Chapter 1

Workstation flow for Xilinx CoolRunner CPLDs

This tutorial provides Xilinx's workstation flow for Xilinx Cool-Runner (XCR) CPLD designs. The XPLA Workstation flow is different from the Xilinx Design Manager flow used for the XC9500 CPLDs. XPLA Workstation is a command line flow.

For Cadence, Exemplar, Mentor Graphics, Synplicity, Synopsys Design Compiler - FPGA Compiler, and Viewlogic designs, the design entry for targeting XCR devices is the same as that for targeting XC9500 devices. The exception is that LogiBLOX is not supported. This tutorial provides the procedure for implementing the calc design from the Mentor Graphics Interface/Tutorial Guide, and the watch design used in the Exemplar, Synopsys, and Synplicity tutorials. This tutorial contains the following sections.

- "Before Beginning the Tutorial"
- "Design Flow"
- "RTL Simulation"
- "Implementing the Watch and Calc design"
- "XCR Timing simulation"

Before Beginning the Tutorial

Set up your system to use XPLA Workstation as follows.

1. The XPLA Workstation software is available from ftp.xilinx.com, compressed in the xpla2_solaris.tar.Z and xpla2_hpux.tar.Z files.
2. Create the install directory **xpla** for XPLA Workstation.
3. From <install_path>/xpla, **uncompress** xpla2_solaris.tar.Z (xpla2_hpux.tar.Z) to generate the xpla2_solaris.tar (xpla2_hpux.tar) archive file.

4. Extract the following directories/files (**tar** xvf xpla2_solaris.tar) : xdsnrtut.pdf (this file), examples, xpla2_solaris.tar, bin, and lib. The <install_path>/xpla/examples directory contains the calc and watch design files used in this tutorial. The xpla2_solaris.tar file may be deleted.
5. Set the environment variable XPLA_PATH to the XPLA Workstation install directory in your .cshrc or .kshrc file.

C Shell:

```
% setenv XPLA_PATH <install_path>/xpla
Korn Shell:
% export XPLA_PATH=<install_path>/xpla
```

6. Add $XPLA_PATH/bin to the PATH environment variable.
7. To test XPLA Workstation installation:

```
cd $XPLA_PATH/examples
mkdir install_test
cp watch.edf install_test
cp watch.ctl install_test
cd install_test
xsh watch (this should produce jedec,fit,tim files)
cd ..
rm -r install_test
```

Design Flow

The design flow is to do a functional simulation using ModelSim or other third party simulator, and then use the third party CAE vendor to compile the Verilog, VHDL, or schematic files to an edif file. The <design>.edf file is input into XPLA Workstation, which produces a jedec file for programming the device and various results files, including timing simulation models. A timing simulation is then run using the timing simulation model.

For designs targeting the XC9500/XL/XV CPLDs, Xilinx Design Manager or WebPACK can be used for implementation. WebPACK is a PC based design system, and supports third party edif entry for both XC9500/XL/XV and XCR series CPLDs. Designs targeting the

XCR CPLDs can use either the workstation based XPLA Workstation or WebPACK, but not Xilinx Design Manager.

Functional simulation is the same for both XC9500/XL/XV and XCR devices. Simulating the timing of the XC9500/XL/XV is slightly different from that of the XCR CPLDs. The XV9500/XL/XV uses the simprims library and generates a verilog or vhdl and sdf file. Designs targeting XCR devices use a delay-annotated verilog (.vo) or vhdl (.vho) file for timing simulation.

RTL Simulation

Functional simulation is the same for the XC9500/XL/XV and XCR devices. A simulation library is not required for functional simulation.

For Verilog simulation, all behaviorally described (inferred) and instantiated registers should have a common signal which asynchronously sets or resets the registers. Toggling the global set/reset emulates the Power-On-Reset of the CPLD. If this is not done, the flip-flops and latches enter an unknown state.

Implementation

XPLA Workstation runs from the command line. The steps are to create the edif file using the CAE vendor software, create a control file using a text editor, and to run XPLA Workstation by entering **xsh** (**xsh2**) <design> at the Unix prompt. The **xsh** script is used to target XPLA 1 devices, and the **xsh2** script is used to target XCR22V10 and XPLA 2 devices.

1. Using the design entry tool and targeting the XC9000 or XCR library, create an edif file.
2. From the project directory, create <design> .ctl using a text editor. Alternately, copy an existing .ctl file such as $XPLA_PATH/examples/watch.ctl to <design>.ctl.
3. In <design>.ctl, provide the <design>.edf argument to the **-i** command.
4. Enter **xsh** (**xsh2**) with no argument to obtain device list. In <design>.ctl, edit **-dev** <device> to target appropriate device.
5. Run **xsh** (**xsh2**) <design>

This produces the jedec, tim, and fit files. It is recommended that users run three iterations of the design, assigning the **th** parameter to 10, 20, and 30. The fit and timing results should then be analyzed for efficiency and performance objectives.

