Mixed-Language Simulation with Lattice IP Designs Using ModelSim

February 2007 Technical Note TN1125

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

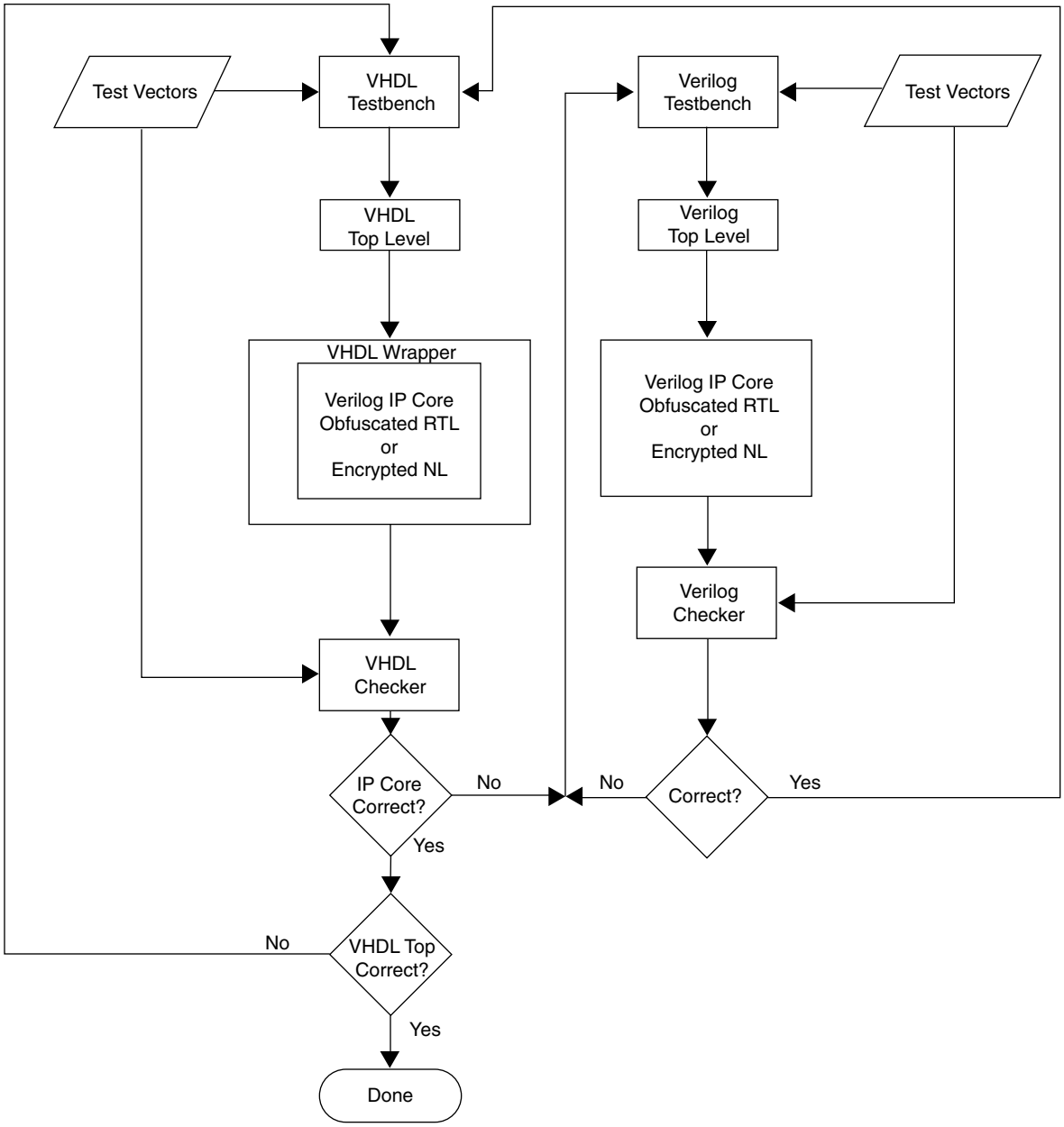
Lattice IP cores are distributed using an obfuscated Verilog RTL simulation model and an encrypted Verilog gate level model. In Verilog-based designs, the IP cores are directly instantiated in the top-level of the design as modules. A Verilog testbench is provided for all Lattice IPs. The testbench is 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 ModelSim, a designer creates a component using a utility named vgencomp. This utility reads in the Verilog top-level module port list and I/O declarations from which it creates a VHDL component port map. The user creates a matching VHDL 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 ModelSim environment. The following example assumes that the user is experienced in using the ispLEVER®, IPexpress and ModelSim 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 ModelSim to Instantiate a Verilog RTL Design Into a VHDL Project

This example assumes the following:

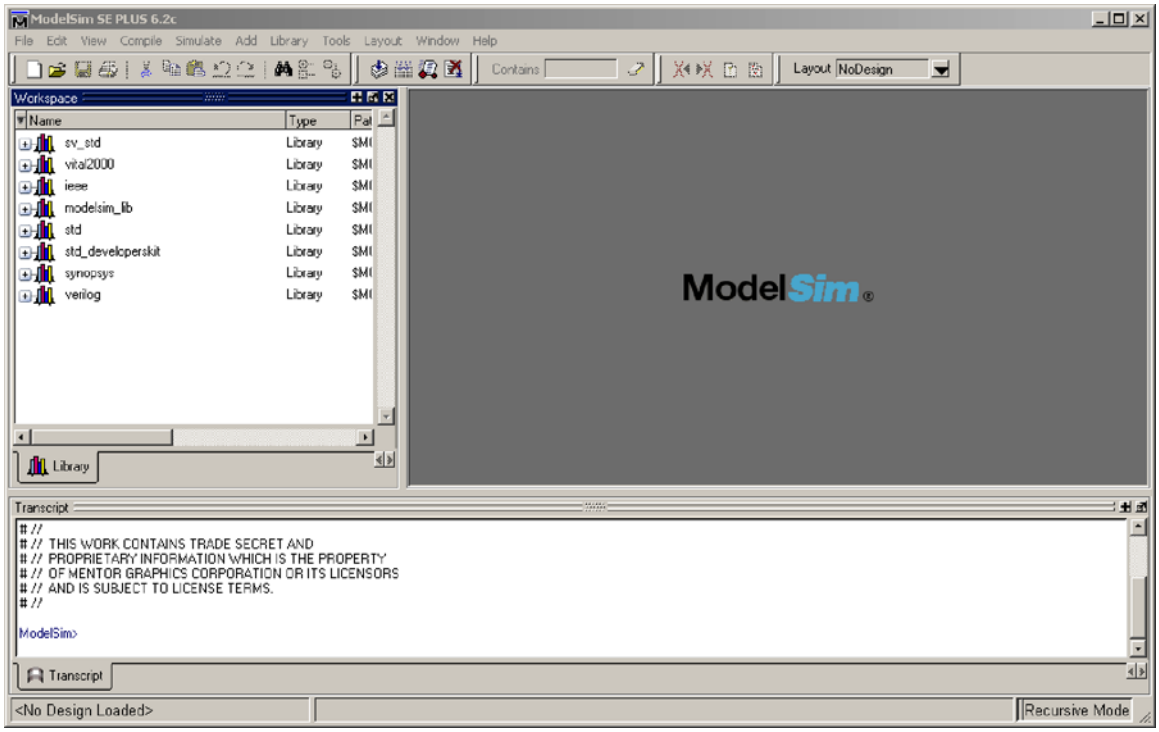
- The user has installed Lattice ispLEVER 6.1 SP1 in the directory C:/ispTOOLS6_1.
- The user has already utilized the Lattice IPexpress tool to install the DDR SDRAM Controller v 6.2 in the directory C:/DDR_ML_Example.
- The targeted file name is ddr_sdram.
- The targeted device is the LFECP33E-4F484C.
- The user has installed ModelSim SE 6.2 C in the directory C:\Modeltech_6.2c.

Launch ModelSim and navigate to the simulation directory.

1. Click on Start->Programs->ModelSim->Modelsim

1.1. The **ModelSim opening screen** now appears as shown in Figure 2.

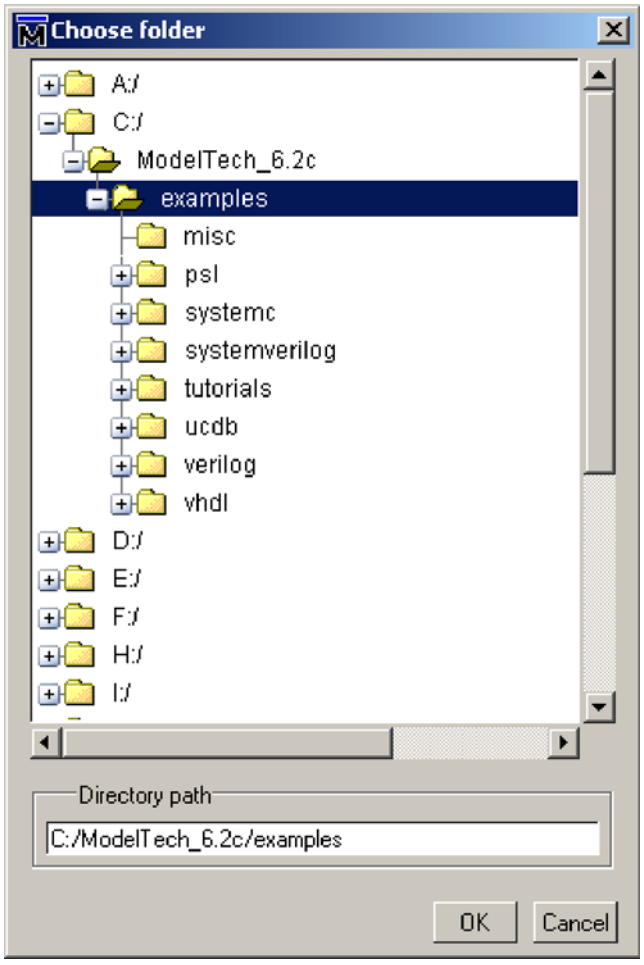
Figure 2. ModelSim Opening Screen



1.2. Click on **File->Change Directory**

The **Choose Folder** window now appears as shown in Figure 3.

