Mixed-Language Simulation with Lattice IP Designs Using Active-HDL

February 2007 Technical Note TN1146

Introduction

Lattice provides pre-tested, reusable functions that can be easily integrated into designs; thereby, allowing the designer to focus on their unique system architecture. These Intellectual Property (IP) cores eliminate the need to recreate many industry-standard functions. These IP Cores are optimized for Lattice Field Programmable Gate Array (FPGA) architectures, which results in fast, compact cores that utilize the latest Lattice architectures to their fullest.

The IPexpress™ design flow enables users to fully parameterize IP in real-time. IPexpress generates the IP in the Verilog hardware description language (HDL). The designer can then instantiate the user-configured IP and complete the design process, including simulation and bitstream generation. For those designers who prefer a VHDL environment for simulation, the use of a single-kernel mixed-language simulator with Lattice FPGA device library support is required. EDA tools such as ModelSim® from Mentor Graphics® and Active-HDL® from Aldec® provide this feature.

Verilog Simulation Flow

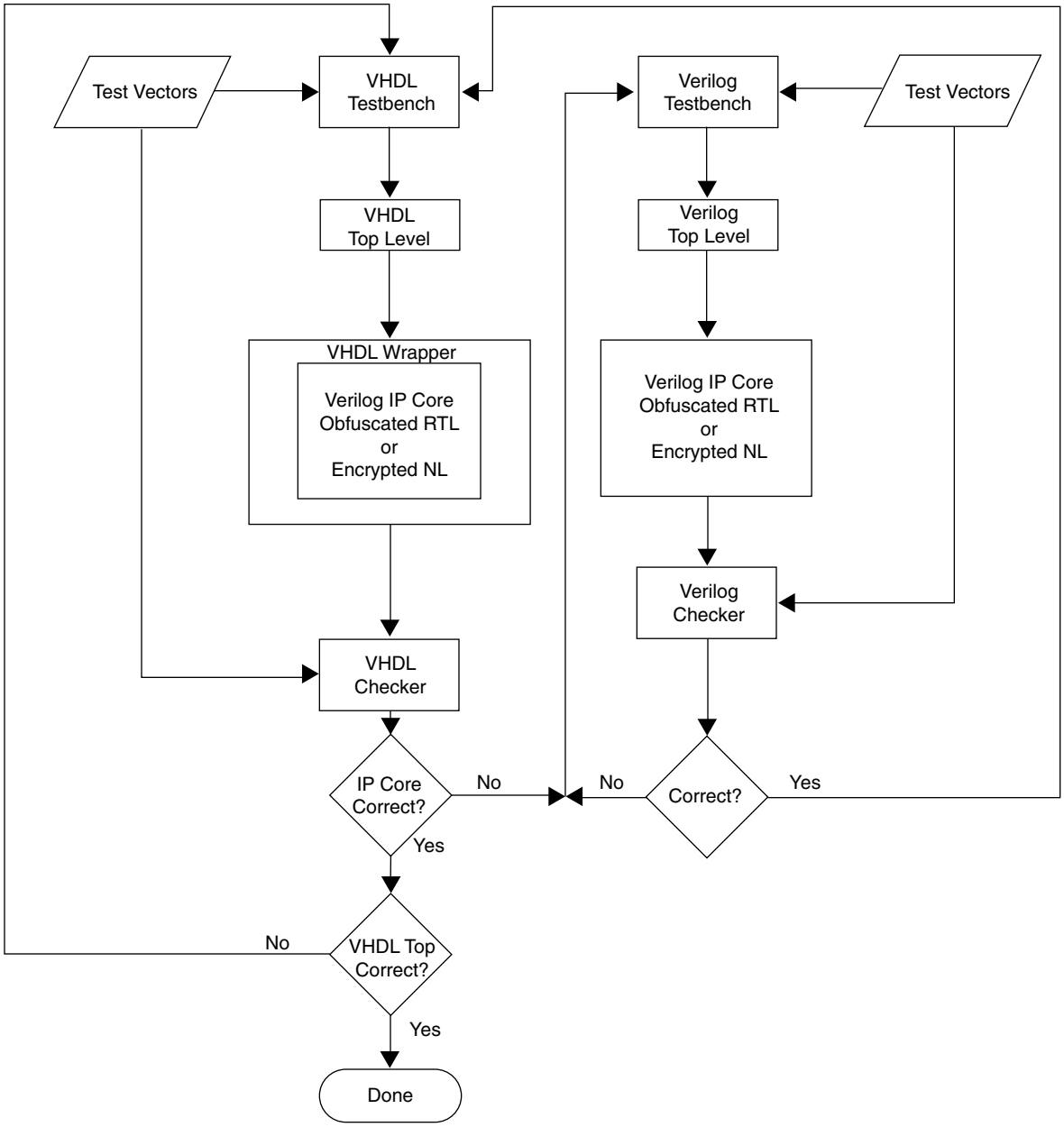
Lattice IP cores are distributed using an obfuscated Verilog RTL simulation model and an encrypted Verilog gate level model. In Verilog-based designs, the IP cores are directly instantiated in the top-level of the design as modules. A Verilog testbench is provided for all Lattice IPs. The testbench is used to verify the correct operation of the IP prior to use in a design.

Mixed Language Simulation Flow

If the FPGA application is being developed in VHDL, the IP must be instantiated as a component. The entire FPGA application can then be simulated as though the IP were a VDHL design. This process is shown in Figure 1.

Once the core has been generated by IPexpress, a VHDL wrapper must be created for instantiating the obfuscated Verilog RTL simulation model. In Active-HDL, a designer creates a Entity/Architecture pair complete with the component declaration and component instantiation using the Block Diagram Editor Tool. This tool reads in the Verilog top-level module port list and creates a symbol. The designer then connects I/O ports and signals to the symbol. After selecting VHDL as the target language, the Entity/Architecture pair is push-button generated. Entity/Architecture pair and instantiates the component within to create the VHDL wrapper. The newly created VHDL wrapper can then be instantiated in a VHDL testbench or in a top level VHDL design. The design may now be compiled for simulation using vcom for the VHDL design units and vlog for the Verilog modules.

Figure 1. Mixed Mode Simulation Flow for IP Express Generated IP Cores



Examples

The example below illustrates a VHDL instantiation of a Lattice DDR Verilog core generated by IPexpress. The IP core can be created and simulated in the Active-HDL environment. The following examples assume that the user is experienced in using the ispLEVER®, IPexpress and Active-HDL tool flows to implement a Lattice Semiconductor IP. It also assumes that the user has used IPexpress to download and install the Lattice Semiconductor DDR SDRAM Controller v6.2 in the directory C:\DDR_ML_Example.

Using Active-HDL® to Instantiate a Verilog RTL Design Into a VHDL Project

This example assumes the following:

- The user has installed Lattice ispLEVER 6.1 SP1 in the directory C:/ispTOOLS6_1.
- The user has already utilized the Lattice IPexpress tool to install the DDR SDRAM Controller v 6.2 in the directory C:/DDR_ML_Example.
- The targeted file name is ddr_sdram.
- The targeted device is the LFECP33E-4F484C.
- The user has already installed Aldec Active-HDL 7.2 PE in the directory C:\Active-HDL 7.2.

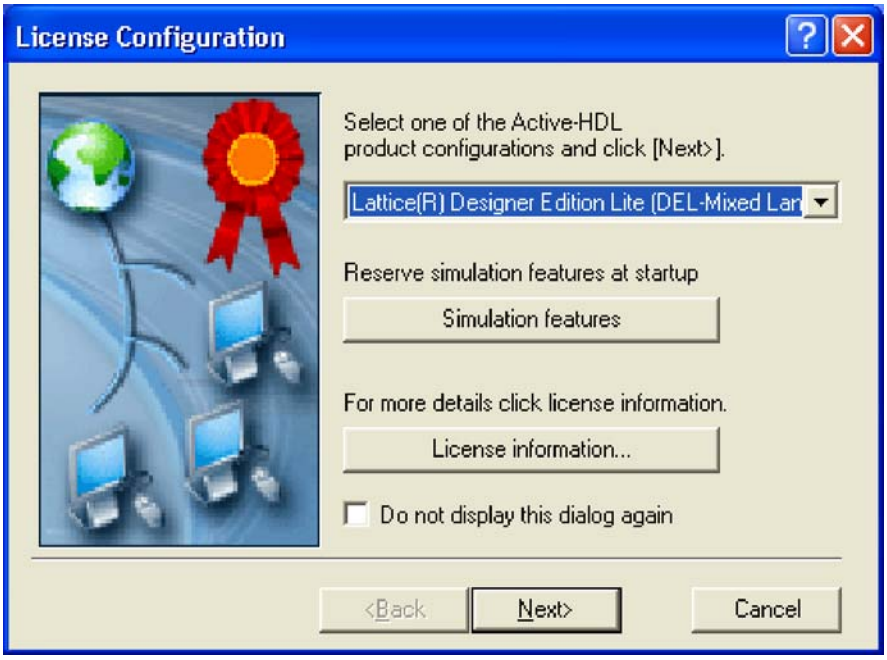
Create the directory **C:\DDR_ML_Example\ddr_p_eval\ddr_sdram\sim\aldec** using Windows Explorer.

Launch **Active-HDL**.

Click on **Start->Programs->Active-HDL 7.2**.

The Active-HDL License Configuration screen appears as shown in Figure 2.

