Virtex 2.1i tutorial: VHDL using FPGA Compiler and VSS

This Tutorial describes the Virtex design flow with Synopsys FPGA Compiler, and simulation flow with VSS simulator. It includes the following sections:

- INTRODUCTION
- OVERVIEW
- SETUP
- TUTORIAL FILES
- FINAL NOTES
- FUNCTIONAL SIMULATION
- SYNTHESIS and IMPLEMENTATION
- TIMING SIMULATION

INTRODUCTION

The purpose of this tutorial is help the users to familiarize the A2.1i FPGA Compiler/VSS design flow. A VHDL design that can be downloaded to a Virtex board is provided. This is a front to back tutorial, which will take the user from functional simulation, synthesis and implementation to timing simulation. The emphasis is on documenting the steps to reach a particular design stage. This tutorial assumes that you are fluent in VHDL and familiar with FPGA compiler and VSS. If not, you may want to complete the Synopsys tutorial first.

OVERVIEW

To use this A2.1i XSI tutorial, you must be using A2.1i XSI and Synopsys v1998.02 or newer. You will need to use FPGA Compiler and VSS for this tutorial. If you are using Synopsys v1998.02 or newer, make sure that you have all the libraries compiled properly. If not, you will need to complete the SETUP process.

SETUP

To use this tutorial, first setup your A2.1i environment and Synopsys environment. Refer to the installation instructions for A2.1i, and the installation instructions for Synopsys for the correct procedures. After setting up the A2.1i and XSI environments, you need to determine if the XSI XDW and simulation libraries are setup and compiled correctly. If they are, you can skip the SETUP section and go to the TUTORIAL FILES section. If not, the DesignWare libraries and the simulation libraries must be compiled. The A2.1i XSI XDW libraries are a collection of Synopsys DesignWare libraries provided by Xilinx. There is a separate XSI XDW DesignWare library for each chip family in A2.1i. For example, there are separate libraries for Spartan, 4000XL, Virtex, etc. It is only necessary to compile the libraries for the device family you will be using. In general, it is a good idea to compile all these libraries at the same time. The A2.1i simulation libraries come in two parts, a functional simulation part and a timing simulation part. The A2.1i XSI VHDL functional simulation libraries are called UNISIM. The A2.1i XSI VHDL timing simulation libraries are called SIMPRIM libraries. The SIMPRIM library is a VITAL simulation library.

For this tutorial, only the libraries related to simulating and synthesizing a Virtex device need

to be compiled if you are using a version of Synopsys newer than v1998.02. For this tutorial, make sure that the Virtex XDW DesignWare libraries, XDW simulation libraries: UNISIM simulation libraries, and SIMPRIM libraries are compiled. All of these libraries have compile scripts which allows you to compile them in the $XILINX area. However, to use these scripts, a user must have write permissions to the $XILINX area. If you do not have write permissions, copy the $XILINX/synopsys/libraries directory to a local directory that you have write access to and follow the instructions below, but instead of going to the directory $XILINX/synopsys/libraries, go to the directory of your local copy area:
(1) cd to the $XILINX/synopsys/libraries/dw/src/virtex or your local directory.
(2) Type at the unix prompt: dc_shell -f install_dw.dc
Note: the following warning message may show up when running this command:
Warning: Can't read link_library file 'your_library.db'. (UID-3)
The warning does not affect compiling, and can be safely ignored.
(3) cd to the $XILINX/synopsys/libraries/sim/src/simprims or your local directory.
(4) Type at the unix prompt: ./analyze.csh
(5) cd to the $XILINX/synopsys/libraries/sim/src/unisims or your local directory
(6) Type at the unix prompt: ./analyze.csh.

After compiling the libraries, you must create a directory where you will run the tutorial. This directory will contain the tutorial HDL files, along with a .synopsys_dc.setup and .synopsys_vss.setup you will create.