Figure 3. Choose Folder Screen Before Changing Directory

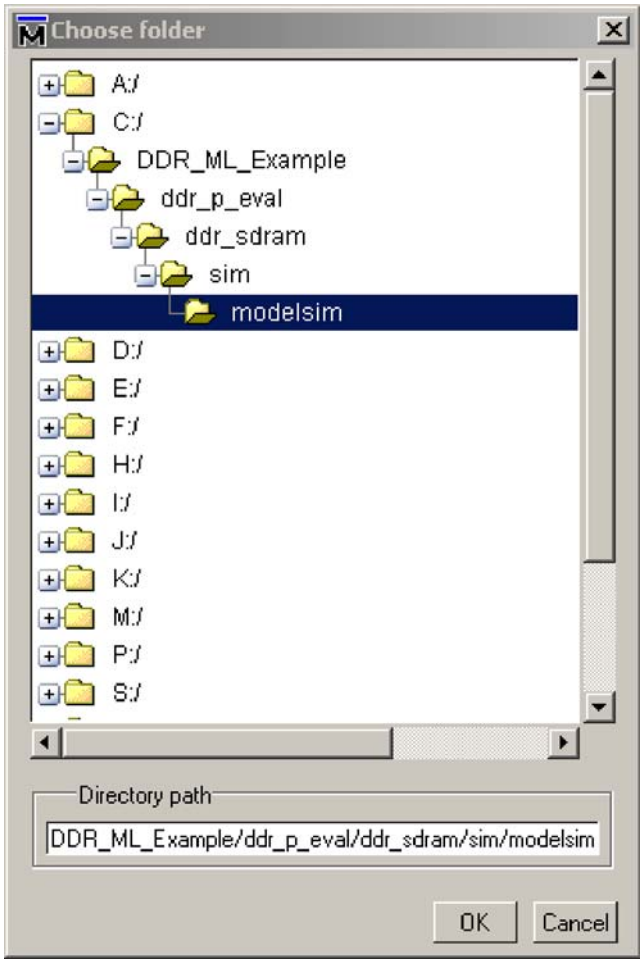


1.3. Browse to the project's simulation directory.

Browse to **C:/DDR_ML_Example/ddr_p_eval/ddr_sdram/sim/modelsim**.

The **Choose Folder** window now appears as shown in Figure 4.

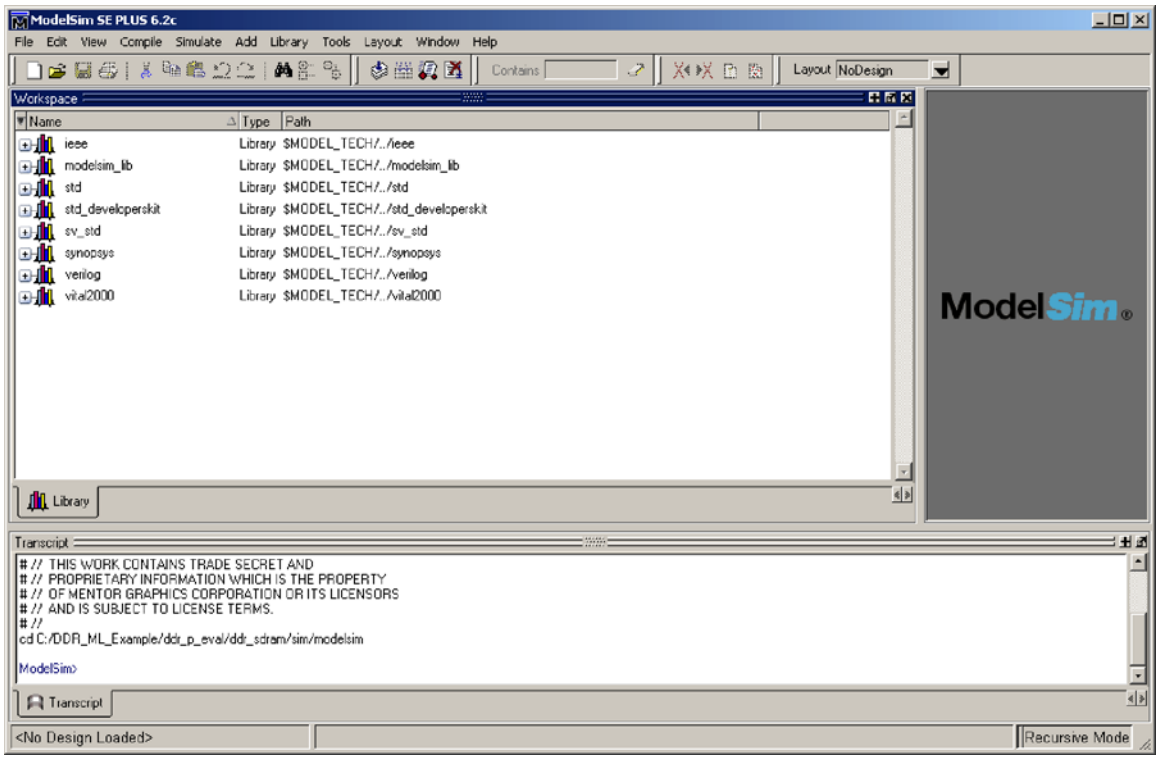
Figure 4. Choose Folder Screen After Browsing to Project Simulation Directory



1. 4. Click on **OK** to change to the project's simulation directory.

The **Choose Folder** screen will close and the ModelSim screen will appear as in Figure 5.

Figure 5. ModelSim Screen After Changing to the Project's Simulation Directory

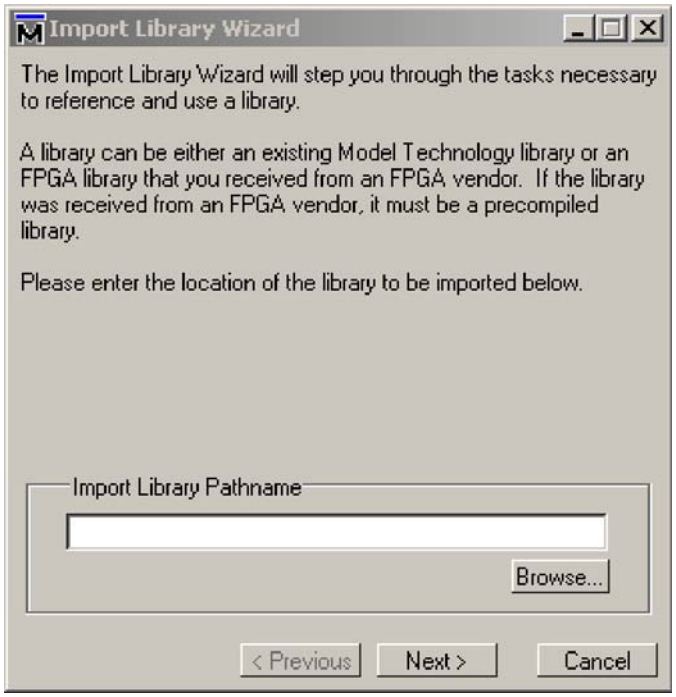


2. Import the ECP Verilog library.

2.1. Click on **File->Import->Library**

The **Import Library Wizard** screen will appear as shown in Figure 6.

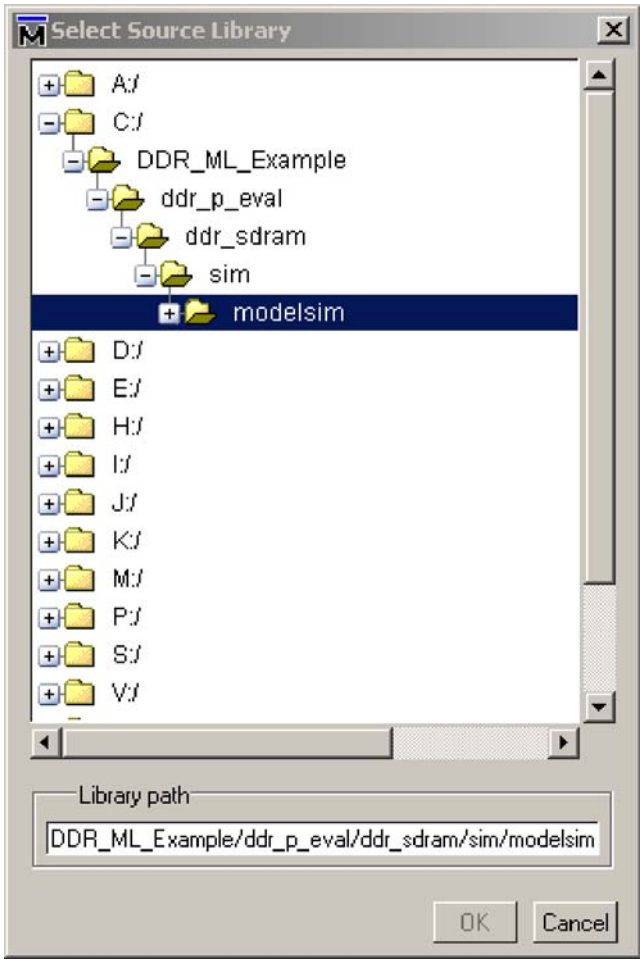
Figure 6. Import Library Wizard Screen Before Import



2.2. Click on the **Browse** button.

The **Select Source Library** screen will appear as shown in Figure 7.

Figure 7. Select Source Library Screen

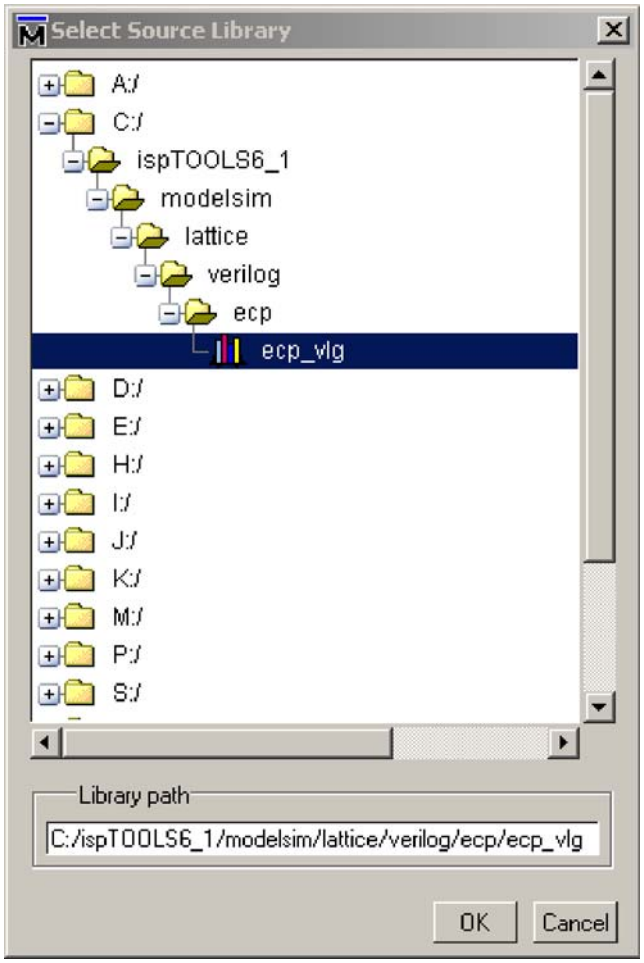


2.3. Navigate to and select the ModelSim Verilog simulation library for the required Lattice device family.

Browse to **C:/ispTOOLS6_1/modelsim/lattice/verilog/ecp/ecp_vlg**.

