2. Command-Line Scripting

QII52002-10.1.0

FPGA design software that easily integrates into your design flow saves time and improves productivity. The Altera® Quartus® II software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

The benefits provided by command-line executables include:

- Command-line control over each step of the design flow
- Easy integration with scripted design flows including makefiles
- Reduced memory requirements
- Improved performance

The command-line executables are also completely compatible with the Quartus II GUI, allowing you to use the exact combination of tools that you prefer.

This chapter describes how to take advantage of Quartus II command-line executables, and provides several examples of scripts that automate different segments of the FPGA design flow. This chapter includes the following topics:

- "Benefits of Command-Line Executables"
- "Introductory Example" on page 2–2
- "Compilation with quartus_sh --flow" on page 2–7
- "The MegaWizard Plug-In Manager" on page 2–11
- "Command-Line Scripting Examples" on page 2–17

Benefits of Command-Line Executables

The Quartus II command-line executables provide control over each step of the design flow. Each executable includes options to control commonly used software settings. Each executable also provides detailed, built-in help describing its function, available options, and settings.

Command-line executables allow for easy integration with scripted design flows. You can easily create scripts in any language with a series of commands. These scripts can be batch-processed, allowing for integration with distributed computing in server farms. You can also integrate the Quartus II command-line executables in makefile-based design flows. These features enhance the ease of integration between the Quartus II software and other EDA synthesis, simulation, and verification software.

Command-line executables add flexibility without sacrificing the ease-of-use of the Quartus II GUI. You can use the Quartus II GUI and command-line executables at different stages in the design flow. For example, you might use the Quartus II GUI to edit the floorplan for the design, use the command-line executables to perform place-and-route, and return to the Quartus II GUI to perform debugging with the Chip Editor.

Command-line executables reduce the amount of memory required during each step in the design flow. Because each executable targets only one step in the design flow, the executables themselves are relatively compact, both in file size and the amount of memory used when running. This memory usage reduction improves performance, and is particularly beneficial in design environments where computer networks or workstations are heavily used with reduced memory.

For a complete list of the Quartus II command-line executables, refer to *Using the Quartus II Executables in Shell Scripts* in Quartus II Help.

Introductory Example

The following introduction to command-line executables demonstrates how to create a project, fit the design, and generate programming files.

The tutorial design included with the Quartus II software is used to demonstrate this functionality. If installed, the tutorial design is found in the <*Quartus II directory*>**/qdesigns/fir_filter** directory.

Before making changes, copy the tutorial directory and type the four commands shown in Example 2–1 at a command prompt in the new project directory.

The <*Quartus II directory*>**/quartus/bin** directory must be in your PATH environment variable.

Example 2–1. Introductory Example

```
quartus_map filtref --source=filtref.bdf --family=CYCLONE ↵
quartus_fit filtref --part=EP1C12F256C6 --fmax=80MHz ↵
quartus_asm filtref ↵
quartus_sta filtref ↵
```

The `quartus_map filtref --source=filtref.bdf --family=CYCLONE` command creates a new Quartus II project called **filtref** with **filtref.bdf** as the top-level file. It targets the Cyclone® device family and performs logic synthesis and technology mapping on the design files.

The `quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz` command performs fitting on the **filtref** project. This command specifies an EP1C12Q240C6 device and the Fitter attempts to meet a global f_{MAX} requirement of 80 MHz and a global t_{SU} requirement of 8 ns.

The `quartus_asm filtref` command creates programming files for the **filtref** project.

The `quartus_sta filtref` command performs basic timing analysis on the **filtref** project using the Quartus II TimeQuest Timing Analyzer, reporting worst-case setup slack, worst-case hold slack, and other measurements.

The TimeQuest Timing Analyzer employs Synopsys Design Constraints to fully analyze the timing of your design. For more information about using all of the features of the **quartus_sta** executable, refer to the *TimeQuest Timing Analyzer Quick Start Tutorial*.

