Running Questa CDC on FPGA Designs

This application note describes how to analyze clock-domain crossings in FPGA designs using Questa CDC.

The process includes:
- Exporting FPGA project file
- Using the correct FPGA library files
- Compiling the FPGA libraries
- Compiling the design files
- Running CDC static analysis
- Reviewing and debugging the static analysis results

This application note describes how to run CDC analysis in batch mode using a Makefile or shell scripts.
# Table of Contents

Overview .......................................................................................................................... 1  
Table of Contents .............................................................................................................. 2  
Introduction ....................................................................................................................... 3  
Vivado Export Flow ........................................................................................................... 4  
Phase 1: Vivado Button Setup ......................................................................................... 4  
Phase 2: Run Vivado Questa CDC Export ....................................................................... 7  
Phase 3: Run Questa CDC .............................................................................................. 9  
FPGA Project Export for Xilinx ISE, Intel Quartus, Microsemi Libero ......................... 12  
FPGA Compiled Libraries ............................................................................................. 15  
Phase 1: Specify FPGA Libraries .................................................................................. 15  
Phase 2: Compiling the Design ..................................................................................... 15  
Phase 3: Compiling a CDC Model of the Design .......................................................... 16  
Phase 4: Running GUI Debug ....................................................................................... 21  
Summary ......................................................................................................................... 21  
Appendix A: Installing FPGA Libraries ......................................................................... 22  
Appendix B: Compiling FPGA Source Libraries ........................................................... 23  
Appendix C: Compilation Targets for Xilinx and Intel/Altera Libraries ....................... 25  
Appendix D: Example templates for non-synthesizable Xilinx models ....................... 27  
Appendix E: Example templates for non-synthesizable Intel/Altera models ................ 28  
Appendix F: Incorporating XilinxCoreLib IP Models .................................................. 29
Introduction

CDC analysis is critical for FPGAs as it will identify CDC issues early in the design flow and avoid triage of CDC bugs in the lab. Triage of CDC bugs in the lab is very time consuming and frustrating, since the CDC bugs manifest in intermittent design failures that are dependent on the operating environment. Users may manually setup a FPGA design for CDC analysis, but this appnote will illustrate the main 2 ways of exporting an FPGA project into Questa CDC: the Vivado export flow and the standard FPGA project export.

CDC analysis of an FPGA design requires special handling. Vendor-supplied versions of these models are developed for simulation performance, so they may not be synthesizable. Synthesizable libraries are needed because CDC analysis (unlike simulation) is based on a netlist of the design. Building a netlist requires synthesizable code for compiling the design and for compiling the FPGA source libraries. The pre-compiled FPGA libraries designed for the Questa Verify tools are both synthesizable and formal-friendly.

Several types of models are provided by FPGA vendors:
   1) Functional models
      Sometimes vendors provide RTL models especially for library primitives.
   2) Behavioral models
      Behavioral models are not synthesizable and are compiled as black boxes for the Questa CDC and Formal tools.
   3) Empty models
   4) Encrypted models

Questa CDC/Formal distribution software comes with synthesizable versions of the unisim/simprim Xilinx and the Altera libraries. The 10.6 software includes both pre-compiled libraries and source code. Also included in the Xilinx and Altera subdirectories of fpga_libs are Makefiles and control files for running the flows described in the following sections.

The Questa CDC tools run on Linux/Unix. No support is available for Microsoft Windows. If a Linux machine is not available, it is possible to run Questa CDC in a Linux virtual machine under Windows.
Vivado® Export Flow

The Vivado export flow allows Vivado users to utilize their FPGA setup to create a Questa CDC environment. Users may download the Questa CDC Tcl script from the Xilinx TclApp Store. Alternatively, the Tcl script may be accessed from the Questa CDC installation.

Phase 1: Vivado Button Setup

1) Download the Questa CDC TclApp from the Xilinx TclApp Store

The Questa CDC Tcl script may be downloaded from the Xilinx TclApp Store (Vivado version 2016.4 and later) or referenced from the Questa CDC installation ($QHOME/share/fpga_libs/Xilinx/write_questa_cdc_script.tcl).

2) Source Questa CDC TclApp script

Start Vivado:
% vivado &

Go to the Vivado “Tcl Console” window and source the TclApp script:
% source write_questa_cdc_script.tcl
3) Install Questa CDC button
Run the Tcl procedure to create the Questa CDC button.
% write_questa_cdc_script -add_button

4) Exit Vivado
Users must exit Vivado, then restart Vivado.
After restarting Vivado, the Questa CDC button will be available in the Vivado GUI.

Alternate Vivado button setup methods:
In addition to installing the button from the Vivado GUI, users can also install the Questa CDC button with the following CSH scripts.

