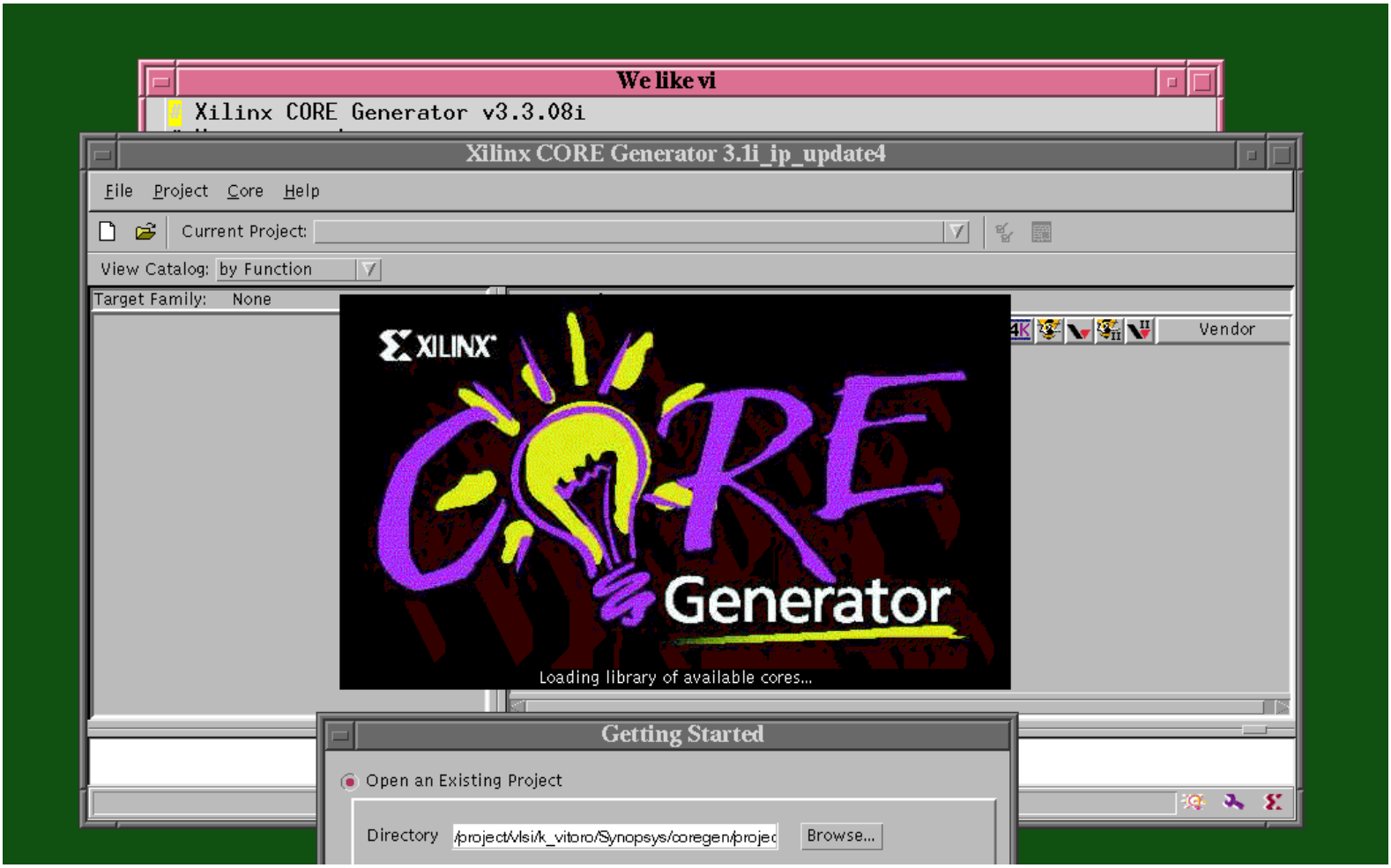
USING COREGEN TO SYNTHESIZE COMPONENTS TOWARDS A VIRTEX TECHNOLOGY.

A TUTORIAL



Developed By
Konstantinos Vitoroulis

Advised By
Tadeusz S. Obuchowicz
VLSI Engineer/CAD Specialist

Electrical and Computer Engineering,
Concordia University,
July 2001.

Trademarks
Spartan, Virtex are trademarks of Xilinx, Inc.
Alliance Series,AllianceCORE,CORE Generator, LogiCOREare trademarks of Xilinx, Inc.
VSS, DC Professional are trademarks of Synopsys, Inc.

Table of contents:

USING COREGEN TO SYNTHESIZE COMPONENTS TOWARDS A VIRTEX TECHNOLOGY.

1. Quick Introduction: Cores, CORE Generator (Coregen) and related terminology.

Xilinx introduced the *Xilinx IP (Intellectual Property) Center* in order to facilitate and improve the design process with their line of software and hardware products. The Xilinx IP Center offers an extensive variety of configurable, predesigned components that are optimized for the Xilinx FPGA families. These components can be incorporated into your designs improving both performance and design time. The Xilinx IP Center is available on the web at address:

`www.xilinx.com/ipcenter`

A *core*, also referred to as an IP core, is a pre-made component that can be used directly in your HDL design. Usually the available cores are optimized for time and/or space performance. Cores can be configured to suit your design's requirements. There are two divisions of available cores, namely the LogiCOREs and the AllianceCOREs.

LogiCOREs are those cores provided free by Xilinx. Available LogiCORE components range from simple gate components to memory components, filters, networking components, image processing components and many others.

AllianceCORE components are components contributed by third party developers. Again a very large library of components exist under the AllianceCORE program.

A third source of core-related information can be found under the Reference Designs collection available by Xilinx. Though no components are available through Reference Designs, useful information on how optimization is achieved can be found here. This information will help you optimize more efficiently your own custom designs. The Reference Designs are available at:

`www.xilinx.com/products/logicore/refdes.htm`

The CORE Generator (Coregen) tool allows one to customize and generate an available core. Coregen provides a list of the available cores installed on the

system and an easy to use Graphical User Interface for customization and generation.

A final note: The available core and reference designs database is always being updated. It is recommended for someone who wishes to utilize a core to search on the Xilinx IP Center web page (provided above) for core availability.

2. About this tutorial

This tutorial is intended to be a quick and complete guide on how to use Coregen to integrate core components in a VHDL design targeting a Xilinx Virtex/VirtexE architecture.

The procedures involved are presented through an example where a memory component is generated, simulated (both RTL and gate level simulations are performed) synthesized and implemented. Screen captions are provided to illustrate every step of the procedure. Usually these captions will indicate clearly the options that you will have to select.

The simulation and synthesis steps are performed with the Synopsys VSS simulator and the Synopsys Design Compiler tools. Implementation is achieved with the Xilinx Alliance software.

Familiarity with those tools is essential in order to be able to complete this guide successfully. Refer to Digital Logic Synthesis Using Synopsys and Xilinx - A Tutorial located at www.ece.concordia.ca/Documentation/synopsys/tutorial.html *if you are not comfortable using those tools. Also basic knowledge of the UNIX operating system is assumed* .

Throughout this guide you will need to create files, generate directories and copy files to several locations that probably do not exist in your current directory structure. In the Appendix a suggested directory structure that you may use (or adapt to your existing directory structure) in order to avoid confusion is provided.

3. Running Coregen

The file `xilinx.vM3.1i.env` must be sourced before one can run any Xilinx tools. The file is located at:

```
/CMC/ENVIRONMENT/xilinx.vM3.1i.env
```

and can either be copied to another location or sourced directly from there.

```
source /CMC/ENVIRONMENT/xilinx.vM3.1i.env
coregen &
```

Note: The filename provided here might change in the case of a software upgrade. This applies for many other full path names in this tutorial.

4. Creating a New Project

A project can be viewed as a directory in your account where Coregen will generate a set of files for a specific component. Several components can exist within a project directory

Once Coregen is invoked, a window titled `'Getting Started'` will open (Figure 1). Select `'Create a New Project'` and press on the `'OK'` button. In the new window that will appear (Figure 2); specify the working directory where the project will be saved and select the following options:

```
Target Architecture: Virtex
Design Entry:  VHDL Synopsys
```

Select `'OK'`. Coregen will offer to create the specified directory if it does not already exist. If any warnings appear at this point in a new window, read and ignore them.

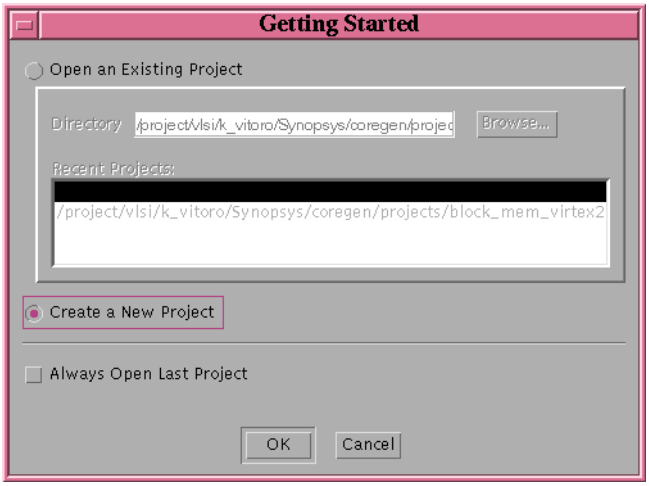


Figure 1: 'Getting Started' with a new project.

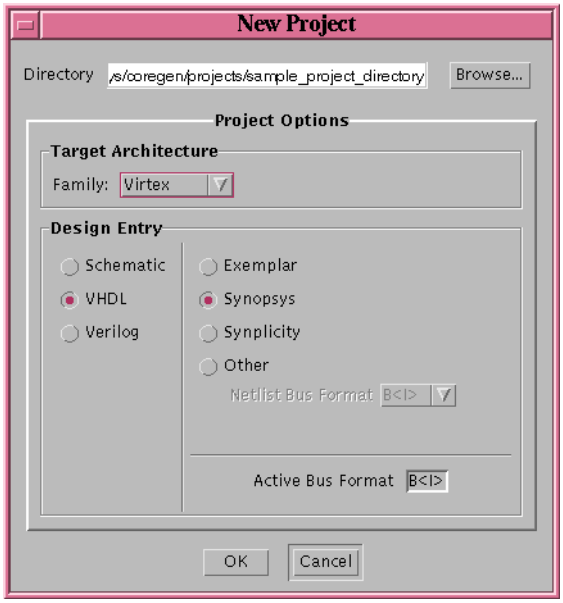


Figure 2: New Project window**.**

A project can also be created by selecting the `'New...'` option in the `'Project'` menu. In the window that will appear, select the options specified above. Figure 3, shows the Coregen main.