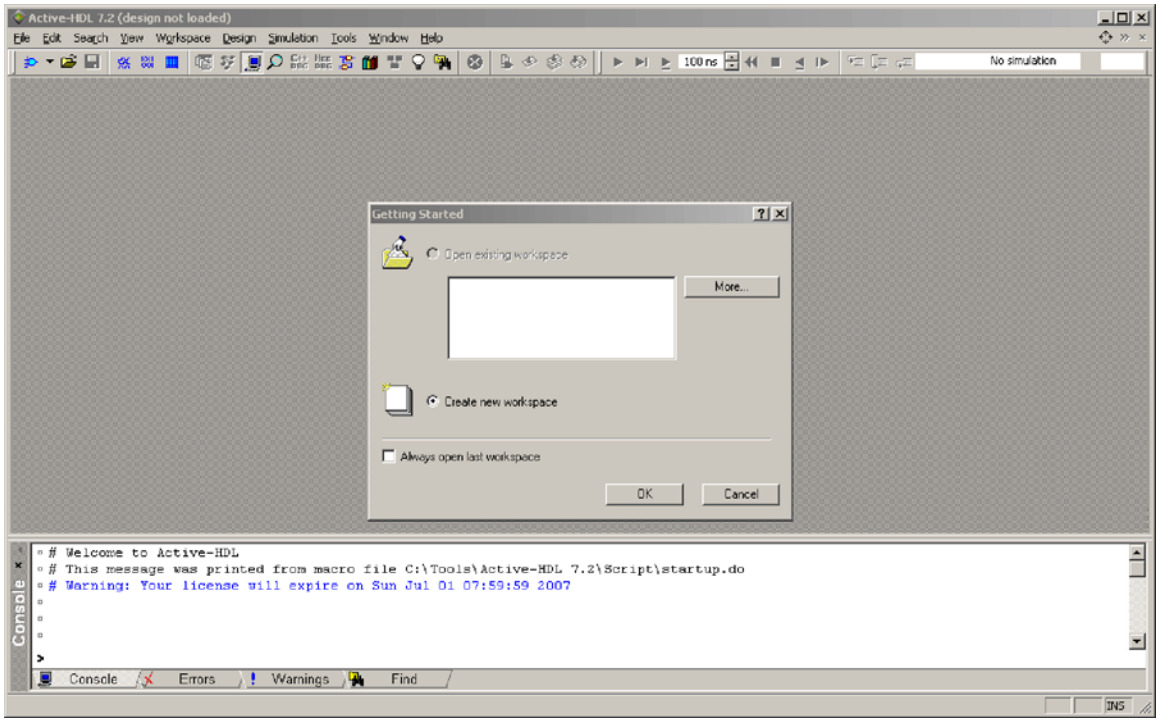
Figure 2. Active-HDL License Configuration Screen



Click on **Next** to launch Active-HDL.

The Active-HDL Getting Started screen appears as shown in Figure 3.

Figure 3. Active-HDL Getting Started Screen

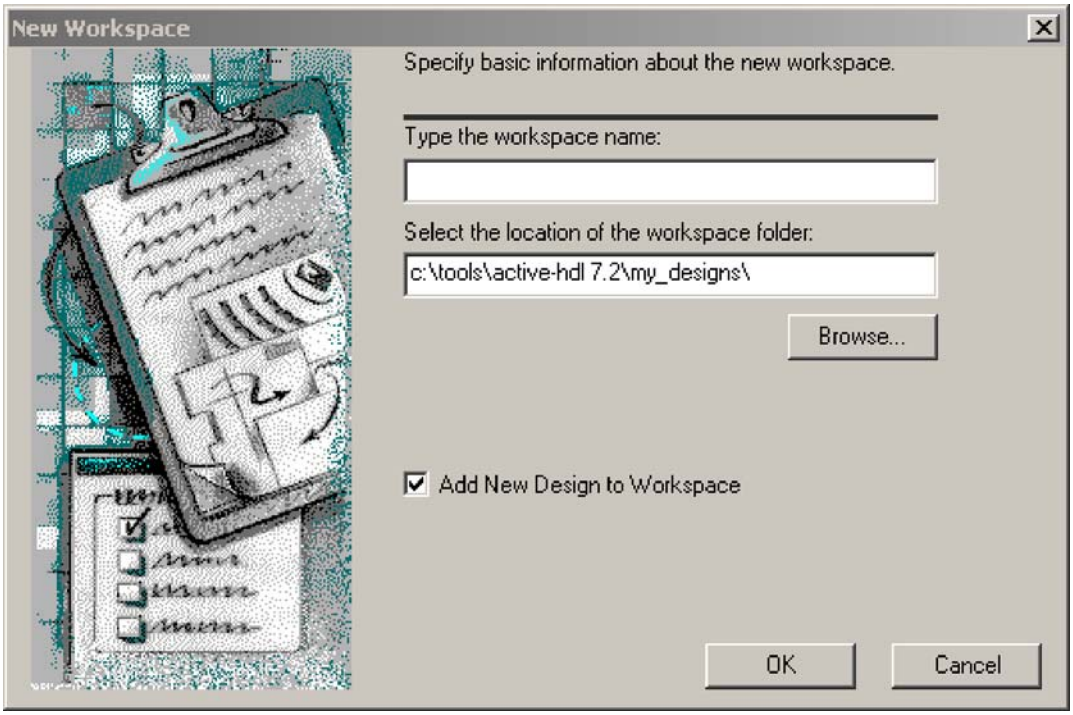


Click on **Create New Workspace** if not already selected.

Click on **OK** to continue

The New Workspace screen appears as shown in Figure 4.

Figure 4. New Workspace Screen



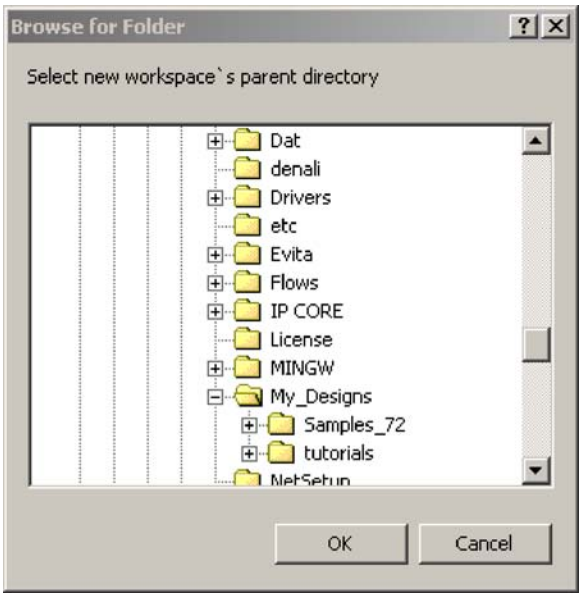
Enter **ddr_sdram_test** in the workspace name text entry box.

Use the **Browse** button to select the location of the workspace folder.

Click on **Browse**.

The **Browse for Folder** screen appears as shown in Figure 5.

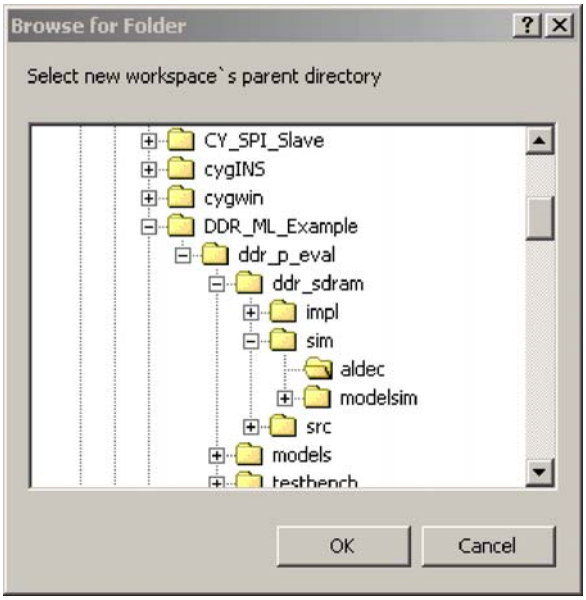
Figure 5. Browse for Folder Screen



Navigate to the directory **C:\DDR_ML_Example\ddr_p_eval\ddr_sdram\sim\aldec**

The Browse for Folder screen is now as shown in Figure 6.

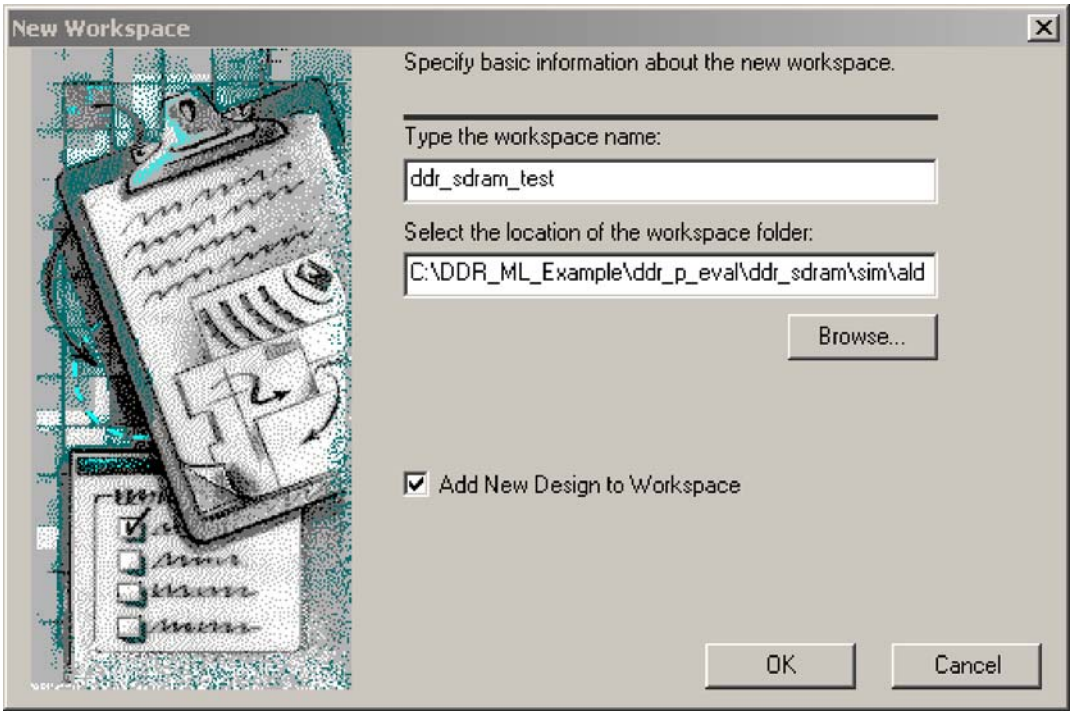
Figure 6. Browse for Folder Screen with Workspace Directory Selected



Click on **OK** to continue.

The New Workspace screen is now show as shown in Figure 7.

