Creating Testbench

Learn how to create a Testbench

Testbench

- **HDL Testbenches** are HDL programs that describes simulation input by using standard HDL language procedures**.**
- The **testbench** is a top level hierarchical model which instantiates the **Unit Under Test (UUT)**, drives it with a set of test vectors and compares the generated results with expected responses.

Testbench Concepts

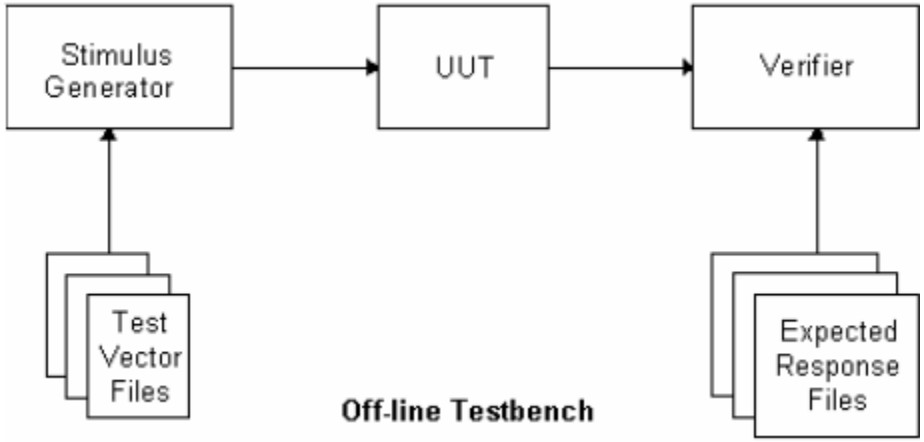
- A typical HDL testbench is composed of three main elements:

- **Stimulus Generator** - driving the UUT with certain signal conditions (correct and incorrect transactions, minimum and maximum delays, fault conditions, etc.).
- **Unit Under Test (UUT) -**representing the model undergoing verification.
- **Verifier -**automatically checks and reports any errors encountered during simulation. It also compares model responses with expected results.

Testbench

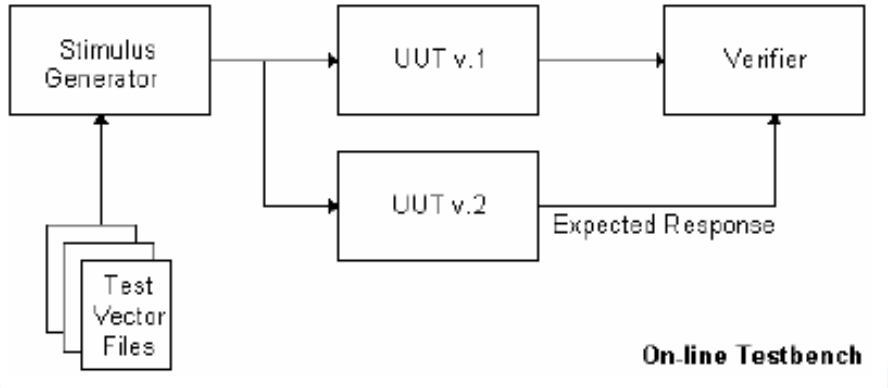
TB Types: Off-line Configuration

- In Off-line configuration, the **Stimulus Generator** and the **Verifier** read all data (test vectors, expected results) from the previously saved files. The **Stimulus Generator** reads all input signals from a file and provides clock processes. The **Verifier** compares the UUT responses with the expected results and reports any faulty behavior.



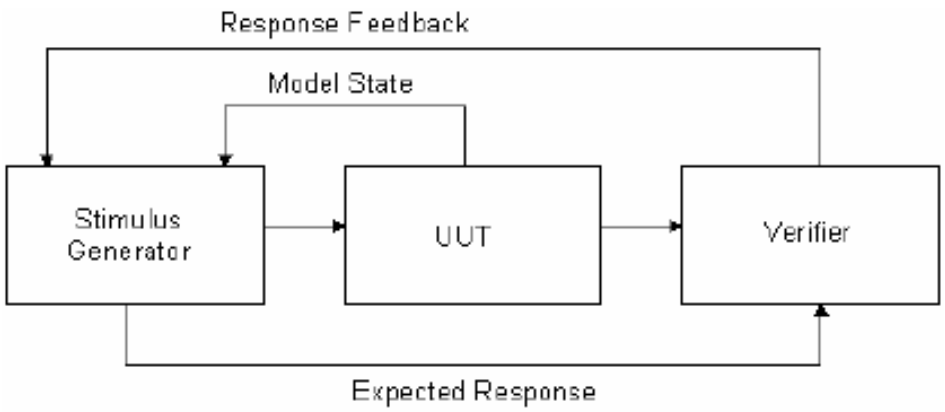
TB Types: On-line Configuration

- The **Stimulus Generator** provides the same input signals to each tested model. Thus, the response of all models are simultaneously generated without any user interaction such as exchanging the components.
- The **Verifier** operation is much simpler than in the off-line configuration because it only gathers simulation results from each model and compares them, detecting any differences and deciding whether to continue a simulation or not.



TB Types: Adaptive Configuration

- The **Stimulus Generator** uses high-level abstraction techniques to adapt test vectors to the changing conditions and responses of a tested model. As a result, test vectors are generated in response to feedback from the UUT and the **Verifier**.



