Virtex 2.1i tutorial: Verilog using FPGA Compiler and VerilogXL

This tutorial describes the Virtex design flow with Synopsys FPGA Compiler, and simulation flow with VerilogXL 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 to help the users to familiarize the A2.1i FPGA Compiler/VerilogXL design flow. A Verilog 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 Verilog and familiar with FPGA compiler and VerilogXL. If not, you may want to complete the Synopsys tutorial first.

OVERVIEW

To use this A2.1i XSI tutorial, you must use A2.1i, XSI Synopsys v1998.02 or newer, and VerilogXL v2.5 or better. You will need to use FPGA Compiler and VerilogXL 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 libraries are setup and compiled correctly. If they are, you can skip the SETUP step and go to the TUTORIAL FILES section. If not, the DesignWare 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 families 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.

For this tutorial, only the A2.1i XSI libraries related to synthesizing a Virtex device need to be compiled if you are using a version of Synopsys newer than v1998.02. All A2.1i XSI libraries have compile scripts which let you 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 into the directory, $XILINX/synopsys/libraries, go to the directory of your local copy area:
(1) Go 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.

TUTORIAL FILES

In your home directory, create a directory that will contain the files for this tutorial. In this empty directory, copy the file virtexxsiverilog.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 .synopsys_dc.setup file. 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 recommended to use the template.synopsys_dc.setup file as the Virtex .synopsys_dc.setup file due to certain Virtex specific features. You will then need to add the virtex libraries in the .synopsys_dc.setup file. This can be done 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 xfpga_virtex is the general library for all the virtex families. The virtex library currently has not been broken down into individual ones for each family. -4 stands for the speed grade.
The synlibs command will append its output 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 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 virtex above should be edited to reflect the /home/data location:

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

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

.synopsys_dc.setup
WORK

FINAL NOTES

The purpose of this tutorial is familiarize you with the overall XSI A2.1i flow and common issues using FPGA Compiler and VerilogXL. This tutorial assumes that you are fluent in Verilog, familiar with FPGA Compiler Syntax, and VerilogXL 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 were 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.
(1) cd to the /home/user/tutorial directory. You should have already setup a .synopsys_dc.setup file in this directory, along with a WORK directory. If you have not setup these files along with the XSI libraries appropriate to your version of Synopsys, please go over the information in 'SETUP' and 'Tutorial Files' first.

(2) In the /home/user/tutorial directory, in addition to the WORK directory, .synopsys_dc.setup file, you will have the following files:

cnt60.v
filesf.f
filest.f
hex2led.v
smallcntr.v
stmchine.v
stopwatch.v
tenths.v
testbenchf.v
testbencht.v
run.script
pr.script

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

(3) Functional simulation is performed by running the VerilogXL command “verilog” with the data file filesf.f.

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

verilog -f filesf.f

(5) You will get an many error from VerilogXL when you attempt to simulate. Here are three of them:

```
Compiling source file “testbenchf.v”
Compiling source file “stopwatch.v”
Compiling source file “stmchine.v”
Compiling source file “hex2led.v”
Compiling source file “cnt60.v”
Compiling source file “smallcntr.v”
Compiling source file “tenths.v”

Error!   Module or primitive (BUFGDLL) not defined        [Verilog-MOPND]
         "stopwatch.v", 27: BUFGDLL DLL(.I(CLOCK), .O(
          oscout));
Error!   Module or primitive (BUFG) not defined           [Verilog-MOPND]
         “stopwatch.v”, 24: BUFG CLOCKBUF(.I(oscout), .O(
         clkint));
```

(6) The Verilog code you tried to simulate is not 100% RTL code. The tutorial design contains instantiations XSI synthesis library cells (like CLKDLL, FDCE, BUFG etc.). In this case, you must tell the Verilog simulator where the models for these cells are located. You will do this by placing the ‘uselib directive in the top-level file of this design, stopwatch.v. Add this line to the top of your stopwatch.v file:

```
`uselib dir=$XILINX/verilog/src/UNIVIRTEX libext=.v
```

When you do this, replace the $XILINX text with the explicit path in your environment. If $XILINX is set to /home/software/xilinx, then the ‘uselib’ line would appear in the top of the stopwatch.v file as:

```
`uselib dir=/home/software/xilinx/verilog/src/UNIVIRTEX libext=.v
```

(7) After making the change, re-run:

verilog -f filesf.f

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 by using bitgen or 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/template.fpga.script.virtex into your /home/user/tutorial directory, which contains the .synopsys_dc.setup file 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 “*”
   set_pad_type -slewrate HIGH all_outputs()
   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
```

```
edifout_design_name = 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.     */
/* ================================================ */


read -format verilog “smallcntr.v”
compile
read -format verilog “hex2led.v”
compile
read -format verilog “cnt60.v”
current_design “cnt60”
set_dont_touch “lsbcount”
set_dont_touch “msbcount”
read -format verilog “stmchine.v”
read -format verilog “tenths.v”
read -format verilog TOP + “.v”

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


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

current_design TOP

set_dont_touch “lsbled”
set_dont_touch “msbled”

/* ================================================ */
/* 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’ 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.

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

```
#!/bin/csh -f
ngdbuild -p v50PC84-4 watch.sedif
map watch.ngd
par watch.ncd stopwatch_r.ncd
ngdanno stopwatch_r.ncd
ngd2ver -ul -w stopwatch_r.nga
```

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

TIMING SIMULATION

To perform timing simulation, a SDF file and structural Verilog produced by ngd2ver must be used, along with a testbench.

(1) To perform timing simulation, run the following command at the command-line:

verilog -f filest.f

filest.f contains three files. The glbl.v, timesim.v and testbench.v. By default, the Verilog file produced by ngd2ver (timesim.v) will use the $sdf_annotate directive which will annotate the SDF file with the Verilog file from ngd2ver. In Alliance 2.1i, the general procedure for specifying global set/reset during post-ngdbuild simprim simulation involves defining the global reset/set signals with the $XILINX/verilog/src/glbl.v module. The glbl.v module connects the global set/reset signal to the design. Therefore, it is necessary to compile this module with the other design files and load it along with the toplevel.v or the testbench.v for simulation.

Here is the filest.f:

glbl.v
testbench.v
timesim.v

Make sure that ngd2ver was run with the -ul option before running timing simulation.