Format of the control file

The control file allows the user to define pinout and compile options. The control filename must be the same as the design name, plus the extension of ".ctl", e.g. watch .ctl. The contents of a control file is provided in three sections: command, property, and pin_assignment. In the description below, boldtype indicates the default value used by the tool if an alternative is not specified. Only the command section is required, and only the **-i**, **-it**, and **-dev** commands in the command section are required. The # character is used to indicate a comment line in the control file.

To assign pins, the **-pre** command is set to **keep**, and the pin assignments are specified in the pin_assignment section. In some cases, the design entry tool or XPLA Workstation renames signals slightly, particularly bus signals. For example, a signal named din[7] may be renamed to din_7_. The pin assignments in the control file must match the signal names assigned by the fitter. If there is a problem, first fit the design using the **-pre ignore** command. Then re-fit the design, ensuring that the signal names provided in the .ctl file mathc those in the .fit or .spf file, using the **-pre keep** command.

COMMAND SECTION ([command])

Specify all the command line options here:

-i file	input filename
-it type	input type (edif or blif)
-th number	max pterm for each equation (5 - 37)
-fi number	max fanin for each equation
-bfi number	max fanin for each logic block (**36** - 40)
-vho	generate vhdl timing model
-vo	generate verilog timing model
-reg	apply register synthesis to the design

-co type	collapsing method (**best** or none)
- effort	synthesis effort (exhaust or **fast**)
-xor type	xor synthesis type (all, exp, or **none**)
-dev name	device name
-pre type	preassign handling: KEEP, **TRY**, or IGNORE

PROPERTY SECTION ([property])

maxpt : specify maximum pterm for each pin/node

keep : specify a signal's attribute as keep

retain : specify a signal's attribute as retain

mode : 0, 1, **4**

dut on | **off**

isp off | **on**

tri-state all

fm_group : group signals within a fast module

lb_group : group signals within a logic block

slow_slew_rate : slow or **fast**

PIN ASSIGNMENT SECTION ([pin_assignment])

[pin_assignment]

<signal_name>:<pin_location>

Control file example

```
[command]
-it edif
-i watch.edf
-dev xcr3128as10be
-reg
-pre keep
-th 30
# Rerun w th 20
```

```
[property]
maxpt bit0 12 bit 1 14
dut on
isp off # frees up 4 pins
[pin_assignment]
din_7_:5
din_6_:4
```

XCR Timing Simulation

Timing simulation is discussed in the Exemplar, Synopsys, and Synplicity tutorials, so this section is brief. The watch design is used as an example

VHDL

For timing simulation of a VHDL design using a XCR CPLD, two files are required.

- watch.vho - generated when **-vho** command is set in .ctl file
- testbencht.vhd

The testbencht.vhd file is an edited version of the original testbench.vhd file. In the VHDL timing model (watch.vho), the tensout, onesout, and tenthsout bus signals are broken into discrete signals. For simulation, the component signals and uut signals in the testbench and design model must match. The component and uut instantiation statements in testbencht.vhd have been edited to match those in watch.vho.

To perform timing simulation, follow these steps.

6. Create the work directory.

   ```
   vlib work
   ```

7. Compile the VHDL source files and the testbench.

   ```
   vcom watch.vho testbencht.vhd
   ```

8. Read in the files for timing simulation.

   ```
   vsim tbx_watch tbx_arch
   ```

Alternatively, select **`File`** → **`Load New Design`**. Click the Add button. Browse and select the design file. Type **`uut`** in the Apply to Region field and click the Load button.

9. View the necessary debugging windows by typing the following command at the ModelSim prompt.

   ```
   view wave signals source
   ```

10. View and add the signals of the design to the waveform window. Use the ModelSim **Combine** command to group the tensout and onesout signals into buses.

11. At the ModelSim prompt type.

    ```
    run 100000 ns
    ```

Verilog

For timing simulation of the Verilog design two files are needed.

- watch.vo - generated when **-vo** command is specified in .ctl file
- testfixturet.v

The testfixturet.v file is an edited version of the original tesfixture.v file. In the Verilog timing model (watch.vo), the tensout, onesout, and tenthsout bus signals are broken into discrete signals. For simulation, the component signals and uut signals in the testbench and design model must match.

To perform timing simulation, follow these steps.

1. Create the work directory.

   ```
   vlib work
   ```

2. Compile the Verilog file and the testfixture.

   ```
   vlog testfixturet.v watch.vo
   ```

3. Simulate the design

   ```
   vsim -L simprims test
   ```

   Now that the HDL netlist has been resolved into primitives, we must provide the simulation models to the SIMPRIM library.

4. View the necessary debugging windows by typing the following command at the ModelSim prompt.

```
view wave signals source
```

5. View and add the signals of the design to the waveform window.
6. At the ModelSim prompt type.

```
run 100000 ns
```

The XPLA Workstation Tutorial is now completed!