The **Select Source Library** screen will appear as shown in Figure 8.

Figure 8. Select Source Library Screen After Selecting ecp_vlg



2.4. Click on **OK** to return to the Import Library Wizard.

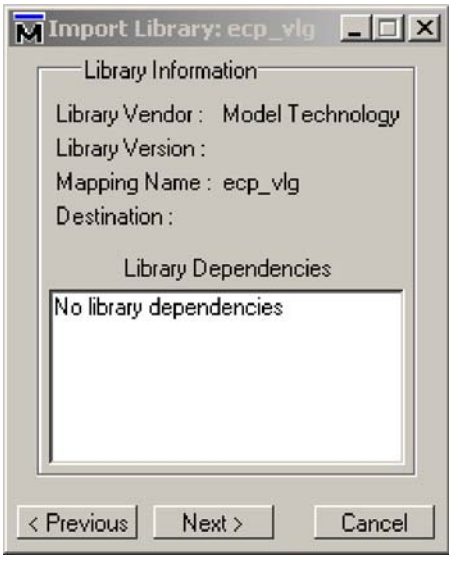
The **Import Library Wizard** screen will appear as shown in Figure 9.

Figure 9. Import Library Wizard Screen After Selecting ecp_vlg Library

2.5. Click on **Next** to open the Import Library screen.

The **Import Library** screen will appear as shown in Figure 10.

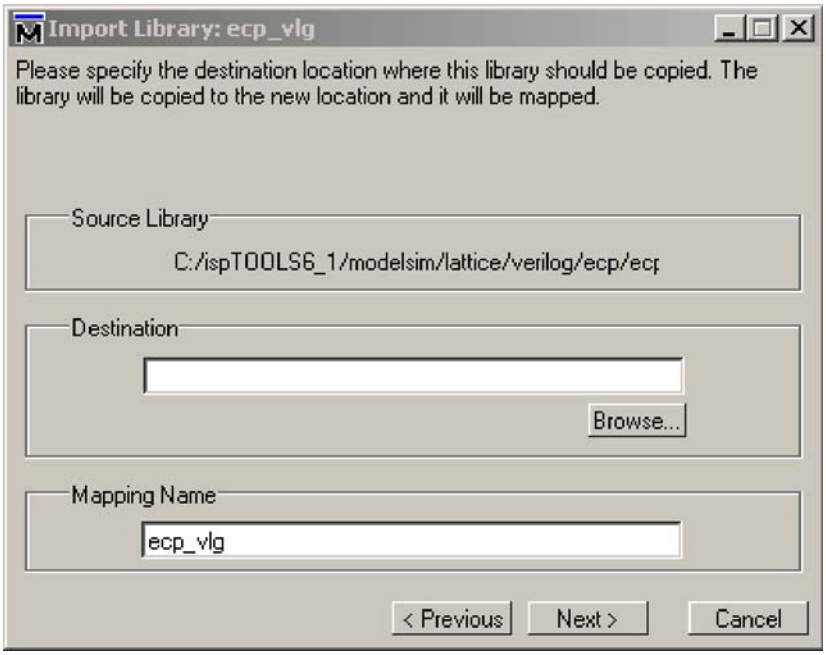
Figure 10. Import Library Screen with ecp_vlg Library



2.6. Click on **Next** to select the destination folder.

The **Import Library** screen will appear as shown in Figure 11.

Figure 11. Import Library Screen

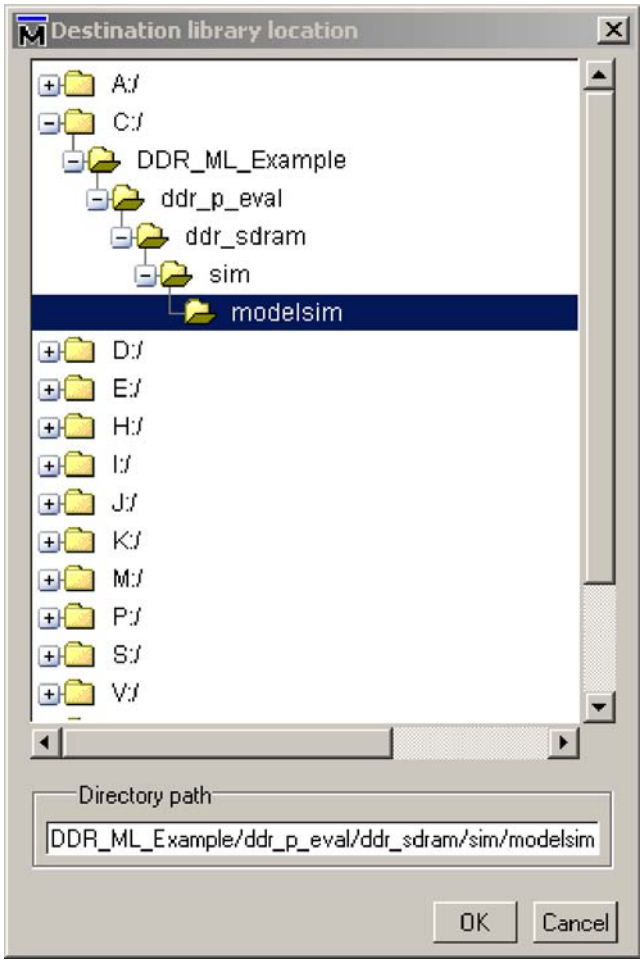


2.7. Browse to select the library destination folder.

Browse to **C:/DDR_ML_Example/ddr_p_eval/ddr_sdram/sim/modelsim**.

The **Destination Library Location** screen will appear as shown in Figure 12.

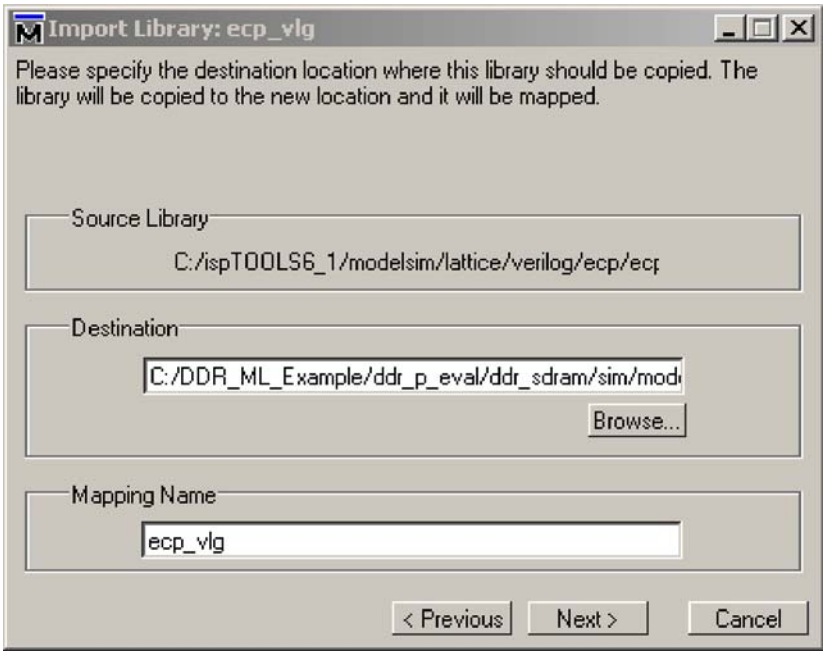
Figure 12. Destination Library Selected



2.8. Click on **OK** to return to the Import Library Wizard.

The **Import Library** screen will appear as shown in Figure 13.

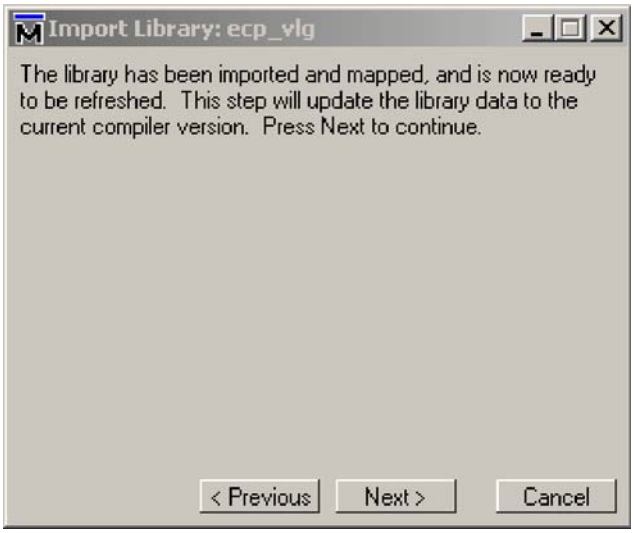
Figure 13. Import Library Screen Showing Destination Path



2.9. Click on **Next** to import and map the simulation library.

The **Import Library** screen will appear as shown in Figure 14.

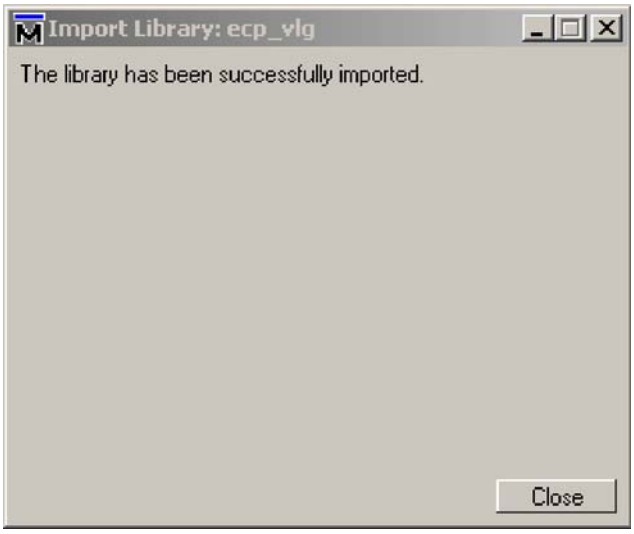
Figure 14. Import Library Screen After Import and Mapping



2.10. Click on **Next** to update the simulation library.

The **Import Library** screen will appear as shown in Figure 15.