TUTORIAL FILES

In your home directory, create a directory that will contain the files for this tutorial. In this empty directory, copy the file virtexxsivhdl.tar.Z. Uncompress and untar this file. Next, in the same directory, create a directory literally called WORK. Type the following at the unix prompt to create WORK:

mkdir WORK

After creating the work directory, you must create a .synospys_dc.setup file and a .synopsys_vss.setup. Templates for these files are provided in the $XILINX/synopsys/examples area. In the $XILINX/Synopsys/examples directory, copy the file template.synopsys_dc.setup_virtex into the same directory that contains WORK. Rename template.synopsys_dc.setup_virtex to .synopsys_dc.setup. It's strongly recommened to use the template.synopsys_dc.setup file as the Virtex .synopsys_dc.setup file due to certain Virtex specific features. In the $XILINX/synopsys/examples directory, copy the file template.synopsys_vss.setup into the same directory that contains WORK. Rename the file template.synospys_vss.setup to .synopsys_vss.setup. The .synopsys_dc.setup file is missing several lines related to synthesis libraries. The missing lines can be added by using the A2.1i XSI tool called `synlibs', which displays the synthesis library information needed for a given die-speed combination. For this tutorial, you will be synthesizing the design in a Virtex device. To add the correct information into the .synopsys_dc.setup file for a Virtex, type in the same directory as the .synopsys_dc.setup file:

synlibs -fc xfpga_virtex-4 >> .synopsys_dc.setup
where fc stands for FPGA compiler.
Note that -4 represents the speed grade -4.

This command will append the output of synlibs into the .synopsys_dc.setup file. The output of the synlibs command should be like the following:

link_library = {xfpga_virtex-4.db }
target_library = {xfpga_virtex-4.db }
symbol_library = {virtex.sdb}
define_design_lib xdw_virtex -path XilinxInstall + /synopsys/libraries/dw/lib/virtex
synthetic_library = {xdw_virtex.sldb standard.sldb}

If you have compiled the Virtex XDW libraries in the $XILINX tree, you can proceed to checking the .synopsys_vss.setup file. If the XDW libraries were compiled in the $XILINX area, then no modification of the define_design_lib line in the .synopsys_dc.setup file is needed. If the $XILINX/synopsys/libraries directory was copied locally, then the path in the .synopsys_dc.setup file must be changed to reflect the copied directory path. For example, if $XILINX/libraries was copied to /home/data, then the `define_design_lib' setting made by synlibs for the virtex device above should be edited to reflect the /home/data location:

define_design_lib xdw_virtex -path /home/data/libraries/dw/lib/virtex

If you do not plan on simulating in VSS, then you do not need to create a .synopsys_vss.setup file. To create the right .synopsys_vss.setup file, copy the $XILINX/synopsys/examples/template.synopsys_vss.setup to the same directory as the .synopsys_dc.setup file you made above. Rename template.synopsys_vss.setup to .synopsys_vss.setup. If you had to compile the A2.1i XSI simulation libraries outside of the $XILINX area, then the paths in the .synopsys_vss.setup file must be changed to reflect this new info. if the $XILINX/synopsys/libraries directory was copied locally, then the paths for the various simulation libraries inside .synopsys_vss.setup would be modified as follows:

UNISIM : /home/data/libraries/sim/lib/unisims
SIMPRIM : /home/data/libraries/sim/lib/simprims
LOGIBLOX : /home/data/libraries/sim/lib/logiblox
XC9000 : /home/data/libraries/sim/lib/xc9000/ftgs

Before starting on the tutorial, type the ls -l command in the directory where you created your .synopsys_dc.setup, .synopsys_vss.setup, and where you uncompressed and untar'd the file virtexxsivhdle.tar.Z file. Make sure that directory has the following items: .synopsys_dc.setup, .synopsys_vss.setup, and WORK.

FINAL NOTES

The purpose of this tutorial is familiarize you with the overall XSI A2.1i flow and common issues by using FPGA Compiler and VSS. This tutorial assumes that you are fluent in VHDL,

and familiar with FPGA Compiler, and VSS syntax. The design for this tutorial is a stopwatch. The tutorial has 3 major parts: functional simulation, synthesis and implementation, and timing simulation. Since most users will be using a version of Synopsys later than v1998.02, and most users do not have write permissions to their $XILINX area, the tutorial assumes that the user had to copy his $XILINX/synopsys/libraries directory to a local writable area. For this tutorial, it is assumed that the contents of $XILINX/synopsys/libraries was copied to /home/data, which was the path used as an example in the 'SETUP' section.