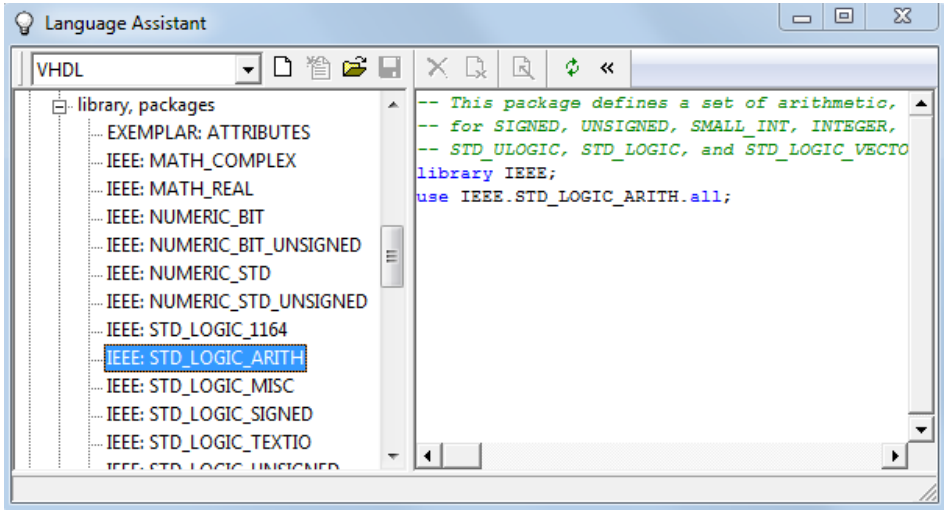
Adaptive Testbench

Testbench Example

```
architecture TB_ARCHITECTURE of counter_tb is

    component counter
    port(
        CLK : in STD_LOGIC;
        RESET : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR(3 downto 0) );
    end component;
    signal CLK : STD_LOGIC;
    signal RESET : STD_LOGIC;
    signal Q : STD_LOGIC_VECTOR(3 downto 0);
begin
    UUT : counter
        port map (
            CLK => CLK,
            RESET => RESET,
            Q => Q
        );
STIMULUS: process
begin
    CLK <= '1';

    wait;
end process;
end TB_ARCHITECTURE;
```

Writing a Simple Testbench

- For the *counter* created in the *Bottom-Up Design Methodology*, we will create a simple testbench.



- Open the design
- Double-click the ***counter.vhd*** file
- Open the **Language Assistant** window
- Select **Languages templates | library, packages**
- Select the **IEEE:STD_LOGIC_ARITH** library

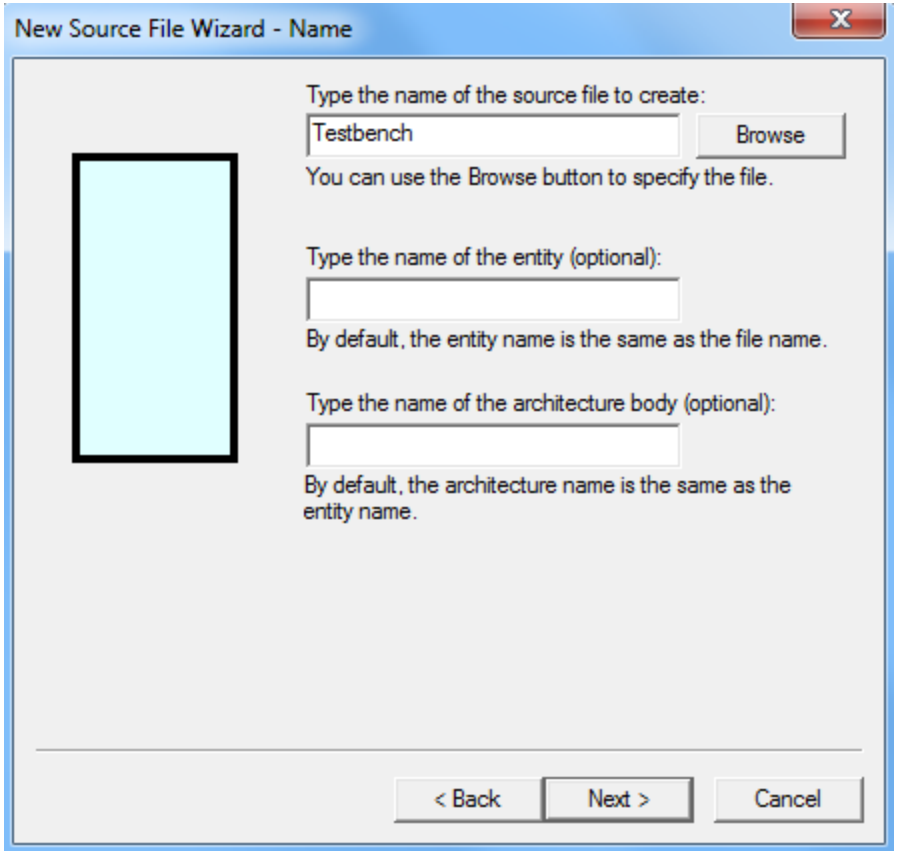
Writing a Simple Testbench (cont.)

- Drag the selected library to the **HDL Editor** window
- Drop it under the IEEE library clause
- Repeat this process for **IEEE.STD_LOGIC_UNSIGNED**
- Compile the design

```
25  library IEEE;
26  use IEEE.STD_LOGIC_1164.all;
27  --This package defines a set of arithmetic, conversion, and comp
28  --  for SIGNED, UNSIGNED, SMALL_INT, INTEGER,
29  --  STD_ULOGIC, STD_LOGIC, and STD_LOGIC_VECTOR.
30  library IEEE;
31  use IEEE.STD_LOGIC_ARITH.all;
32  --This package defines a set of unsigned arithmetic, conversion,
33  -- and comparision functions for STD_LOGIC_VECTOR.
34  library IEEE;
35  use IEEE.STD_LOGIC_UNSIGNED.all;
36
37  entity counter is
38       port(
39            CLK : in STD_LOGIC;
40            RESET : in STD_LOGIC;
41            Q : out STD_LOGIC_VECTOR(3 downto 0)
42            );
43  end counter;
```

Writing a Simple Testbench (cont.)