Figure 15. Import Library Screen after Updating the Simulation Library



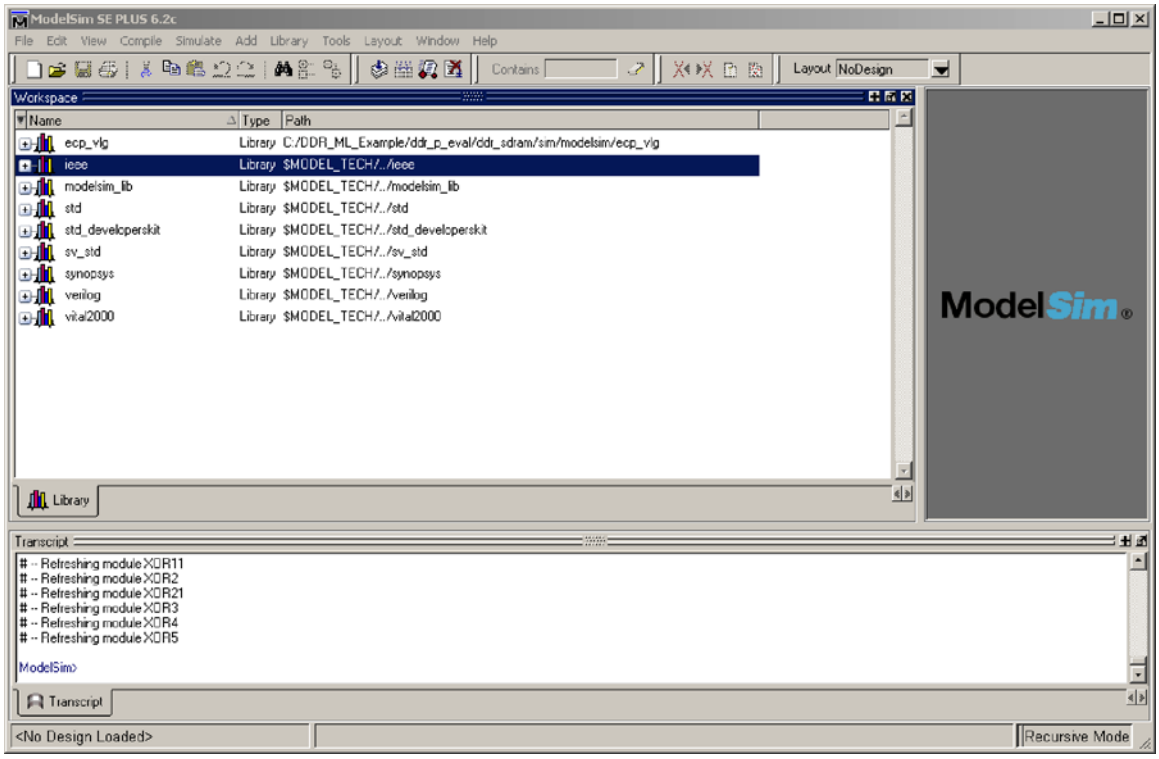
.2.11. Click on **Close** to complete importing the simulation library.

ModelSim will now refresh the importing modules in the library and import it into the project workspace.

2.12. Return to the **ModelSim** screen.

The **ModelSim** screen will now appear as shown in Figure 16.

Figure 16. ModelSim Screen After Successfully Importing the ecp_vlg Simulation Library

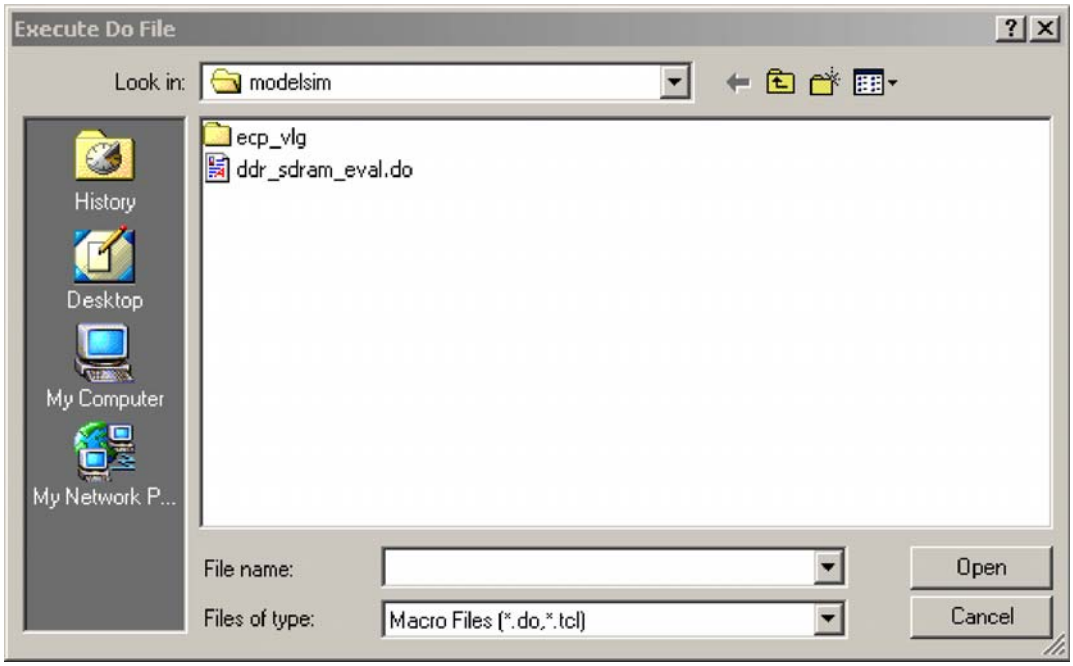


3. Create the work library and run the Verilog simulation.

3.1 Click on **TOOLS->TCL->Execute**

The **Execute Do File** screen will appear as shown in Figure 17.

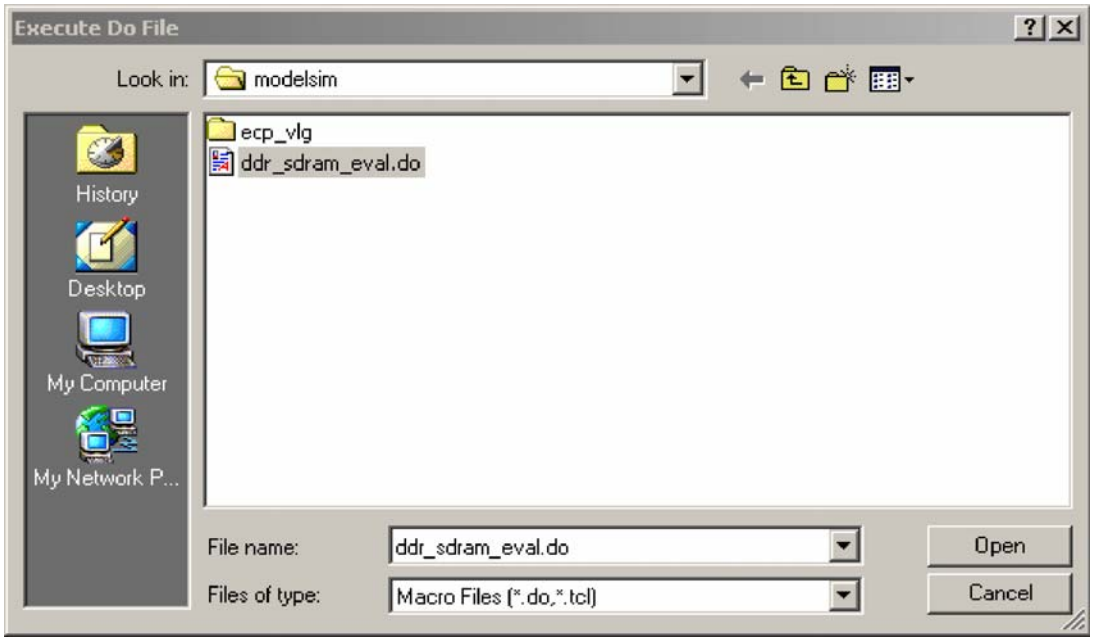
Figure 17. Execute Do File Screen



3.2. Click on **ddr_sdram_eval.do** to select it.

The **Execute Do File** screen will appear as shown in Figure 18.

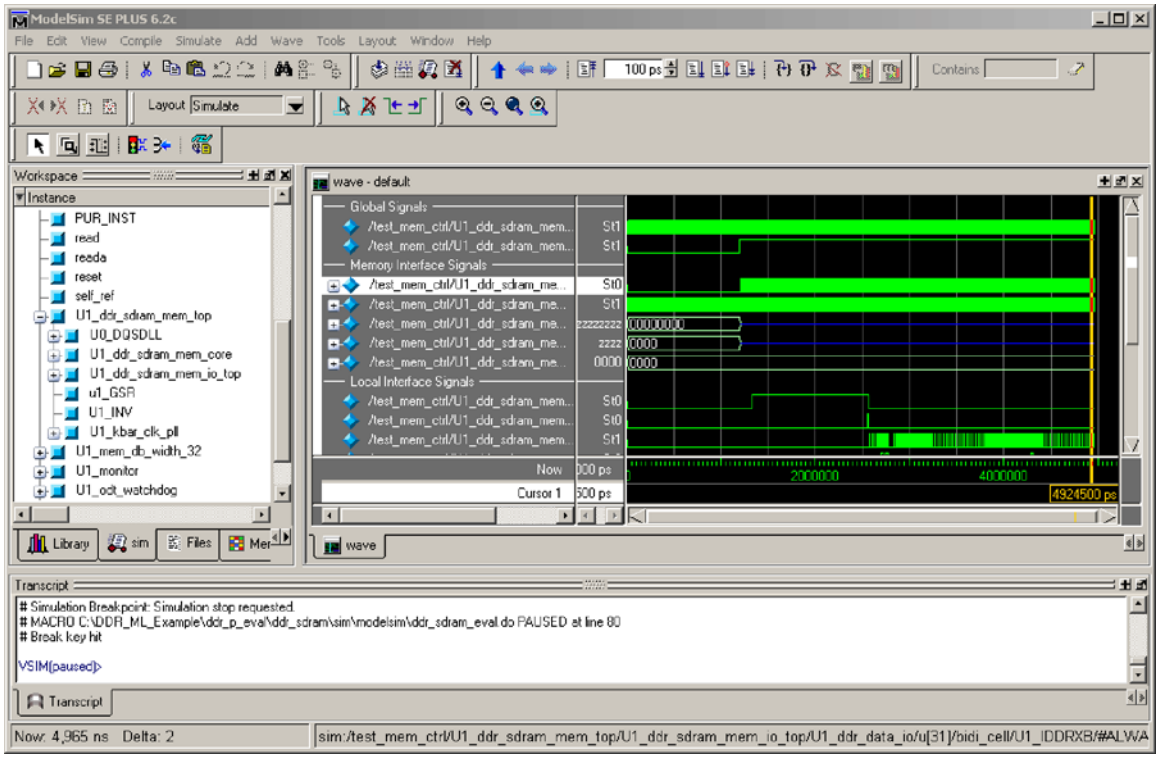
Figure 18. Execute Do File Screen Ready to Execute the Simulation Macro



3.3. Click on **Open** to execute the macro.

The simulator will execute the macro file and run the simulation. The **ModelSim** screen will appear as shown in Figure 19.