You can put the four commands from Example 2–1 into a batch file or script file, and run them. For example, you can create a simple UNIX shell script called **compile.sh**, which includes the code shown in Example 2–2.

Example 2–2. UNIX Shell Script: compile.sh

```
#!/bin/sh
PROJECT=filtref
TOP_LEVEL_FILE=filtref.bdf
FAMILY=Cyclone
PART=EP1C12F256C6
FMAX=80MHz
quartus_map $PROJECT --source=$TOP_LEVEL_FILE --family=$FAMILY
quartus_fit $PROJECT --part=$PART --fmax=$FMAX
quartus_asm $PROJECT
quartus_sta $PROJECT
```

Edit the script as necessary and compile your project.

Command-Line Scripting Help

Help for command-line executables is available through different methods. You can access help built in to the executables with command-line options. You can use the Quartus II Command-Line and Tcl API Help browser for an easy graphical view of the help information.

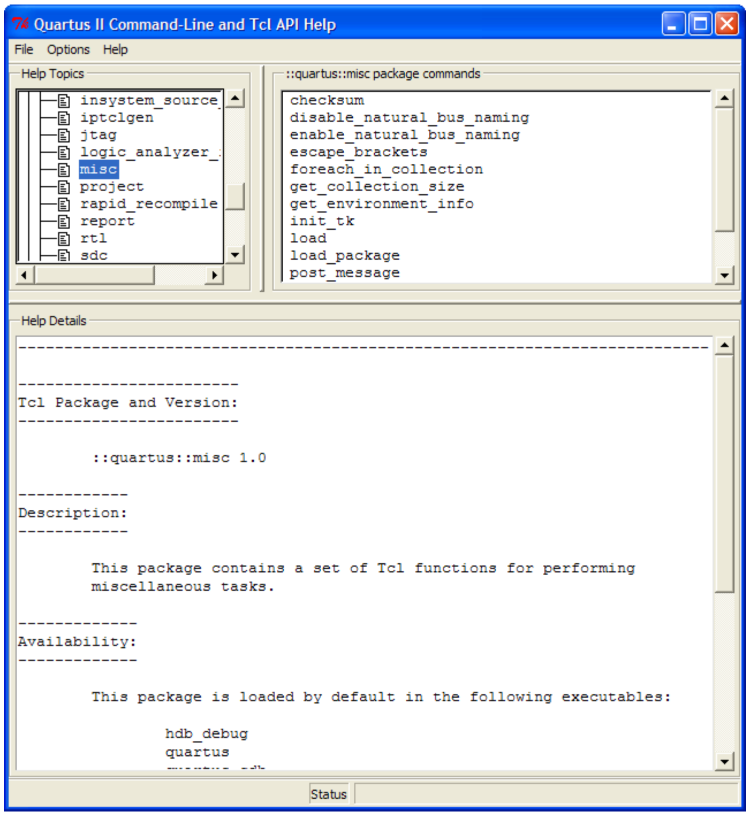
To use the Quartus II Command-Line and Tcl API Help browser, type the following command:

```
quartus_sh --qhelp ↵
```

This command starts the Quartus II Command-Line and Tcl API Help browser, a viewer for information about the Quartus II Command-Line executables and Tcl API (Figure 2–1).

Use the `-h` option with any of the Quartus II Command-Line executables to get a description and list of supported options. Use the `--help=`<*option name*> option for detailed information about each option.