- Double-click **Add New File** in the **Design Browser**
- Choose the **Wizards** tab and select **VHDL Source Code**
- Type *Testbench* as the name of the file
- Continue through to **Finish** so you do not need to specify any ports.



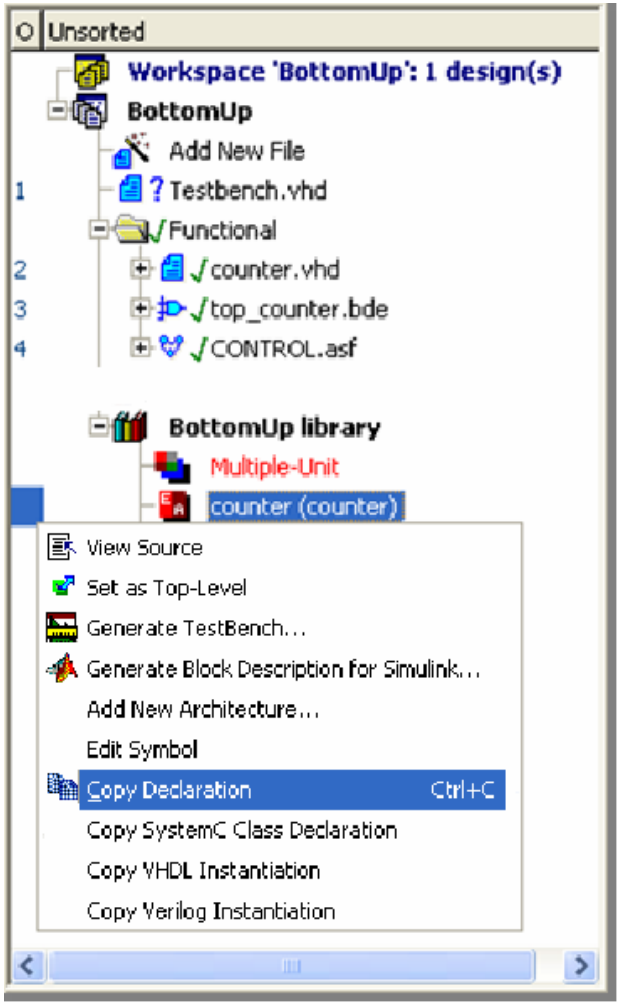
Writing a Simple Testbench (cont.)

- The HDL Editor will open the *Testbench* file with the following contents:

```
15  ------------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ------------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {Testbench} architecture {Testbench}}
24
25  library IEEE;
26  use IEEE.STD_LOGIC_1164.all;
27
28  entity Testbench is
29  end Testbench;
30
31  --}} End of automatically maintained section
32
33  architecture Testbench of Testbench is
34  begin
35
36       -- enter your statements here --
37
38  end Testbench;
```

design flow | testbench.vhd

- To insert the UUT Component Declaration:
  - Copy the *Counter* declaration from the working library and paste the declaration about the *begin* keyword.



Writing a Simple Testbench (cont.)

- The *Testbench* file with the contents should look as follows:

```
25  library IEEE;
26  use IEEE.STD_LOGIC_1164.all;
27
28  entity Testbench is
29  end Testbench;
30
31  --}} End of automatically maintained section
32
33  architecture Testbench of Testbench is
34      -- Component declaration of the "counter(counter)" unit defined in
35      -- file: "c:\My_Designs\BottomUp\BottomUp/src/Functional/counter.vhd"
36      component counter
37      port(
38          CLK : in std_logic;
39          RESET : in std_logic;
40          Q : out std_logic_vector(3 downto 0));
41      end component;
42      for all: counter use entity WORK.counter(counter);
43  begin
44
```

design flow | testbench.v...

- You will need to create a clocking process to force the UUT model with the appropriate pattern.
- You will also need to add signals to bind the testbench with the UUT model.

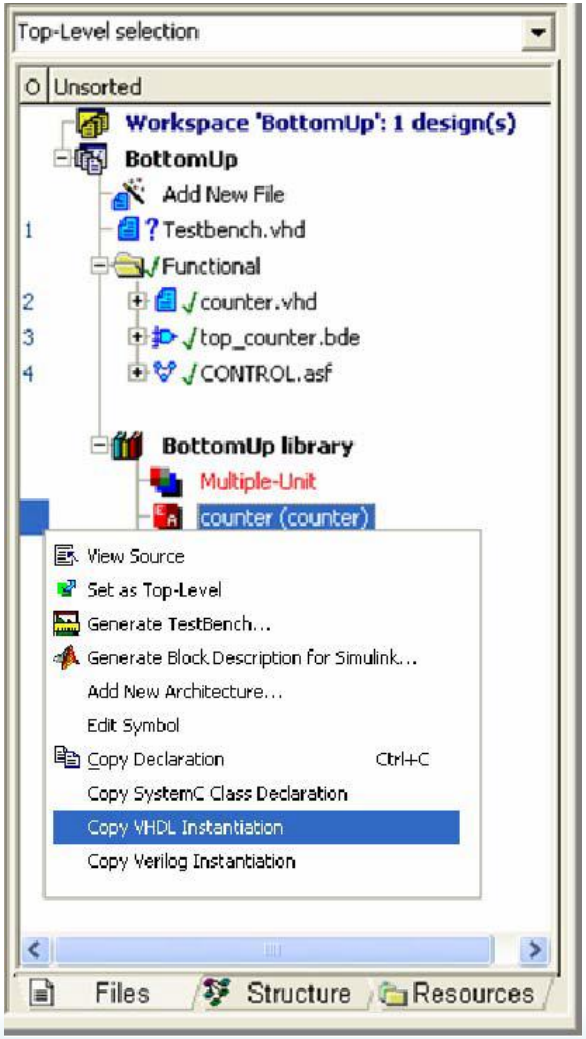
Writing a Simple Testbench (cont.)

