Design Compiler® FPGA FPGA Synthesis Process for Design Compiler Users

This document gives you guidelines for performing the basic steps for processing your FPGA design using Design Compiler FPGA and interfacing with FPGA vendors place and route tools. The target audience is designers that are familiar with the Design Compiler ASIC synthesis flow.

This document contains the following sections:

1 FPGA Synthesis Process

Figure 1 shows the FPGA synthesis process required to synthesize an FPGA design targeting either a Xilinx or an Altera FPGA device. This process is for a basic Design Compiler FPGA flow. Each step in the process is described further in this document. For more information, see the *Design Compiler FPGA User Guide*.

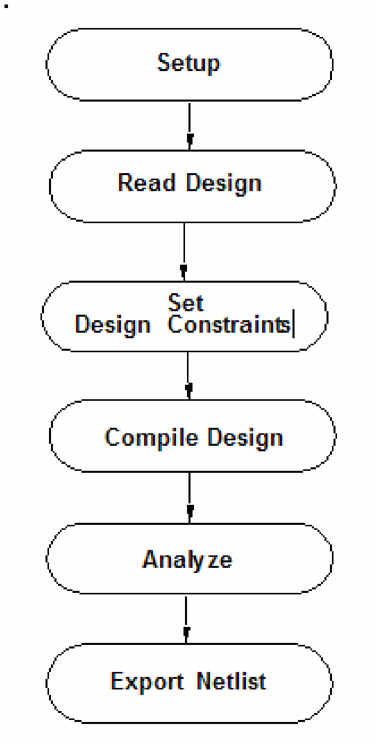

Figure 1 FPGA Synthesis Process for a Basic Design Compiler FPGA Flow

1.1 Guidelines for Using Design Compiler Scripts

Attention must be taken when migrating ASIC synthesis scripts directly from Design Compiler to Design Compiler FPGA. Although the scripts are compatible, Design Compiler scripts can typically have many commands that are specific only for an ASIC synthesis flow and in most cases will have no effect in Design Compiler FPGA. To make the migration effortless, any ASIC script that you want to use in Design Compiler FPGA should be confined within the boundary of the basic steps described in the following sections.

For default Design Compiler FPGA synthesis scripts targeting an Altera and a Xilinx device, see Example 1 and Example 2 at the end of this document.

1.2 Setting up the Environment for Synthesis

The first step in the synthesis process is to specify the tool and design related setup information. For proper tool setup, you need to use the FPGA device specific Synopsys setup files located under the Design Compiler FPGA installed library location. The setup files give the location of the device target, link and synthetic libraries, and other variables and commands such as the `set_fpga_defaults` command. The `set_fpga_defaults` command is used to predefine a set of values for synthesis switches specific to Design Compiler FPGA. These set of values have been shown to produce optimum circuit implementation for selected FPGA architectures. The formats of these commands are as follows:

```
source <$DCFPGA-LIB>/synosys_dc_stratix.setup
set_fpga_defaults –verbose altera_stratix
```

Starting with version W-2004.12, if you want to perform formal verification with Formality after synthesis, you will need to enable the Verification Friendly Mode (VFM) compatibility in Design Compiler FPGA by using the command `set_verification_friendly_mode`. This command allows Design Compiler FPGA to report special optimizations that were performed during synthesis to Formality in a Synopsys Verification File (SVF) specified with the `set_svf` command.

For example,

```
source <$DCFPGA-LIB>/synosys_dc_virtex2.setup
set_fpga_defaults –verbose xilinx_virtex2

set_svf <design.svf>
set_verification_friendly_mode
```

1.3 Reading Designs into Design Compiler FPGA

Analyze and elaborate your VHDL or Verilog design source files with Presto compiler enabled for VHDL, which allows high-level behavior structure such as extracting memory. For example,

```
set hdlin_enable_presto_for_vhdl true
analyze -f VHDL  -library WORK entity.vhd
analyze -f verilog  -library WORK design.v

elaborate <top_level design>
```

If your design contains the Xilinx coregen component, you can also read in the EDIF netlist with the `read_edif` command so that timing analysis in DC FPGA is able to time through the coregen component. However, you might need to use the `bus_naming_style`, `bus_inference_style`, or `bus_extraction_style` variables to specify the bus notation style in your EDIF netlist if it is different from the default bus notation style in Design Compiler FPGA, which uses brackets ([ ]).