Figure 2–1. Quartus II Command-Line and Tcl API Help Browser


Quartus II Command-Line and Tcl API Help
File
Options
Help
Help Topics
::quartus::misc package commands
Help Details
Status

```
insystem_source
iptclgen
jtag
logic_analyzer_
misc
project
rapid_recompile
report
rtl
sdc
```

```
checksum
disable_natural_bus_naming
enable_natural_bus_naming
escape_brackets
foreach_in_collection
get_collection_size
get_environment_info
init_tk
load
load_package
post_message
```

```
---------------------------------------------------------------------------


------------------------
Tcl Package and Version:
------------------------

        ::quartus::misc 1.0

------------
Description:
------------

        This package contains a set of Tcl functions for performing
        miscellaneous tasks.

-------------
Availability:
-------------

        This package is loaded by default in the following executables:

                hdb_debug
                quartus
```

Command-Line Option Details

Command-line options are provided for many common global project settings and for performing common tasks. You can use either of two methods to make assignments to an individual entity. If the project exists, open the project in the Quartus II GUI, change the assignment, and close the project. The changed assignment is updated in the Quartus II Settings File. Any command-line executables that are run after this update use the updated assignment. For more information refer to “Option Precedence” on page 2–5. You can also make assignments using the Quartus II Tcl scripting API. If you want to completely script the creation of a Quartus II project, choose this method.

For more information about the Quartus II Tcl scripting API, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. For more information about Quartus II project settings and assignments, refer to the *QSF Reference Manual*.

Option Precedence

If you use command-line executables, you must be aware of the precedence of various project assignments and how to control the precedence. Assignments for a particular project exist in the Quartus II Settings File for the project. Assignments for a project can also be made with command-line options. Project assignments are reflected in compiler database files that hold intermediate compilation results and reflect assignments made in the previous project compilation.

All command-line options override any conflicting assignments found in the Quartus II Settings File or the compiler database files. There are two command-line options to specify whether the Quartus II Settings File or compiler database files take precedence for any assignments not specified as command-line options.

☞ Any assignment not specified as a command-line option or found in the Quartus II Settings File or compiler database file is set to its default value.

The file precedence command-line options are `--read_settings_files` and `--write_settings_files`.

By default, the `--read_settings_files` and `--write_settings_files` options are turned on. Turning on the `--read_settings_files` option causes a command-line executable to read assignments from the Quartus II Settings File instead of from the compiler database files. Turning on the `--write_settings_files` option causes a command-line executable to update the Quartus II Settings File to reflect any specified options, as happens when you close a project in the Quartus II GUI.

If you use command-line executables, be aware of the precedence of various project assignments and how to control the precedence. Assignments for a particular project can exist in three places:

- The Quartus II Settings File for the project
- The result of the last compilation, in the **/db** directory, which reflects the assignments that existed when the project was compiled
- Command-line options

Table 2–1 lists the precedence for reading assignments depending on the value of the `--read_settings_files` option.

Table 2–1. Precedence for Reading Assignments

Option Specified	Precedence for Reading Assignments
`--read_settings_files = on` (Default)	1. Command-line options 2. Quartus II Settings File 3. Project database (db directory, if it exists) 4. Quartus II software defaults
`--read_settings_files = off`	1. Command-line options 2. Project database (db directory, if it exists) 3. Quartus II software defaults

Table 2–2 lists the locations to which assignments are written, depending on the value of the `--write_settings_files` command-line option.

Table 2–2. Location for Writing Assignments

Option Specified	Location for Writing Assignments
`--write_settings_files = on` (Default)	Quartus II Settings File and compiler database
`--write_settings_files = off`	Compiler database

Example 2–3 assumes that a project named **fir_filter** exists, and that the analysis and synthesis step has been performed (using the `quartus_map` executable).

Example 2–3. Write Settings Files

```
quartus_fit fir_filter --fmax=80MHz ↵
quartus_sta fir_filter ↵
mv fir_filter_sta.rpt fir_filter_1_sta.rpt ↵
quartus_fit fir_filter --fmax=100MHz --write_settings_files=off ↵
quartus_sta fir_filter ↵
mv fir_filter_sta.rpt fir_filter_2_sta.rpt ↵
```

The first command, `quartus_fit fir_filter --fmax=80MHz`, runs the `quartus_fit` executable and specifies a global f_{MAX} requirement of 80 MHz.