- Below the *Counter* component, declare the following signals:
- **signal CLK: std_logic :=‘0’;**
- **signal RESET: std_logic;**
- **signal Q: std_logic_vector(3 downto0);**

- To create a clocking process, you need a variable that will stop the simulation. Declare the following below the signal declarations:
- **shared variable END_SIM : boolean:=FALSE;**

Writing a Simple Testbench (cont.)

- The next step is to map the **Counter** component with the declared signals.
- Expand the **BottomUp** library in the **Design Browser** and select the **counter** component.
- Right-click and select **Copy VHDL Instantiation**
- Paste the instantiation under the *begin* keyword and change the label name from **Label1** to **UUT**



Writing a Simple Testbench (cont.)

- To define the clocking process, insert the following below the port mapping section:

```
CLK_IN: process
begin
if END_SIM = false then
CLK <= '1';wait for 10 ns;
CLK <= '0';wait for 10 ns;
else wait;
end if;
end process;
```

Writing a Simple Testbench (cont.)

- The following is a reset process that resets the *Counter* output 0ns and stops the simulation at 250ns.

```
RESET_IN: process
begin
RESET <= '1';
wait for 10 ns;
RESET <= '0';
wait for 250 ns;
END_SIM := TRUE;
wait;
end process;
```

- The simulation will stop because the END_SIM value is set to TRUE

Writing a Simple Testbench (cont.)

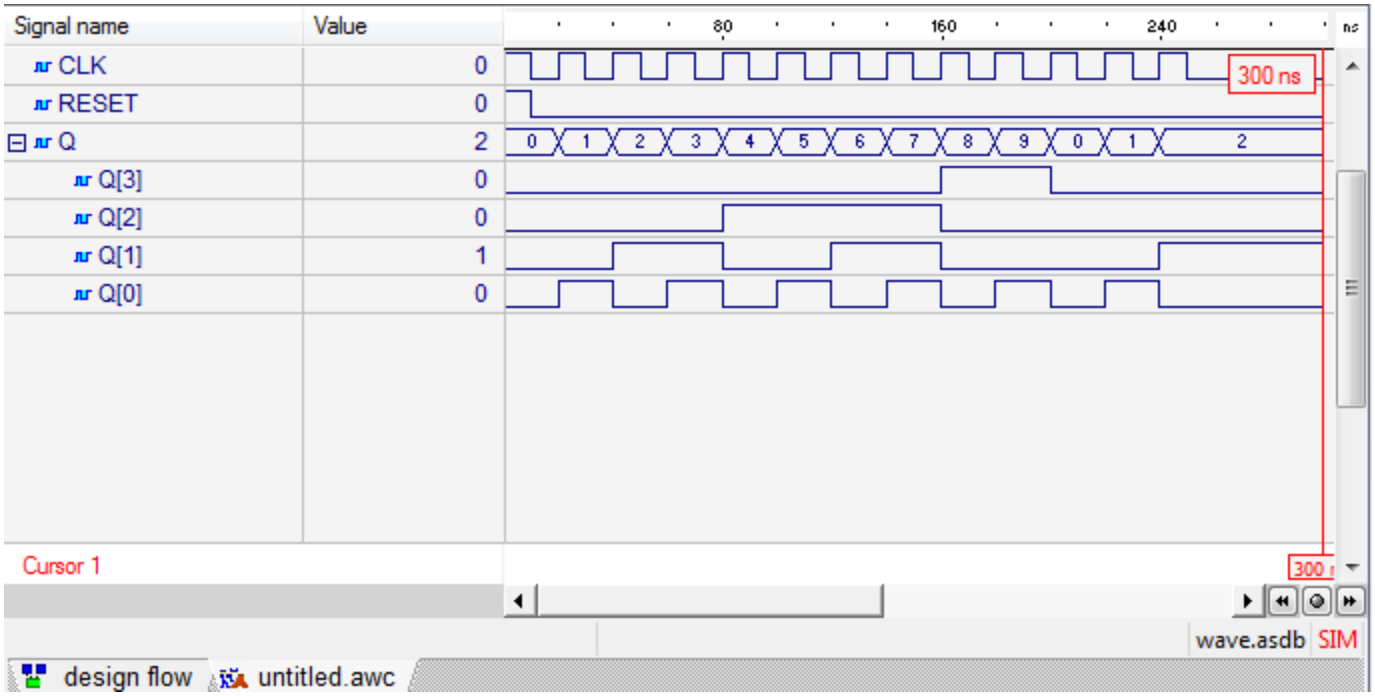
- The complete *testbench* architecture should look as follows:

```
architecture Testbench of Testbench is
        -- Component declaration of the "counter(counter)" unit defined in
    -- file: "./src/counter.vhd"
    component counter
    port(
        CLK : in STD_LOGIC;
        RESET : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    for all: counter use entity work.counter(counter);

    signal CLK : std_logic := '0';
    signal RESET : std_logic;
    signal Q : std_logic_vector(3 downto 0);

    shared variable END SIM : boolean := FALSE;
```

```
begin

    UUT : counter
    port map(
        CLK => CLK,
        RESET => RESET,
        Q => Q
    );
     -- enter your statements here --
    CLK_IN : process
    begin
        if END_SIM = FALSE then
            CLK <= '1'; wait for 10ns;
            CLK <= '0'; wait for 10ns;
        else wait;
        end if;
    end process;

    RESET_IN : process
    begin
        RESET <= '1';
        wait for 10ns;
        RESET <= '0';
        wait for 250ns;
        END_SIM := TRUE;
        wait;
    end process;

end Testbench;
```

Writing a Simple Testbench

- Compile the Testbench file and set it as a top-level unit. Simulate it for 300ns. The results are shown below:



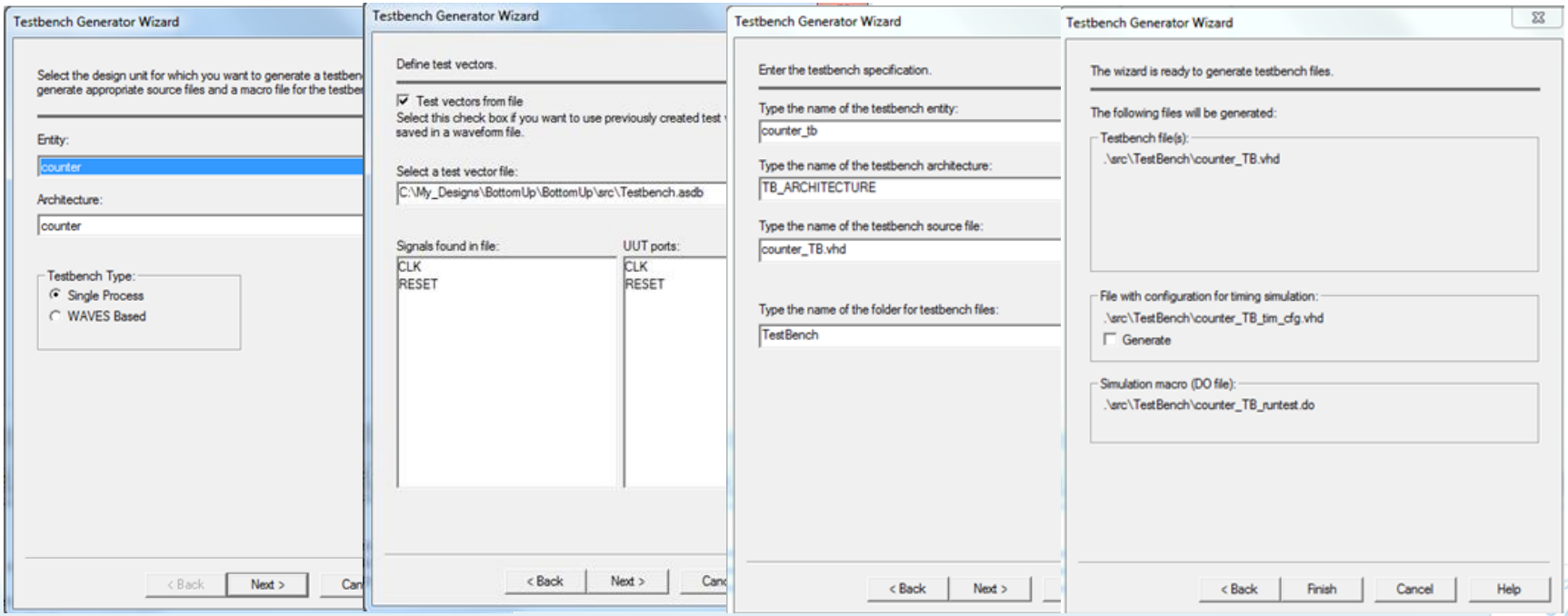
Using Testbench Generator Wizard

- Active-HDL comes with a **Testbench Generator Wizard** that automates the process of the testbench creation. You can create skeleton testbench files or completely functional testbenches from previously saved waveform files.

- There are two ways to start the wizard:
- **Tools | Generate Testbench**
- Right-click the **Testbench** component in the library and select **Generate Testbench**.

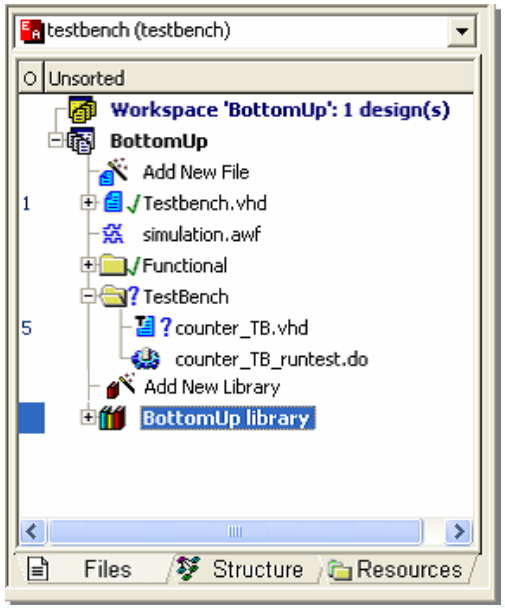
Testbench Generator Wizard (cont.)

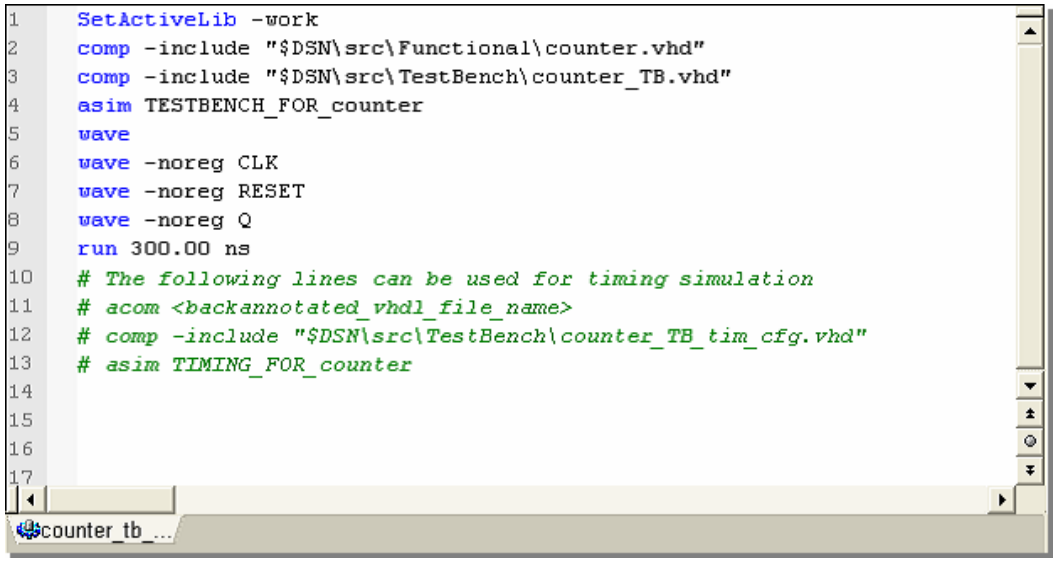
- Select the *Counter* architecture and **Single Process**
- In the next window, check the **Test Vectors from file** option and browse for the file of the previously generated waveform.
- Continue through the wizard accepting the default options and press **Finish**.



