source code breakpoint allows you to stop simulation at a certain location in the source code. A breakpoint must be set prior to executing the run command or using the Run option in the GUI. The simulator stops simulation when it is about to execute the line where a breakpoint has been set.

A breakpoint can be set in a source file only if debugging information was generated when that file was compiled. The compilers (**`acom, alog`**) generate debugging information if they are invoked with the **`-dbg`** argument. If you run compilation using the GUI, you need to make sure that the **Enable Debug** option is enabled in the **Design Settings** dialog box.

Setting Breakpoints

To set a code breakpoint in the HDL Editor, place the insertion point at the line where you want to halt execution and use the keyboard shortcut F9. The HDL Editor shows a breakpoint marker (a white hand) in front of each line where a breakpoint has been set. The marker does not disappear if you close the file and reopen it.

If you set breakpoints in several source files, they can be conveniently viewed in the HDL Code Breakpoints Editor as shown below. The editor allows you not only to view breakpoints but also to set the breakpoint scope and conditions related to hit counts. The **Breakpoints Editor** can be opened using **Breakpoints** from the **Simulation** menu.

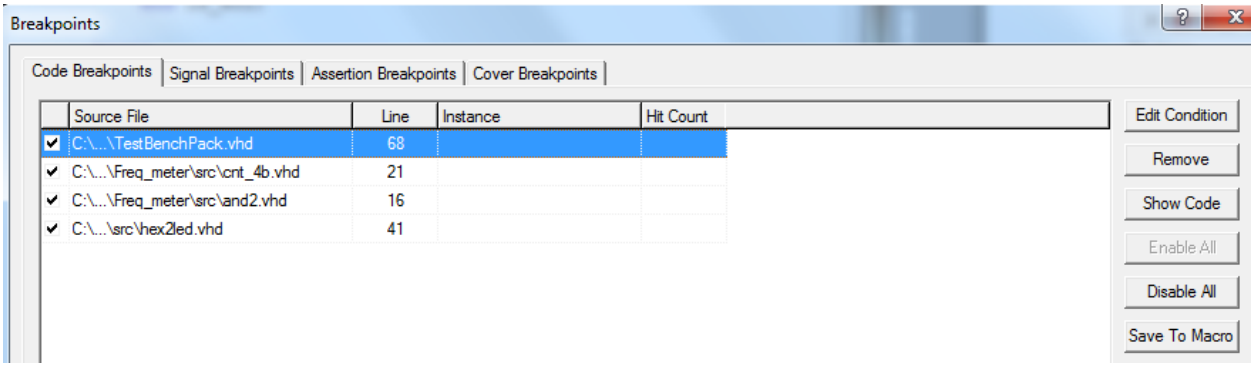

Figure 1 Breakpoint Editor – Code Breakpoints

Editing Breakpoint Properties

You can use the HDL Code **Breakpoints** window to edit the properties of a breakpoint after the breakpoint has been set. To do so, select the **Edit Condition** button on the right hand side of the editor. The following properties can be set:

Instance

Limits the scope of a breakpoint to the selected instance.

Hit Count

Specifies the hit count that will trigger the breakpoint. You can set the number of hits and any of the following options: *break when less, break when equal, break when greater*, and *break when multiple of*. The default option is break always. (The hit count cannot be set when *break always* is selected.)

Count by Instance

Checking this option affects breakpoints set on lines describing units instantiated more than once in the design. When selected, breakpoint hits are counted individually for each instance. (For example, when, a breakpoint is set on a line describing a multiplexor, and four such multiplexors are instantiated in the design, the simulator will keep track of four hit counters.)

Signal Breakpoints

Signal breakpoints allow you to stop simulation when a signal changes value or is written to. Three types of signal breakpoints are available:

- **Event breakpoint**; an event breakpoint is hit when the signal changes value.
- **Value breakpoint**; a value breakpoint is hit when the signal acquires a specific value.
- **Transaction breakpoint**; a transaction happens when a value is assigned to a signal even if the signal does not change its value. When a transaction breakpoint is set on signal vhdl_sig, the following assignment will trigger a transaction breakpoint.

  ```
  vhdl_sig <= '1'
  ```

  Even if the value of signal vhdl_sig equals '1' before the assignment. Transactions breakpoints can only be set on VHDL signals. Setting a transaction breakpoint on a Verilog object does not trigger an error. Such breakpoint is, however, equivalent to a breakpoint set on an event.