The second command, `quartus_sta fir_filter`, runs timing analysis for the results of the previous fit.

The third command uses the UNIX `mv` command to copy the report file output from **quartus_sta** to a file with a new name, so that the results are not overwritten by subsequent timing analysis.

The fourth command reruns the Fitter with a global f_{MAX} requirement of 100 MHz. By specifying the `--write_settings_files=off` option, the command-line executable does not update the Quartus II Settings File to reflect the changed f_{MAX} requirement. The compiler database files reflect the changed f_{MAX} requirement. If the `--write_settings_files=off` option is not specified, the command-line executable updates the Quartus II Settings File to reflect the 100-MHz global f_{MAX} requirement.

The fifth command reruns timing analysis, and the sixth command renames the report file, so that it is not overwritten by subsequent timing anlysis.

Use the options `--read_settings_files=off` and `--write_settings_files=off` (where appropriate) to optimize the way that the Quartus II software reads and updates settings files. Example 2–4 shows how to avoid unnecessary reading and writing.

Example 2–4. Avoiding Unnecessary Reading and Writing

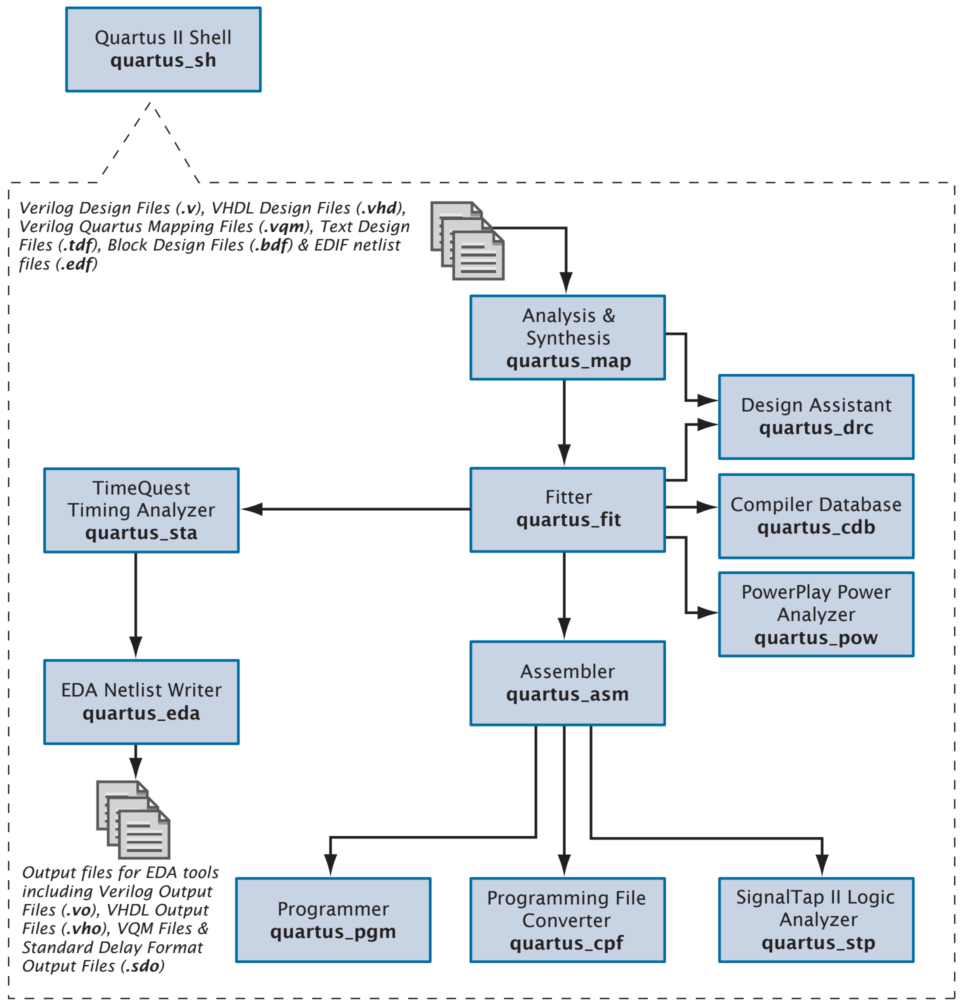
```
quartus_map filtref --source=filtref --part=EP1C12F256C6 ↵
quartus_fit filtref --fmax=100MHz --read_settings_files=off ↵
quartus_asm filtref --read_settings_files=off --write_settings_files=off ↵
```