Testbench Generator Wizard (cont.)

- The testbench wizard has created a folder *Testbench* in your **Design Browser**. It contains two files:
- *Counter_TB.vhd:* the testbench file
- *Counter_TB_runtest.do:* The macro command file that compiles and executes the testbench for simulation





```
SetActiveLib -work
comp -include "$DSN\src\Functional\counter.vhd"
comp -include "$DSN\src\TestBench\counter_TB.vhd"
asim TESTBENCH_FOR_counter
wave
wave -noreg CLK
wave -noreg RESET
wave -noreg Q
run 300.00 ns
# The following lines can be used for timing simulation
# acom <backannotated_vhdl_file_name>
# comp -include "$DSN\src\TestBench\counter_TB_tim_cfg.vhd"
# asim TIMING_FOR_counter
```

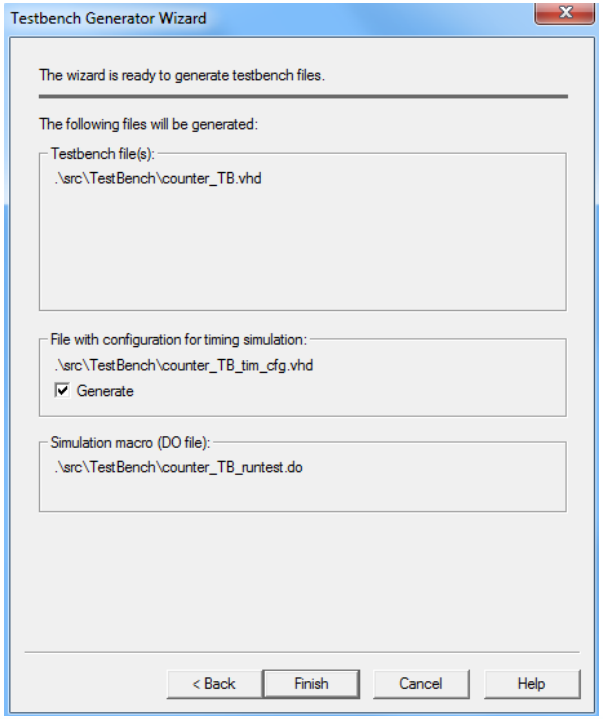
Testbench Generator Wizard (cont.)

- In the complete *Counter_TB.vhd* file, you can observe that the clocking and reset processes have been replaced by a *STIMULUS* process.
- It contains a sequence of assignments and *wait* instructions.

```
48  begin
49
50      -- Unit Under Test port map
51      UUT : counter
52          port map (
53              CLK => CLK,
54              RESET => RESET,
55              Q => Q
56          );
57
58      --Below VHDL code is an inserted
59      --User can modify it ....
60
61  STIMULUS: process
62  begin  -- of stimulus process
63  --wait for <time to next event>; -- ·
64
65      CLK <= '1';
66      RESET <= '1';
67      wait for 10 ns; --0 fs
68      CLK <= '0';
69      RESET <= '0';
70      wait for 10 ns; --10 ns
71      CLK <= '1';
72      wait for 10 ns; --20 ns
73      CLK <= '0';
74      wait for 10 ns; --30 ns
75      CLK <= '1';
76      wait for 10 ns; --40 ns
77      CLK <= '0';
78      wait for 10 ns; --50 ns
```

Testbench for a Timing Simulation

- You can use the same waveform file to create a testbench for a timing simulation. To do so, use the same testbench wizard as before. The only difference, is in the last window, select the **Generate** check box. The Testbench folder will now contain three files:



- *Counter_TB.vhd*
- *Counter_TB_runtest.do*
- *Counter_TB_tim_cfg.vhd*

Timing Simulation (cont.)

- The Timing Simulation can be performed after the synthesis and Place&Route processes are finished. They can generate a VHDL netlist file for the timing simulation.
- To enable the timing simulation, change the contents of the timing configuration testbench to reflect the name of the VHDL netlist.

```
23  configuration TIMING_FOR_counter of counter_tb is
24      for TB_ARCHITECTURE
25          for UUT : counter
26  --
27  -- The user should replace :
28  -- ENTITY_NAME with an  entity name from a backannotated VHDL file,
29  -- ARCH_NAME   with an architecture name from a backannotated VHDL file,
30  -- and uncomment the line below
31  --          use entity work.ENTITY_NAME (ARCH_NAME);
32          end for;
33      end for;
34  end TIMING_FOR_counter;
```

- You only need to uncomment the *use* line and change the architecture and entity names accordingly.

Timing Simulation (cont.)