FUNCTIONAL SIMULATION

The purpose of this part of the tutorial is to run a functional simulation. You have already finished most of the setup when you created the WORK directory and made the .synopsys_vss.setup file. Now, you need to compile the design files and testbench, followed by running the VSS simulation tool. This tutorial assumes that you have uncompressed and untar'd and placed your setup files in /home/user/tutorial and replace your paths appropriately. One of the features of the A2.1i XSI VSS functional simulation flow is that instantiated XSI cells, like FDCE, or CLKDLL can be simulated.

(1) cd to the /home/user/tutorial/ directory. You should have already setup a .synopsys_dc.setup and .synopsys_vss.setup file in this directory, along with a WORK directory. If you have not setup these files along with the appropriate XSI libraries to your version of Synopsys, please go over the information in 'SETUP' first.

(2) In the /home/user/tutorial directory, you will have the following files:

cnt60.vhd
hex2led.vhd
smallcntr.vhd
stmchine.vhd
stopwatch.vhd
tenths.vhd
testbenchf.vhd
testbencht.vhd
func.sim
timing.sim
commandf.txt
commandt.txt
run.script
pr.script

These are the design files for the tutorial. There is a testbench for timing simulation and a testbench for functional simulation. The top-level file in this design is stopwatch.vhd.

(3) The functional simulation for this tutorial will be performed by running the following script in the /home/user/tutorial directory: func.sim. Open the contents of func.sim in a text editor and take note of the following:

Note that vhdlan is used with the -i option. Always use vhdlan with the -i option. By default, vhdlan uses the -c option, which will only work if your system is setup to use a certain type of C-compiler. For more information regarding C-compiler, please check out our solution 2311. http://support.xilinx.com/techdocs/2311.htm. If you want to use the -c option with vhdlan, please refer to the Synopsys web site for the proper setup.

Contents of the func.sim file:

```
#!/bin/csh -f
rm -r WORK
mkdir WORK
vhdlan -i tenths.vhd
vhdlan -i smallcntr.vhd
vhdlan -i cnt60.vhd
vhdlan -i hex2led.vhd
vhdlan -i stmchine.vhd
vhdlan -i stopwatch.vhd
vhdlan -i testbenchf.vhd
vhdlsim -e commandf.txt overall
```

Note: vhdlan is the VSS command to analyze vhdl files.

(4) Run the functional simulation by typing at the UNIX prompt:

./func.sim

This will start the functional simulation process by running the vhdlan commands. Did you get any errors? Do you know why the error happens? Do you know where to fix it? Before going to step (5), try to figure out the solution to the error.

(5) The code you are trying to simulate is not 100% RTL. If it was, the code would have simulated with no problems. VHDL code that is 100% RTL doesn't need any additional VHDL libraries (except for IEEE) for simulation. When VHDL code contains an instantiated component that does not have an underlying RTL behaviorial description, a RTL model must be made for functional simulation. In this case, the BUFGDLL has been instantiated. For more information on the BUFGDLL, refer to the Libraries Guide and the Databook. The UNISIM libraries must be created so that library cells instantiated from the XSI libraries can be functionally simulated. The VHDL code that instantiates any XSI library cell must contain the following two lines:

```
library UNISIM;
use UNISIM.vcomponents.all;
```

In general, you don't have to do this in every VHDL file in your design, only in the files that contain instantiated XSI library cells (like FDCE, RAM32X1S, BUFGDLL, etc.). It doesn't

hurt to place the above two lines in every VHDL file in your design.

For this tutorial design, what file needs the above two lines to make the func.sim file compile with no errors?