Figure 19. ModelSim Screen Displaying Successful Execution of the DDR SDRAM Behavioral Simulation

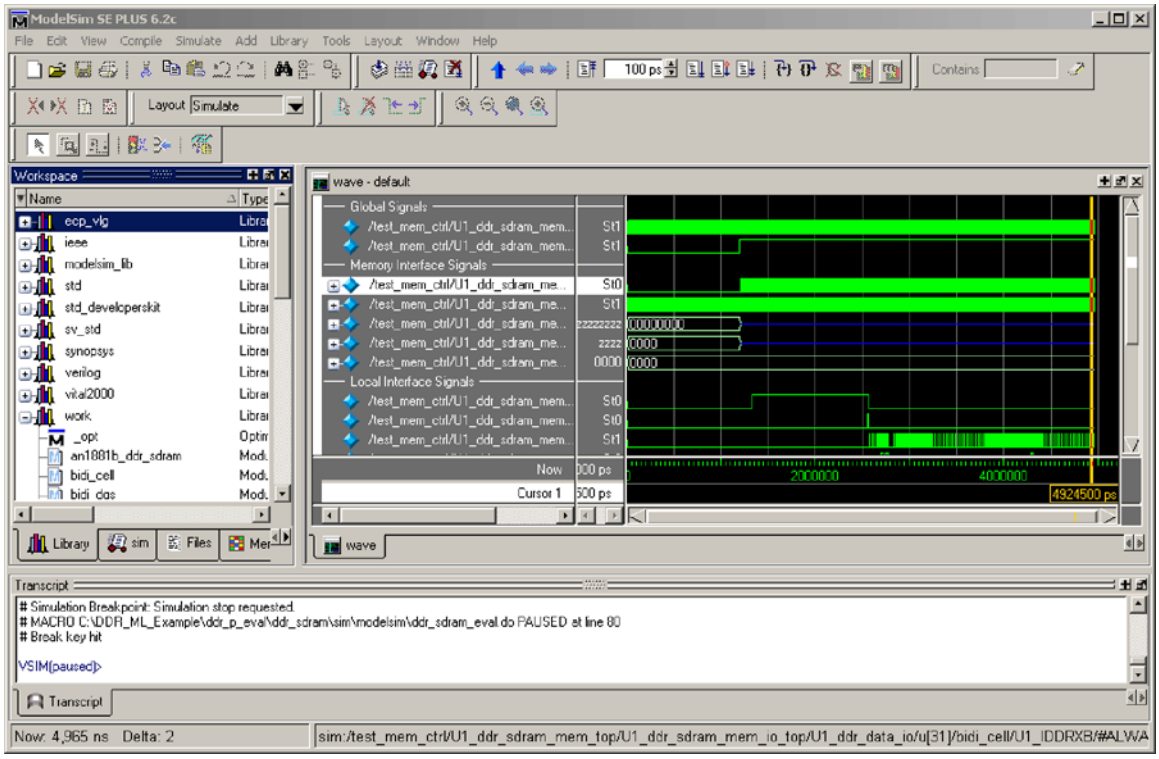


4. Creating the VHDL Top Level Wrapper.

4.1. On the **ModelSim - Workspace - Library** tab, expand the work library by clicking on the **+** sign.

The **ModelSim** opening screen now appears as shown in Figure 20.

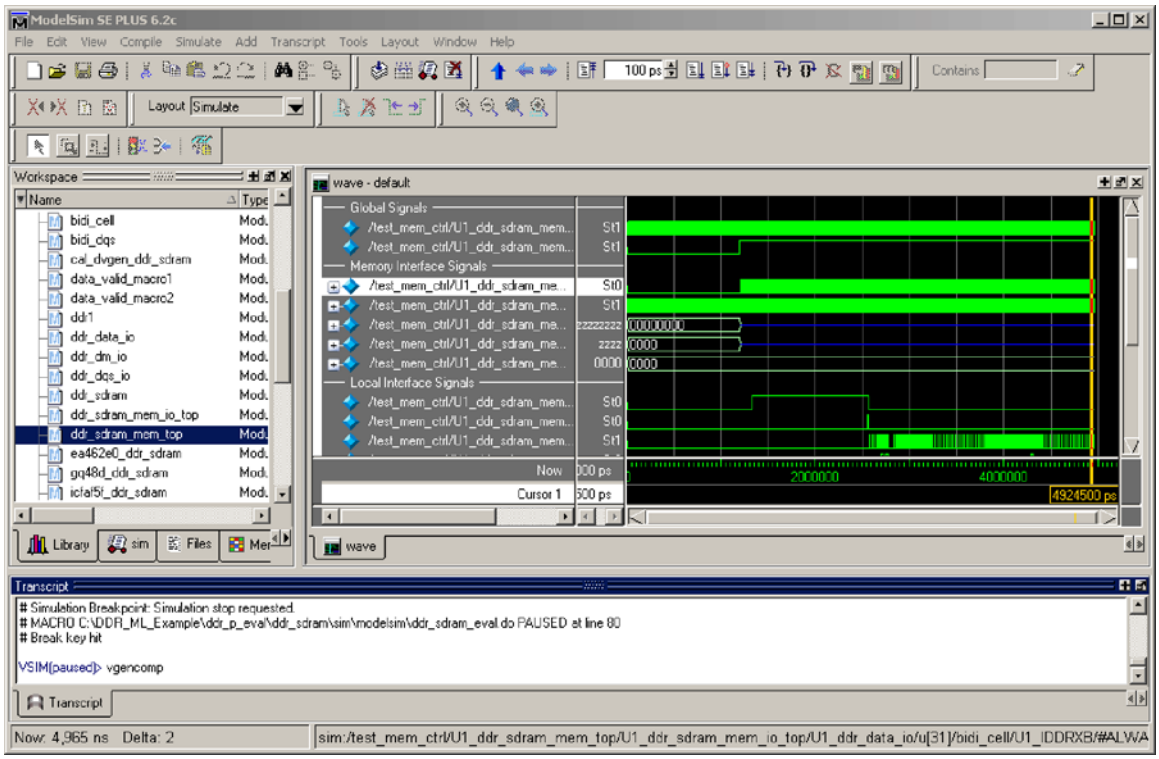
Figure 20. ModelSim screen with expanded work library.



4.2. Scroll down the Library window to click on **ddr_sdram_mem_top** to select it.

The **ModelSim** opening screen now appears as shown in Figure 21.

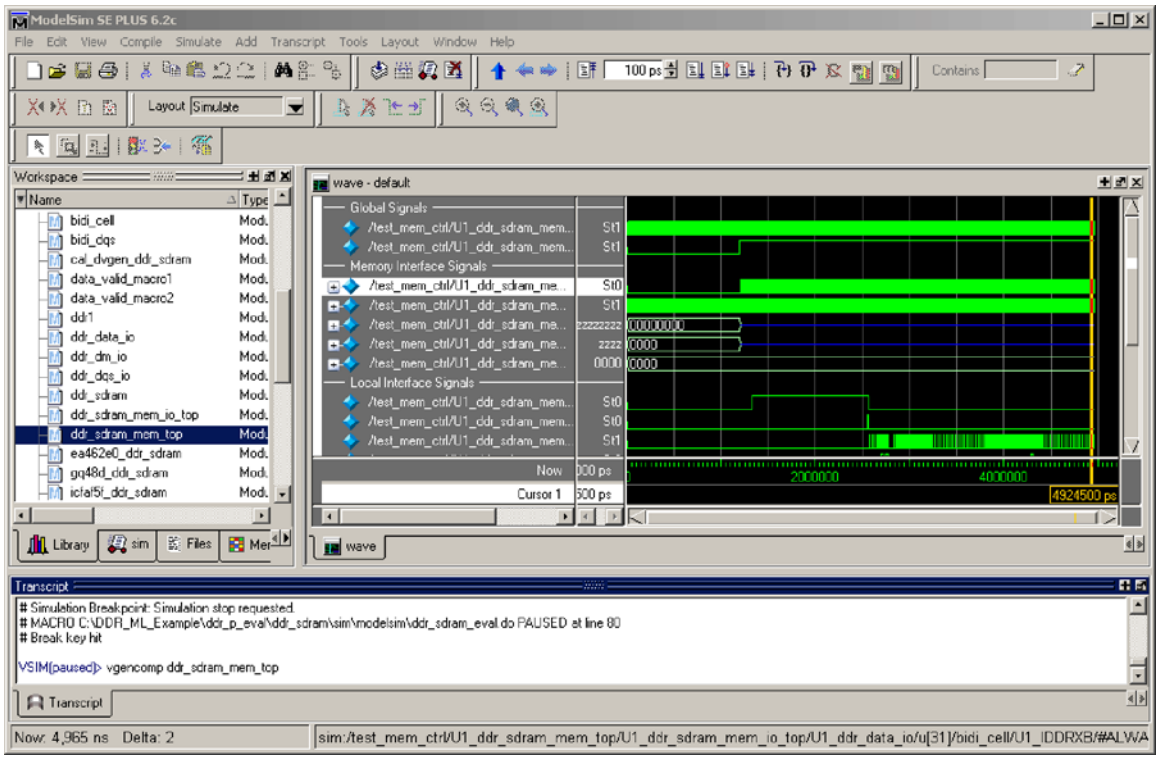
Figure 21. ModelSim Screen with ddr_sdram_mem_top Module Selected



4.3. Type **vgencomp ddr_sdram_mem_top** on the transcript window command line.

The **ModelSim** opening screen now appears as shown in Figure 22.