Signal breakpoints can be set on VHDL signals, Verilog nets and variables, SystemC signals, and EDIF nets. A signal breakpoint cannot be set on a VHDL variable or an object created dynamically during the execution of an HDL subprogram.

Setting Breakpoints

You can set a breakpoint by opening the **Breakpoints Editor** window using **Breakpoints** from **Simulation** menu. Select the **Signal Breakpoints** tab, and select the **Add Signals** button on the right hand side. The **Add Signals** window will pop up and you can specify which signals you would like to set a breakpoint for.

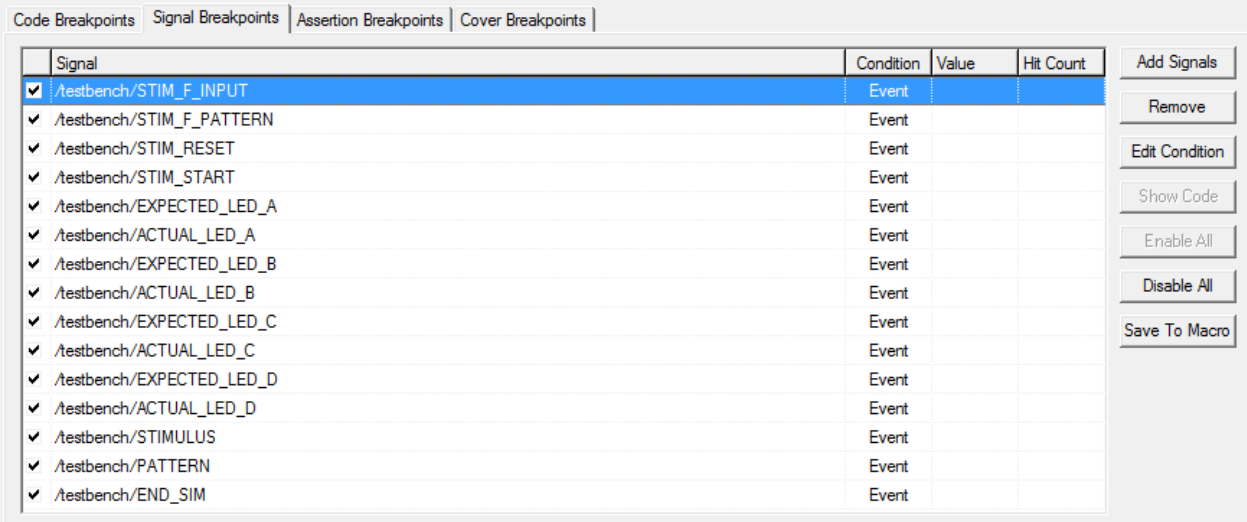

Figure 2 Breakpoint Editor – Signal Breakpoints

Editing Breakpoints Properties

You can change the breakpoint type from event to value or transaction by selecting the **Edit Condition** button. Additionally, you can use this dialog box to specify breakpoint hit count conditions.

Transaction

Sets a transaction breakpoint; a transaction happens when a value is assigned to a signal even if the signal does not change its value.

Event

Sets an event breakpoint; an event breakpoint is hit when the signal changes value.

Value

Sets a value breakpoint; a value breakpoint is hit when the signal acquires a specific value.

Step/Trace Simulation

Active-HDL allows you to trace through single HDL statements and analyze the model execution line by line or by processes, subprograms, and procedures. This is especially useful when debugging test benches and behavioral code.

When you trace through the source code, Active-HDL highlights the statement currently being executed with a yellow line. Required source files are loaded automatically in the HDL Editor.