5. Selecting a component.

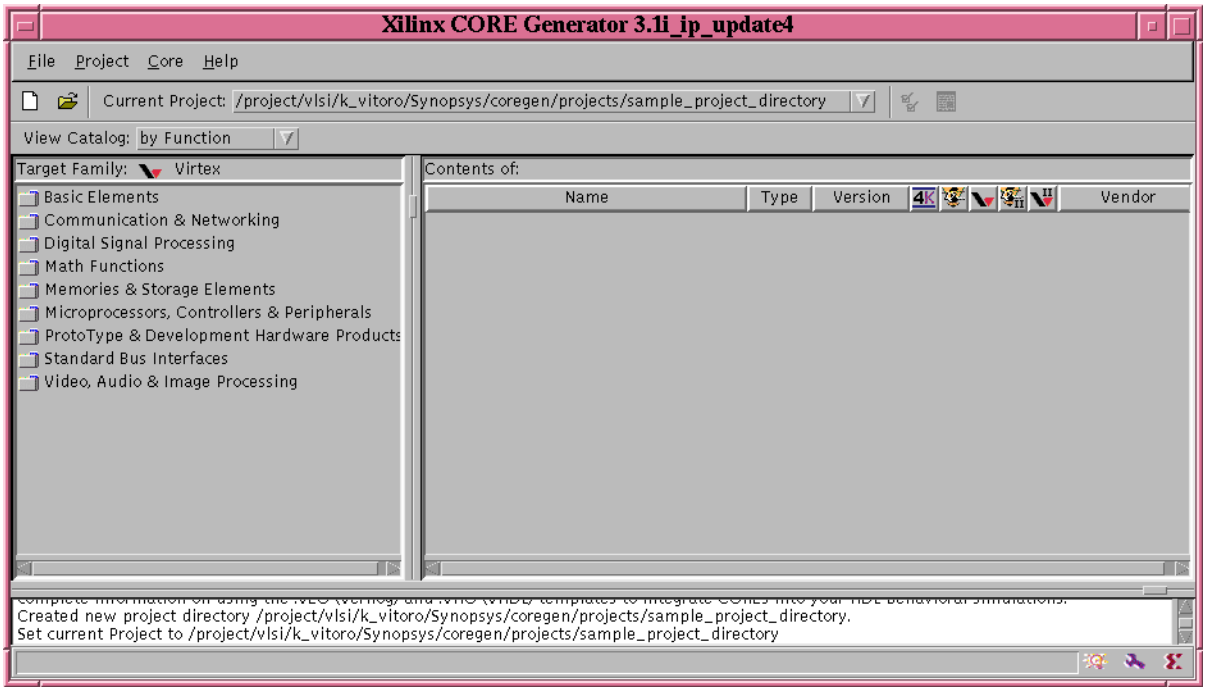


Figure 3: Xilinx CORE Generator main window.

In this window, (Figure 3 above) locate the `'View Catalog'` pull down menu (near the top left corner of the window) and select the method of displaying the components. The available options are:

```
by Function
Alphabetically
by Vendor
by Family
by Type
```

If not selected already choose `'by Function'`. On the left side of the Coregen window the component categories can be selected.

In this tutorial we will be implementing a small, random access memory (RAM) component to illustrate the general procedures involved in creating and using Coregen generated components in a design.

Double click on the 'Memories & Storage Elements' entry and then on newly listed 'RAMs & ROMs' entry (Figure 4).

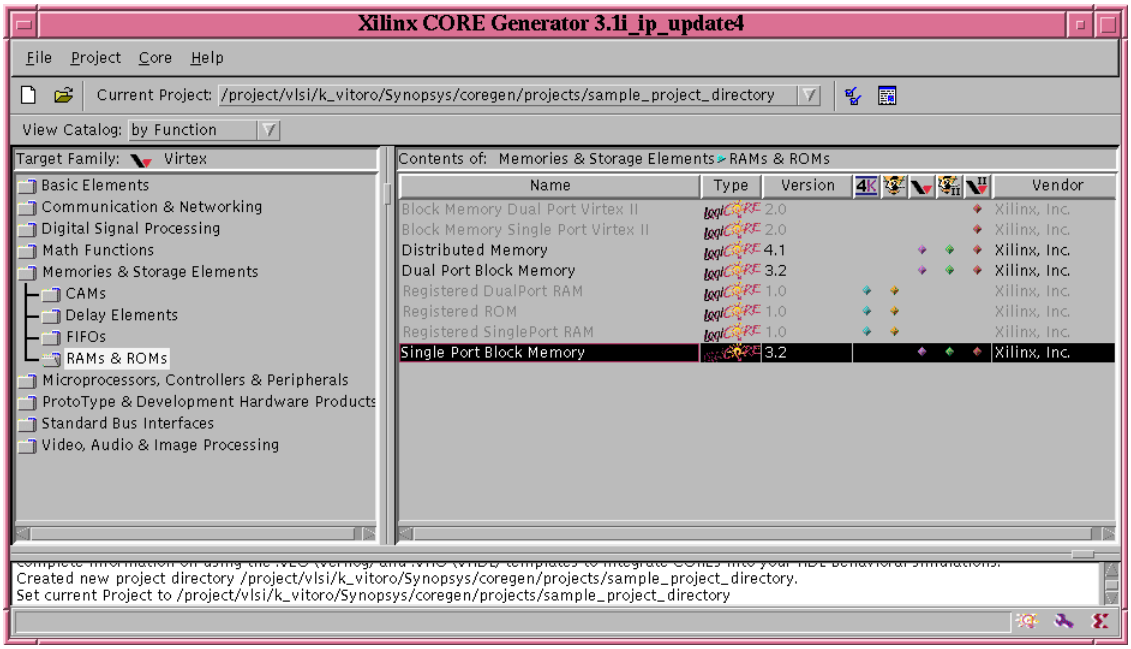


Figure 4: Specifying the component you want to generate.

The available components will be displayed in the right-hand side of the window. Notice that some of the component names appear in a light grey color which indicates that those components are not available for the selected FPGA family.

Locate the component named 'Single Port Block Memory' and verify that the version number next to the name is '3.2'. It is important to ensure that the version of the Coregen 'Single Port Block Memory' component is 3.2, as different versions will most likely require different customization options than what is specified in this guide.

When the name of a component is shaded then that component is not available for the selected family and it cannot be selected. The colored diamond shapes indicate the FPGA families for which the component is available. The 'Single Port Block Memory' component, for example, is available for the Virtex, Spartan II and Virtex II technologies.

Double click on the name of the component `'Single Port Block Memory'`, version 3.2, in order to select it and a new window will appear. This is the Component Customization window (Figure 5).

6. Customizing a component.

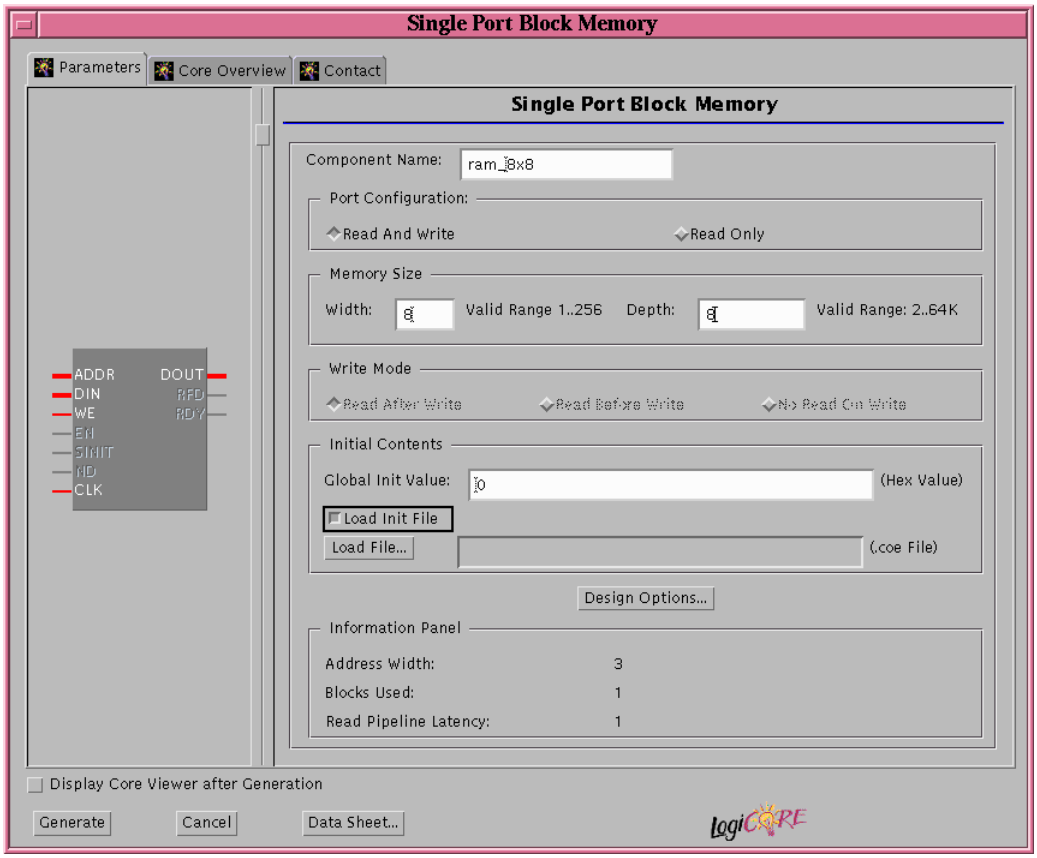


Figure 5: The Component Customization window.

The parameters of the selected component are specified in this window. The available options differ depending on the type and version of a component. For complete documentation on a component refer to the corresponding data sheet, which can be viewed by selecting the button `'Data Sheet...'` located near the middle of the bottom on the component customization window.