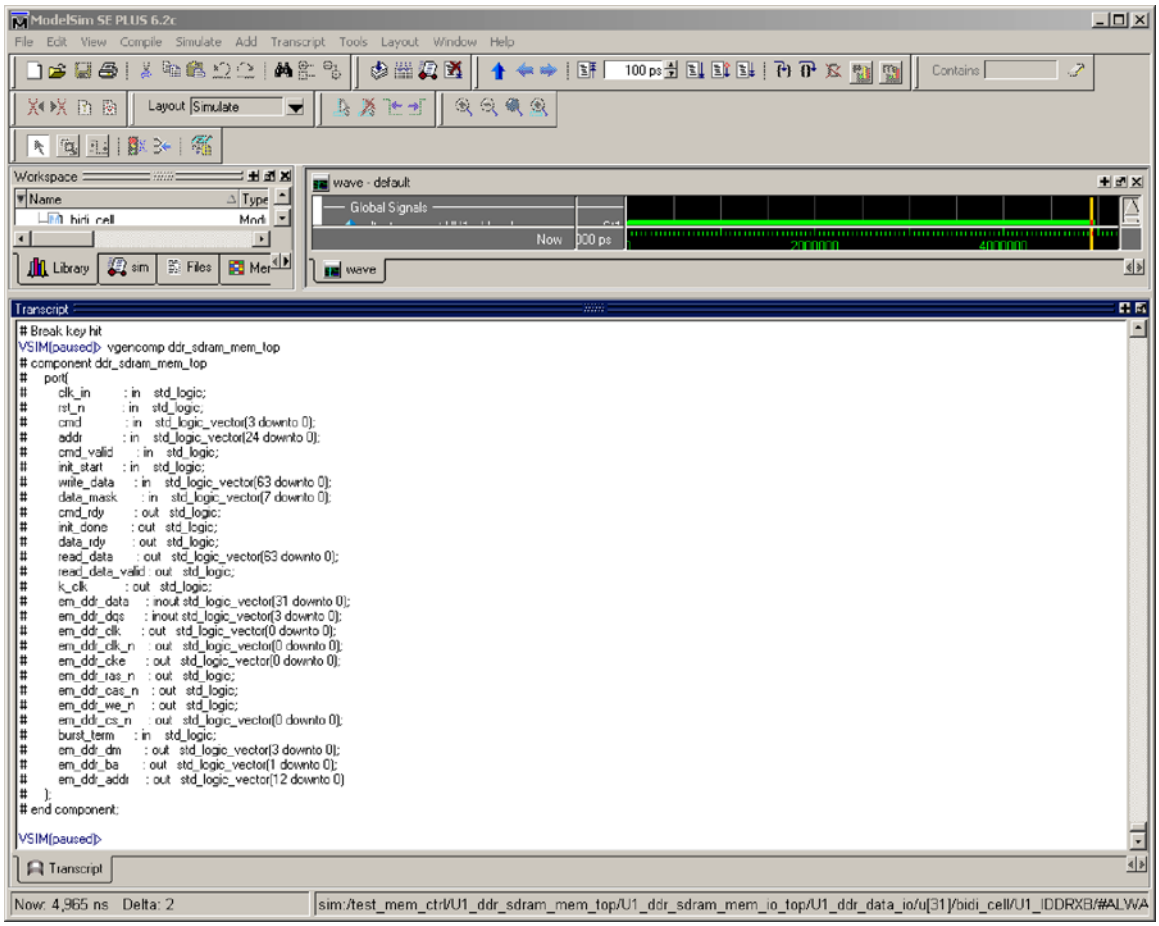
Figure 22. ModelSim screen with "vgencomp ddr_sdram_mem_top" entered.



4.4. Enter a **carriage return** to execute the transcript window command line.

The **ModelSim** opening screen now appears as shown in Figure 23.

Figure 23. ModelSim Screen After Execution of vgencomp.



ModelSim has created a VHDL component port map that directly maps the ddr_sdram_mem_top Verilog module. The component port map is shown in the Figure 24.

Note that the VHDL component port map is configuration-specific. Making any changes to the configuration in IPexpress will also require matching changes to the component and its associated Entity/Architecture pair.

Figure 24. ModelSim Created VHDL Component Port Map

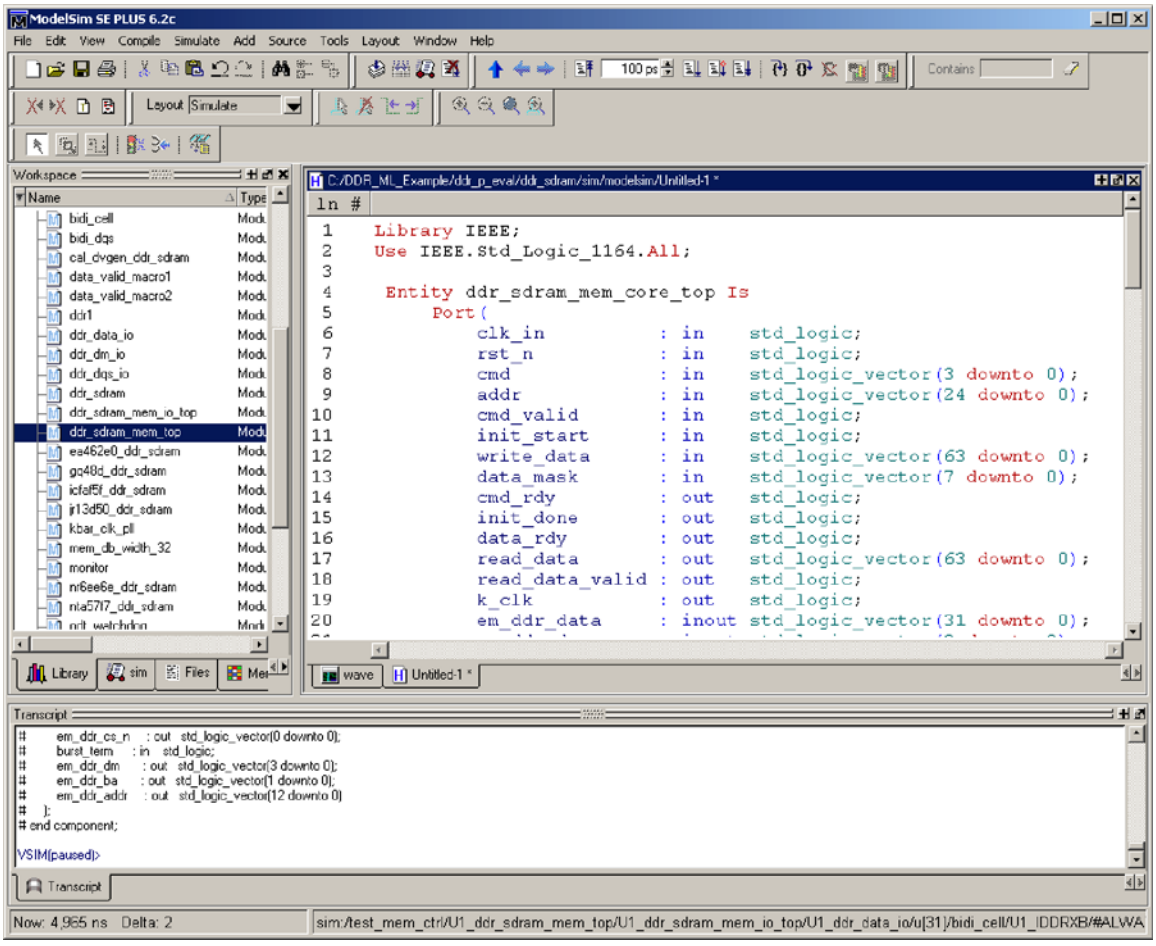
```
# component ddr_sdram_mem_top
#    port(
#       clk_in         : in    std_logic;
#       rst_n          : in    std_logic;
#       cmd            : in    std_logic_vector(3 downto 0);
#       addr           : in    std_logic_vector(24 downto 0);
#       cmd_valid      : in    std_logic;
#       init_start     : in    std_logic;
#       write_data     : in    std_logic_vector(63 downto 0);
#       data_mask      : in    std_logic_vector(7 downto 0);
#       cmd_rdy        : out   std_logic;
#       init_done      : out   std_logic;
#       data_rdy       : out   std_logic;
#       read_data      : out   std_logic_vector(63 downto 0);
#       read_data_valid : out   std_logic;
#       k_clk          : out   std_logic;
#       em_ddr_data    : inout std_logic_vector(31 downto 0);
#       em_ddr_dqs     : inout std_logic_vector(3 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_cke     : out   std_logic_vector(0 downto 0);
#       em_ddr_ras_n   : out   std_logic;
#       em_ddr_cas_n   : out   std_logic;
#       em_ddr_we_n    : out   std_logic;
#       em_ddr_cs_n    : out   std_logic_vector(0 downto 0);
#       burst_term     : in    std_logic;
#       em_ddr_dm      : out   std_logic_vector(3 downto 0);
#       em_ddr_ba      : out   std_logic_vector(1 downto 0);
#       em_ddr_addr    : out   std_logic_vector(12 downto 0)
#    );
# end component;
```

4.5. Implementing the Top-Level Wrapper.

The next step is to create a VHDL entity-architecture pair using the ModelSim generated component port map.

Edit a new VHDL source file and add the source code provided in the VHDL Wrapper Listing in this document. The ModelSim screen is shown in Figure 25. Note that the component declaration has been copied into the VHDL file and the entity/architecture pair have been built around it.

Figure 25. ModelSim Screen with New VHDL Wrapper in Text Editor



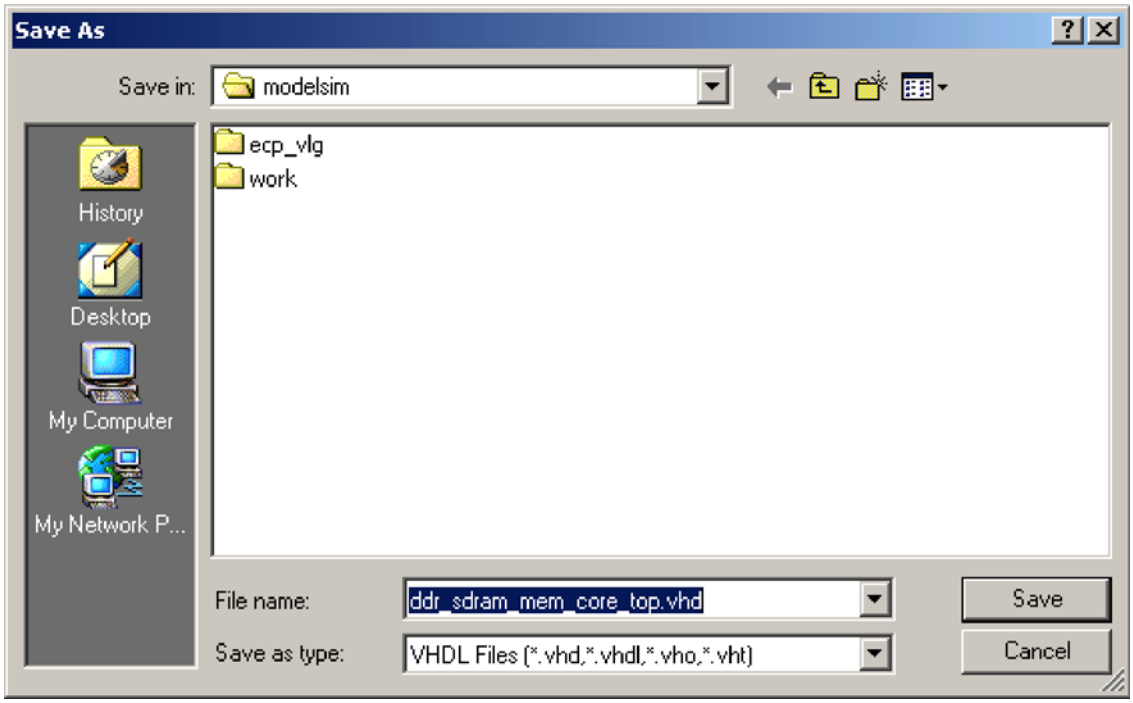
4.6. Saving the new VHDL Wrapper.

The next step is to save the new VHDL Wrapper in the top-level source directory

4.6.1. Click on **File->Save As** and enter **ddr_sdram_mem_core_top.vhd** into the file name text box.

The new ModelSim screen is shown in Figure 26.