- From Questa CDC installation:
  % $QHOME/share/fpga_libs/Xilinx/setup_qcdc_vivado_button.csh

- 2) From Vivado installation (starting from 2017.3)
  %<vivado_installation>/data/XilinxTclStore/tclapp/mentor/questa_cdc/setup_qcdc_vivado_button.csh
Phase 2: Run Vivado Questa CDC Export

1) Start Vivado:
% vivado &

2) Open the Vivado project
Select “Open Project”, then select the project file. For example, select the “project_fpga.xpr” from the “project_fpga” directory.

3) Open Synthesized Design
To enable writing the SDC constraints file, the synthesized design must be loaded.

This design has already been synthesized. Select “Open Synthesized Design”. 
Note: If the “Open Synthesized Design” fails, select “Run Synthesis” to re-run the design synthesis.

Note: In the case that a design does not synthesize, specify the Questa CDC Tcl procedure option to bypass the SDC writing (“-use_existing_xdc” option).

4) Run the Questa CDC export

Click the Questa CDC button in the Vivado GUI. This will run Questa CDC setup, then invoke the Questa CDC GUI.

After the “Custom Command Arguments Entry” window pops up, click “OK”
Review the output files

The Tcl procedure generates 4 output files:

- **qcdc_run.sh**
  
  This shell script runs the design compilation and CDC analysis:

  ```bash
  rm -rf qft CDC_RESULTS
  $QHOME/bin/qverify -c -licq -l qcdc_demo_top.log -od CDC_RESULTS -do "\n  onerror {exit 1}; \
  do qcdc_ctrl.tcl; \ 
  sdc load demo_top_syn.sdc; \ 
  netlist load lib /Vivado/2016.4/data/parts/xilinx/artix7/devint/artix7.lib; \ 
  do qcdc_compile.tcl; \ 
  cdc run -d demo_top -L xil_defaultlib -formal -formal_effort high; \ 
  cdc generate report demo_top_detailed.rpt; \ 
  exit 0"
  
  - **qcdc_compile.tcl**
    
    This Tcl file compiles the design files.

  - **qcdc_ctrl.tcl**
    
    This Tcl file contains the design and CDC constraints

  - **demo_top_syn.sdc**
    
    **Phase 3: Run Questa CDC**

    This SDC constraints file is generated by the Vivado write_xdc command.

    1) Generate & review the design results

    From the GUI menu, click “Verify” => “Analyze Clocks”
Check the "CDC Setup Checks" and "CDC Policy Checks" tabs for any errors.
Go to the “Clocks” tab to review the clock groups.

<table>
<thead>
<tr>
<th>Group</th>
<th>Signal</th>
<th>Expression</th>
<th>Register Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group_cpu_clk_in (1)</td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Group_core_clk_in (1)</td>
<td></td>
<td></td>
<td>119</td>
</tr>
<tr>
<td>Group_mac_clk_in (1)</td>
<td></td>
<td></td>
<td>148</td>
</tr>
<tr>
<td>Primary Port Inferred (0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Box Inferred (0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undriven Inferred (0)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2) Run CDC analysis
Click the run button to run CDC analysis.

See the CDC tutorial for more information on running CDC analysis:
<installation>/share/doc/pdfdocs/tutorials_user.pdf
FPGA Project Export for Xilinx ISE, Intel Quartus, Microsemi Libero

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Library</th>
<th>File Extension(s)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel (Altera)</td>
<td>Quartus</td>
<td>.qsf</td>
<td></td>
</tr>
<tr>
<td>Microsemi</td>
<td>Libero</td>
<td>.prj .prjx</td>
<td></td>
</tr>
<tr>
<td>Xilinx</td>
<td>Vivado</td>
<td>.xpr</td>
<td>Support via Vivado Export Flow</td>
</tr>
<tr>
<td>Xilinx</td>
<td>ISE</td>
<td>.xise</td>
<td></td>
</tr>
</tbody>
</table>

FPGA Project Export – GUI Mode

1) Invoke the Questa CDC GUI
   % qverify &
2) Go to File => Import => FPGA Project File
3) Select the project file type

![Image of Import FPGA Project File dialog box]

- Choose "Inte(Altera) Quartus"
- Click "OK"

Run Vivado and import the script 'write_questa_cdc_script.tcl' from the Xilinx Tcl Store or from '<Questa CDC install>/share/fpga_libs/Xilinx'.

4) Specify the project file

![Image of Import Intel(Altera) Quartus Project File dialog box]

- Select "golden"
- Click "Open"

5) Run the batch file

```bash
% qverify -c -do fpga_run.tcl
```
FPGA Project Export – Batch Mode
The “netlist generate tcl” command is used to export a Tcl run script from the FPGA project file.

1) Run the netlist generate tcl command

```bash
qverify 1> netlist generate tcl -help
# Usage: netlist generate tcl
#       <tcl file> -project <project_file>
# Generate do file from FPGA project file.