To customize the Single Port Block Memory, set the following values in the appropriate fields:

```
Component Name:ram_8x8
Port Configuration:Read And Write
Memory Size: Width: 8  Depth: 8
```

To initialize the contents of a memory (and this applies for either RAMs ,ROMs or any other components that requires initialization) an appropriate text file with extension .coe is required. Change to the project directory (the one created according to the steps in section 2) and with a text editor of your choice (Figure 6) create the file: ram_8x8_init.coe containing the following:

```
MEMORY_INITIALIZATION_RADIX=2;
MEMORY_INITIALIZATION_VECTOR=
00000000,
00000001,
00000010,
00000011,
00000100,
00000101,
00000110,
00000111;
```

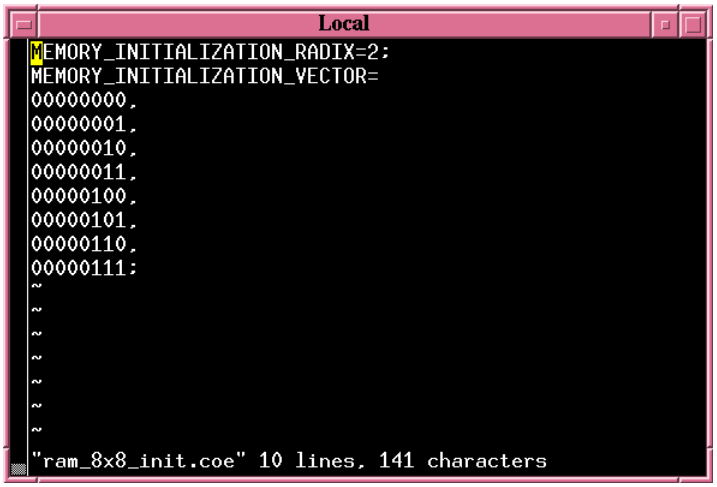


Figure 6: Use your preferred editor to edit the .coe file.

In the `'initial contents'` field check the option `'Load Init File'` and select the `'Load File...'` button. A new dialog box will appear for the .coe (Figure 7) file selection. Specify the full path to the .coe file.

The MEMORY_INITIALIZATION_RADIX keyword seen in the file content specifies that the initial data provided in the MEMORY_INITIALIZATION_VECTOR command will be in binary format. Other possible formats are:

```
MEMORY_INITIALIZATION_RADIX=10;  for decimal
MEMORY_INITIALIZATION_RADIX=16;  for hexadecimal
```

The MEMORY_INITIALIZATION_VECTOR is followed by the data values separated by commas, terminated with a semicolon.

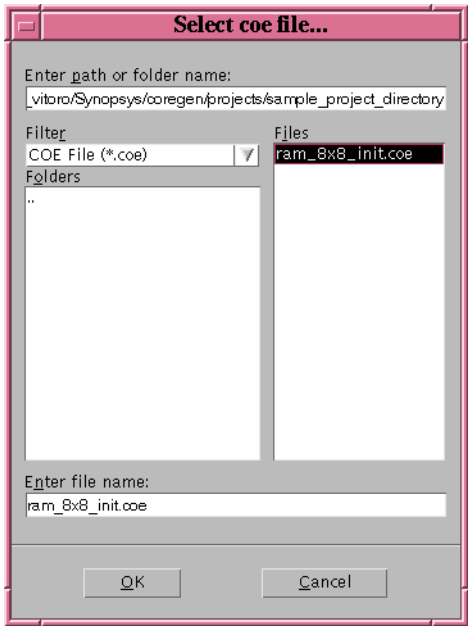


Figure 7: Selecting the .coe file.

If the file was read correctly (Figure 8) by the tool, no error messages will appear and the text field next to the `'Load File...'` button will contain the full path to the .coe file in black color. If any errors occurred during the translation of the .coe file, then an error message will appear and the textbox with the .coe file location will contain red characters. In the case of errors you need to correct the file. A .coe file is needed in order to provide initial values for a component. Refer to the component data sheet for further details regarding the syntax of the initialization for a particular component (e.g. the coefficients for a FIR filter).

7. Generating a component

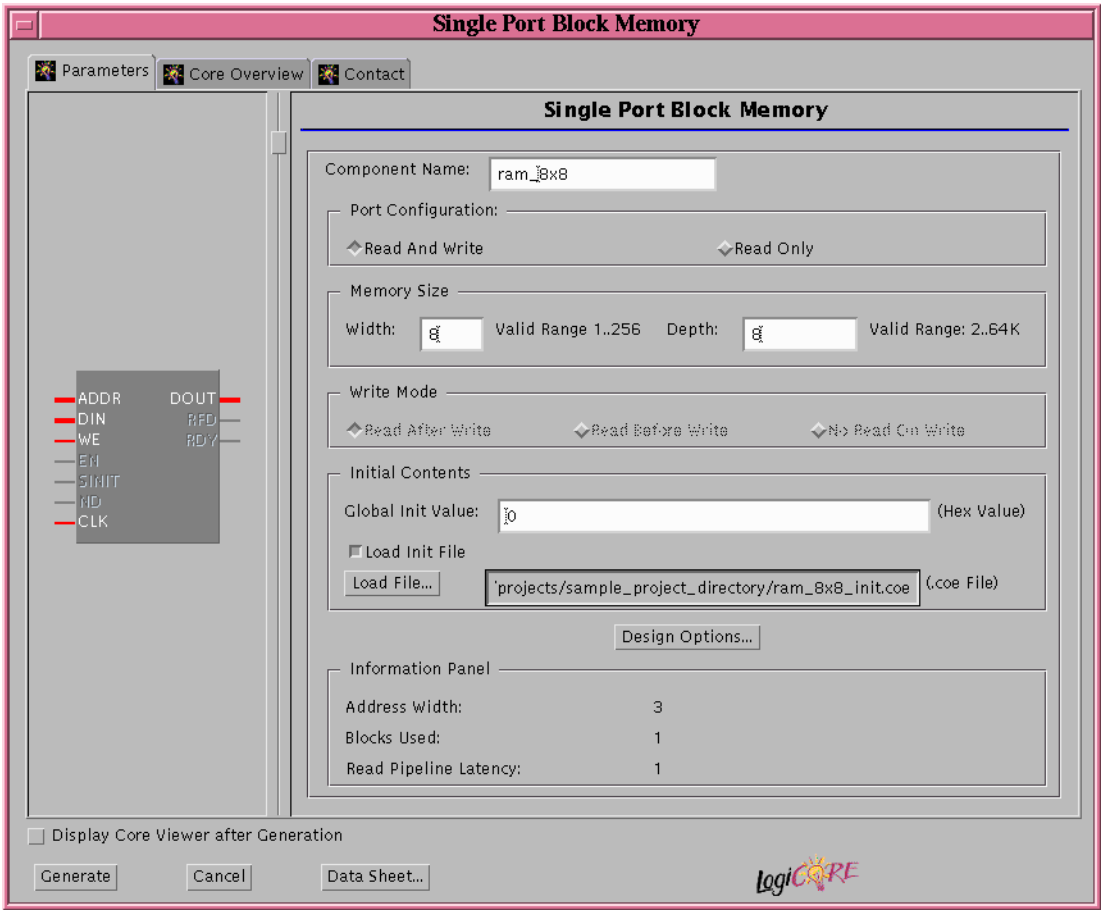


Figure 8: Complete customization options. The component can now be generated.

Once a component is customized, it can be generated by selecting the `'Generate'` button located at the bottom left of the component customization window. Upon generation, Coregen will create the files that are necessary in order to use the core. Typical such files are:

`.ASY`	Graphical symbol file, used by the Xilinx Foundation iSE tools.
`.EDN`	EDIF netlist file, used by the Xilinx implementation tools.

`.MIF`	Memory Initialization File. This file is used by the simulation tools to provide the initial contents of the memory.
`.VHO`	A template file containing information on how to use a core in a VHDL design.
`.VEO`	A template file containing information regarding the usage of the core in Verilog designs.
`.XSF`	A pin file used by Xilinx Foundation tools to create a graphical symbol for the core.

Select the 'Generate' button and Coregen will create the following files in the specified project directory:

```
-rw-r--r-- 1 k_vitoro beng 384  Jun 28 12:16 ram_8x8.asy
-rw-r--r-- 1 k_vitoro beng 9294 Jun 28 12:16 ram_8x8.edn
-rw-r--r-- 1 k_vitoro beng 72   Jun 28 12:15 ram_8x8.mif
-rw-r--r-- 1 k_vitoro beng 3370 Jun 28 12:16 ram_8x8.vho
-rw-r--r-- 1 k_vitoro beng 1037 Jun 28 12:16 ram_8x8.xco
```

Now the memory core is ready to be used as a component in an RTL design.

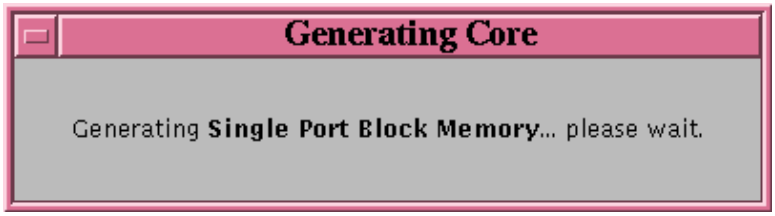


Figure 9: Your core component is being generated.

8. Creating VHDL designs containing core(s).

Before the generated core can be used as a component in an RTL design, the XilinxCoreLib library path must be specified in your '.synopsys_vss.setup' file. Recall that the '.synopsys_vss.setup' file is used by the Synopsys VSS simulator tool. The sample '.synopsys_vss.setup' file provided below shows the necessary addition:

```
WORK > DEFAULT
DEFAULT: ../work
TIMEBASE = NS
XILINXCORELIB : /CMC/tools/xilinx.vM3.1i/synopsys/libraries/sim/lib/xilinxcorelib
```

After the addition the XILINXCORELIB library can be declared and used in your VHDL code.