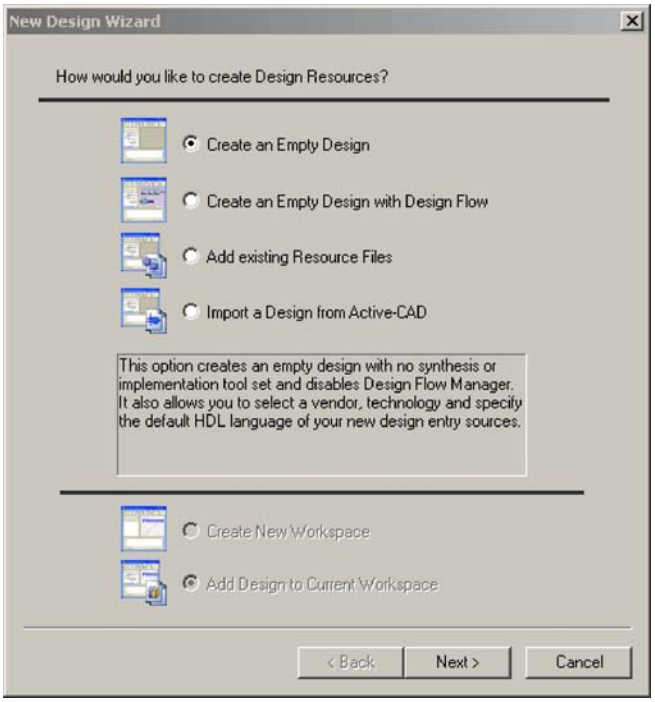
Figure 7. New Workspace Screen with Required Information Entered



Click **OK** to continue.

.The New Design Wizard screen appears as shown in Figure 8.

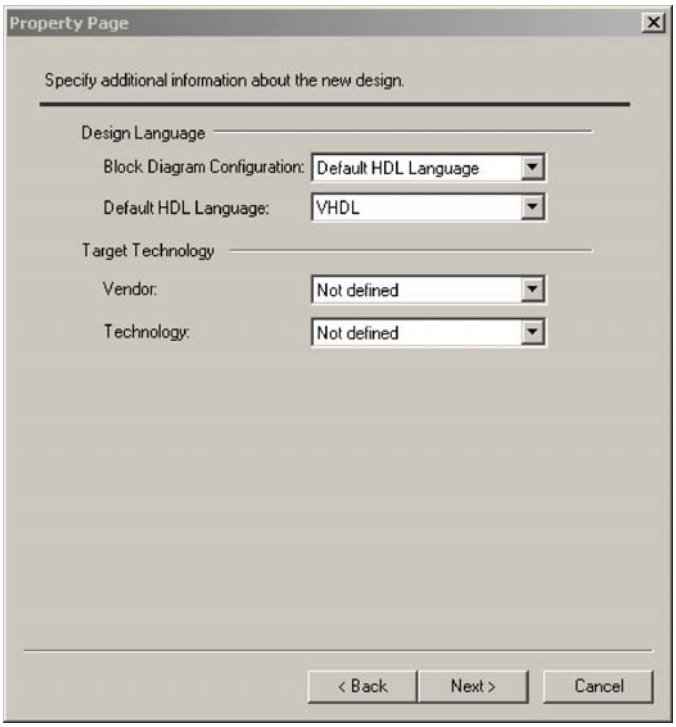
Figure 8. New Design Wizard Screen



Click on **Next** to continue creating an Empty Design.

The **Property Page** screen appears as shown in Figure 9.

Figure 9. Property Page Screen



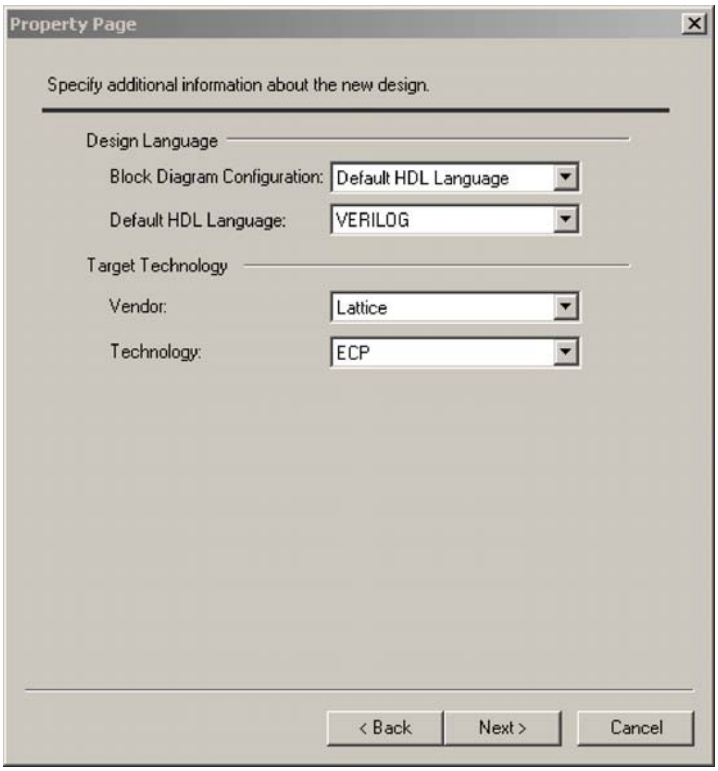
Using the drop-down list for the Default HDL Language, select **VERILOG**.

Using the drop-down list for the Vendor, select **Lattice**.

Using the drop-down list for the Technology, select **ECP**.

The Property Page screen is now as shown in Figure 10.

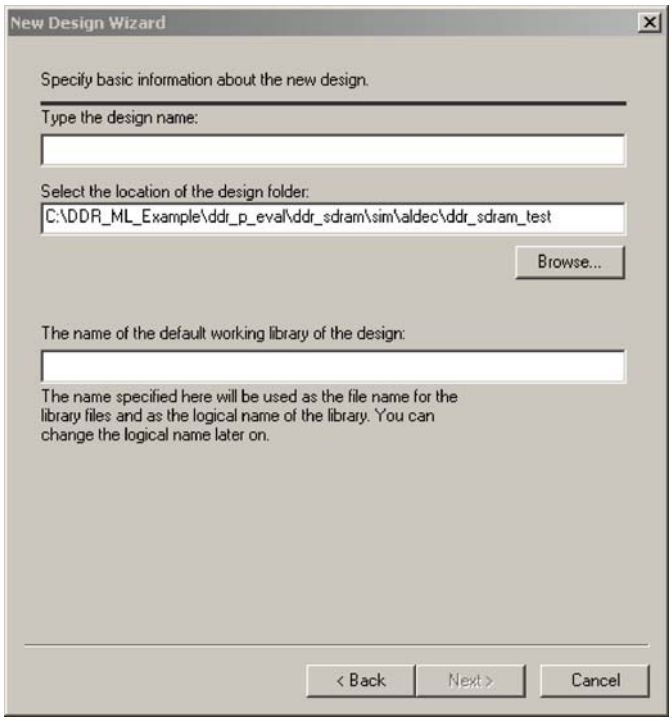
Figure 10. Property Page with Information Entered



Click on **Next** to continue.

The New Design Wizard screen is now as shown in Figure 11.

Figure 11. New Design Wizard Screen with Design Folder Location Selected

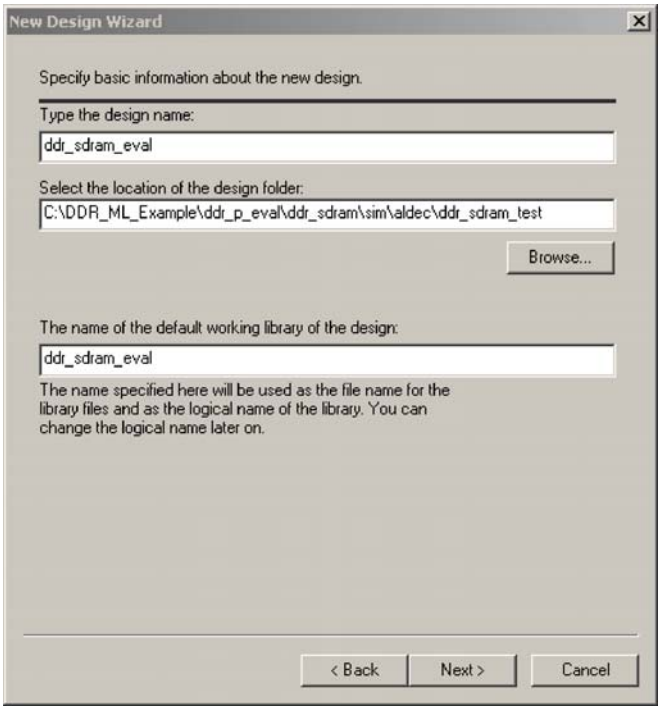


Enter **ddr_sdram_eval** in the design name text entry box.

Note that **ddr_sdram_eval** is also automatically entered into the default working library text box.

The New Design Wizard screen is now as shown in Figure 12.

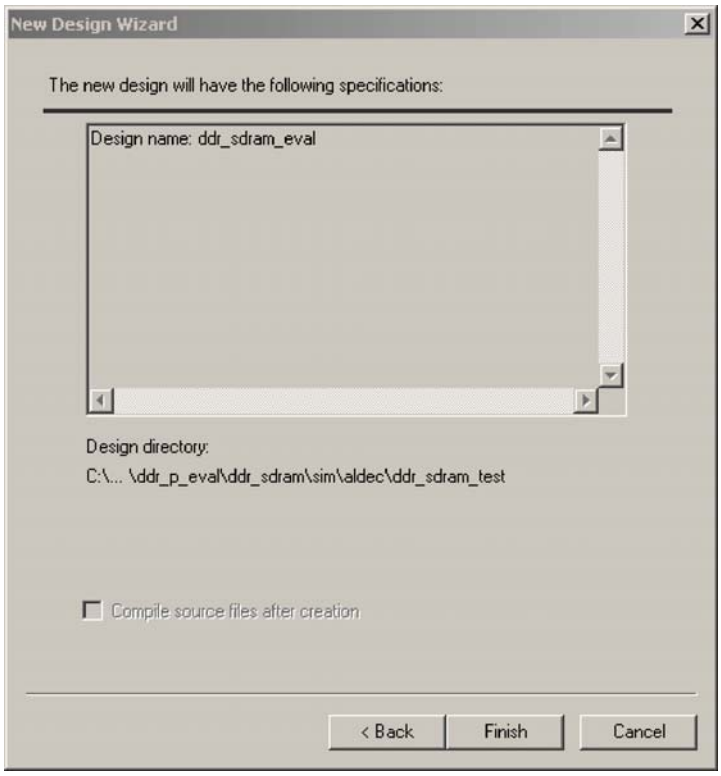
Figure 12. New Design Wizard Screen with Information Entered



Click on **Next** to continue.