- The *Counter_TB_runtest.do* requires some modifications to run the timing simulation.
- Uncomment the last three lines in the file and move them to the top of the file.
- Comment the three lines beneath those lines
- Replace <backannotated_VHDL_file_name> with the timing netlist file name.

```
1   SetActiveLib -work
2   comp -include "$DSN\src\counter.vhd"
3   comp -include "$DSN\src\TestBench\counter_TB.vhd"
4   asim TESTBENCH_FOR_counter
5   wave
6   wave -noreg CLK
7   wave -noreg RESET
8   wave -noreg Q
9   run 300.00 ns
10  # The following lines can be used for timing simulation
11  # acom <backannotated_vhdl_file_name>
12  # comp -include "$DSN\src\TestBench\counter_TB_tim_cfg.vhd"
13  # asim TIMING_FOR_counter
```

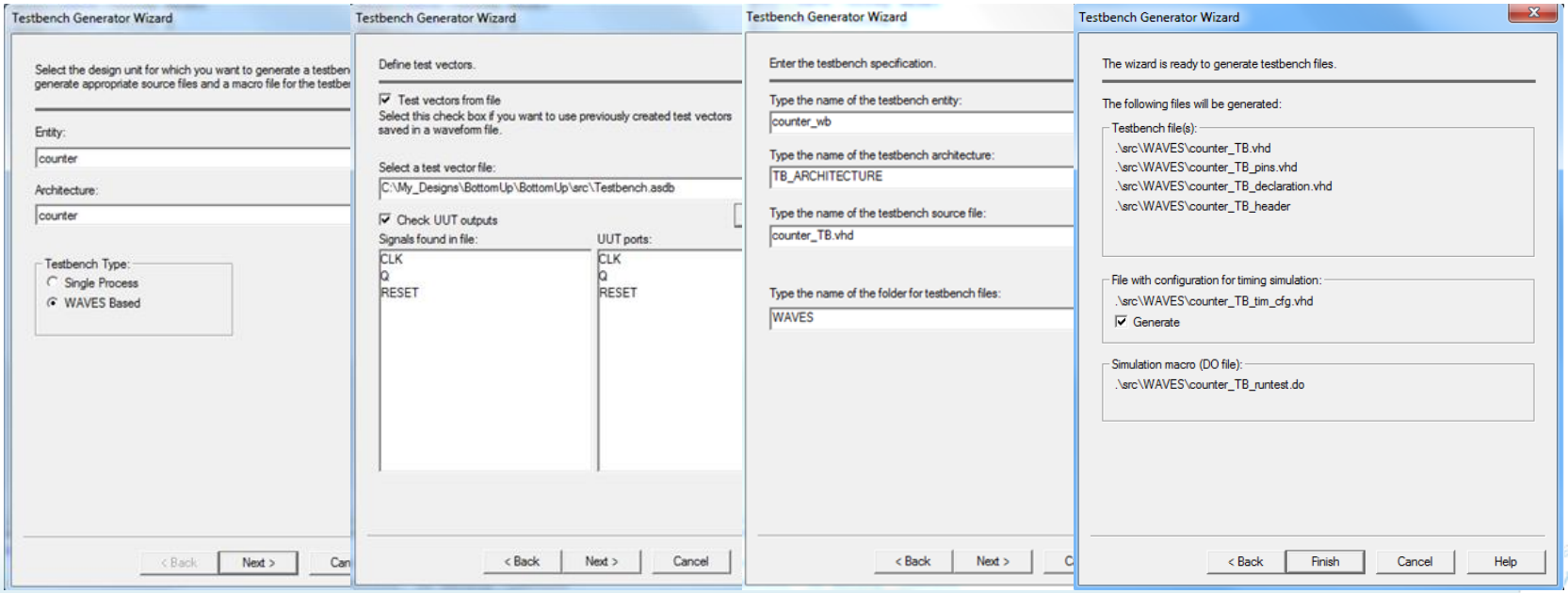
```
1   SetActiveLib -work
2   # The following lines can be used for timing simulation
3   acom <backannotated_vhdl_file_name>
4   comp -include "$DSN\src\TestBench\counter_TB_tim_cfg.vhd"
5   asim TIMING_FOR_counter
6   #comp -include "$DSN\src\counter.vhd"
7   #comp -include "$DSN\src\TestBench\counter_TB.vhd"
8   #asim TESTBENCH_FOR_counter
9   wave
10  wave -noreg CLK
11  wave -noreg RESET
12  wave -noreg Q
13  run 300.00 ns
```

WAVES Testbench

- The WAVES-based testbench simultaneously drives the inputs and compares the output response with a previously saved pattern.
- During simulation, the test vectors (stimulus and output response) are taken from the test vector file (*.VEC).
- The Test Bench Generator Wizard generates the test vector file (*.VEC) from a waveform file (*.AWC) created using the Waveform Editor.
- Typically, a WAVES testbench is used to create a testbench during the design stage and during design development.
- The second possibility is to use WAVES testbenches to compare timing and functional simulation results.
- The WAVES testbench is compliant with the IEEE Standard and can also be used in the electrical test environment.

WAVES Testbench (cont.)

- Select the *Counter* architecture and **Single Process**
- In the next window, check the **Test Vectors from file** option and browse for the file of the previously generated waveform.
- Continue through the wizard accepting the default options
- Check the **Generate** box and click **Finish**.

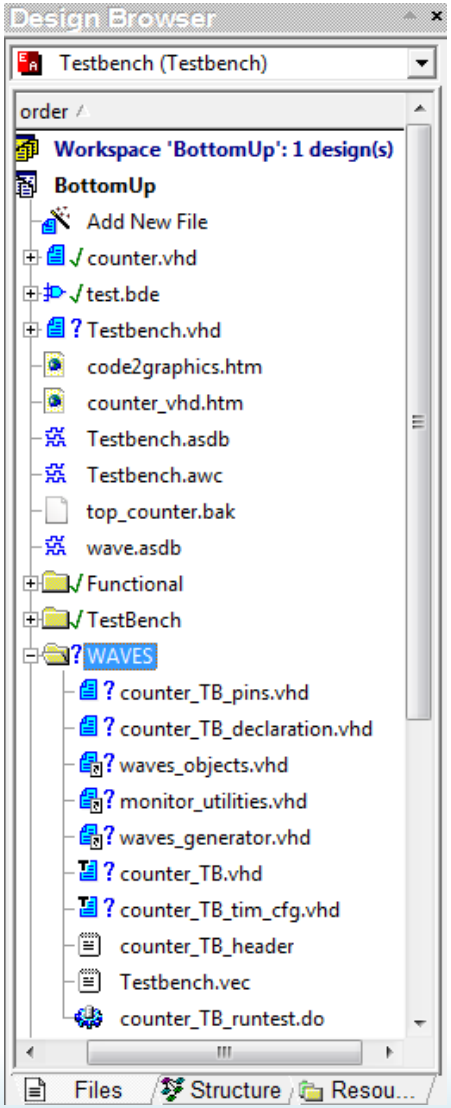


WAVES Testbench (cont.)