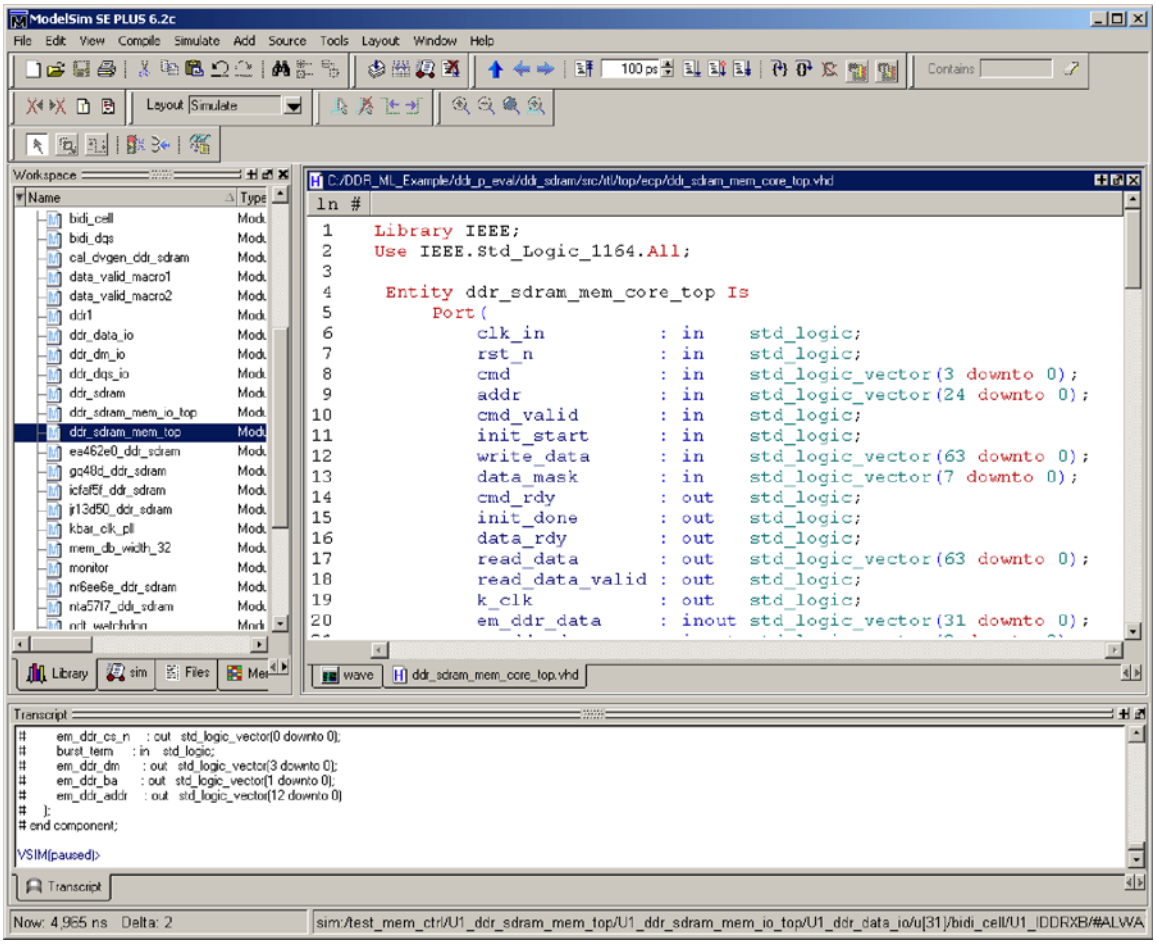
Figure 26. Save As ddr_sdram_mem_core_top.vhd.



4.6.2. Browse to the directory **C:\DDR_ML_Example\ddr_p_eval\ddr_sdram\src\rtl\top\ecp** and click on **Save** to save the new VHDL Wrapper.

The new ModelSim screen is shown in Figure 27.

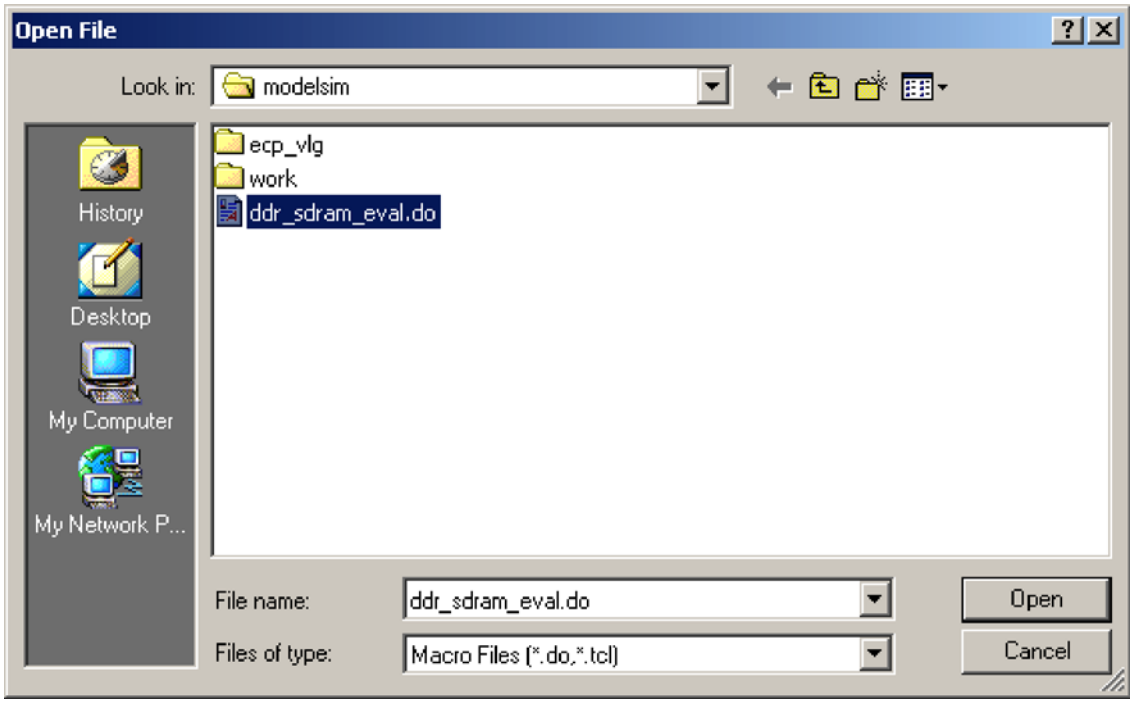
Figure 27. ModelSim Screen with Saved VHDL Wrapper



4.7. Adding the new VHDL Wrapper to the work library.

The next step is to add the new VHDL Wrapper to the work library. This is done by modifying the ddr_sdram_eval.do macro file. Click on **File->Open** to open the **Open File** window. Click on the selection arrow next to the File Type text entry box and select **Macro Files** (*.do,*.tcl). Select **ddr_sdram_eval.do**. The **Open File** screen is shown in Figure 28.

Figure 28. Open File Window with Macro File Showing

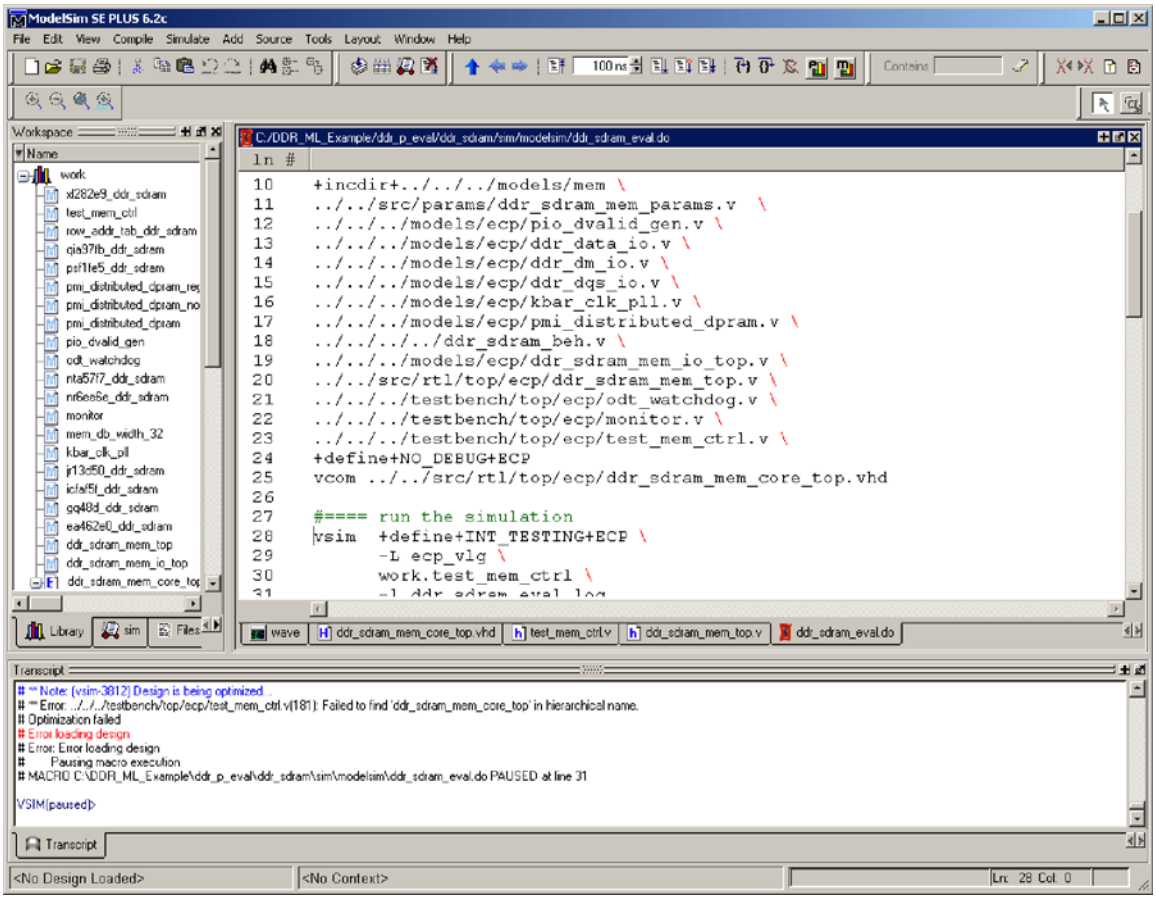


Click on **Open** to open the file in the text editor pane.

Add the following text to the file on line 25: **vcom ../../src/rtl/top/ecp/ddr_sdram_mem_core_top.vhd**

The ModelSim window with the modified file will appear as shown in Figure 29.

Figure 29. Modified VHDL Wrapper file.