A: stopwatch.vhd

fixed stopwatch.vhd file:

```
library IEEE;
use IEEE.std_logic_1164.all;
library UNISIM;
use UNISIM.vcomponents.all;

entity stopwatch is

   port (    RESET : in STD_LOGIC;
             STRTSTOP : in STD_LOGIC;
             TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
             ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
             TENSOUT : out STD_LOGIC_VECTOR(6 downto 0);
             CLOCK: in STD_LOGIC
 );
end stopwatch;

architecture inside of stopwatch is

component BUFGDLL
   port (I: in STD_LOGIC; O: out STD_LOGIC);
end component;

component BUFG
   port (I : in STD_LOGIC;
         O : out STD_LOGIC);
end component;

component stmchine
   port (     CLK : in STD_LOGIC;
              RESET : in STD_LOGIC;
              STRTSTOP : in STD_LOGIC;
              CLKEN : out STD_LOGIC;
              RST : out STD_LOGIC
 );
end component;

component tenths
   port (     CLOCK : in STD_LOGIC;
```

```
            CLK_EN : in STD_LOGIC;
            ASYNC_CTRL : in STD_LOGIC;
            TERM_CNT : out STD_LOGIC;
            Q_OUT : out STD_LOGIC_VECTOR(9 downto 0));
end component;

component cnt60
   port (    CE : in STD_LOGIC;
            CLK : in STD_LOGIC;
            CLR : in STD_LOGIC;
            LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
            MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component hex2led
   port (  HEX : in STD_LOGIC_VECTOR(3 downto 0);
           LED : out STD_LOGIC_VECTOR(6 downto 0));
end component;

signal strtstopinv : STD_LOGIC;
signal oscout : STD_LOGIC;
signal clkint : STD_LOGIC;
signal clkenable : STD_LOGIC;
signal rstint : STD_LOGIC;
signal xcountout : STD_LOGIC_VECTOR(9 downto 0);
signal xtermcnt : STD_LOGIC;
signal cnt60enable : STD_LOGIC;
signal lsbcnt : STD_LOGIC_VECTOR(3 downto 0);
signal msbcnt : STD_LOGIC_VECTOR(3 downto 0);

begin

DLL:BUFGDLL port map(I=>CLOCK,O=>oscout);

CLOCKBUF:BUFG port map(I=>oscout,O=>clkint);

MACHINE:stmchine port map(CLK=>clkint,
            RESET=>RESET,
            STRTSTOP=>strtstopinv,
            CLKEN=>clkenable,
            RST=>rstint
 );

XCOUNTER:tenths port map(CLOCK=>clkint,
            CLK_EN=>clkenable,
            ASYNC_CTRL=>rstint,
```

```
            TERM_CNT=>xtermcnt,
            Q_OUT=>xcountout
                );

sixty: cnt60 port map(CE=>cnt60enable,
          CLK=>clkint,
          CLR=>rstint,
          LSBSEC=>lsbcnt,
          MSBSEC=>msbcnt
              );

lsbled:hex2led port map(HEX=>lsbcnt,
LED=>ONESOUT
                );

msbled:hex2led port map(HEX=>msbcnt,
LED=>TENSOUT
                );
cnt60enable<=xtermcnt and clkenable;
TENTHSOUT<=not(xcountout);
strtstopinv<=not(STRTSTOP);

end inside;
```

(6) After you have corrected the stopwatch.vhd file, the func.sim file will compile with no errors. The simulation will start and display the waveform viewer and the unix shell will display the vhdlsim prompt.

SYNTHESIS and IMPLEMENTATION

In this section of the tutorial, you will synthesize the design and create a placed and routed ncd file. After creating the place and routed ncd file, you can optionally proceed to create a .bit file for downloading to a Virtex board, using bitgen and promgen, and the Hardware Debugger.

(1) Functional simulation is performed to make sure that the desired RTL behavior has been implemented. Once the desired RTL behavior has been confirmed, the design can be synthesized. To synthesize a design with FPGA Compiler, you need to create a compile script. A default compile script is provided for your modification in the A2.1i software. Copy the file $XILINX/synopsys/example/template.fpga.script.virtex into your /home/user/tutorial directory, which contains the .synopsys_dc.setup and .synopsys_vss.setup files you created earlier. The XSI compile script is already provided in this tutorial, users can use run.script.