For example,

```
set bus_naming_style "%s(%d)"
set bus_inference_style "%s(%d)"
set bus_extraction_style “%s(%d:%d)”
read_edif <coregen.edn>
```

1.4 Defining Constraints for Synthesis

For timing constraints, you need to specify your required clock frequency with the `create_clock` command and set the input and output delay with the `set_input_delay` and `set_output_delay` commands respectively. You must also identify any timing exception paths such as multicycle and false-paths so that logic optimization can be performed on true critical paths in the design. You can automatically disable timing analysis for cross-clock domain paths by setting the `fpga_cross_clock_paths_false` variable to true. At the same time, any ASIC type gated-clocks in the design should also be transformed to the FPGA clock enabled implementation with the `fpga_clock_gate_removal` variable set on the appropriate base clock domains.

As part of design constraining, the target FPGA device is specified with the `set_fpga_target_device` command, and I/O buffer inference is set with the `set_port_is_pad` command, this information is supplied in your vendor supplied setup file. For example:

```
set_port_is_pad "*"
set_fpga_target_device 2vp100ff1696-5
```

1.5 Compiling the Design to FPGA Specific Primitives

For Design Compiler FPGA to perform optimum logic optimization, it is recommended that you use the top-down compile approach with ungrouping of any blocks containing less than 500 instances. This synthesis approach has been shown to produce efficient circuit implementation in the selected FPGA architecture and primitives. For example,

```
ungroup -small 500
compile
```

1.6 Reporting and Resolving Design Problems

Various reporting commands can be used to help analyze the synthesis results. The `report_fpga` command can be use to report the overall resource utilization in the target device while the `report_timing` command can be used to view the designs critical paths. However, the post place and route timing values that are reported should be the determining factor that the design meets the intended timing goal.

1.7 Exporting the Netlist for Place and Route Tools

The primary interface between Design Compiler FPGA and the FPGA vendor place and route tools is by using the `write` and `write_par_constraint` commands. You can use the `write` command to generate a post-synthesis EDIF netlist for place and route and a gate-level Verilog or VHDL netlist for formal verification in Formality and functional simulation. The timing constraints used for synthesis is forward annotated to the place and route tool using the `write_par_constraint` command, which generates a Xilinx NCF or an Altera Tcl constraint file. However, the `–formality` option needs to be used with the `write_par_constraint` command in an Altera flow when formal verification with Formality is to be performed later.

For example,

```
write  -format edif -hier -o <design>.edf
write  -format verilog -hier -o <design>.v
write  -format vhdl -hier  -o <design>.vhd
write_par_constraint design.[ncf|tcl]
```

1.7.1 Xilinx Place and Route Flow

The following script can be used to run place and route on the post-synthesis EDIF netlist and NCF constraint file through Xilinx ISE tool.

```
ngdbuild -p 2vp100ff1696-5 -uc design.ucf design.edf
design.ngd map -ol high -o map.ncd design.ngd design.pcf
par -ol high -w map.ncd design.ncd design.pcf
trce -e 5  -o design.twr design.ncd design.pcf
trce -u -v -o design.twr design.ncd design.pcf

# For Formality Formal Verification Flow
netgen -ecn formality -w design.ncd design_fm.v

# For Primetime Static Timing Analysis Flow
netgen   -sta -pcf design.pcf design.ncd design_pt.v
pcf2sdc  -i design.pcf -o design_pt.sdc -pcs design_pt.pcs
```