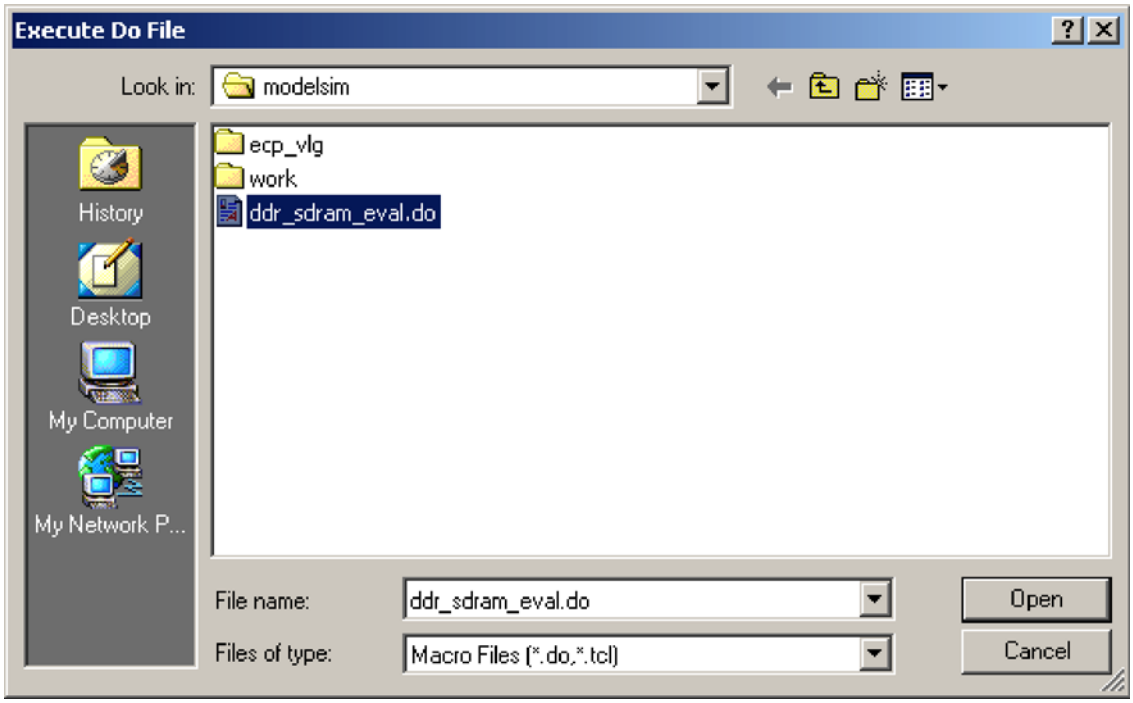


Click on **File->Save** to save the edited file.

Click on **Tools->TCL->Execute Macro** and select **ddr_sdram_eval.do**.

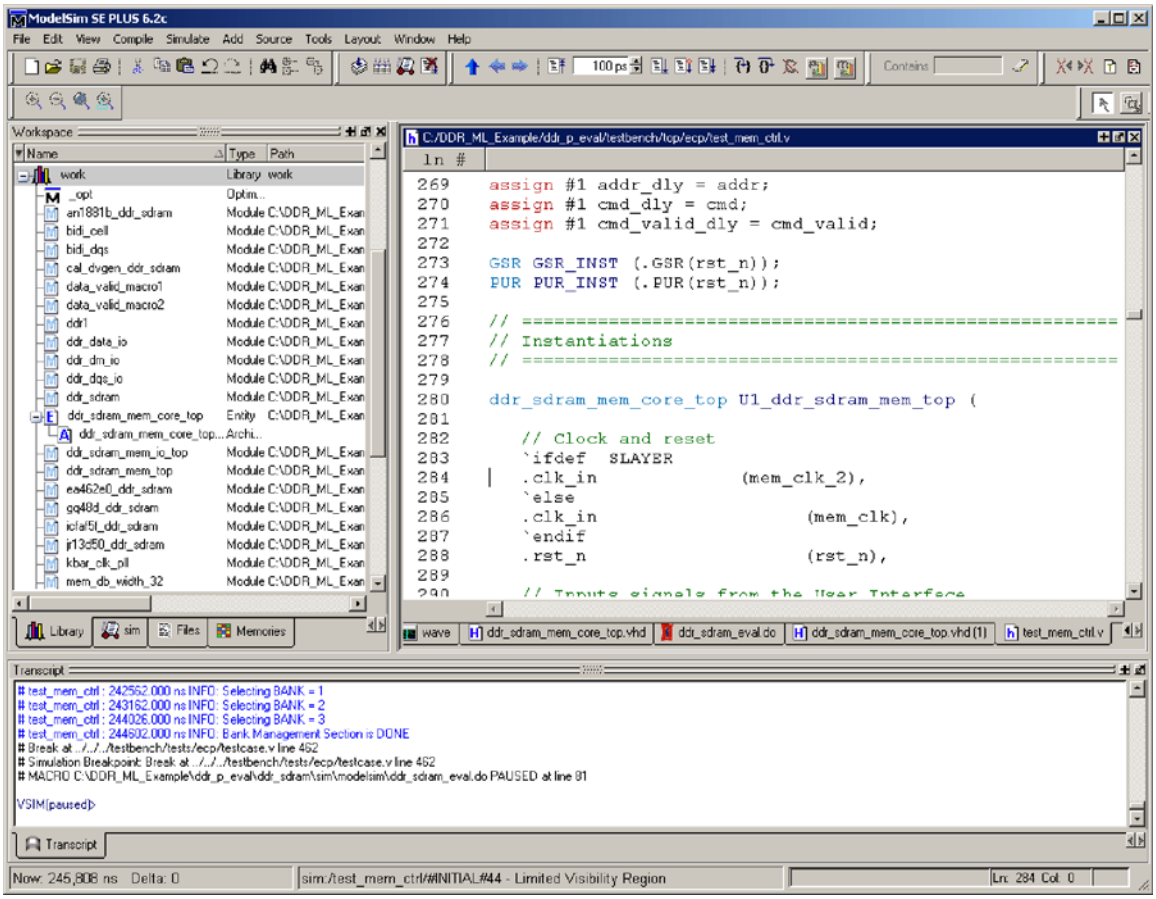
The **Execute Do File** screen will appear as shown in Figure 30.

Figure 30. Execute Do File Screen with ddr_sdram_eval.do Selected



Click on **Open** to execute the macro file and compile the VHDL Wrapper into the work library. After compiling, the simulator runs the simulation without using the new VHDL Wrapper. A new entry ddr_sdram_mem_core_top is created in the work library. The ModelSim screen will appear as shown in Figure 31. Note the addition of the ddr_sdram_mem_core_top Entity/Architecture pair.

Figure 31. ModelSim Window with 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

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;

 Entity ddr_sdram_mem_core_top Is
     Port(
         clk_in          : in    std_logic;
         rst_n           : in    std_logic;
         cmd             : in    std_logic_vector(3 downto 0);
         addr            : in    std_logic_vector(24 downto 0);
         cmd_valid       : in    std_logic;
         init_start      : in    std_logic;
         write_data      : in    std_logic_vector(63 downto 0);
         data_mask       : in    std_logic_vector(7 downto 0);
         cmd_rdy         : out   std_logic;
         init_done       : out   std_logic;
         data_rdy        : out   std_logic;
         read_data       : out   std_logic_vector(63 downto 0);
         read_data_valid : out   std_logic;
         k_clk           : out   std_logic;
         em_ddr_data     : inout std_logic_vector(31 downto 0);
         em_ddr_dqs      : inout std_logic_vector(3 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_cke      : out   std_logic_vector(0 downto 0);
         em_ddr_ras_n    : out   std_logic;
         em_ddr_cas_n    : out   std_logic;
         em_ddr_we_n     : out   std_logic;
         em_ddr_cs_n     : out   std_logic_vector(0 downto 0);
         burst_term      : in    std_logic;
         em_ddr_dm       : out   std_logic_vector(3 downto 0);
         em_ddr_ba       : out   std_logic_vector(1 downto 0);
```

```
        em_ddr_addr     : out   std_logic_vector(12 downto 0)
    );
end ddr_sdram_mem_core_top;

architecture ddr_sdram_mem_core_arch of ddr_sdram_mem_core is

component ddr_sdram_mem_core -- created by vgencomp
    port(
        clk_in          : in    std_logic;
        rst_n           : in    std_logic;
        cmd             : in    std_logic_vector(3 downto 0);
        addr            : in    std_logic_vector(24 downto 0);
        cmd_valid       : in    std_logic;
        init_start      : in    std_logic;
        write_data      : in    std_logic_vector(63 downto 0);
        data_mask       : in    std_logic_vector(7 downto 0);
        cmd_rdy         : out   std_logic;
        init_done       : out   std_logic;
        data_rdy        : out   std_logic;
        read_data       : out   std_logic_vector(63 downto 0);
        read_data_valid : out   std_logic;
        k_clk           : out   std_logic;
        em_ddr_data     : inout std_logic_vector(31 downto 0);
        em_ddr_dqs      : inout std_logic_vector(3 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_cke      : out   std_logic_vector(0 downto 0);
        em_ddr_ras_n    : out   std_logic;
        em_ddr_cas_n    : out   std_logic;
        em_ddr_we_n     : out   std_logic;
        em_ddr_cs_n     : out   std_logic_vector(0 downto 0);
        burst_term      : in    std_logic;
        em_ddr_dm       : out   std_logic_vector(3 downto 0);
        em_ddr_ba       : out   std_logic_vector(1 downto 0);
        em_ddr_addr     : out   std_logic_vector(12 downto 0)
    );
end component;

begin

sdram_mem_core_inst: ddr_sdram_mem_core
    port map(
        clk_in          => clk_in,
        rst_n           => rst_n,
        cmd             => cmd,
        addr            => addr,
        cmd_valid       => cmd_valid,
        init_start      => init_start,
        write_data      => write_data,
        data_mask       => data_mask,
        cmd_rdy         => cmd_rdy,
        init_done       => init_done,
        data_rdy        => data_rdy,
        read_data       => read_data,
```

```
        read_data_valid => read_data_valid,
        k_clk           => k_clk,
        em_ddr_data     => em_ddr_data,
        em_ddr_dqs      => em_ddr_dqs,
        em_ddr_clk      => em_ddr_clk,
        em_ddr_clk_n    => em_ddr_clk_n,
        em_ddr_cke      => em_ddr_cke,
        em_ddr_ras_n    => em_ddr_ras_n,
        em_ddr_cas_n    => em_ddr_cas_n,
        em_ddr_we_n     => em_ddr_we_n,
        em_ddr_cs_n     => em_ddr_cs_n,
        burst_term      => burst_term,
        em_ddr_dm       => em_ddr_dm,
        em_ddr_ba       => em_ddr_ba,
        em_ddr_addr     => em_ddr_addr
    );

end ddr_sdram_mem_core_arch;
```

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.