The following sample code illustrates how a core can be used in an RTL design. Note the code segments that were copied directly from the ram_8x8.vho file mentioned in the previous section.

```
-- Sample VHDL code that instances a Coregen
-- generated component.

library IEEE;
use IEEE.std_logic_1164.all;

library XILINXCORELIB;
use XILINXCORELIB.all;

entity coregen_ram_8x8 is
        port(
                addr: IN std_logic_VECTOR(2 downto 0);
                clk: IN std_logic;
                din: IN std_logic_VECTOR(7 downto 0);
                dout: OUT std_logic_VECTOR(7 downto 0);
                we: IN std_logic);
end coregen_ram_8x8;


architecture coregen of coregen_ram_8x8 is

-- The following component declaration code
-- was pasted directly from the file
-- ram_8x8.vho which was generated by coregen
```

```
------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component ram_8x8
        port (
        addr: IN std_logic_VECTOR(2 downto 0);
        clk: IN std_logic;
        din: IN std_logic_VECTOR(7 downto 0);
        dout: OUT std_logic_VECTOR(7 downto 0);
        we: IN std_logic);
end component;


-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of ram_8x8: component is "true";

-- COMP_TAG_END ------ End COMPONENT Declaration ------------


begin

-- The following component instantiation code
-- segment was copied from the ram_8x8.vho
-- file generated by coregen and then modified
-- to comply with the rest of the code.


------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
ram: ram_8x8
                port map (
                        addr => addr,
                        clk => clk,
                        din => din,
                        dout => dout,
                        we => we);
-- INST_TAG_END ------ End INSTANTIATION Template ------------

end coregen;
```

After this code is saved in a file it can be analyzed by the Synopsys tools. Analyze the file using the `'gvan'` or the `'vhdlan'` command.

9. Simulating a design that contains a core component.

In order to simulate a VHDL RTL design, an appropriate VHDL configuration file must be created. The .vho file created by Coregen in the project directory contains the main portion of the code that must appear in the configuration file.

The following configuration file can be used as a template. Notice the code segments that were copied directly from the 'ram_8x8.vho' file:

```
-- Template configuration file
-- needed for simulation of a
-- design with core components

library XILINXCORELIB;
use XILINXCORELIB.all;


configuration virtex_ram_8x8 of coregen_ram_8x8 is

for coregen

-- copy the configuration code excerpt from the
-- .vho file generated by coregen here.

------- Begin Cut here for CONFIGURATION snippet ---- CONF_TAG

-- synopsys translate_off

for all : ram_8x8 use entity XilinxCoreLib.blkmemsp_v3_2(behavioral)
                generic map(
                        c_has_en => 0,
                        c_has_din => 1,
                        c_has_limit_data_pitch => 0,
                        c_has_sinit => 0,
                        c_limit_data_pitch => 8,
                        c_width => 8,
                        c_sinit_value => "0",
                        c_addr_width => 3,
                        c_has_rfd => 0,
                        c_has_we => 1,
                        c_depth => 8,
                        c_write_mode => 0,
                        c_pipe_stages => 0,
                        c_has_nd => 0,
                        c_default_data => "0",
                        c_has_default_data => 0,
```

```
                        c_mem_init_file => "ram_8x8.mif",
                        c_reg_inputs => 0,
                        c_enable_rlocs => 0,
                        c_has_rdy => 0);
        end for;

  -- synopsys translate_on

  -- CONF_TAG_END ------ End CONFIGURATION snippet ------------

  end for;
  end;
```

Pay attention to the line in the code that specifies where the .mif file is located:

```
c_mem_init_file => "ram_8x8.mif",
```

This line provides the path to the .mif Memory Initialization File. Since no full path to the .mif file is specified, the simulation tool will search in the current directory (the one where the simulation tool was invoked from) in order to find it. This is why you need to have a copy of the .mif file in the directory where you invoke the simulator from. Note that this code line can be edited to contain the full path to the .mif file.

Once the configuration file is created and analyzed using the `'gvan'` or the `'vhdlan'` command, the VHDL RTL code can be simulated. Copy the `'ram_8x8.mif'` file from the project directory to the directory where the source code of the design is and invoke the Synopsys VHDL simulator using the command `'vhdldbx'`. For simulation, choose the name of the configuration and not the top level entity name (Figure 10). Simulate as usual. Figure 11 shows the Synopsys VSS simulator and Figure 12 gives simulation results for our RAM component.

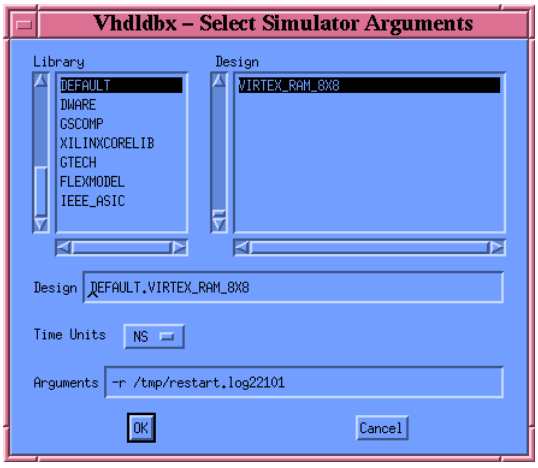


Figure 10: Selecting a configuration for simulation.

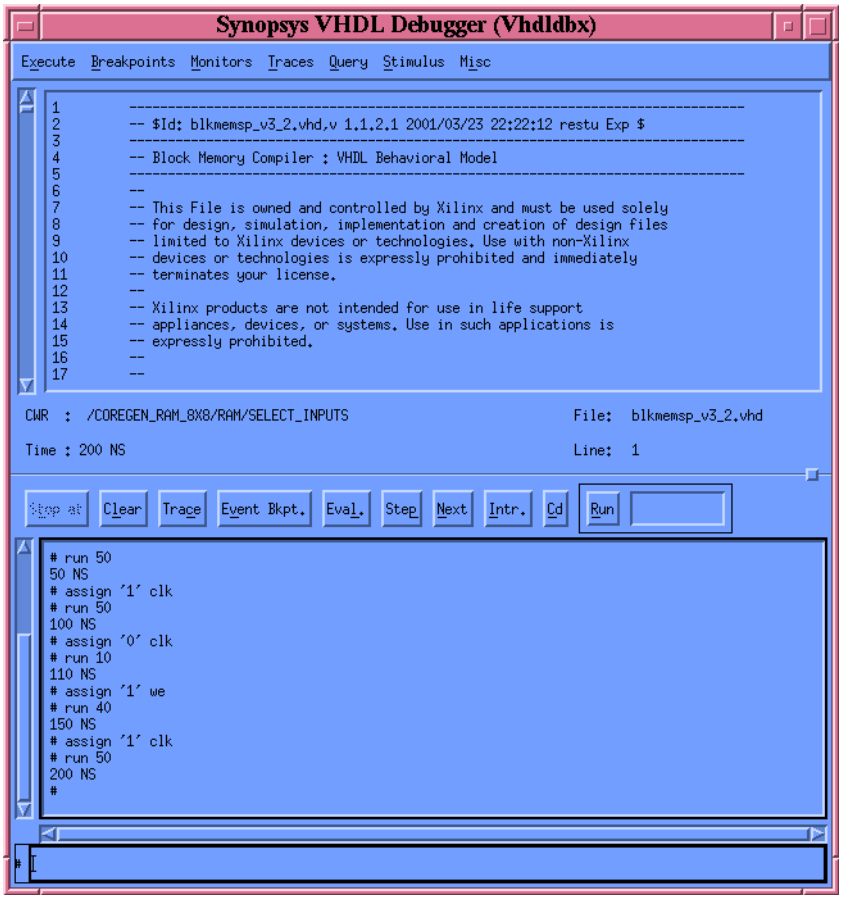


Figure 11: The Synopsys VSS vhdldbx simulator.

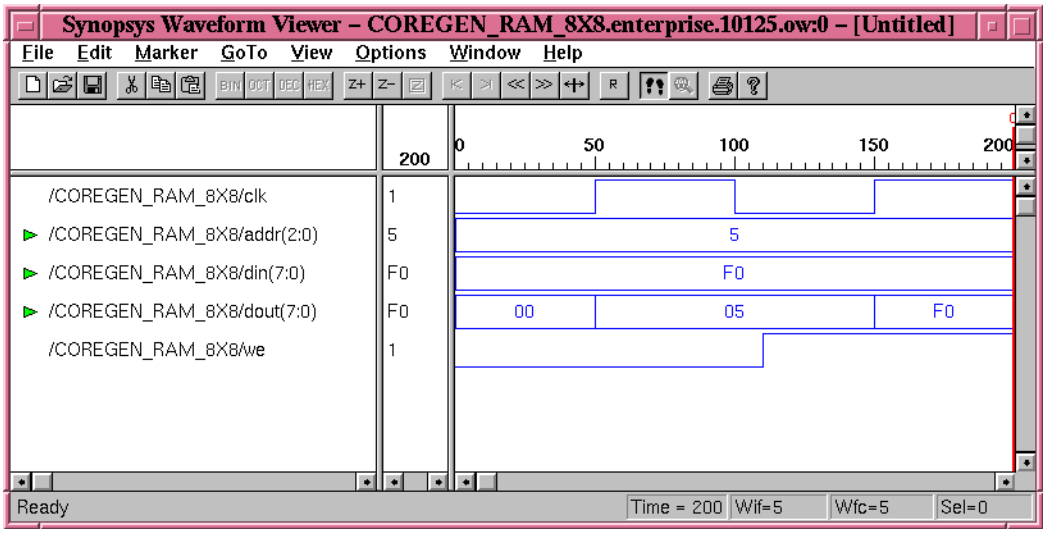


Figure 12: Some simulation results for our example RAM component.

10. Synthesizing a design containing Coregen components towards a Virtex technology.

The following steps are necessary in order to synthesize towards a Virtex technology using Synopsys' Design Compiler:

* target the necessary libraries in the setup file of design compiler (.synopsys_dc.setup)
* prepare a script file with all the commands needed by Design Compiler to synthesize.
* run the script.

The end result of the synthesis process will be an EDIF (Electronic Design Interchange Format) netlist file (filename extension .sedif) which will be used by the Xilinx implementation tools. Note that for the Virtex technology the implementation tools REQUIRE a netlist in EDIF format; other netlist formats Design Compiler can create will not be accepted for implementation.

The setup file `'.synopsys_dc.setup'` must target the appropriate libraries. Template dc.setup files for several Xilinx FPGA families can be found at location:

```
/CMC/tools/xilinx.vM3.1i/synopsys/examples
```

Again, note the possibility that this pathname may change in future updates of the software.