Example:
qverify -c -do “netlist generate tcl fpga_run.tcl -project project_1.qsf”
```

```tcl
fpga_run.tcl:
netlist fpga -vendor Intel -library Quartus -device cycloneiii -version 12.1
vlog \
.../test5/rtl/afifo.v \
.../test5/rtl/Clk_ctrl.v \
.../test5/rtl/eth_miim.v \
...
-work work

cdc run -d top
```

2) Run the generated Tcl file

```bash
% qverify -c -do fpga_run.tcl
```
FPGA Compiled Libraries

Note: Due to its large size, FPGA libraries are no longer included in the release and must be installed separately.

If your installation software is used for analyzing designs with supported FPGA models, you must install the distribution FPGA libraries (See Appendix A). These have FPGA models engineered and verified to support the requirements for the Questa CDC/Formal model compilers and are designed to work efficiently with Questa CDC/Formal tools.

Phase 1: Specify FPGA Libraries
Pre-compiled libraries are available for Xilinx and Altera FPGA designs. Users should specify. The CDC analysis will use the specified pre-compiled FPGA libraries for FPGA models when compiling formal and CDC models.

Specifying FPGA libraries prior to CDC analysis:
  netlist fpga -vendor xilinx -library vivado -version 2016.3
  OR
  netlist fpga -vendor intel -library quartus -version 16.0
  OR
  netlist fpga -vendor microsemi -library libero -version 11.7_sp1
  OR
  netlist fpga -vendor lattic -library diamond -version 3.6

Phase 2: Compiling the Design
You likely have created one or more filelists that identify the source code files for your design. The files within each individual library must be compiled in the correct order. Also, different libraries that depend on each other must be compiled in the correct order. Using filelists from simulation handles this.

With the filelists (or filelists) for your design, run the design compilation commands.

The following example shows the compiler invocation for a Verilog design:
  vlib qstatic_libs/work
  vmap work qstatic_libs/work
  vlog -f verilog_files.list -work work +incdir+src/Verilog

The following example compiles a VHDL-2002 design into the work library.
  vlib qstatic_libs/work
  vmap work qstatic_libs/work
  qverify -c "netlist fpga -vendor xilinx -library vivado -version 2015.4; \ vcom -f vhdl_files.list -work work -skipsynthoffregion"

Note: vcom is being compiled within the qverify Tcl shell, so the netlist fpga directive will setup the referenced VHDL libraries (i.e. unisims and simprims).

The -skipsynthoffregion argument directs the compiler to obey synthesis-off regions. These are regions of code surrounded by synthesis_off/synthesis_on (or
translate_off/translate_on) pragmas. The compiler parses this code to pick up declarations, but otherwise ignores it. One reason to use -skipsynthoffregion is to ignore VHDL library and use statements for libraries needed for simulation only. Rather than using a filelist to compile files with one invocation, you can set up a script to compile the file one-by-one:

```sh
qverify -c "netlist fpga -vendor xilinx -library vivado -version 2015.4; \
vcom vhdl_file1.vhd -work work; \
vcom vhdl_file2.vhd -work work; \
vcom vhdl_file3.vhd -work work -skipsynthoffregion"
```

**Phase 3: Compiling a CDC Model of the Design**

The target design is the top-level block for CDC analysis. This can be a VHDL entity (or configuration) or a Verilog module. The -d <design> argument to the cdc run command is a required argument that identifies the target design.

Once the FPGA resource libraries and the design library have been compiled, you can compile the CDC logic model using the cdc run command. Here are two examples:

```sh
cdc run -d DUT_top -work work
    Compiles the target design DUT_top from work using libraries mapped in ./modelsim.ini (the default mapping file).