The file template.fpga.script.virtex is a template for a compile script that can be used for synthesizing into Virtex.
Here is the contents of the file template.fpga.script.virtex:

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                FPGA Compiler                  */
/*                                               */
/*  Targets the Xilinx XCV150PQ240-3 and assumes a   */
/*   VHDL source file by way of an example.          */
/*                                               */
/*   For general use with VIRTEX architectures.      */
/* ================================================*/

/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */

  TOP = <design_name>
                /* ========================== */
                /* Note: Assumes design file- */
                /* name and entity name are   */
                /* the same (minus extension) */
                /* ========================== */

  designer = "XSI Team"
  company  = "Xilinx, Inc"
  part     = "XCV150PQ240-3"

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

  analyze -format vhdl TOP + ".vhd"

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

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

```
  current_design TOP

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

  remove_constraint -all

/* If setting timing constraints, do it here.
  For example:                                */
/*
  create_clock <clock_pad_name> -period 50
*/


/* ================================================ */
/* 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 "*"
  insert_pads

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

  compile -map_effort med

/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */

  report_fpga > TOP + ".fpga"
  report_timing > TOP + ".timing"

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

  set_attribute TOP "part" -type string part

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

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

  write -format edif -hierarchy -output TOP + ".sedif"

/* ================================================ */
/* Write out the design to a DB.                    */
/* ================================================ */

  write -format db -hierarchy -output TOP + ".db"

/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy */
/* needs to be flattened before the constraints are */
/* written-out.)                                    */
/* ================================================ */

  write_script > TOP + ".dc"

/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */

  sh dc2ncf -w TOP + ".dc"

/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */

  exit

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

Here are the contents of run.script file in this tutorial. Use the file run.script here

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using    */
/*              FPGA Compiler                       */
/*                                                  */
/*  Targets the Xilinx XCV150PQ240-3 and assumes a  */
/*   VHDL source file by way of an example.         */
```

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

/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */

  TOP = stopwatch
                /* ========================= */
                /* Note: Assumes design file- */
                /* name and entity name are   */
                /* the same (minus extension) */
                /* ========================= */

  designer = "XSI Team"
  company  = "Xilinx, Inc"
  part     = "XCV150PQ240-3"

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


analyze -format vhdl "smallcntr.vhd"
elaborate smallcntr
compile

analyze -format vhdl "hex2led.vhd"
elaborate hex2led
compile

analyze -format vhdl "cnt60.vhd"
elaborate cnt60
current_design cnt60
set_dont_touch "lsbcount"
set_dont_touch "msbcount"

analyze -format vhdl "stmchine.vhd"
analyze -format vhdl "tenths.vhd"
analyze -format vhdl TOP + ".vhd"
```

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

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

current_design TOP
set_dont_touch “lsbled”
set_dont_touch “msbled”
set_dont_touch “DLL”
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */

  remove_constraint -all

/* If setting timing constraints, do it here.
  For example:                                */
/*
  create_clock <clock_pad_name> -period 50
*/
/* ================================================ */
/* 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 “*”
  remove_attribute find(port,”CLOCK”) port_is_pad
  insert_pads

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

  compile -map_effort med

/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */
```

```
/*   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing" */

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

   set_attribute TOP "part" -type string part

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

   write -format edif -hierarchy -output TOP + ".sedif"

/* ================================================ */
/* Write out the design to a DB.                    */
/* ================================================ */

/*   write -format db -hierarchy -output TOP + ".db" */

/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
/* ================================================ */

/*   write_script > TOP + ".dc" */

/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */

/*   sh dc2ncf -w TOP + ".dc" */

/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */

/*    exit */

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

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

Before using this script. Note the following items.
A. The design is compiled from the bottom up.

B. when compiling a Virtex design in FPGA Compiler, the proper type of output for place and route in A2.1i is an .sedif file.

C. there are ‘dont_touch’ commands added to the script. Whenever you instantiate a component from the XSI synthesis library, you must place a dont_touch on the instance to prevent Synopsys from deleting or modifying the library cell.

D. When synthesizing a Virtex design with FPGA Compiler, do not use uniquify;

This design is synthesized by compiling modules that are instantiated more than once, and uses ‘dont_touch’ attributes on these instances.

(2) Synthesize the design.
First, start the Design Analyzer tool by typing the following command in the directory that contains the .synopsys_dc.setup file for this design:

design_analyzer &

This will bring up the Design Analyzer GUI. When the GUI appears, Run the script run.script by selecting ‘Execute Script’ in the Setup menu. A pop-up window will appear where you can select the script ‘run.script’ to run. The user also has the option to synthesize the design in the dc_shell, use the following command:

dc_shell -f run.script

If the script run successfully, the script will stop and a .sedif file will be created. If an error is reported, keep the following information in mind:

(a) Make sure that the .synopsys_dc.setup file is setup correctly. Make sure that the ‘XDW’ synthesis libraries are compiled for the version of Synopsys you are using. Make sure that the paths referenced in the .synopsys_dc.setup file exist. Refer to the Setup section of this tutorial for more information.

(b) When you start the Design Analyzer, make sure you run the command ‘design_analyzer &’ in the directory that contains the .synopsys_dc.setup file.

Note: If you received the following error during synthesis:

Error: The package ’VITAL_Timing’ depends on the package ’std_logic_1164’
which has been analyzed more recently.

Please re-analyze the source file for 'VITAL_Timing' and try again. (LBR-28)

This error will occur because the UNISIM library is not for synthesis, it's for functional simulation purposes only. There are two ways to avoid this error:

A. Comment out

library unisim;
use unisim.vcomponents.all;

in the stopwatch.vhd file.

B. Add the following line in the .synopsys_dc.setup file:

hdlin_translate_off_skip_text = true
Please see Xilinx solution 4945 for more details
http://support.xilinx.com/techdocs/4945.htm

(3) Take the .sedif file produced by Design Analyzer, and place and route the design for timing simulation using the following script (pr.script is included in the tutorial directory):