- The testbench wizard created a ***WAVES*** folder containing:
  - **Counter_TB_pins.vhd**-package contains declaration of enumerated type with the UUT ports names
  - **Counter_TB_declaration.vhd**-package contains declaration of all used in testbench constants and types
  - **Counter_TB_header**-TEXT header file contains the main information about WAVES testbench files and objects
  - **waves_objects.vhd**-standard WAVES package
  - **monitor_utilities.vhd**-package contains procedures for monitoring and comparison the UUT outputs
  - **waves_generator.vhd**-package contains procedure for reading test vectors from the file and generating stimulus and output patterns
  - **simulation.vec**-file with test vectors
  - **Counter_TB.vhd**-top level testbench entity
  - **Counter_TB_tim_cfg.vhd**-top level testbench configuration for timing simulation
  - **Counter_TB_runtest.do**-macro for running compilation files, initialization of simulation, waveform creation and simulation of the whole testbench

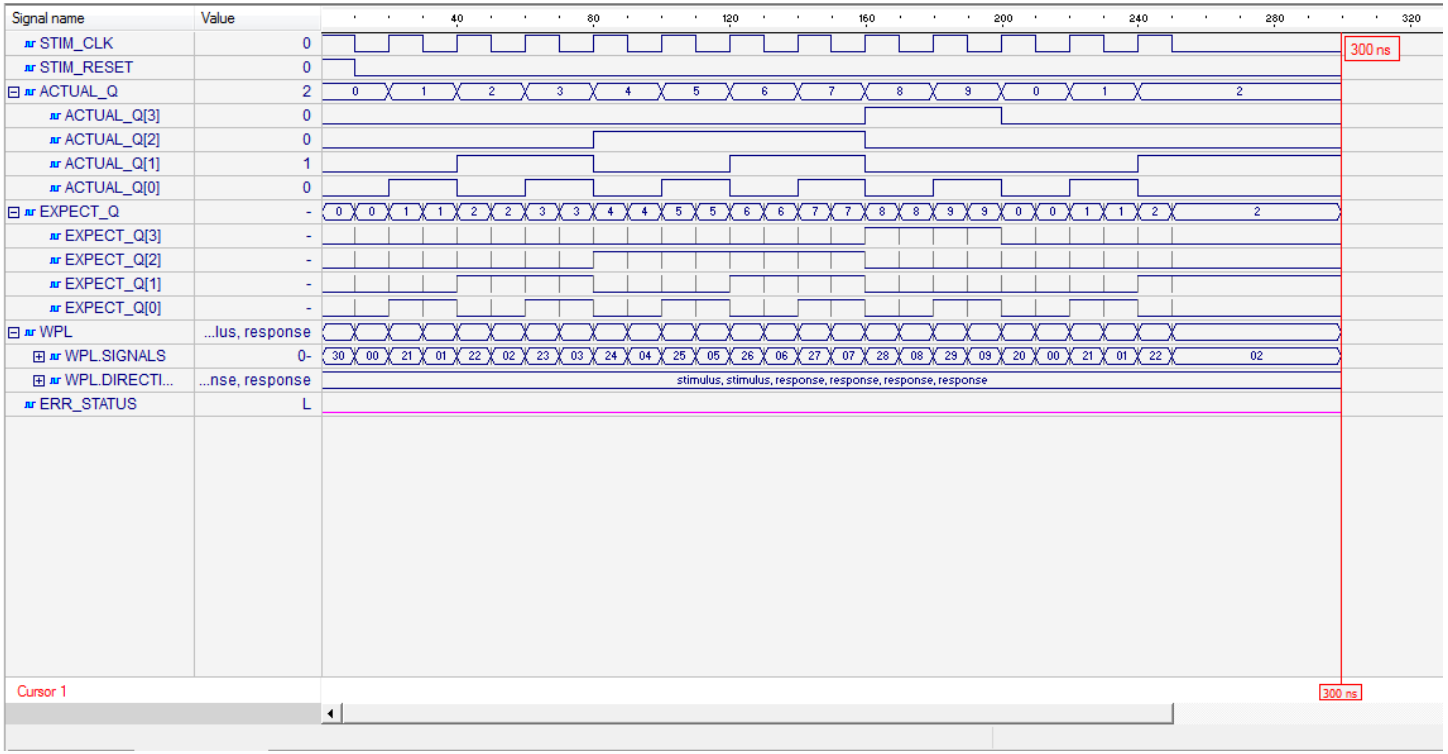
WAVES Testbench (cont.)

- Files with the icon are not generated by the wizard. These files are constant and remain the same for each WAVES testbench.
- The rest of the files are specific to the design. The two most important are:
- ***_declaration.vhd**
  - The user can change some objects' declaration important for simulation
- ***_runtest.do**
  - The simulation macro can be customized by the user



WAVES Testbench (cont.)

- To simulate the design with a WAVES testbench, run the *Counter_TB_runtest.do* macro file. The simulation will look as follows:



WAVES Testbench (cont.)

- The WAVES testbench compares the output response with a previously saved pattern within the ‘comparison window’
- Default size of this window is only 1 ps narrower than the test vector step time.
- To display simulation results you can use the Zoom to fit toolbar button.