The New Design Wizard screen is now as shown in Figure 13.

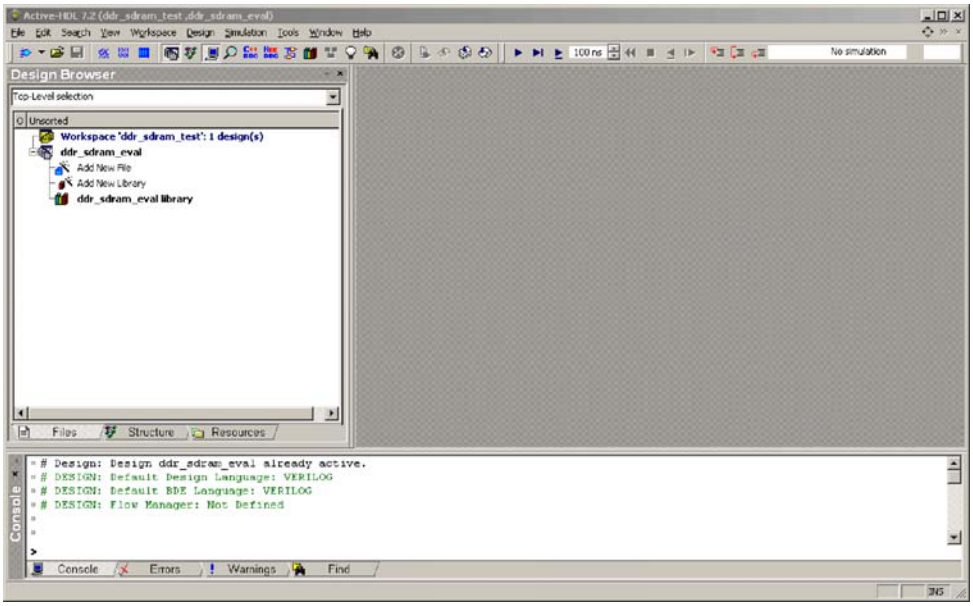
Figure 13. New Design Wizard Screen with Information Entry Completed



Click on **Finish** to create the new design.

The Active-HDL Design Environment screen now appears as shown in Figure 14.

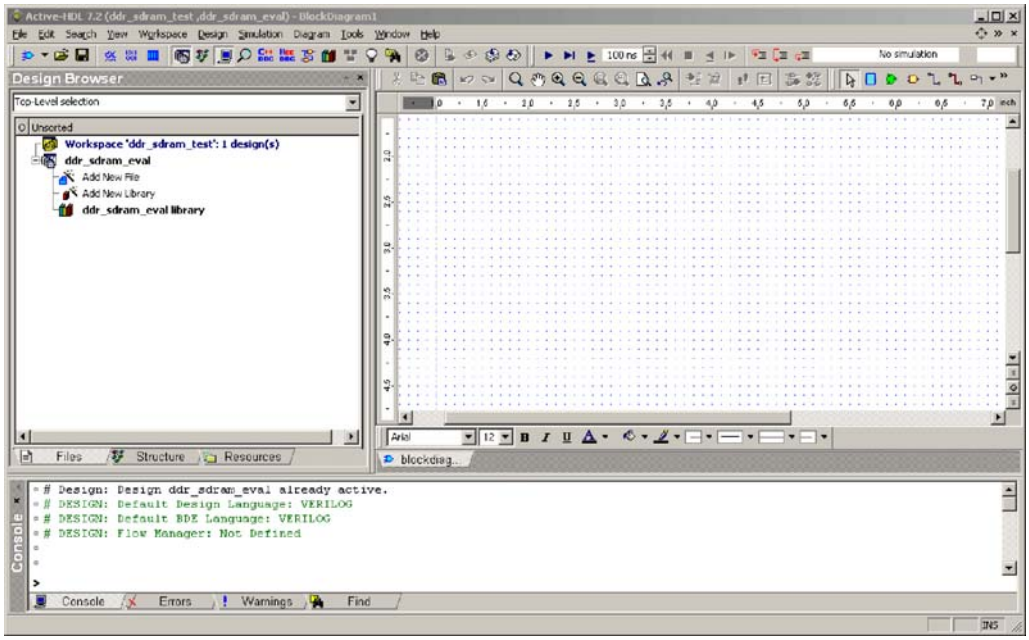
Figure 14. Active-HDL Design Environment Screen with Workspace Active



Click on the **Blue Logic Gate icon** on the **Standard toolbar** to create a new Block Diagram.

A new Block Diagram Entry sheet appears in the Active-HDL Design Environment screen as shown in Figure 15.

Figure 15. Active-HDL Design Environment Screen with New BDE Sheet



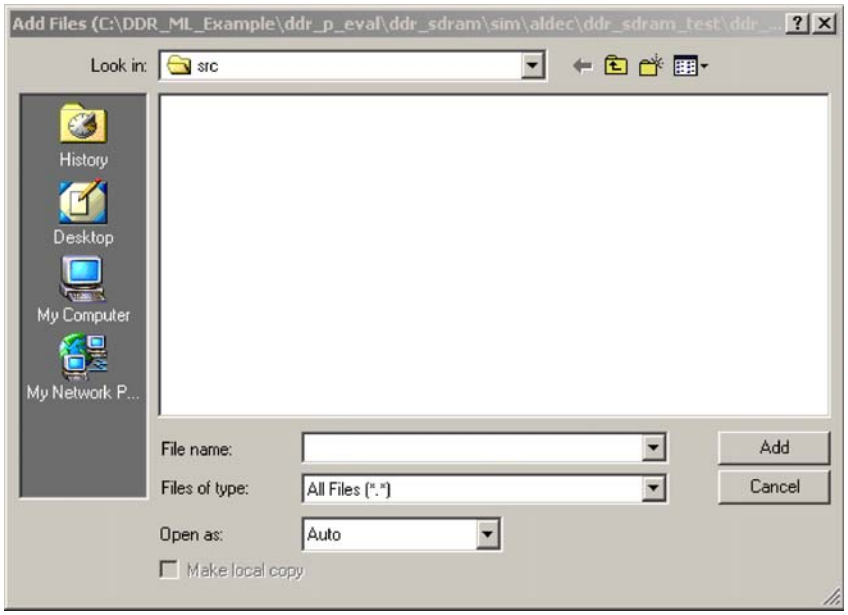
In the Design Browser, left-click on **ddr_sdram_eval** to select.

Right-click and hover on **Add Files to Design**.

Left-click to select.

The **Add Files** screen appears as shown in Figure 16.

Figure 16. Add Files Screen



Navigate to the **C:\DDR_ML_Example\ddr_p_eval\ddr_sdram\sim\modelsim** directory.

Left-click on **ddr_sdram_eval.do** to select it.

Make sure the **Make Local Copy** radio button is selected.

Click on **Add** to copy the macro file into the **ddr_sdram_eval** directory.

Double-click on **ddr_sdram_eval.do** in the **Design Browser** to open it for editing.

Delete the following lines: 1, 2, 3 and 4.

Append **$DSN/../../** to the relative paths defined on lines 8 through 23.

Delete the following lines: 24, 25, 26 and 27.

Add the following text on lines 24, 25 and 26:

```
vsim-L ovi_ecp \
   ddr_sdram_eval.test_mem_ctrl \
   -l ddr_sdram_eval.log
```

Change quietly to **quiet** on line 30.

Delete **-noupdate** in lines 31, 47 and 59.

Comment out the following lines: 60 through 75

Click on **File-> Save**.

The following table details the contents of the ddr_srdam_eval.do macro file.

Table 1. Modified ddr_sdram_eval.do Macro File

```
File Name: ddr_sdram_eval.do

#==== compile
vlog +libext+.v -y $DSN/../../../../../models/mem \
+incdir+$DSN/../../../../../testbench/tests/ecp \
+incdir+$DSN/../../../../src/params \
+incdir+$DSN/../../../../../models/mem \
$DSN/../../../../src/params/ddr_sdram_mem_params.v  \
$DSN/../../../../../models/ecp/pio_dvalid_gen.v \
$DSN/../../../../../models/ecp/ddr_data_io.v \
$DSN/../../../../../models/ecp/ddr_dm_io.v \
$DSN/../../../../../models/ecp/ddr_dqs_io.v \
$DSN/../../../../../models/ecp/kbar_clk_pll.v \
$DSN/../../../../../models/ecp/pmi_distributed_dpram.v \
$DSN/../../../../../../ddr_sdram_beh.v \
$DSN/../../../../../models/ecp/ddr_sdram_mem_io_top.v \
$DSN/../../../../src/rtl/top/ecp/ddr_sdram_mem_top.v \
$DSN/../../../../../testbench/top/ecp/odt_watchdog.v \
$DSN/../../../../../testbench/top/ecp/monitor.v \
$DSN/../../../../../testbench/top/ecp/test_mem_ctrl.v \
+define+NO_DEBUG+ECP
#vcom ../../src/rtl/top/ecp/ddr_sdram_mem_core_top.vhd

#==== run the simulation
vsim  -L ovi_ecp \
      ddr_sdram_eval.test_mem_ctrl \
      -l ddr_sdram_eval.log

view wave
onerror {resume}
```

```
quiet WaveActivateNextPane {} 0
add wave -divider {Global Signals}
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/clk_in
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/rst_n
add wave -noupdate -divider {Memory Interface Signals}
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_clk
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_clk_n
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_cke
add wave -noupdate -format Literal -radix hexadecimal /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_addr
add wave -noupdate -format Literal /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_ba
add wave -noupdate -format Literal -radix hexadecimal /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_data
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_dqs
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_dm
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_ras_n
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_cas_n
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_we_n
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/em_ddr_cs_n
add wave -divider {Local Interface Signals}
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/init_start
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/init_done
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/cmd_rdy
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/cmd_valid
add wave -noupdate -format Literal /test_mem_ctrl/U1_ddr_sdram_mem_top/cmd
add wave -noupdate -format Literal -radix hexadecimal /test_mem_ctrl/U1_ddr_sdram_mem_top/addr
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/data_rdy
add wave -noupdate -format Literal -radix hexadecimal /test_mem_ctrl/U1_ddr_sdram_mem_top/write_data
add wave -noupdate -format Literal /test_mem_ctrl/U1_ddr_sdram_mem_top/data_mask
add wave -noupdate -format Literal -radix hexadecimal /test_mem_ctrl/U1_ddr_sdram_mem_top/read_data
add wave -noupdate -format Logic /test_mem_ctrl/U1_ddr_sdram_mem_top/read_data_valid
add wave -divider {End of the Core}
run -all
```

In Design Browser, left-click on **ddr_sdram_eval.do** to select it.