Below is a sample setup file:

```
/* ================================================ */
/* Template .synopsys_dc.setup file for Xilinx designs */
/*       For use with Synopsys Design Compiler.      */
/* ================================================ */


/*  This setup file will result in the .db file      */
/*  being represented in terms of GATES and not      */
/*  LUTs                                             */

/* if you want the .db in terms of LUTs              */
/* use the libraries as specified by the             */
/* synlibs xfpga_virtex-4                            */
/* instead of synlibs xdc_virtex-4                   */

/* ================================================ */
/* The Synopsys search path should be set to point   */
/* to the directories that contain the various       */
/* synthesis libraries used by FPGA Compiler during  */
/* synthesis.                                        */
/* ================================================ */


search_path =  {. /CMC/tools/xilinx.vM3.1i/synopsys/libraries/syn \
                /CMC/tools/synopsys/syn/libraries/syn } ;




                /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */
                /* Ensure that your UNIX environment */
                /* includes the two environment var- */
                /* iables: $XILINX (points to the    */
                /* Xilinx installation directory) and*/
                /* $SYNOPSYS (points to the Synopsys */
                /* installation directory.)          */
                /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */

/* ================================================ */
/* Define a work library in the current project dir  */
```

```
/* to hold temporary files and keep the project area */
/* uncluttered. Note: You must create a subdirectory */
/* in your project directory called WORK.            */
/* ================================================= */


   define_design_lib WORK -path ./Code/Work


bus_extraction_style = "%s<%d:%d>"
bus_naming_style = "%s<%d>"
bus_dimension_separator_style = "><"

hdlin_translate_off_skip_text=true
edifin_lib_logic_1_symbol = "VCC"
edifin_lib_logic_0_symbol = "GND"
edifout_ground_name = "GND"
edifout_ground_pin_name = "G"
edifout_power_name = "VCC"
edifout_power_pin_name = "P"
edifout_netlist_only = "true"
edifout_no_array = "true"
edifout_power_and_ground_representation = "cell"
edifout_write_properties_list = \
{"DUTY_CYCLE_CORRECTION" "INIT_00" "INIT_01" "INIT_02" "INIT_03" \
 "INIT_04"  "INIT_05" "INIT_06" "INIT_07" "INIT_08" "INIT_09" "INIT_0A"\
 "INIT_0B" "INIT_0C" "INIT_0D" "INIT_0E" "INIT_0F" "INIT" "CLKDV_DIVIDE"\
 "IOB" "EQN" "lut_function" "instance_number" "pad_location" "part"}

/* ================================================= */
/* Set the link, target and synthetic library        */
/* variables. Use synlibs (with the -dc switch) to   */
/* determine the link and target library settings.   */
/* You may like to copy this file to your project    */
/* directory, rename it ".synopsys_dc.setup" and     */
/* append the output of synlibs. For example:        */
/* synlibs xdc_virtex-4 >> .synopsys_dc.setup      */
/* ================================================= */


link_library = {xdc_virtex-4.db xdw_virtex.sldb}
target_library = {xdc_virtex-4.db }
symbol_library = {virtex.sdb}
synthetic_library = {xdw_virtex.sldb standard.sldb}
define_design_lib xdw_virtex -path \
/CMC/tools/xilinx.vM3.1i/synopsys/libraries/dw/lib/virtex
```

This file should be located in the directory where the 'dc_shell' command is be invoked from and must be named '.synopsys_dc.setup'.

Once the setup file is prepared, a script file with all the necessary commands to the Design Compiler for the synthesis procedure is needed. The following script file will synthesize the simple ram8x8 example; it can be used as an example for writing a script file. A general template file for synthesis towards a Virtex FPGA can be found in the directory:

/CMC/tools/xilinx.vM3.1i/synopsys/examples

under the name: 'template.fpga.script.virtex'.

```
/*                                                    */
/*   For general use with VIRTEX architectures.       */
/* ================================================== */

/* ================================================== */

/*==================================================*/
/* Enable Synopsys to write out top level design name*/
/* as <design_name> instead of synopsys_edif.       */
/*==================================================*/



   designer = "Kostas Vitoroulis"
   company  = "Concordia University"
   part     = "XCV300PQ240-4"

/* ================================================== */
/* Analyze and Elaborate the design file and specify */
/* the design file format.                           */
/* ================================================== */

analyze -format vhdl /project/vlsi/k_vitoro/Synopsys/Code/COREGEN_TESTING/ram_8x8.vhd


                    /* ========================== */
                    /* You must analyze lower-level */
                    /* hierarchy modules here       */
                    /* ========================== */

   elaborate coregen_ram_8x8

/* ================================================== */
```

```
/* Set the current design to the top level.          */
/* ================================================= */

   current_design coregen_ram_8x8

/* ================================================= */
/* Set the synthesis design constraints.             */
/* ================================================= */

/* uncomment the following line if the design is hierarchical */

/*   uniquify  */

   remove_constraint -all

/*  if your design is sequential, modify the following line */

/*  create_clock clk  -period 5000   */


/* ================================================= */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                        */
/* ================================================= */

   set_port_is_pad "*"
   set_pad_type -slewrate HIGH all_outputs()

/* ================================================= */
/* Substitute the required input and output pads for */
/* IBUF and OBUF, respectively in the following two  */
/* lines.                                            */
/* ================================================= */

   set_pad_type -exact IBUF all_inputs()
   set_pad_type -exact OBUF all_outputs()

   insert_pads


/*  ================================================  */
/*                                                    */
/*  Set don't touch attributes on any instances       */
/*  which are generated by Coregen                    */
```

```
/*                                                      */
/*  ================================================  */

 set_dont_touch ram

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*               Compile the design                     */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++ */

   compile -map_effort low


/* ================================================== */
/* Set the part type for the output netlist.          */
/* ================================================== */


/* The following pin specifications are for the XCV300PQ240   chip  */

   set_attribute  coregen_ram_8x8 "part" -type string "XCV300PQ240-4"

   set_attribute "clk" "pad_location" -type string "P3"
   set_attribute "addr<0>" "pad_location" -type string "P4"
   set_attribute "addr<1>" "pad_location" -type string "P6"
   set_attribute "addr<2>" "pad_location" -type string "P7"
   set_attribute "din<0>" "pad_location" -type string "P10"
   set_attribute "din<1>" "pad_location" -type string "P13"
   set_attribute "din<2>" "pad_location" -type string "P17"
   set_attribute "din<3>" "pad_location" -type string "P18"
   set_attribute "din<4>" "pad_location" -type string "P20"
   set_attribute "din<5>" "pad_location" -type string "P21"
   set_attribute "din<6>" "pad_location" -type string "P24"
   set_attribute "din<7>" "pad_location" -type string "P25"
   set_attribute "dout<0>" "pad_location" -type string "P27"
   set_attribute "dout<1>" "pad_location" -type string "P28"
   set_attribute "dout<2>" "pad_location" -type string "P31"
   set_attribute "dout<3>" "pad_location" -type string "P34"
   set_attribute "dout<4>" "pad_location" -type string "P35"
   set_attribute "dout<5>" "pad_location" -type string "P38"
   set_attribute "dout<7>" "pad_location" -type string "P39"
   set_attribute "dout<7>" "pad_location" -type string "P41"
   set_attribute "we" "pad_location" -type string "P42"

/* ================================================== */
/* Save design in EDIF format as <design>.sedif       */
/* ================================================== */
```

```
write -format edif -hierarchy -output /project/vlsi/k_vitoro/Synopsys/EDIF/coregen_ram_8x8.sedif

   quit
/* ================================================ */
/* Now run the Xilinx design implementation tools.   */
/* ================================================ */
```

When preparing this file make sure to change all the path names to correspond to files in your account's directory structure. Specifically the following lines should will modification:

```
analyze -format vhdl /project/vlsi/k_vitoro/Synopsys/Code/COREGEN_TESTING/ram_8x8.vhd
write -format edif -hierarchy -output /project/vlsi/k_vitoro/Synopsys/EDIF/coregen_ram_8x8.sedif
```

Save the file under the name `'ram_8x8.scr'` and execute it using the command:

```
dc_shell -f (...the path to your script file)
```

Warnings regarding *"unresolved refences"* will be generated during the execution of the script (Figure 13).

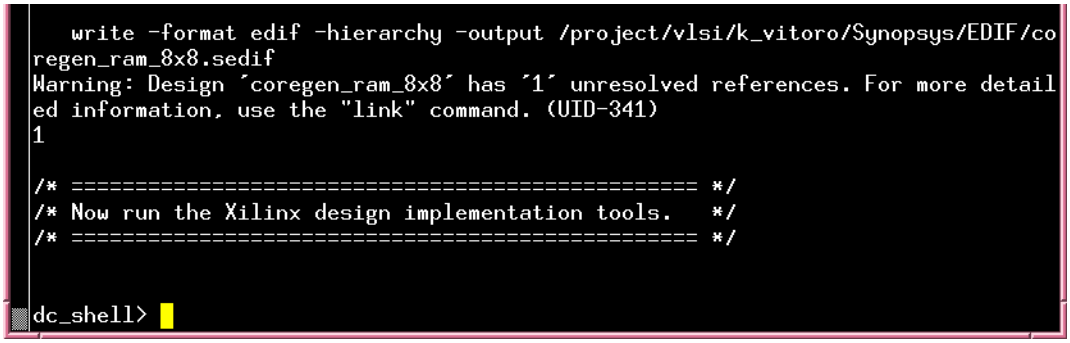


Figure 13: The "unresolved refences" warning.

This warning implies that Design Compiler was unable to synthesize the component instance named `'ram'`, of type `'ram_8x8'`. Although warnings about unresolved references should normally be investigated, in this case they can be safely ignored since the Xilinx implementation tools will have all the information they need to generate the components (This information will be provided through the EDIF file as it will be discussed later).

It is suggested to have a directory where only script files for the Design Compiler are kept and then execute the scripts from that location by specifying the full path name of the file.

The output of the `dc_shell` command can be logged to a file by issuing the following command:

```
dc_shell -f ram_8x8.scr | tee ram_8x8.log
```

Here the output of the `dc_shell` is piped to the `'tee'` command which redirects it to standard output and to the file `'ram_8x8.log'`. Once the output is logged it is easier to find possible errors and warnings that were generated during execution.