```
#!/bin/csh -f
ngdbuild -p v50PC84-4 stopwatch.sedif
map stopwatch.ngd
par stopwatch.ncd stopwatch_r.ncd
ngdanno stopwatch_r.ncd
ngd2vhdl stopwatch_r.nga
```

Optionally, you can place and route the EDIF files by using the A2.1i GUI's. Please refer the Quick Start Guide Tutorial for more information for using the GUI's for place and route.

TIMING SIMULATION

To perform timing simulation, a SDF file and structural VHDL produced by ngd2vhdl must be used, along with a testbench. The script file timing.sim will perform the timing simulation. Run timing.sim in the directory that contains the .synopsys_vss.setup file you created.

(1) Open the script file timing.sim in a text editor and note the following:
vhdlan is used with the -i option. Always use vhdlan with the -i option. By default, vhdlan uses the -c option, which will only work if your system is setup to use a certain type of C-compiler. If you want to use the -c option with vhdlan, please refer to the Synopsys web site for the proper setup.

Since a timing simulation is performed, when vhdldbx or vhdlsim is invoked, the -sdf_top option must be specified first.

Use the file timing.sim here:

```
#!/bin/csh -f
rm -r WORK
mkdir WORK
vhdlan -i stopwatch_r.vhd
vhdlan -i testbencht.vhd
vhdlsim -sdf_top /testbenchf/uut -sdf stopwatch_r.sdf -e commandt.txt overall
```

(2) Close the timing.sim file in the text editor. Run the timing simulation, which will produce a waveform view like the functional simulation, by typing at the unix prompt:

./timing.sim

When the script runs, if you get errors/warnings/messages about ‘Bad Regions’, undefined libraries, or the instance is unbound, make sure the simulation libraries for A2.1i XSI VSS have been setup correctly. Make sure that the paths in the .synopsys_vss.setup file point to paths which exist in your setup. For further information on setup, please refer to ‘Setup’ section in this tutorial.

Note: The following error may occur when loading the design into the VSS simulator in timing simulation:

```
**Error: vhdlsim,260:
   (SDF File: timesim.sdf Line: 2512) generic
   /TESTBENCHT/UUT/DLL_CLKDLL_CLKDLL/TPERIOD_CLKIN_posedge is not
declared.
```

This is caused by our simprim models, and will be fixed in the future service pack of 2.1i. To workaround the problem, you need to modify the simprim_Vcomponents.vhd, and simprim_VITAL.vhd, and then recompile the simulation library. The files are located under $XILINX/vhdl/src/simprim directory. For detailed information on how to modify the files , please check our solution record 6720 at

http://support.xilinx.com/techdocs/6720.htm.