Right-click and hover over **Execute** to select.

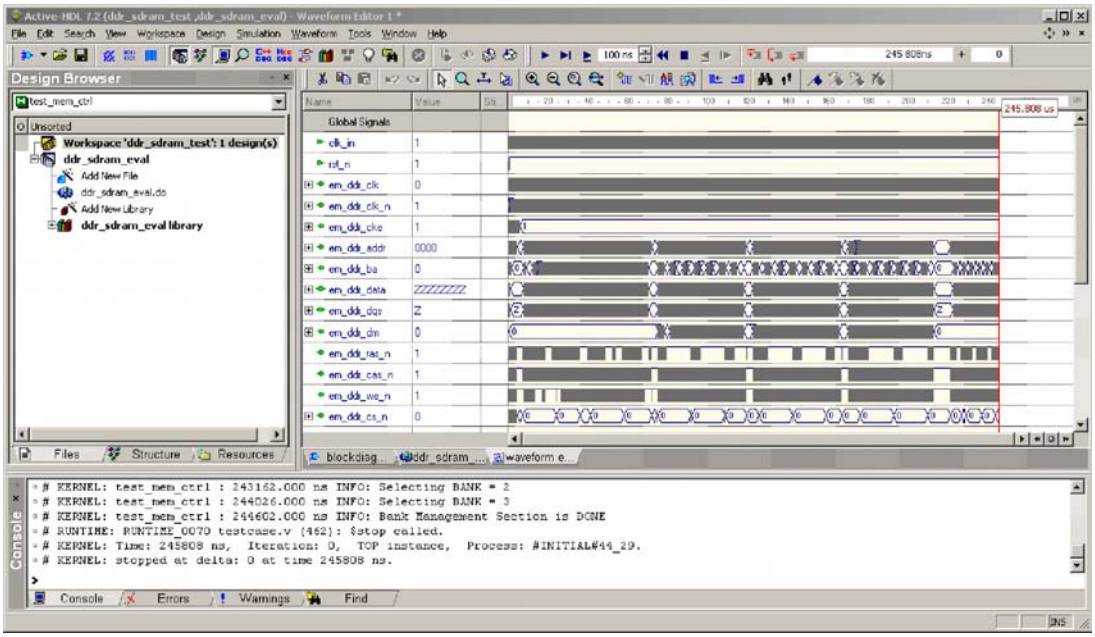
Left-click to **Execute** macro.

The design will now compile, elaborate and simulate.

Left-click on the Design Browser's **Files** tab to select it.

The **Active-HDL Design Environment screen** now appears as shown in Figure 17.

Figure 17. Active-HDL Design Environment Screen

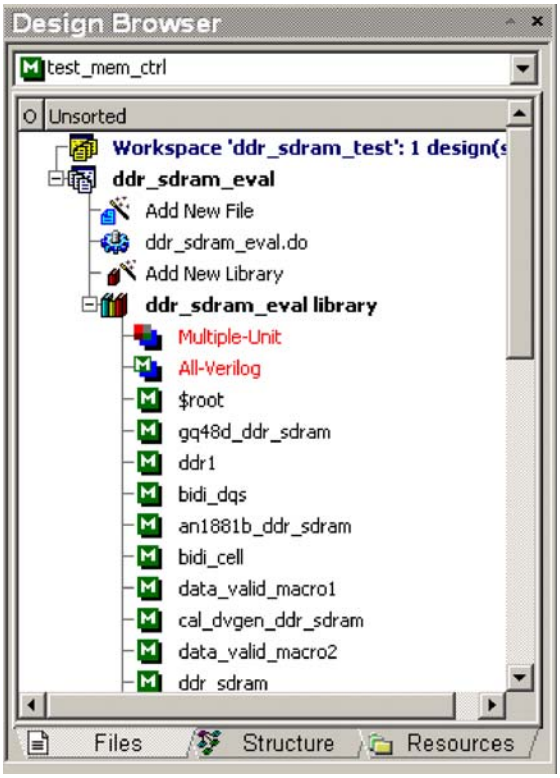


The user can now verify the functionality of the IP core.

Click on the **+** sign next to the **ddr_sdram_eval** library entry in the Design Browser window.

The Active-HDL Design Environment screen now appears as shown in Figure 18.

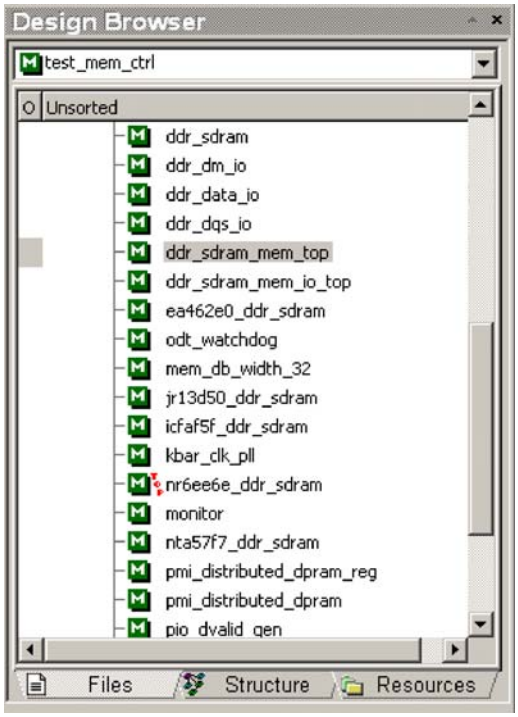
Figure 18. Design Browser Screen with ddr_sdram_eval Library Expanded



Scroll down and left-click to select the **ddr_sdram_mem_top** Verilog module.

The Active-HDL Design Environment screen now appears as shown in Figure 19.

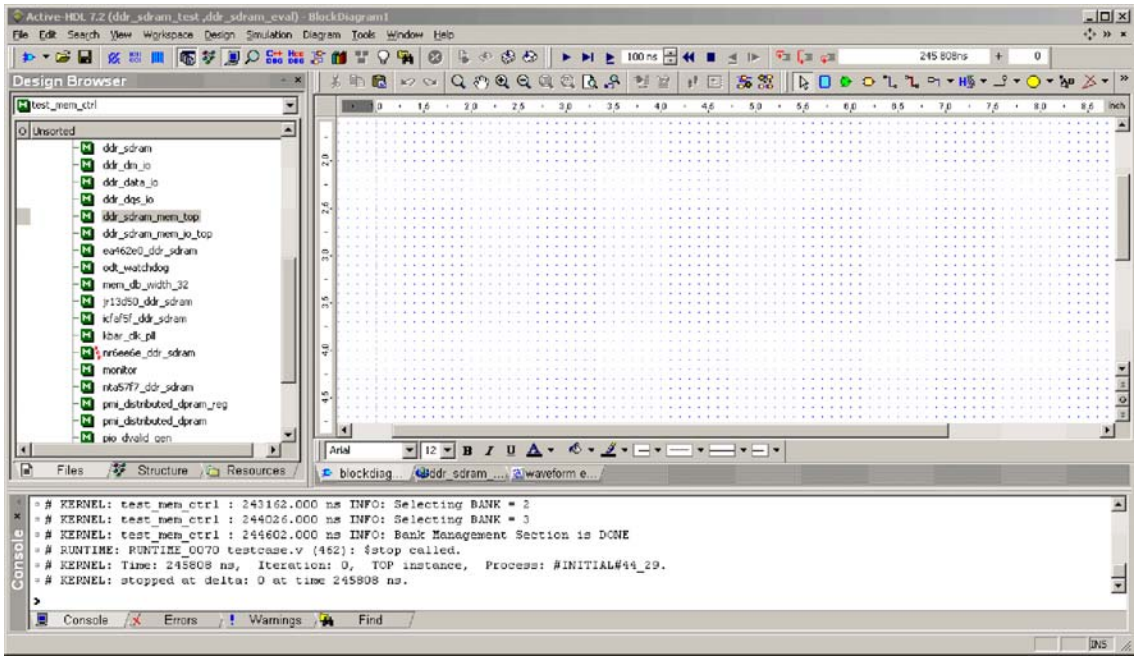
Figure 19. Design Browser Screen with ddr_sdram_mem_top Verilog Module Selected



Click on the **Block Diagram Editor** tab to expose the Block Diagram sheet.

The Design Environment will appear as shown in Figure 20.

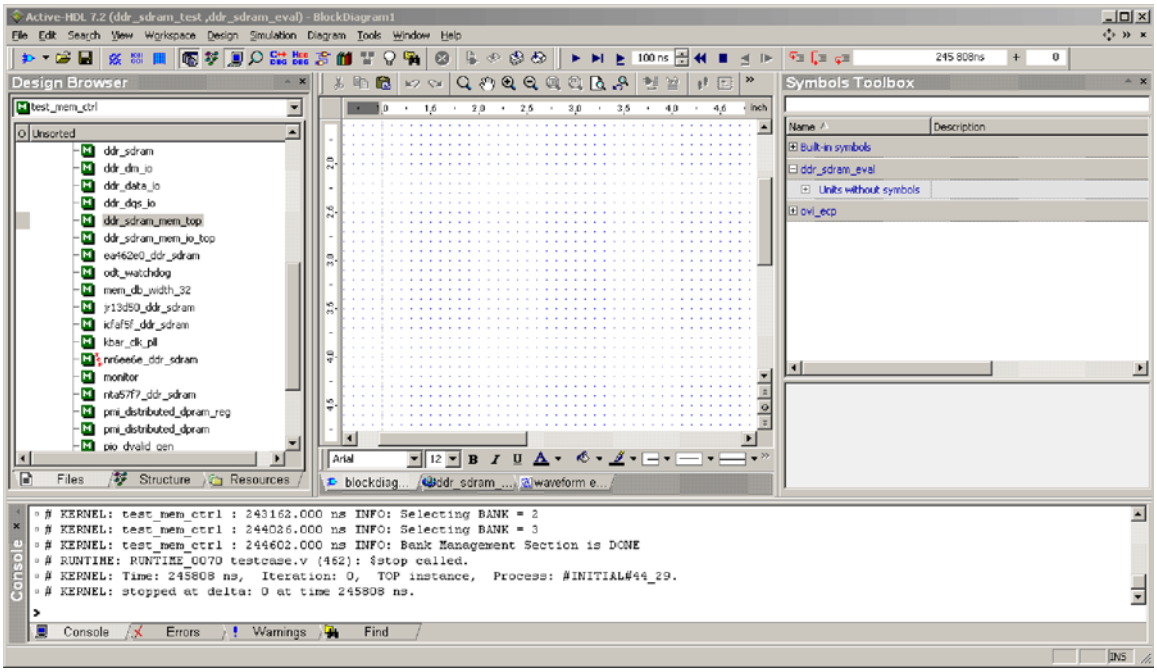
Figure 20. Design Environment Screen with BDE Sheet Exposed



Left-click on the **yellow logic gate** of the **BDE Edit toolbar** to open the **Symbols Toolbox**.