1.7.2 Altera Place and Route Flow

The following script can be used to run place and route on the post-synthesis EDIF netlist and Tcl constraint file through Altera Quartus II version 4.0 or later. To run place and route directly using the Tcl file generated in Design Compiler FPGA, use the following command

```
quartus_sh -t design.tcl
```

If you intend to run formal verification and static timing analysis after place and route, then you need to modify the design.tcl file generated by Design Compiler FPGA for Quartus II. Quartus II creates the necessary files for Formality and PrimeTime. The modifications needed are as follows:

```
# For Formality Formal Verification Flow
set_instance_assignment -name PRESERVE_HIERARCHICAL_BOUNDARY
FIRM -to \|
set_global_assignment -name EDIF_FILE    design.edf
set_global_assignment -name VERILOG_FILE design.v

# For Primetime Static Timing Analysis Flow
set_global_assignment -name EDA_TIMING_ANALYSIS_TOOL
"PtimeTime (Verilog HDL output from Quartus II)"
```

1.8 Overview of Guidelines

The following gives an overview of the guidelines for processing an FPGA design in Design Compiler FPGA:

- Use the Synopsys setup file provided with Design Compiler FPGA located in the library directory. Otherwise, make sure the target, link, and synthesis library variables point to the correct version and that the `set_fpga_defaults` is set correctly.
- Specify only applicable timing constraints such as clock constraints, I/O delay, and timing exceptions.
- Remove any wire load model specification or any other ASIC type library constraints.
- Specify the target FPGA device with the `set_fpga_target_device` variable.
- Enable IOB insertion with the `set_port_is_pad`.
- Compile with an `ungroup -small 500` command only.
- For good results, it is recommended that you use the `compile` command alone (the default) instead of applying various command options that are typically used for ASIC synthesis.

- Do not flatten the entire top-level design but rather selectively dissolve critical path blocks as needed.
- Perform a top-down compile instead of a bottom-up compile flow.
- Forward annotate your timing constraints to the place and route tool using the `write_par_constraint` command.

Methods for Improving Timing Quality of Result (QoR)

To achieve a high performance synthesis goal, it is best to perform a top-down compile and to synthesize with the same timing constraints that are used for place and route. Sometimes the bottom-up synthesis might be needed to perform a targeted compile on critical blocks. However, it is considered bad practice to do a top-down compile with all hierarchy flattened, this method results in a larger and slower netlist implementation. The following guidelines are recommended to help you meet your high performance synthesis goals:

- If your post place and route critical path reported is different from the DC FPGA post-synthesis critical path, use the `set_max_delay` or other timing constraints to focus the Design Compiler FPGA optimization efforts on the actual critical path of the design.
- If post place and route critical path crosses multiple hierarchy boundaries, use the `ungroup` command to dissolve the hierarchy boundaries to isolate the critical path in a single level of hierarchy. This method allows Design Compiler FPGA to perform better optimization in the absent of a hierarchy boundary restriction.
- Perform register retiming.
- If the critical path starts or ends at an embedded FPGA macro, such as blockRAM, MULT18S, SRL16, and so forth, explore a different implementation that might result in better timing at the expense of higher area usage. For example, reimplement the blockRAM with distributed RAM, or the SRL16 with its registers equivalents.
- If the critical source or destination is from a register to an IOB, try performing IOB register insertion during synthesis or during place and route.
- If the critical path contains a high fanout net, try setting a lower fanout load limit in synthesis.
- If the critical path reported in place and route start or end at a black-box block, such as a coregen component, try and include the gate-level EDIF netlist of the black-box block in synthesis so that Design Compiler FPGA is able to take into account the timing through this block during synthesis.

- If the critical path involves finite state machine (FSM), try different state encoding, especially the one-hot encoding scheme.
- If the critical path contains any clock skew due to gated-clocks present in the design, then you should perform gated-clock transformation to remove the clock skew.
- Specify proper clock relationships if your design contains any multiple (unrelated) clock domains.