At this point the file `'coregen_ram_8x8.sedif'` should exist in the directory you specified earlier within the script file at line:

```
write -format edif -hierarchy -output .../coregen_ram_8x8.sedif
```

This file contains the netlist (in EDIF format) for your synthesized design. The netlist will be used in the implementation steps, which are described next.

11. Implementing a Synthesized design using the Xilinx implementation tools.

As described in section 5 Coregen creates a set of files when a specified core is generated. One of these is an EDIF netlist file, recognized by the `'.edn'` filename extension. This file *must* be copied to the directory where Synopsys' Design Compiler has saved the `'.sedif'` file. Recall that Design Compiler saves the EDIF netlist when it executes the following line in the script:

```
write -format edif -hierarchy -output .../coregen_ram_8x8.sedif
```

(The ellipses stand for your specified directory)

We need to place the two EDIF netlist files in the same directory for the following reason: The EDIF netlist generated by the Synopsys Design Compiler tool does not contain any information regarding the implementation of the core (hence the warning message regarding unresolved references mentioned near the end of section 8). This information will be provided to the Xilinx implementation tools by the Coregen generated EDIF (.edn) file.

To proceed with the implementation of the ram_8x8 memory example, locate the `'ram_8x8.edn'` file in the project directory where Coregen created the memory core component and copy it as `'ram_8x8.sedif'` in the directory where Design Compiler wrote the file `'coregen_ram_8x8.sedif'`.

Run the Xilinx Design Manager using the command `'dsgnmgr&'`(Figure 14).

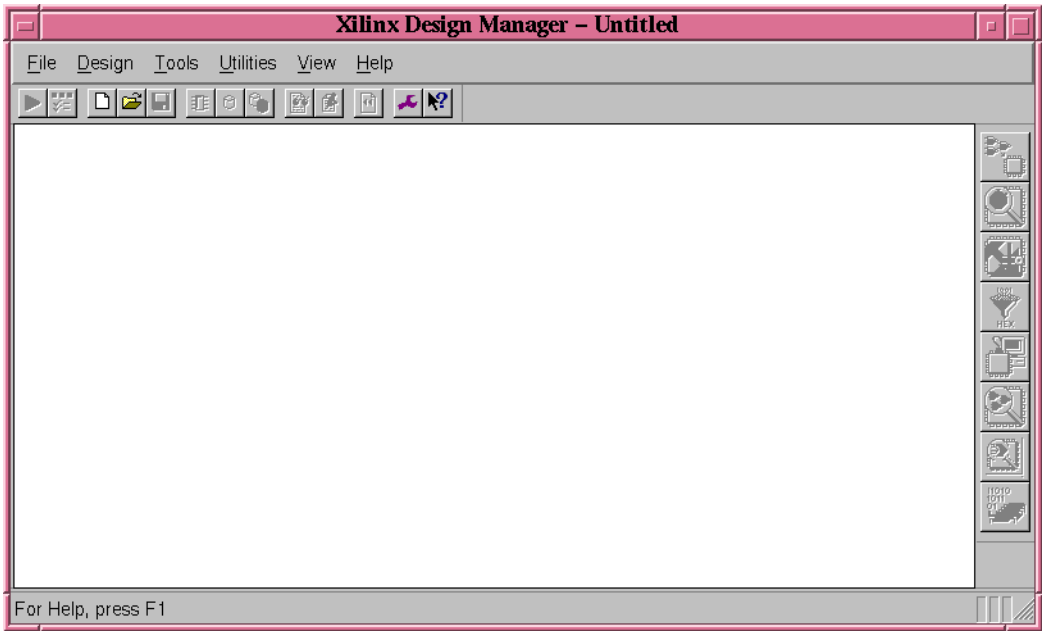


Figure 14: Design Manager's main window.

Select `'New Project'` in the file menu. A new window will appear (Figure 15).

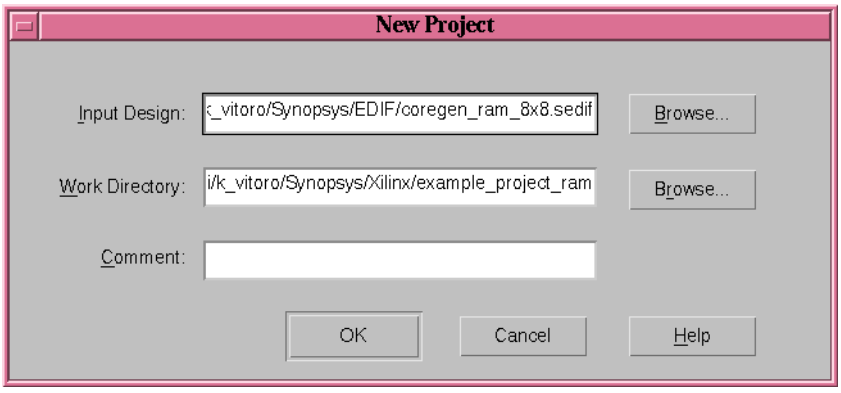


Figure 15: New project specifications widow.

In the 'Input Design' field specify the EDIF file 'coregen_ram_8x8.sedif', generated by Design. In the field 'Work Directory' specify a project directory and give the name 'ram_8x8' to the project. In this working directory the Design Manager will create all files which relate to the project.

Select the 'OK' button and a new window titled 'New Version' will displayed (Figure 16).

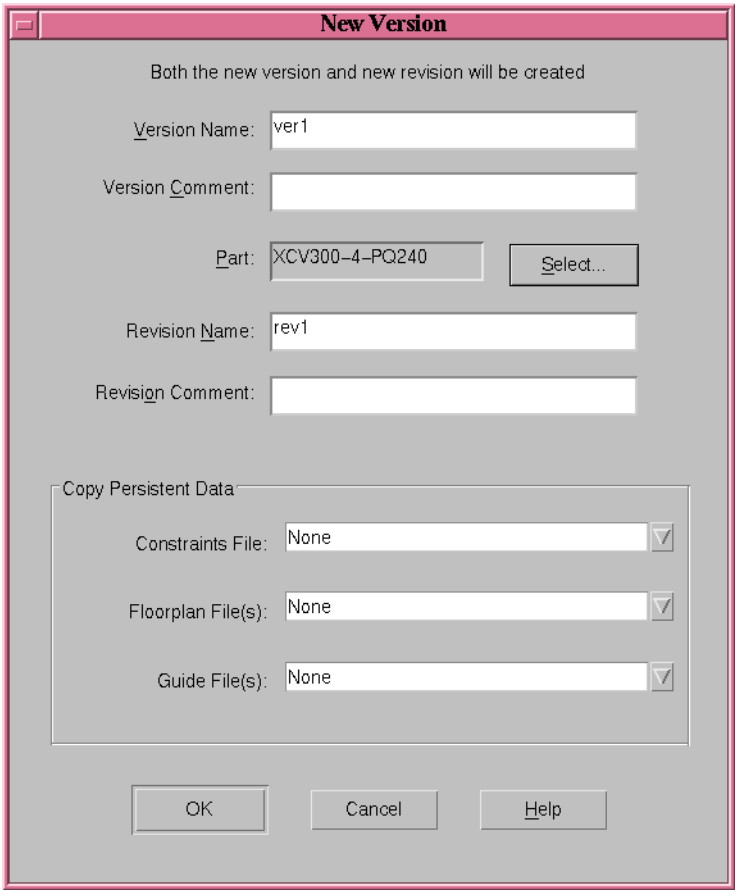


Figure 16: Select the target device for implementation.

In this window choose the 'Select...' button next to the 'Part' text field and a new window titled 'Part Selector' will appear (Figure 17). Use the pull down menus to specify the target chip for the implementation and select the 'OK' button. For the purposes of this tutorial a Virtex XCV300 chip with package type PQ240 and speed grade 4 is available to you then the 'Part' field will contain the string 'XCV300-4-PQ240'

.

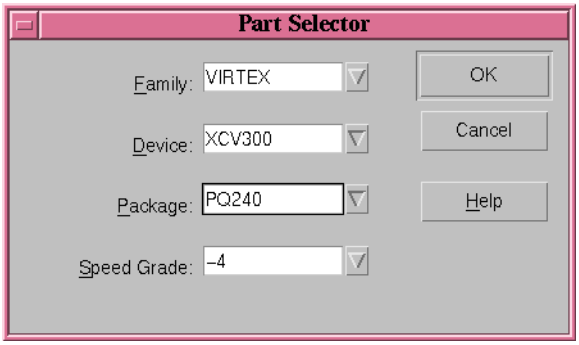


Figure 17: Part Selector window.

Select the `'OK'` button in the `'New Version'` and the Design Manager's main window will be available to you (Figure 18).

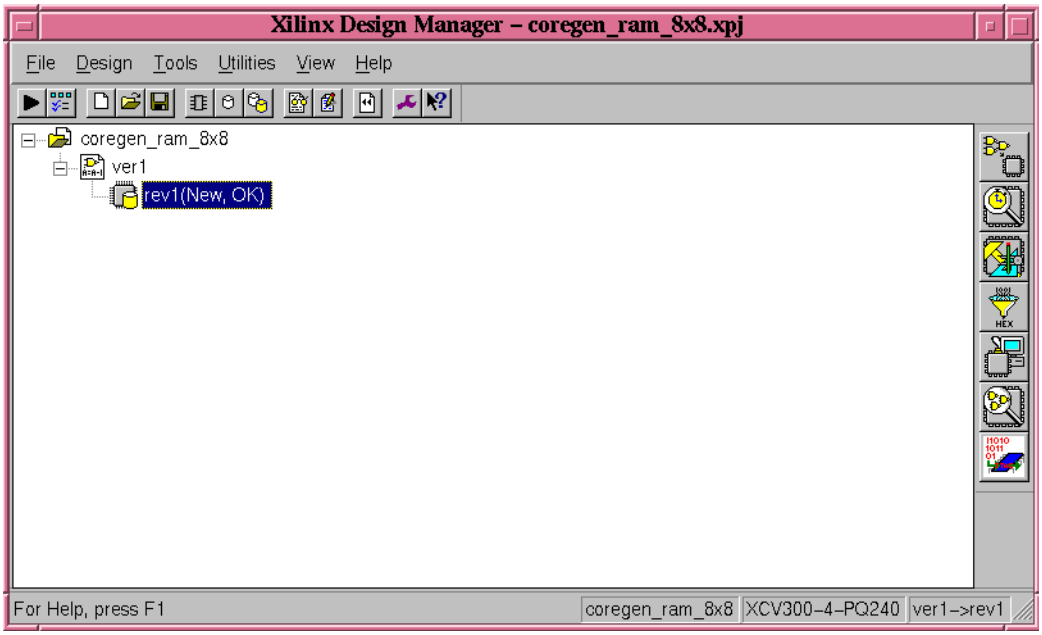


Figure 18: Ready to implement the design.