The Design Environment will appear as shown in Figure 21.

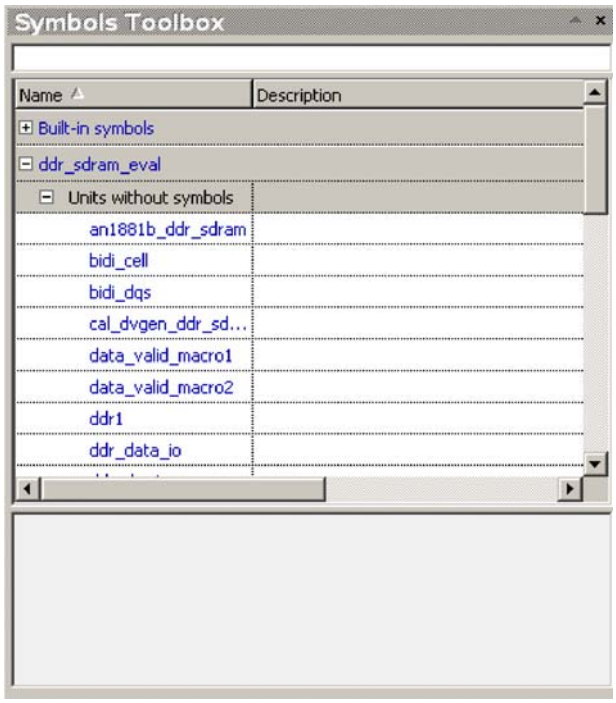
Figure 21. Design Environment Screen with Symbols Toolbox Window Open



In the Symbols Toolbox window, left click on the **+** sign next to the **Units without symbols** label to expand it.

The Symbols Toolbox now appears as shown in Figure 22.

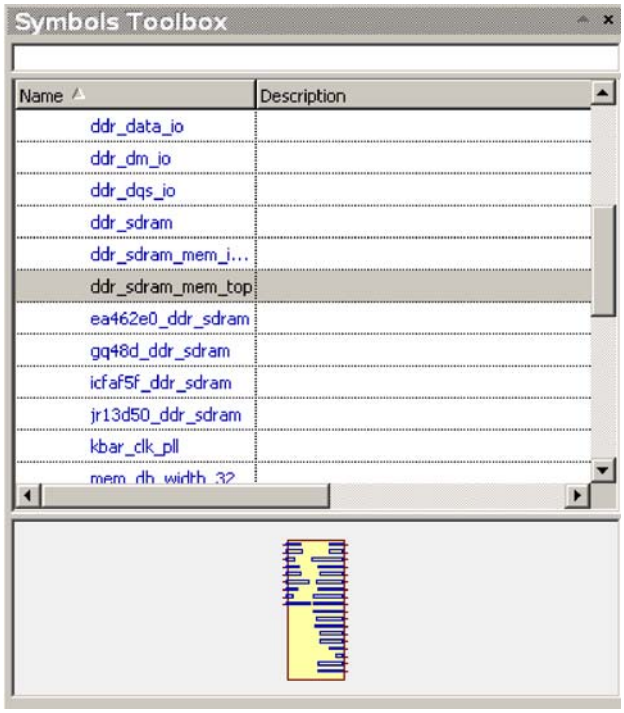
Figure 22. Symbols Toolbox with "Units without symbols" Label Expanded



Scroll down to `ddr_sdram_mem_top` and double-click to select it.

The Symbols Toolbox now appears as shown in Figure 23.

Figure 23. Symbols Toolbox window with ddr_sdram_mem_top Selected

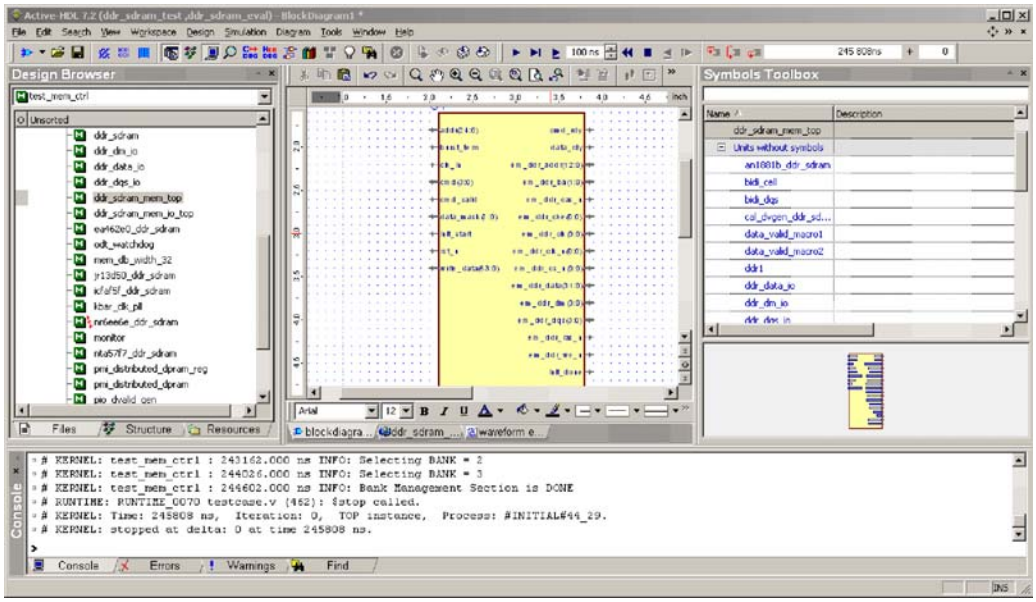


Left-click on the **ddr_sdram_mem_top symbol** icon.

Drag and drop it into the **BDE Edit** window.

The Design Environment Screen now appears as shown in Figure 24.

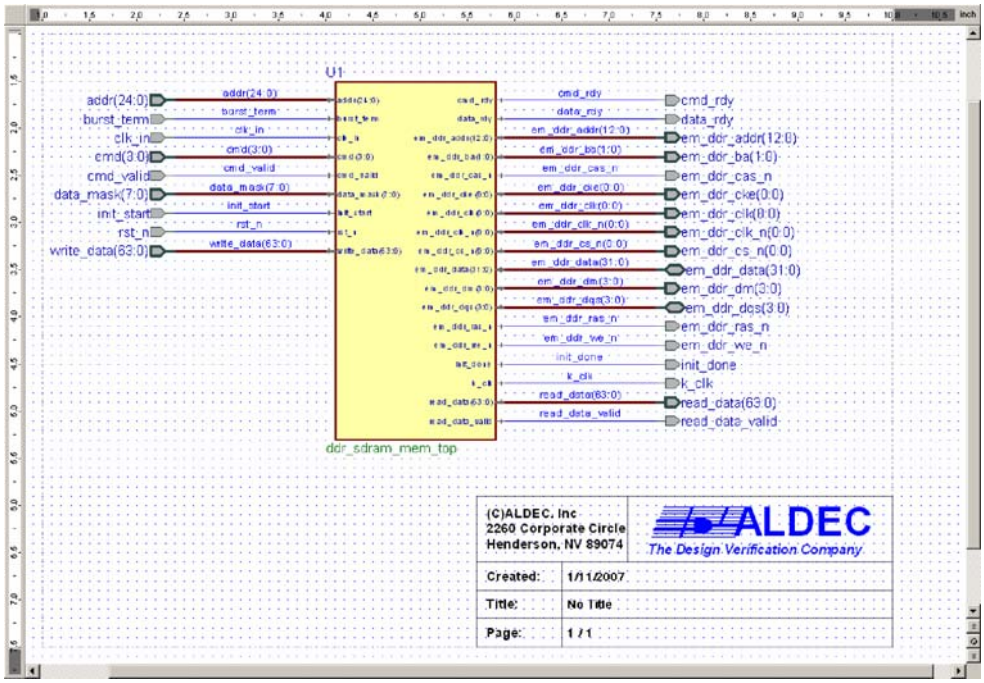
Figure 24. Design Environment Screen with ddr_sdram_mem_top Symbol Added to the BDE Edit Window



Use the tools provided on the **BDE Edit toolbar** to add signals, buses, ports and names to the Block Diagram.

After additions the Block Diagram Editor sheet now appears as shown in Figure 25.

Figure 25. Block Diagram Editor Widow with Symbol and Ports Connected and Named



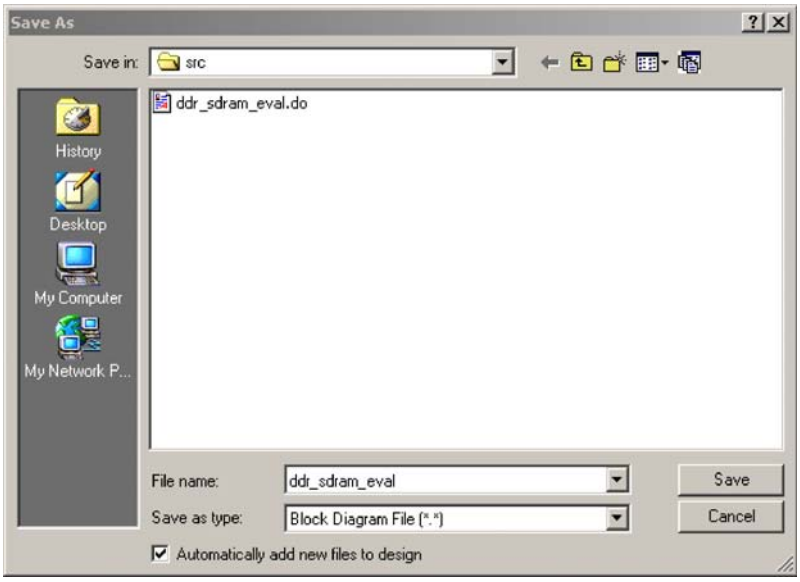
Save the Block Diagram by clicking on **File-> Save**.