2 Methods for Improving Area Quality of Result (QoR)

If your synthesis primary objective is to obtain the lowest area usage possible, then you might try the following to get the smallest gate-level netlist implementation for the design:

- Synthesize the design without specifying any timing constraints so that Design Compiler FPGA optimizes for area goal.
- Enable Presto Compiler so that embedded FPGA device macros such as blockRAM and DSP block are inferred and fully utilized.
- Make sure that constant propagation and register merging variables are enabled.
- Ungroup multiple referenced blocks or blocks containing duplicated logic so that those duplicated logic elements are removed during optimization.
- Explore different finite state machine encoding schemes depending on the register-to-look-up table ratio in the design.
- Set a high cell fanout load constraint.

Example 1   Synthesis Script for an Altera Device

```
set outputdir ./test_cg_distributed_ram.syn_dir
file delete -force $outputdir
file mkdir $outputdir
file mkdir $outputdir/mywork
define_design_lib WORK -path $outputdir/mywork

set_fpga_defaults altera_stratix
set top test_cg_distributed_ram
set hdlin_enable_presto_for_vhdl true

analyze -format VHDL -library WORK ../source/mod_test.vhd
analyze -format verilog -library WORK
../source/test_cg_distributed_ram.v

elaborate $top
write -hier -f db -o $outputdir/$top.elab.db

current_design $top
link

set_fpga_target_device "EP1S40F780C6"

#set_port_is_pad "*"

create_clock [find port "clk*"] -period 10
set_input_delay   5 -clock clk [remove_from_collection
[all_inputs] [get_ports “clk”]]
set_output_delay 5 -clock clk [all_outputs]

set_fpga_clock_gate_removal [get_clocks]

compile

write -hier -f edif    -o $outputdir/$top.edf
write -hier -f db      -o $outputdir/$top.db
write -hier -f verilog -o $outputdir/$top.v
write -hier -f vhdl    -o $outputdir/$top.vhd
write_par_constraint      $outputdir/$top.tcl

report_fpga
report_timing -from clk -net -in

ungroup -all -flatten
report_reference
```

Example 2   Synthesis Script for Xilinx Device

```
set outputdir ./test_coregen_edif_property.syn_dir
file delete -force $outputdir
file mkdir $outputdir
file mkdir $outputdir/mywork
define_design_lib WORK -path $outputdir/mywork

set_fpga_defaults xilinx_virtexe

set top test_coregen_edif_property

set hdlin_enable_presto_for_vhdl true

analyze -format verilog -library WORK ../source/mod_test.v
analyze -format VHDL    -library WORK
../source/dcfpga_edn_pkg.vhd
analyze -format VHDL    -library WORK
../source/test_coregen_edif_property.vhd

set bus_naming_style "%s<%d>"
set bus_inference_style "%s<%d>"

read_file -format edif -library WORK
../source/trig_lookup.edn

set bus_naming_style "%s\[%d\]"

elaborate $top
write -hier -f db -o $outputdir/$top.elab.db

current_design $top
link

set_fpga_target_device "V2000EFG680-6"
set_port_is_pad "*"

create_clock [find port "clk*"] -period 40 -name clk

set_input_delay  5 -clock clk [remove_from_collection
[all_inputs] [get_ports “clk”]]
set_output_delay 5 -clock clk [all_outputs]

set_fpga_clock_gate_removal [get_clocks]

ungroup -small 500
compile

write -hier -f edif    -o $outputdir/$top.edf
write -hier -f db      -o $outputdir/$top.db
write -hier -f verilog -o $outputdir/$top.v
write -hier -f vhdl    -o $outputdir/$top.vhd
```

```
write_par_constraint      $outputdir/$top.ncf

report_fpga
report_timing -from clk -net -in

ungroup -all -flatten
report_reference
```