The design can now be implemented by selecting the 'Design' --> 'Implement' menu option.

The 'Flow Engine' will run in a separate window that displays the separate steps of the implementation process (Figure 19).

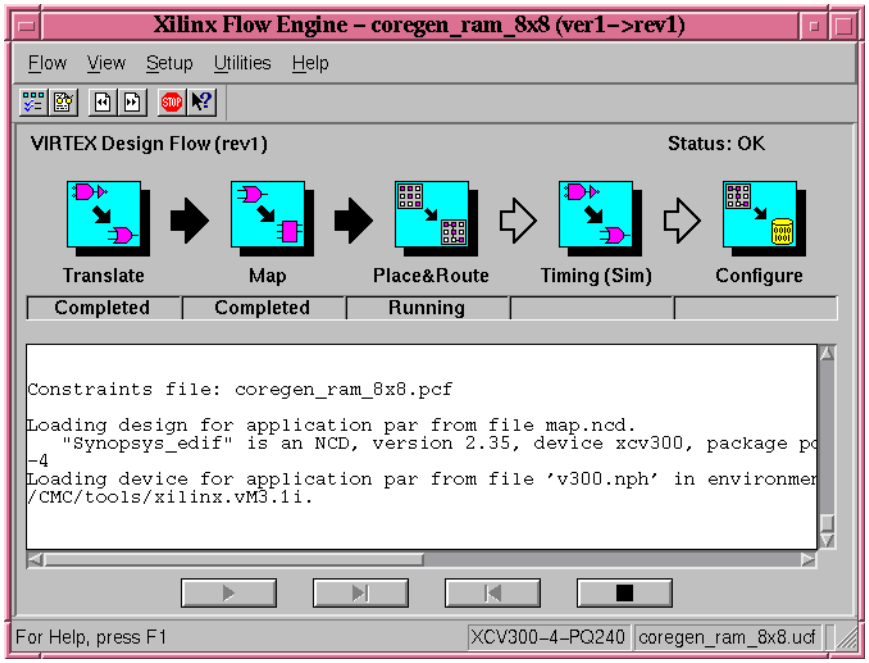


Figure 19: The flow engine window.

Once the implementation phase is complete, a new window will be displayed. Select 'Ok' to close it and in the main window of the 'Design Manager' notice the item 'rev1(Implented, OK)' that indicates that the implementation is complete (Figure 20). If errors occur refer to the fe.log file located in the Xilinx directory project in the subdirectory: .../ver1/rev1

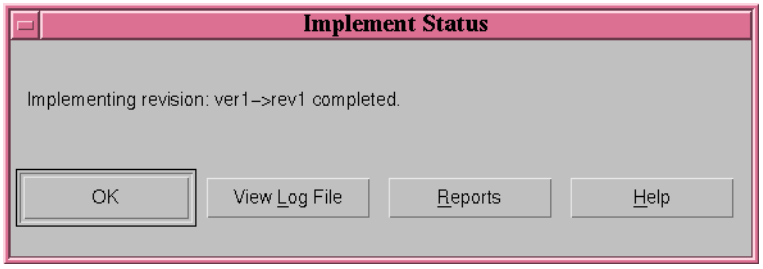


Figure 20: Implementation completed without errors.

The implementation of the design is now complete and the 'Hardware Debugger' tool can be used to download the bit file to the device.

12. Performing gate level simulation.

RTL simulation can be used to verify the functional correctness of an RTL design but it cannot guarantee that the design will function correctly after implementation. This is because RTL simulation does not take into consideration any gate level, physical timing issues.

Gate level simulation, on the other hand, provides a sufficient indication that an implemented design will be functional with regards to timing issues. To perform gate level simulation the gate level netlist of the design is needed. In the gate level netlist the design is composed of only primitive gate components (hence the name gate level). The timing information of each gate is modelled and taken into consideration during the course of simulation.

The gate level netlist of a design is generated by either the synthesis or the implementation tools. In our example we will derive the gate level netlist from the Xilinx implementation tools. For the Virtex families it is only possible to simulate the gate level netlist created by the Xilinx implementation.

To derive the gate level netlist for the ram_8x8 example, perform the following procedure (the same procedure applies for every Virtex design): From the main window of the Xilinx Design Manager, select the `'Options...'` entry of the `'Design'` menu (Figure 21).

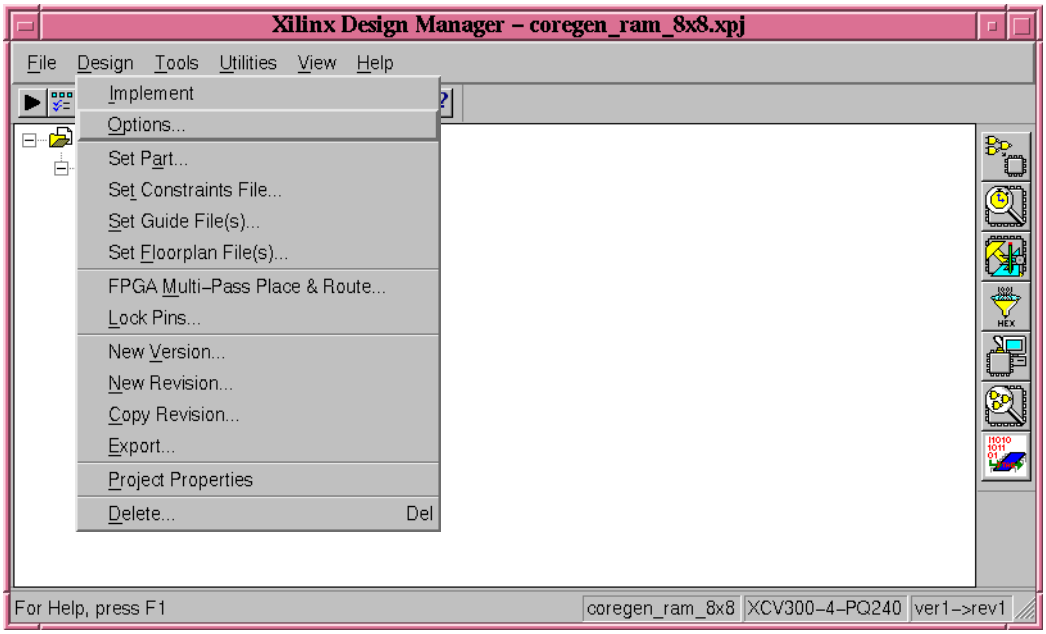


Figure 21: Preparing for Gate Level Simulation.

A new window titled `Options` will appear (Figure 22).

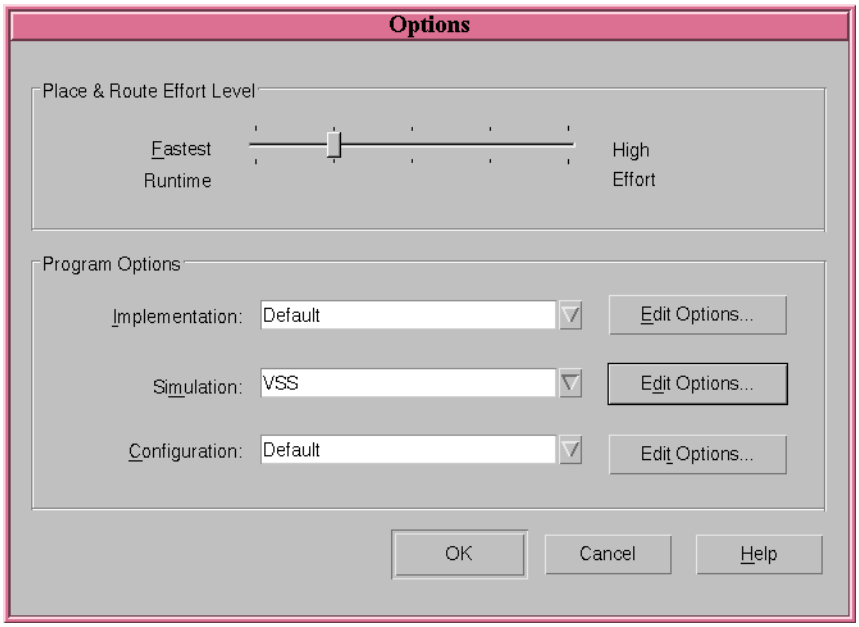


Figure 22: Setting up the options for the Synopsys' VSS simulator

In the `Programs Options` section, select `VSS` in the drop menu titled `Simulation` and then press in the corresponding `Edit Options` button.