cdc run -d DUT_top -work top_lib -modelsimini LibraryMapping.0in
    Compiles the target design DUT_top from top_lib using libraries mapped in LibraryMapping.0in.
```

Use the following steps to compile a CDC model of the design.

1. **Set up the directive files.**
   A directives file is a Tcl-based file of Questa Functional Verification directives. Directives set up operating conditions, define clocks, define black boxes, specify custom synchronizers, modify reported results, create waivers, and so on. You apply significant effort creating and adjusting the directives files because this is how you fine tune CDC analysis. Here is an example:

   ```tcl
   netlist constant scan_mode '0'
   netlist clock CLK_1 -group clk grp A -period 4
   netlist clock CLK_2 -group clk grp A -period 8
   netlist clock CLK_3 -group clk grp B -period 11
   netlist port domain input_port1 -async
   netlist port domain input_port2 -clock CLK_1
   netlist port domain -output -clock CLK_2
   cdc reconvergence -depth 1 -divergence_depth 1
   netlist blackbox syncA* cdi_master
   cdc preference -black_box_empty_module
   ```

In addition to standard CDC directives, the following directives are particularly useful for CDC analysis of FPGA designs.

**netlist constant**
Applies a constant value to input ports (and sometimes to internal nodes) so the cdc compiler can prune irrelevant logic from the design logic and the CDC model. This technique makes the memory footprint smaller, improves performance and ensures only relevant results are returned.

**cdc reconvergence**
Sets the sequential levels that define how deep paths diverge and reconverge to be considered instances of reconvergence and single_source_reconvergence schemes. The deeper the analysis, the greater the decrease in performance. Initially, set the reconvergence depth to 1 and the divergence depth to 1.

**netlist blackbox**
Identifies specific modules/entities/architectures as user-defined black boxes. Use netlist port domain directives to identify the clock domains for the black boxes’ ports (even asynchronous ones) so the logic outside the black box instances can be analyzed properly. Fanin/fanout logic of ports of user-defined black box instances that are not assigned port domains is ignored for CDC analysis.

**cdc preference -black_box_empty_module**
Turns empty modules/entities/architectures into inferred black boxes instead of treating them as regular models. Some elements in a synthesizable FPGA library are “stubs” containing only the port declarations. Specifying the -black_box_empty_module option makes it easier to identify these elements so you can add netlist port domain directives for their ports.

2. Run cdc run in report-clocks mode.
For example:
```
prompt> qverify -c -do "\n  netlist fpga -vendor xilinx -library vivado -version 2004.3;\n  onerror {exit 1} ;\n  do cdc_directives.do ;\n  configure error -inferred black_box ;\n  cdc run -report_clock -d DUT_top -work work ;\n  exit 0"
```
The command performs clock analysis and stops. Check the results:

a) **Check cdc_run.log for errors:**

Error/Warning Summary
----------------------------------------------------------------------------------------------------------------------------------------
Count Type Message ID Summary
----------------------------------------------------------------------------------------------------------------------------------------
1 Error command-188 Design elaboration failed.
1 Error command-195 Design Elaboration (Child process) returned a non zero status.
1 Error parser-284 Vopt error.
Each error/warning is explicitly described in cdc.log. Fix any issues, then rerun design compilation (if the source code changed) and cdc -report_clock.

b) Check the clock groupings in cdc.rpt.

Clock Group Summary for 'demo_top'
=====================================
Total Number of Clock Groups : 4
1. User-Specified : (3)
2. Inferred : (0)
2.1 Primary : 0
2.2 Undriven : 0
2.3 Blackbox : 0
2.4 Gated Mux : 0
2.5 Gated Combo : 0
3. Ignored : (1)

==
1. User-Specified (3)

==
Group 0 (148 Register Bits, 0 Latch Bits)
-------
mac_clk_in
Group 1 (119 Register Bits, 0 Latch Bits)
-------
core_clk_in
Group 2 (0 Register Bits, 0 Latch Bits)
-------
check_en (unused clock)
check_en_r1 (unused clock)

==
2. Inferred (0)

==
2.1 Primary (0)

None
2.2 Blackbox (0)

None
2.3 Undriven (0)

None
2.4 Gated Mux (0)
None

2.5 Gated Combo (0)

None

3. Ignored(1)

==

Group 0(35 Register Bits, 0 Latch Bits)

----------

cpu_clk_in

Synchronous clocks should be grouped together. Clocks in different groups are assumed to be asynchronous and therefore require synchronization on signals that traverse storage elements in different clock domains. CDC analysis results are not meaningful until the clocks are set up correctly (see “Clock Tree Detection” on page 97).

3. Run cdc run.

For example:

```bash
prompt> qverify -c -do "\n  onerror {exit 1} ; \n  do cdc_directives.do ; \n  configure error -inferred black_box ;\n  cdc run -report_clock -d DUT_top -work work ; \n  exit 0"
```

The command performs clock analysis, compiles the CDC model of the design, runs CDC analysis, generates reports on the results and generates a database file to load into the CDC GUI for debugging issues found by static CDC analysis. Among the files generated by cdc:

- cdc.db — .db database of the CDC results for loading into the CDC GUI.
- cdc.rpt — Text report containing CDC results.
- cdc_design.rpt — Text report containing results of clock and design analysis

4. Check the cdc_design.rpt again.

  a) Check the port domains:

Port Domain Information

Port Direction Constraints Clock Domain Type

----------

clock input Clock Bus {clk[1]} Questa CDC
reset input Reset {clk[1]} Questa CDC
in1 input {clk[0]} User
in2 input {clk[0]} User
in3 input {clk[0]} User
out1 output {clk[1]} Questa CDC
out2 output {clk[1]} Questa CDC

Check the inferred port domains (clock domains assigned to the ports). By default, each input port is assigned to the clock domain of its first fan-in register. Any primary inputs or outputs that connect to multiple clock domains or are not assigned to a clock domain are listed. Use netlist port domain directives to make adjustments.

b) Check for black boxes.
Black Boxes:

Module Instance Count File (Line)
----------------------------------------
cdi_master 1 /home/cdc/blackbox/cdi_master.v (2)

Internal logic of the black boxes is unknown and in particular, the connectivity between a black box’s inputs and outputs is unknown. So, black boxes can mask some CDC problems. Check that the port domains of the user-defined black boxes (blackbox in the report) are all specified.

VITAL models, FPGA library elements that are not synthesizable and design blocks with unsynthesizable constructs are inferred black boxes (inferred in the report), unless explicitly specified with netlist blackbox directives. Check the inferred black boxes in cdc.rpt. If an inferred black box affects CDC results, at least one associated black box CDC scheme is reported:

Black Box Crossing. (blackbox)
----------------------------------------
tx_clk: start: tx_sig2 (/u/zin/blackbox/dut.v : 25)
<clock N/A>: end: dut.Di2 (/u/zin/dut.v: 40)(ID:blackbox_12944)

You can declare the black box as a user-defined black box (with netlist blackbox) and specify the port domains for the black box’s I/O ports (with netlist port domain). There are 2 options for resolving black boxes:

Option 1: Specify constraints for black boxes
See Appendix B for example constraints for Xilinx models.
See Appendix C for example constraints for Altera models.

Additionally, see examples in the installation:
See <install_dir>/share/fpga_libs/Xilinx/xilinx_black_box_examples.tcl
See <install_dir>/share/fpga_libs/Altera/altera_black_box_examples.tcl

Option 2: Specify synthesizable models
You might be able to obtain or write synthesizable models of various black boxed elements. For example, using the Xilinx CoreGen tool: run Project >Project
Options; select the Generation tab; and set Simulation Files: Structural. Structural models are synthesizable. Be sure to keep the structural models and behavioral models in different locations to prevent overwriting previously-generated files.

**Phase 4: Running GUI Debug**

To run qverify in GUI mode, specify the CDC results database as the only argument:

```
prompt> qverify cdc.db
```

At this point, debugging CDC issues with the CDC GUI is the same for FPGA-based design as it is with other designs. As you analyze the CDC results, you will find RTL issues to fix, to waive and to filter out. You might want to add or change directives in your control file to:

- Adjust clock configurations (netlist clock)
- Set clock domains of I/O ports (netlist port domain)
- Declare custom synchronizers (cdc custom sync)
- Define characteristics of certain signals in the design (cdc signal)
- Reclassify the results (cdc report)

**Summary**

The Makefile scripts are an easy way to compile FPGA libraries for CDC analysis. The Tcl templates provide directives for constraining the non-synthesizable FPGA models.

For more information:
- Go to: [http://www.mentor.com/supportnet](http://www.mentor.com/supportnet)
- Email support@mentor.com
- Call 1-800-547-4303
Appendix A: Installing FPGA Libraries

The standard installation does not include FPGA libraries. If you would like to analyze designs with FPGA models, you must install the distribution FPGA libraries. These have FPGA models engineered to support the requirements for the Questa CDC/Formal model compilers and are designed to work efficiently with the CDC/Formal tools.

1. Select the V10.6x.fpga.tgz link and download the file. Do the same for install_fpga_libs.
2. Make the installer executable and run it.
   ```
   prompt> chmod u+x install_fpga_libs
   prompt> install_fpga_libs
   ```
3. Be sure the installation software is already installed
   This will install Questa formal FPGA libraries.
   Questa formal release should have been already installed.
   OK [o, Enter], Cancel [c]
4. Enter o and then you will be prompted to enter the path to the installation software:
   ```
   Please enter Questa formal installation path.
   [/zin/tools/latest]
   /u/qformal/10.6
   ```
   The files are extracted from the tgz file and installed in the proper location
   (<install_dir>/share/fpga_libs).
5. Check that the libraries are installed.
   ```
   prompt> ls install_dir/share/fpga_libs
   Actel XILINX_ISE_13.4_HDM_CELLS.tcl
   Altera XILINX_ISE_14.7_FORMAL_CELLS.lst
   ALTERA_QUARTUS_14.0_FORMAL_CELLS.lst
   XILINX_ISE_14.7_FORMAL_CELLS.tcl
   ALTERA_QUARTUS_14.0/FormAL_CELLS.tcl
   XILINX_ISE_14.7_HDM_CELLS.lst
   ALTERA_QUARTUS_14.0_HDM_CELLS.lst XILINX_ISE_14.7_HDM_CELLS.tcl
   ALTERA_QUARTUS_14.0_HDM_CELLS.tcl
   XILINX_VIVADO_2014.3_FORMAL_CELLS.lst
   Encryption_FORMAL.tcl XILINX_VIVADO_2014.3_FORMAL_CELLS.tcl
   fpga.libmap.txt XILINX_VIVADO_2014.3_HDM_CELLS.lst
   fpga.support.txt XILINX_VIVADO_2014.3_HDM_CELLS.tcl
   lib XILINX_VIVADO_2014.4_FORMAL_CELLS.lst
   modelsim.ini XILINX_VIVADO_2014.4_FORMAL_CELLS.tcl
   Xilinx XILINX_VIVADO_2014.4_HDM_CELLS.lst
   XILINX_ISE_13.4_HDM_CELLS.lst XILINX_VIVADO_2014.4_HDM_CELLS.tcl
   ```
   When running a Questa CDC/Formal tool with an installed FPGA library, you must specify the desired library in the qverify session using the “netlist fpga” directive before running netlist compile to create the netlist used to generate the analysis models.
Appendix B: Compiling FPGA Source Libraries

If you have compiled your FPGA source libraries already for Questa simulation, you can use them for CDC analysis if:

- The libraries’ RTL is synthesizable VHDL, Verilog or SystemVerilog code.
- The libraries were compiled using the versions of Questa vcom/vlog commands that match those shipped with the Questa CDC/Formal distribution software.

Additionally, the standard FPGA library installation includes pre-compiled libraries for the latest Xilinx and Altera libraries. If your design requires a version that is not pre-compiled, you can manually compile the needed library version.

Set up and compile the FPGA source libraries as illustrated in the following examples. If FPGA library elements are instantiated in VHDL code, you must compile the VHDL components for the library. The logical library name for this VHDL library has no _ver suffix. The synthesizable library elements are specified in Verilog, you must compile the Verilog elements for the library. The logical library name for this Verilog library has a _ver suffix.

**Xilinx**

A Makefile script will compile the VHDL files containing the component and package declarations from the standard Xilinx simulation library, then compile the synthesizable Verilog models of the library.

**Step 1: Copy the Makefile from the Questa CDC installation**

```
% cp <installation>/share/fpga_libs/Xilinx/Makefile .
```

**Step 2: Edit the Xilinx path in the Makefile**

In the Makefile, modify the XILINX_SIM_LIB variable to point to your Vivado installation:

```
```

**Step 3: Compile the Xilinx libraries**

```
% make xilinx_libs
```
Intel/Altera
A Makefile script will compile the VHDL files containing the component and package declarations from the standard Altera simulation library, then compile the synthesizable Verilog models of the library.

Step 1: Copy the Makefile from the Questa CDC installation
   % cp <installation>/share/fpga_libs/Altera/Makefile .

Step 2: Edit the Quartus path in the Makefile
In the Makefile, modify the ALTERA_SIM_LIB variable to point to your Quartus installation:

   ALTERA_SIM_LIB=/tools/quartus/

Step 3: Compile the Altera libraries

   Option 1: Compile libraries for all device families
   To compile for all device families:
   % make altera_libs_all

   Option 2: Compiling for a single device family
   1) Define the device family:
      # device_family=stratixiigx
   2) Compile for the device family
      % make altera_libs

Logical-physical Library Mappings
The vmap command creates a logical-to-physical library mapping. For example, in the previous examples, vmap mapped the logical name altera_mf to the physical location qstatic_libs/altera_mf. The command also updates the modelsim.ini file with the logical-physical mapping. The command creates a new modelsim.ini file if one does not exist. This example shows the library mappings for a VHDL-only Xilinx design:

   [Library]
   unisim = ./qstatic_libs/unisim
   XilinxCoreLib = ./qstatic_libs/XilinxCoreLib
   work = ./qstatic_libs/work
Appendix C: Compilation Targets for Xilinx and Intel/Altera Libraries

The FPGA library installation includes Makefile scripts for manually compiling FPGA libraries. If your design requires a version that is not pre-compiled, you can manually compile the needed library version.

The following describes the Makefile scripts for manually compiling FPGA libraries:

**Xilinx**

- **unisim**
  Used for library elements instantiated in VHDL. Compile the VHDL files containing the component and package declarations from the standard Xilinx simulation library. Then compile the synthesizable Verilog models of the library. The vlog – convertallparams option is needed to convert the Verilog parameters to match the generics types in the VHDL component definitions.
    
    ```
    vlib qstatic_libs/unisim
    vmap unisim qstatic_libs/unisim
    vcom -work unisim $XILINX/vhdl/src/unisims/unisim_VCOMP.vhd
    vcom -work unisim $XILINX/vhdl/src/unisims/unisim_VPKG.vhd
    vlog -work unisim -convertallparams \
    install_dir/share/fpga_libs/Xilinx/ISE/xeclib/unisims/*.v
    ```

- **unisims_ver**
  Used for library elements instantiated in Verilog. Compile the synthesizable Verilog models of the Xilinx library.
    
    ```
    vlib qstatic_libs/unisims_ver
    vmap unisims_ver qstatic_libs/unisims_ver
    vlog -work unisims_ver \
    install_dir/share/fpga_libs/Xilinx/ISE/xeclib/unisims/*.v
    ```

- **simprim**
  Used for library elements instantiated in VHDL. Compile the VHDL files containing the component and package declarations from the standard Xilinx simulation library. Then compile the synthesizable Verilog models of the library. The vlog – convertallparams option is needed to convert the Verilog parameters to match the generics types in the VHDL component definitions.
    
    ```
    vlib qstatic_libs/simprim
    vmap simprim qstatic_libs/simprim
    vcom -work simprim $XILINX/vhdl/src/simprims/simprim_Vcomponents.vhd
    vcom -work simprim $XILINX/vhdl/src/simprims/simprim_Vpackage.vhd
    vlog -work simprim -convertallparams \
    install_dir/share/fpga_libs/Xilinx/ISE/xeclib/simprims/*.v
    ```

- **simprims_ver**
  Used for library elements instantiated in Verilog. Compile the synthesizable Verilog models of the Xilinx library.
    
    ```
    vlib qstatic_libs/simprims_ver
    vmap simprims_ver qstatic_libs/simprims_ver
    vlog -work simprims_ver \
    install_dir/share/fpga_libs/Xilinx/ISE/xeclib/simprims/*.v
    ```

- **XilinxCoreLib**

---

Mentor Graphics Confidential
Used for library elements instantiated in VHDL. First, run xilinxcorelib_compiledo to create a filelist (xilinxcorelib_vhdl_analyze_order.f) that specifies the synthesizable files in the correct compilation order. This script adds $XILINX/vhdl/src/XilinxCoreLib/ to the file names in the source file compile list. Next, compile the VHDL simulation library files.

- **XilinxCoreLib**
  Used for library elements instantiated in VHDL. Compile the VHDL simulation library files.
  ```
  vlib qstatic_libs/XilinxCoreLib
  vmap XilinxCoreLib qstatic_libs/XilinxCoreLib
  vcom -work XilinxCoreLib -f xilinxcorelib_vhdl_analyze_order.f
  ```

- **XilinxCoreLib_ver**
  Used for library elements instantiated in Verilog. Compile the Verilog simulation library files.
  ```
  vlib qstatic_libs/XilinxCoreLib_ver
  vmap XilinxCoreLib_ver qstatic_libs/XilinxCoreLib_ver
  vlog -work XilinxCoreLib_ver $XILINX/verilog/src/XilinxCoreLib/*.v
  ```

**Intel/Altera**

If FPGA library elements are instantiated in VHDL code, you must compile a resource library for that. The logical library name for this library has no _ver suffix. If FPGA library elements are instantiated in Verilog code, you must compile a resource library for that. The logical library name for this library has a _ver suffix.

- **altera_mf**
  Used for library elements instantiated in VHDL. Compile the VHDL files containing the component and package declarations from the standard Altera simulation library. Then compile the synthesizable Verilog models of the library. The vlog +incdir argument is the include directory for the source files.
  ```
  vlib qstatic_libs/altera_mf
  vmap altera_mf qstatic_libs/altera_mf
  vcom -work altera_mf \
  $QUARTUS_ROOTDIR/eda/sim_lib/altera_mf_components.vhd
  vlog -work altera_mf \
  install_dir/share/fpga_libs/Altera/quartus/fv_lib/verilog/*.v \
  +incdir+install_dir/share/fpga_libs/Altera/quartus/fv_lib/Verilog
  ```

- **altera_mf_ver**
  Used for library elements instantiated in Verilog. Compile the synthesizable Verilog models of the Altera library. The vlog +incdir argument is the include directory for the source files.
  ```
  vlib qstatic_libs/altera_mf_ver
  vmap altera_mf_ver qstatic_libs/altera_mf_ver
  vlog -work altera_mf_ver \
  install_dir/share/fpga_libs/Altera/quartus/fv_lib/verilog/*.v \
  +incdir+install_dir/share/fpga_libs/Altera/quartus/fv_lib/Verilog
  ```
Appendix D: Example templates for non-synthesizable Xilinx models

See `<install_dir>/share/fpga_libs/Xilinx/xilinx_black_box_examples.tcl`

Specify the fifo_32x8 as a black box:

```
netlist blackbox fifo_32x8
netlist port domain din -clock wr_clk -module fifo_32x8
netlist port domain wr_en -clock wr_clk -module fifo_32x8
netlist port domain dout -clock rd_clk -module fifo_32x8
netlist port domain rd_en -clock rd_clk -module fifo_32x8
netlist port domain empty -clock rd_clk -module fifo_32x8
netlist port domain full -clock wr_clk -module fifo_32x8
netlist port domain rst -async -module fifo_32x8
```

Specify the dpram_2kx9 as a custom synchronizer:

```
cdc custom sync dpram_2kx8 -type dpram_2kx8
cdc custom sync data -from -to -type dpram_2kx8
hier port domain *a -clock clka -module dpram_2kx8
hier port domain *b -clock clkb -module dpram_2kx8
```
Appendix E: Example templates for non-synthesizable Intel/Altera models

See <install_dir>/share/fpga_libs/Altera/altera_black_box_examples.tcl

Specify altsyncram as a black box:

```tcl
cdc blackbox memory altsyncram \ 
-write_addr address_a -read_addr address_b \ 
-data_in data_a -data_out q_b
netlist port domain address_a data_a -clock clock0 -module altsyncram
netlist port domain address_b q_b -clock clock1 -module altsyncram
netlist port domain wren_a rden_a byteena_a -clock clock0 -module \ 
altsyncram
netlist port domain wren_b rden_b byteena_b -clock clock1 -module \ 
altsyncram
netlist port domain clocken0 clocken1 addressstall_a -clock clock0 -module \ 
altsyncram
netlist port domain clocken1 clocken3 addressstall_b -clock clock1 -module \ 
altsyncram
```

Specify the scfifo as a black box:

```tcl
netlist blackbox scfifo
netlist port domain -input -clock clock -module scfifo
netlist port domain -output -clock clock -module scfifo
```

Specify the dcfifo as a custom synchronizer:

```tcl
cdc custom sync dcfifo -type dcfifo
  cdc custom sync data -from data -to q -module dcfifo
  hier port domain wr* -clock wrclk -module dcfifo
  hier port domain data -clock wrclk -module dcfifo
  hier port domain rd* -clock rdclk -module dcfifo
  hier port domain q -clock rdclk -module dcfifo
  hier port domain aclr -async -module dcfifo
```

Specify altdpram as a custom synchronizer:

```tcl
cdc custom sync altdpram -type altdpram
  cdc custom sync data -from data -to q -module altdpram
  hier port domain wren -clock inclock -module altdpram
  hier port domain data -clock inclock -module altdpram
  hier port domain wraddress -clock inclock -module altdpram
  hier port domain wraddresstall -clock inclock -module altdpram
  hier port domain inclocken -clock inclock -module altdpram
  hier port domain byteena -clock inclock -module altdpram
  hier port domain rden -clock outclock -module altdpram
  hier port domain rdaddress -clock outclock -module altdpram
  hier port domain rdaddressstall -clock outclock -module altdpram
  hier port domain outclocken -clock outclock -module altdpram
  hier port domain q -clock outclock -module altdpram
  hier port domain aclr -async -module altdpram
```
Appendix F: Incorporating XilinxCoreLib IP Models

For models such as the XilinxCoreLib and other complex IPs, there is an automated process for generating CDC models:

1. Compile the IP simulation model
   % vlog ... -y <Xilinx installation>/ISE_DS/ISE/verilog/src/XilinxCoreLib/
   +libext+v

2. Specify IP block name with “hier block” directive
   “hier block BLK_MEM_GEN_V8_2”

3. Run cdc analysis to generate HDM hc dc setup
   The CDC analysis will generate:
   - hc dc_run.Makefile: makefile to generate the CDC model

4. Generate the binary CDC model (hierdb)
   % make -f hc dc_run.Makefile BLOCKS

5. Convert the binary model into a TCL model
   % `qverify -c -do “hier generate tcl \
     hc dc_run/BLK_MEM_GEN_V8_2/hcdc_BLK_MEM_GEN_V8_2.hierdb”

6. Modify the CDC Tcl model
   Remove the comments from the Tcl model.

7. Reference the CDC Tcl model in the CDC run
   % qverify ... -do “\n   ...   hier block BLK_MEM_GEN_V8_2 -user_specified; \n   do hc dc_BLK_MEM_GEN_V8_2.tcl; \n   cdc run -d TOP_MODULE; \n   exit”