A **Save As** screen now appears.

Enter **ddr_sdram_eval** as the new file name.

The **Save As** screen will appear as shown in Figure 26.

Figure 26. Save As Screen with File Name Entered

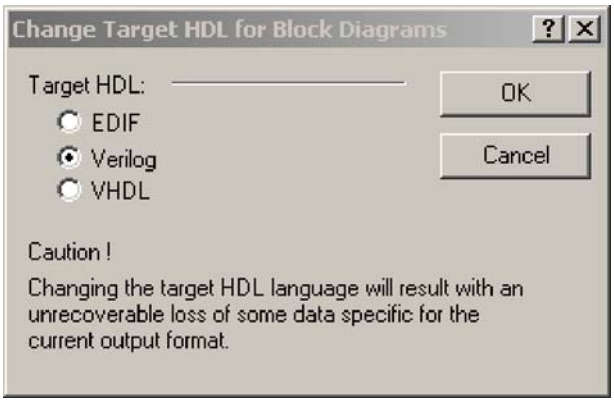


The next step changes the HDL language selection from Verilog to VHDL.

Left-click on **Design->Change Target HDL...**

The Change Target HDL for Block Diagrams screen appears as shown in Figure 27.

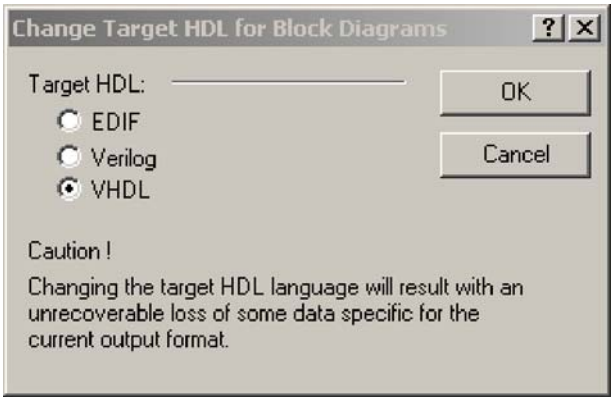
Figure 27. Change Target HDL for Block Diagram Screen



Click on the radio button new to **VHDL** to change the Target HDL.

The Change Target HDL for Block Diagrams screen appears as shown in Figure 28.

Figure 28. Change Target HDL Screen Modified to VHDL



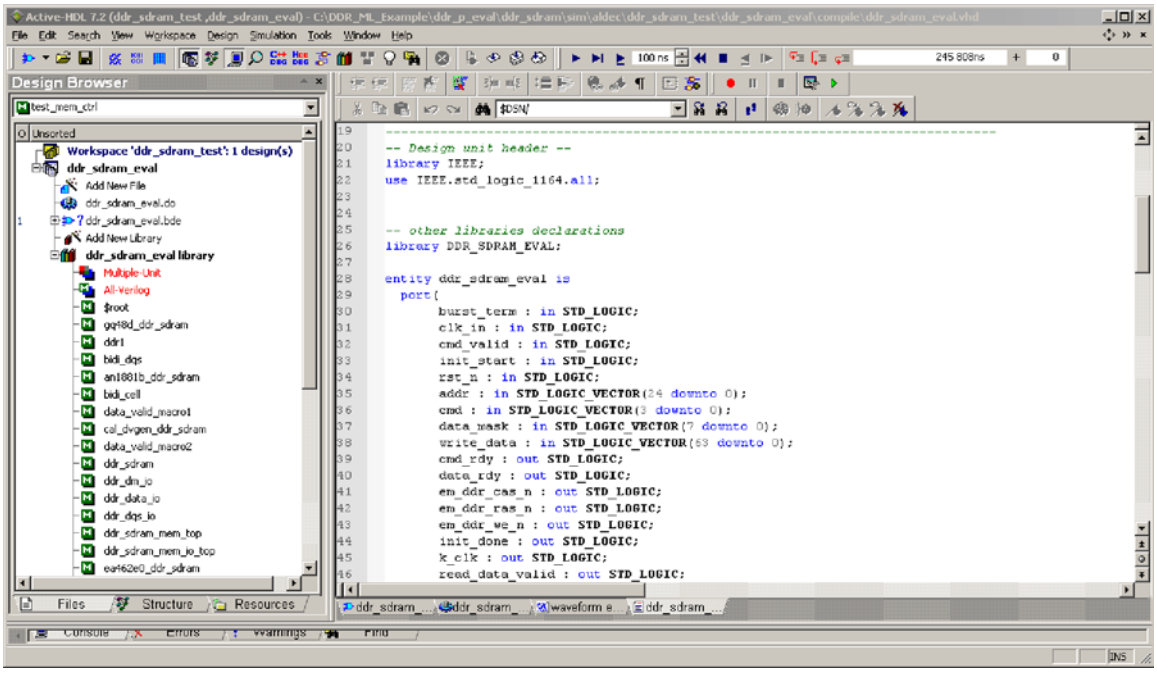
Click on **OK** to complete the change.

Generate the VHDL Wrapper by clicking on the **Generate HDL Code icon** (blue logic symbol, blue lines and black connector) in the **BDE Edit toolbar**.

Click on the **View HDL Code icon** (black glasses and blue lines) to view the generated VHDL wrapper.

The **Text Edit** screen appears with the VHDL wrapper containing the Entity/Architecture pair, component declaration and component instantiation as shown in Figure 29.

Figure 29. Design Environment with VHDL Wrapper Displayed in Text Edit Window

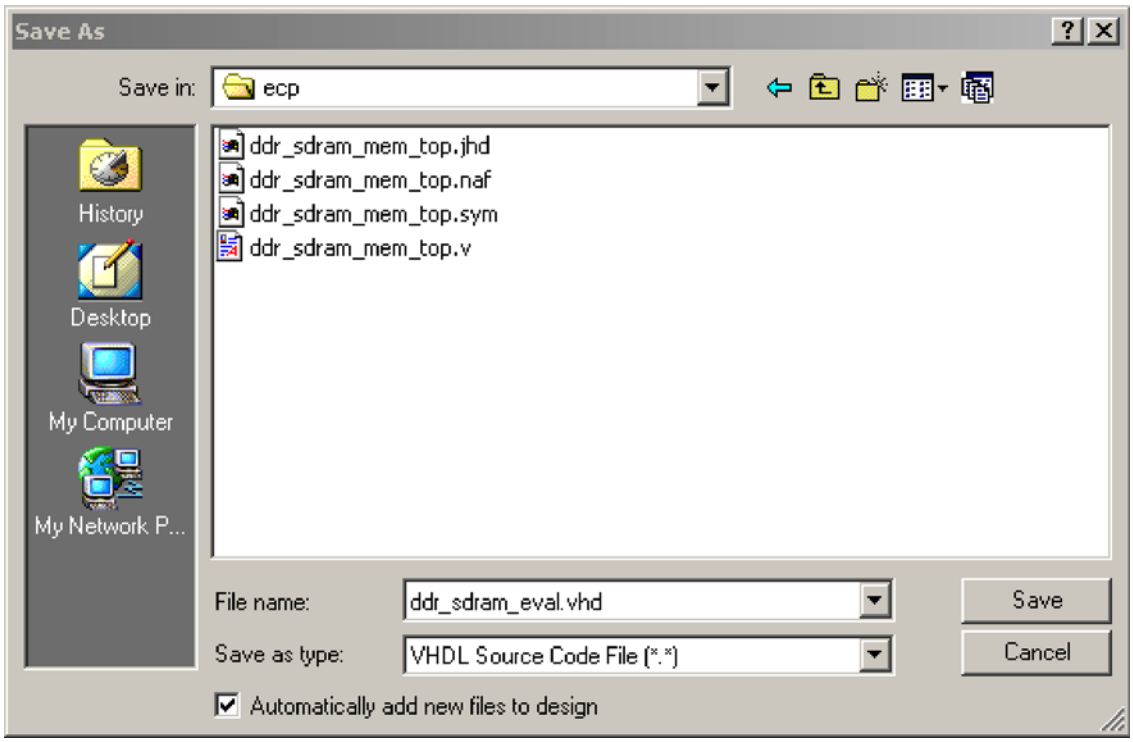


Save the file to the **Lattice IP core source directory**.

Click on **File->Save As** and navigate to the directory **C:\DDR_ML_Example\ddr_p_eval\ddr_sdram\src\rtl\top\ecp**.

The **Save As** screen will appear as shown in Figure 30.

Figure 30. File Save As Screen with Source Directory Selected

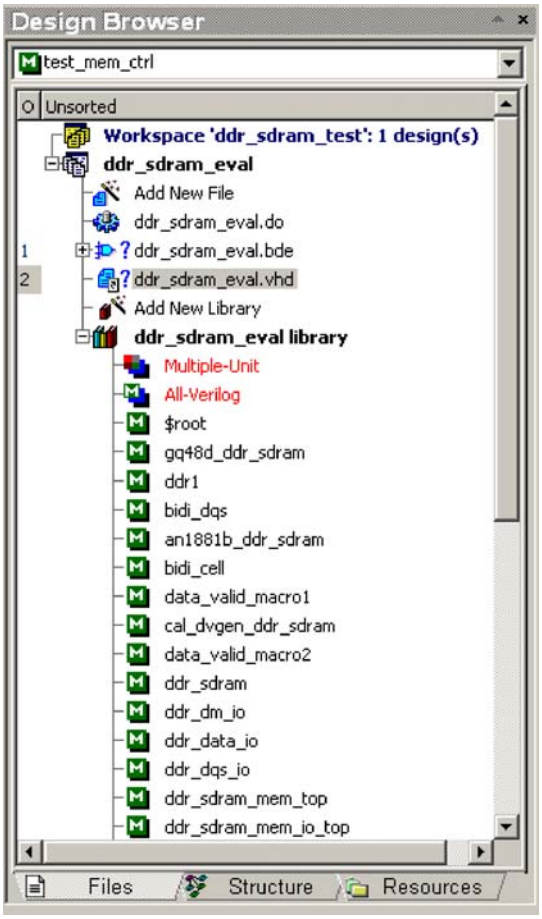


Ensure **Automatically add new files to design** is selected.

Click on **Save** to save the file.

The new VHDL wrapper **ddr_sdram_eval.vhd** now appears in the **Files** tab of the **Design Browser** as shown in Figure 31.