a) The 'general' tab

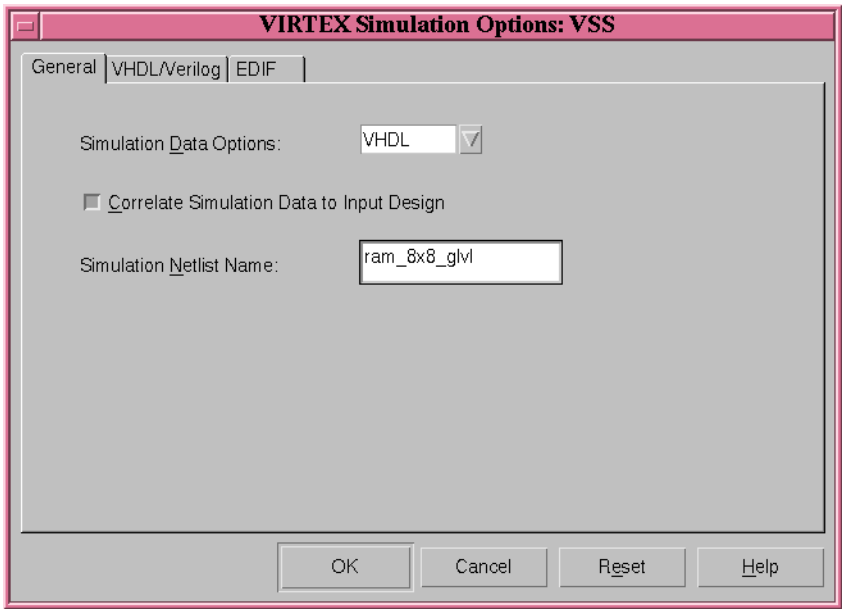
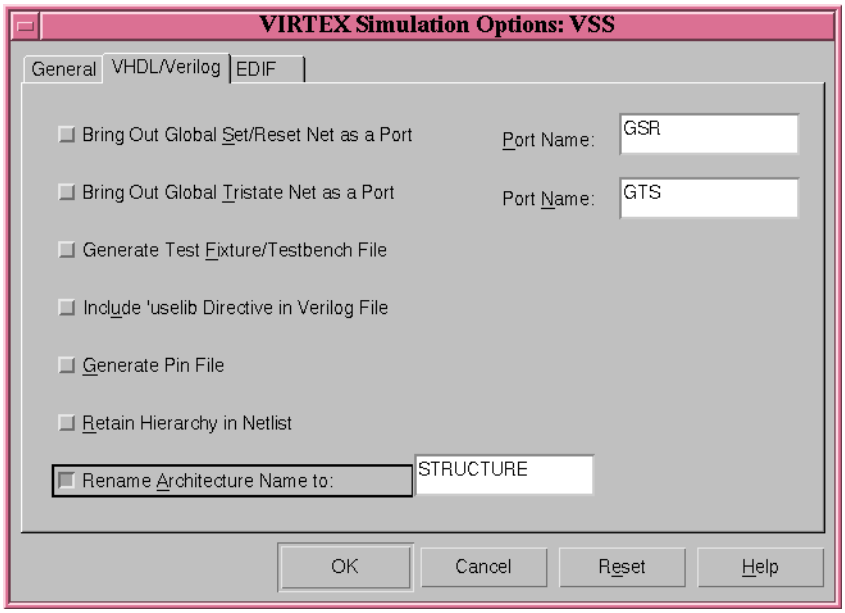


Figure 23: The options for the gate level netlist of the design that will be generated by the tool.

b) The 'VHDL/Verilog' tab



c) The 'EDIF' tab.

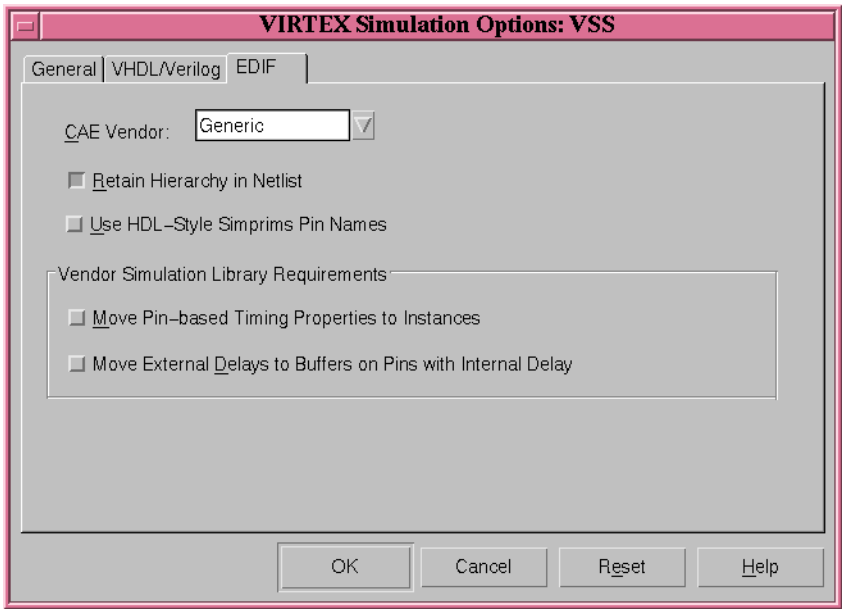


Figure 23: The options for the gate level netlist of the design that will be generated by the tool.

Another window will show (Figure 23) with the options separated three in tab forms.

Select the 'General' tab, and specify the following:

```
Simulation Data Options:  VHDL
Correlate Simulation Data to Input Design:
           check the button on the left
Simulation Netlist Name:ram_8x8_glvl
```

The first setting will cause the gate level code to be in VHDL, the second setting will make the gate level entity names consistent with that of the original design and the third setting specifies the filename of the gate level code (no extension needed; it will be added).

Then select the VHDL/Verilog tab and check the entry 'Rename Architecture Name To:'. Leave the name 'STRUCTURE' in the textfield (of course any name could be chosen). Press the 'OK' button. Press the 'OK' Button in the 'Options' window also and in the main window of the Xilinx Design Manager, select the 'Design' --> 'Implement' Option. The flow engine will be run once again but this time the gate level netlist will be written.

The gate level netlist file will be located in the ram_8x8 project directory:

```
.../ram_8x8/ver1/rev1/ram_8x8_glvl.vhd
```

In order to perform the gate level simulation, a configuration for the gate level code must be created. Then the design in 'ram_8x8_glvl.vhd' and its configuration can be analyzed and simulated.

Locate the 'ram_8x8_glvl.vhd' and view its contents. Note the three entity/architecture pairs , namely ROC/ROC_V, TOC/TOC_V and COREGEN_RAM_8X8/STRUCTURE. Also notice the libraries declared before each entity. The location of these libraries must be specified in the setup file '.synopsys_vss.setup'. Below is an example of a '.synopsys_vss.setup' file that includes these libraries, among others:

```
WORK > DEFAULT
DEFAULT: ./work
TIMEBASE = PS
LOGIBLOX : /CMC/tools/xilinx.vM3.1i/synopsys/libraries/sim/lib/logiblox
SIMPRIM : /CMC/tools/xilinx.vM3.1i/synopsys/libraries/sim/lib/simprims
UNISIM : /CMC/tools/xilinx.vM3.1i/synopsys/libraries/sim/lib/unisims
XDW : /CMC/tools/xilinx.vM3.1i/synopsys/libraries/sim/lib/xdw
XILINXCORELIB : /CMC/tools/xilinx.vM3.1i/synopsys/libraries/sim/lib/xilinxcorelib
```

Note the TIMEBASE=PS setting which sets the time base in picoseconds, the reason being that the simulations libraries have delay values specified in units of picoseconds.

The configuration file needed by the simulation tools will specify which ROC/ROC_V and TOC/TOC_V pairs will be used. A template configuration file follows:

```
-- "SYNOPSYS_EDIF" will be the name of the entity
-- for the gate level code produced by Xilinx Design Manager
-- if it was not specified that the tool should following
-- the naming patterns of the original design.

-- "STRUCTURE", in a similar fashion, will be the name of the
-- corresponding architecture if the Xilinx implementation
-- tools were not told to following the naming conventions
-- of the original design.

library UNISIM;
library SIMPRIM;
library XDW;

configuration ram_8x8_glvl_virtex_CONFIG of COREGEN_RAM_8X8 is
for STRUCTURE
-- if one of the ROC or TOC entities does not appear in
-- you gate level code, comment out the corresponding line.

for ROC_NGD2VHDL : ROC use entity WORK.ROC(ROC_V);
end for;

for TOC_NGD2VHDL : TOC use entity WORK.TOC(TOC_V);
end for;

-- use defaults for everything else found in
-- libraries defined in the .synopsys_vss.setup file
-- found in this directory

end for;
end ram_8x8_glvl_virtex_CONFIG;
```

In order to perform the gate level simulation copy both the configuration file and the gate level code file in the same directory, analyze them (in the order: gate level, configuration) and simulate the entity architecture pair derived from the configuration file.

APPENDIX:

Directory structure.

A well maintained computer account MUST have a proper directory structure. Lack of organization will only cause confusion and chaos in one's work. This applies especially when working with sophisticated software tools that need a large number of files like the Synopsys and Xilinx tools.

The general rule that applies when a directory structure for software tools is created is to have different directories groups of similar files.

The following is an efficient directory structure for the Synopsys and Xilinx tools:

```
Synopsys
   |
   |-xilinx
   |
   |-coregen
   |
   |-EDIF
   |
   |-Scripts
   |
   |-Code
   |   |
   |   |-Work
   |
   |
   |-PostSynthesisCode
   |   |
   |   |-Work
   |
```

The Synopsys directory is the main directory and contains everything else. The environment files for Synopsys and Xilinx (synopsys_2000.env, xilinx.vM3.1i.env) and the .synopsys_dc.setup file can be placed here. The dc_shell command can be executed from this directory to execute scripts:

```
dc_shell -f ./Scripts/script_name.scr
```

The xilinx directory is where all projects created with xilinx can be saved.

The coregen directory is where all projects created with Coregen can be saved.

The Code directory is where the RTL VHDL code can be saved. The .synopsys_vss.setup file would exist in this directory. Also the simulator would be invoked from here.

The PostSynthesisCode directory can be used to store the derived gate level code of a design. Another .synopsys_vss.setup file would live here. The simulator would be invoked from here for gate level simulation.

The Work directories seen in Code and PostSynthesisCode directories are used to store intermediate files for the Synopsys tools. The .synopsys_vss.setup file specifies the Work directory as the working directory for the Synopsys tools.

Summary of steps.

The following is a summary of all the steps followed in this tutorial for the creation, simulation, synthesis/implementation and gate level simulation of the ram_8x8 example.

1) Run Coregen and generate the core(s) you intend to use in your design.

2) Write the VHDL RTL design using the generated core components as VHDL components

3) To simulate, create the necessary configuration files. The Coregen generated .vho contains segments of code that need to be included in your configuration file. Your .synopsys_vss.setup file should specify the location of the XILINXCORELIB on the system.

4) Synthesize as usual using the appropriate dc_shell script and the .synopsys_dc.setup file that targets your available Virtex device. Make sure that the netlist produced from synthesis is in EDIF format (use .sedif as extension).

5) Copy the EDIF file for the core that was generated by Coregen to the same location where your design's EDIF was written. Change the extension from .edn to .sedif

6) Run the Xilinx implementation tools.

7) If you wish to perform gate level simulation, specify to the Xilinx tools to generate a VHDL gate level code of the design. Then generate the appropriate configuration file for the gate level code and proceed simulating.