To debug a specific location in the source code, set a code breakpoint and run the simulation. When the simulation encounters the breakpoint, the simulation is stopped and prints a message to the console “Breakpoint set on signal <signal>“. You can then step through the code.

Debugging Information

Simulation stepping is possible only if debugging information was generated during the compilation of the HDL code. The compilers (**`acom, alog`**) generate debugging information if they are invoked with the **`-dbg`** argument. If you run compilation using the GUI, you need to make sure that the **Enable Debug** option is enabled in the **Design Settings** dialog box.

A library and a simulated design can consist of any mixture of units compiled with and without debugging information. You can step only through those units that contain debugging information. If the debugging information is not available for any unit, the step command (or the Trace GUI option) will advance simulation as if the run command was invoked.

Tracing In, Out and Over

The GUI provides three options for stepping through the source code: *Trace In, Trace Out*, and *Trace Over*. The behavior of those options differs with regards to the execution of subprograms, i.e. VHDL procedures and functions and Verilog tasks and functions.

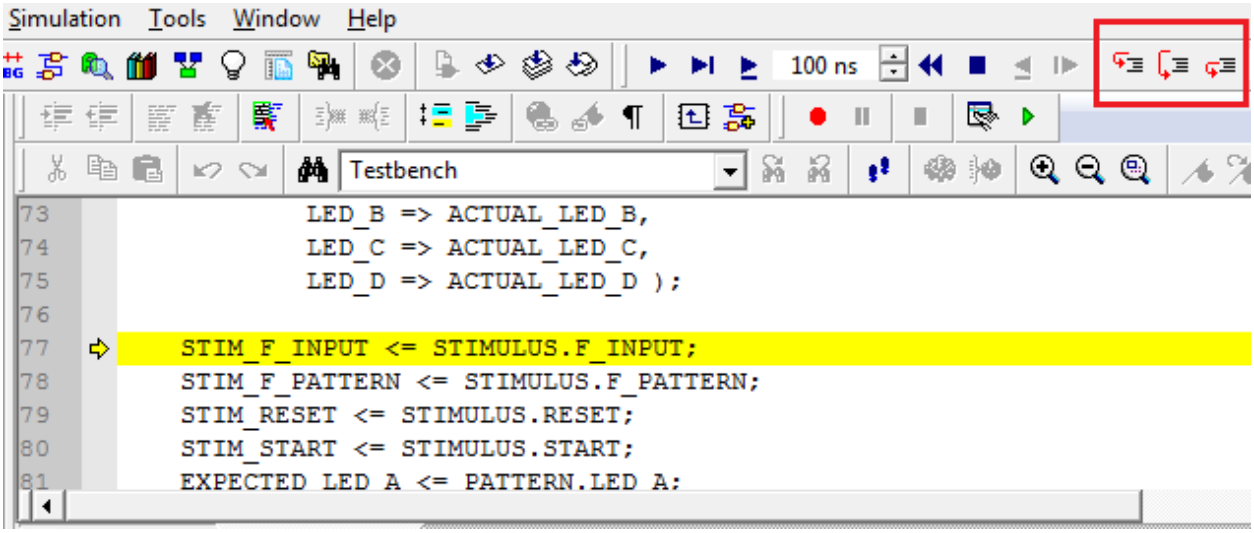

Figure 3 Breakpoints and step switches

Trace In executes a single HDL statement. If a subprogram call is invoked, the control is passed to the first statement in the subprogram body.

Trace Over steps over subprograms. If a subprogram call is encountered, the statements contained within the subprogram body are skipped.

Trace Out executes the current subprogram (function, procedure, or task). If subprograms are nested, the command completes the innermost subprogram only.

Determining the Current Execution Region

Each HDL unit may be instantiated multiple times in the design. For example, a VHDL entity **FF** may be instantiated as **/tb/uut/u1** and **/tb/uut/u2**. To check whether the simulator is executing code for instance **/tb/uut/u1** or **/tb/uut/u2**, use the **Call Stack** window. The **Call Stack** window displays the hierarchical name of the process currently being executed. To open the **Call Stack** window, go to **View | CallStack**.