Figure 31. Design Browser Window Displaying VHDL Wrapper



Compile the new VHDL Wrapper by left-clicking on **ddr_sdram_eval.vhd** to select it.

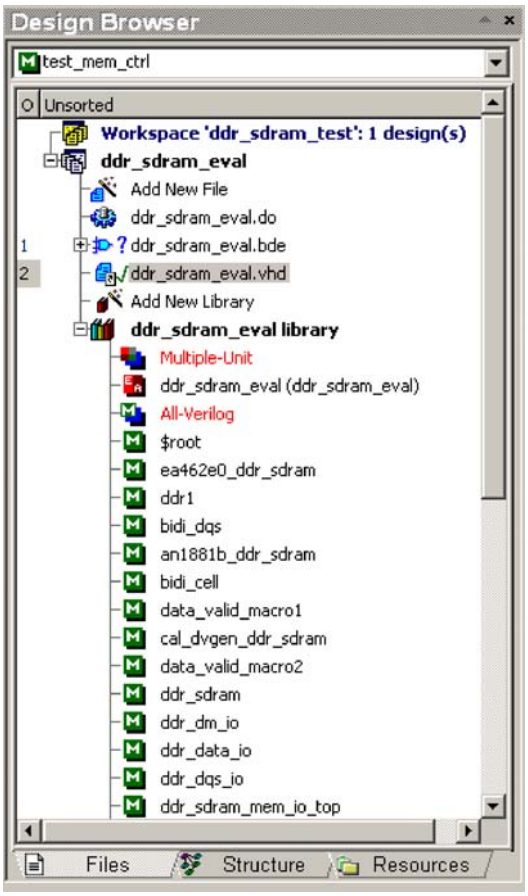
Right-click on **ddr_sdram_eval.vhd**.

Left-click on **Compile** to run the compiler.

Active-HDL compiles the VHDL wrapper and adds the Entity/Architecture pair to the ddr_sdram_eval library for use by the simulator.

The Design Browser's Files tab now appears as shown in Figure 32.

Figure 32. Design Browser's Files Tab with VHDL Wrapper and Entity/Architecture Pair Added



Summary

The previous steps have verified the integrity of the Lattice IPexpress DDR SDRAM Controller 6.2 core and created VHDL top-level wrappers for instantiation of the core into the user's design. As Lattice does not package a VHDL test-bench with the IP core, the user must create a VHDL testbench for simulation of the core using the new wrappers. Optionally the user may instantiate the core into their VHDL design and simulate the core as part of the overall design.

References

Refer to the *ModelSim SE User's Manual* Chapter 9 Mixed-Language Simulation and Chapter 18 Signal Spy for further information.

HDL Source Listings

Verilog Core Top-Level Module

```
module ddr_sdram_mem_top (

      // Clock and reset
      clk_in,                 // System Clock
      rst_n,                  // System reset

      // Inputs signals from the User Interface
      cmd,                    // Command for controller
      addr,                   // Address for Rd/Wr
      cmd_valid,              // Command Valid
      init_start,             // Starts the Iitialization
      write_data,             // Data to be written
      data_mask,              // Data Mask

      // User Interface Output signals
      cmd_rdy,                // Ready for the new Command
      init_done,              // Initialization is done
      data_rdy,               // Ready for more Write_data
      read_data,              // Read data to the User
      read_data_valid,        // Read Data Valid
      k_clk,

      // Bi-directional databus to external memory
      em_ddr_data,            // Data to the memory
      em_ddr_dqs,             // Data_Strobes

      // Output to External memory
      // SDRAM Address, controls and clock
      em_ddr_clk,             // DDR1/DDR2 clock
      em_ddr_clk_n,           // Inverted DDR1/DDR2 clock
      em_ddr_cke,             // Clock Enable
      em_ddr_ras_n,           // Row address strobe
      em_ddr_cas_n,           // Column Address Strobe
      em_ddr_we_n,            // Write Enable
      em_ddr_cs_n,            // Chip select

      burst_term,             // Burst Termination

      em_ddr_dm,              // Data mask
      em_ddr_ba,              // Bank Address
      em_ddr_addr             // Row or Collumn Address
   );

   //--------User Inputs
   input                              clk_in;
   input                              rst_n;
   input   [3:0]                      cmd;
   input   [`ADDR_WIDTH-1:0]          addr;
   input                              cmd_valid;
   input                              init_start;
```

```
input   [`DSIZE-1:0]                write_data;
input   [`USER_DM-1:0]              data_mask;

//--------User Outputs
output                              cmd_rdy;
output                              init_done;
output                              data_rdy;
output  [`DSIZE -1:0]               read_data;
output                              read_data_valid;
output                              k_clk;

//--------DDR Bi-Directionals
inout  [`DATA_WIDTH-1:0]            em_ddr_data;
inout  [`DQS_WIDTH-1:0]             em_ddr_dqs;


//--------DDR Outputs
output [`DATA_WIDTH/8-1:0]          em_ddr_dm;
output [`CLKO_WIDTH-1:0]            em_ddr_clk;
output [`CLKO_WIDTH-1:0]            em_ddr_clk_n;
output [`CKE_WIDTH-1:0]             em_ddr_cke;
output                              em_ddr_ras_n;
output                              em_ddr_cas_n;
output                              em_ddr_we_n;
output  [`CS_WIDTH-1:0]             em_ddr_cs_n;

//--------DDR Inputs
input                               burst_term;

//--------DDR Output Buses
output  [`ROW_WIDTH-1:0]            em_ddr_addr;
output  [`BNK_WDTH-1:0]             em_ddr_ba;
```

VHDL Wrapper Listing

```
--- Design unit header ---
library IEEE;
use IEEE.std_logic_1164.all;
--- other libraries declarations ---
library DDR_SDRAM_EVAL;
entity ddr_sdram_eval is
 port(
   burst_term :      in STD_LOGIC;
   clk_in :          in STD_LOGIC;
   cmd_valid :       in STD_LOGIC;
   init_start :      in STD_LOGIC;
   rst_n :           in STD_LOGIC;
   addr :            in STD_LOGIC_VECTOR(24 downto 0);
   cmd :             in STD_LOGIC_VECTOR(3 downto 0);
   data_mask :       in STD_LOGIC_VECTOR(7 downto 0);
   write_data :      in STD_LOGIC_VECTOR(63 downto 0);
   cmd_rdy :         out STD_LOGIC;
   data_rdy :        out STD_LOGIC;
   em_ddr_cas_n :    out STD_LOGIC;
```

```
    em_ddr_ras_n :    out STD_LOGIC;
    em_ddr_we_n :     out STD_LOGIC;
    init_done :       out STD_LOGIC;
    k_clk :           out STD_LOGIC;
    read_data_valid : out STD_LOGIC;
    em_ddr_addr :     out STD_LOGIC_VECTOR(12 downto 0);
    em_ddr_ba :       out STD_LOGIC_VECTOR(1 downto 0);
    em_ddr_cke :      out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_clk :      out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_clk_n :    out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_cs_n :     out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_dm :       out STD_LOGIC_VECTOR(3 downto 0);
    read_data :       out STD_LOGIC_VECTOR(63 downto 0);
    em_ddr_data :     inout STD_LOGIC_VECTOR(31 downto 0);
    em_ddr_dqs :      inout STD_LOGIC_VECTOR(3 downto 0)
  );
end ddr_sdram_eval;

architecture ddr_sdram_eval of ddr_sdram_eval is
---- Component declarations -----
component ddr_sdram_mem_top
  port (
    addr :            in STD_LOGIC_VECTOR(24 downto 0);
    burst_term :      in STD_LOGIC;
    clk_in :          in STD_LOGIC;
    cmd :             in STD_LOGIC_VECTOR(3 downto 0);
    cmd_valid :       in STD_LOGIC;
    data_mask :       in STD_LOGIC_VECTOR(7 downto 0);
    init_start :      in STD_LOGIC;
    rst_n :           in STD_LOGIC;
    write_data :      in STD_LOGIC_VECTOR(63 downto 0);
    cmd_rdy :         out STD_LOGIC;
    data_rdy :        out STD_LOGIC;
    em_ddr_addr :     out STD_LOGIC_VECTOR(12 downto 0);
    em_ddr_ba :       out STD_LOGIC_VECTOR(1 downto 0);
    em_ddr_cas_n :out STD_LOGIC;
    em_ddr_cke :      out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_clk :      out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_clk_n :    out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_cs_n :     out STD_LOGIC_VECTOR(0 downto 0);
    em_ddr_dm :       out STD_LOGIC_VECTOR(3 downto 0);
    em_ddr_ras_n :    out STD_LOGIC;
    em_ddr_we_n :     out STD_LOGIC;
    init_done :       out STD_LOGIC;
    k_clk :           out STD_LOGIC;
    read_data :       out STD_LOGIC_VECTOR(63 downto 0);
    read_data_valid : out STD_LOGIC;
    em_ddr_data :     inout STD_LOGIC_VECTOR(31 downto 0);
    em_ddr_dqs :      inout STD_LOGIC_VECTOR(3 downto 0)
  );
end component;
```

```
begin
----  Component instantiations  ----
U1 : ddr_sdram_mem_top
 port map(
   addr              => addr,
   burst_term        => burst_term,
   clk_in            => clk_in,
   cmd               => cmd,
   cmd_rdy           => cmd_rdy,
   cmd_valid         => cmd_valid,
   data_mask         => data_mask,
   data_rdy          => data_rdy,
   em_ddr_addr       => em_ddr_addr,
   em_ddr_ba         => em_ddr_ba,
   em_ddr_cas_n      => em_ddr_cas_n,
   em_ddr_cke        => em_ddr_cke,
   em_ddr_clk        => em_ddr_clk,
   em_ddr_clk_n      => em_ddr_clk_n,
   em_ddr_cs_n       => em_ddr_cs_n,
   em_ddr_data       => em_ddr_data,
   em_ddr_dm         => em_ddr_dm,
   em_ddr_dqs        => em_ddr_dqs,
   em_ddr_ras_n      => em_ddr_ras_n,
   em_ddr_we_n       => em_ddr_we_n,
   init_done         => init_done,
   init_start        => init_start,
   k_clk             => k_clk,
   read_data         => read_data,
   read_data_valid   => read_data_valid,
   rst_n             => rst_n,
   write_data        => write_data
 );
end ddr_sdram_eval;
```

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.