In Example 2–4, the `quartus_asm` executable does not read or write settings files because they do not change any settings in the project.

Compilation with quartus_sh --flow

Figure 2–2 shows a typical Quartus II FPGA design flow.

Figure 2–2. Typical Design Flow


Quartus II Shell
quartus_sh
Verilog Design Files (.v), VHDL Design Files (.vhd), Verilog Quartus Mapping Files (.vqm), Text Design Files (.tdf), Block Design Files (.bdf) & EDIF netlist files (.edf)
Analysis & Synthesis
quartus_map
Design Assistant
quartus_drc
TimeQuest Timing Analyzer
quartus_sta
Fitter
quartus_fit
Compiler Database
quartus_cdb
PowerPlay Power Analyzer
quartus_pow
EDA Netlist Writer
quartus_eda
Assembler
quartus_asm
Output files for EDA tools including Verilog Output Files (.vo), VHDL Output Files (.vho), VQM Files & Standard Delay Format Output Files (.sdo)
Programmer
quartus_pgm
Programming File Converter
quartus_cpf
SignalTap II Logic Analyzer
quartus_stp

Use the `quartus_sh` executable with the `--flow` option to perform a complete compilation flow with a single command. (For information about specialized flows, type `quartus_sh --help=flow` ↵ at a command prompt.) The `--flow` option supports the smart recompile feature and efficiently sets command-line arguments for each executable in the flow.

The following example runs compilation, timing analysis, and programming file generation with a single command:

```
quartus_sh --flow compile filtref ↵
```

Text-Based Report Files

Each command-line executable creates a text report file when it is run. These files report success or failure, and contain information about the processing performed by the executable.

Report file names contain the revision name and the short-form name of the executable that generated the report file: <*revision*>.<*executable*>**.rpt**. For example, using the `quartus_fit` executable to place and route a project with the revision name **design_top** generates a report file named **design_top.fit.rpt**. Similarly, using the `quartus_sta` executable to perform timing analysis on a project with the revision name **fir_filter** generates a report file named **fir_filter.sta.rpt**.

As an alternative to parsing text-based report files, you can use the **::quartus::report** Tcl package. For more information about this package, refer to *::quartus::report* in Quartus II Help.

You can use command-line executables in scripts that control a design flow that uses other software in addition to the Quartus II software. For example, if your design flow uses other synthesis or simulation software, and you can run the other software at a command prompt, you can include it in a single script. The Quartus II command-line executables include options for common global project settings and operations, but you must use a Tcl script or the Quartus II GUI to set up a new project and apply individual constraints, such as pin location assignments and timing requirements. Command-line executables are very useful for working with existing projects, for making common global settings, and for performing common operations. For more flexibility in a flow, use a Tcl script, which makes it easier to pass data between different stages of the design flow and have more control during the flow.

For more information about Tcl scripts, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*, or *About Quartus II Tcl Scripting* in Quartus II Help.

For example, your script could run other synthesis software, then place-and-route the design in the Quartus II software, then generate output netlists for other simulation software. Example 2–5 shows how to do this with a UNIX shell script for a design that targets a Cyclone II device.

Example 2–5. Script for End-to-End Flow (Part 1 of 2)

```
#!/bin/sh
# Run synthesis first.
# This example assumes you use Synplify software
synplify -batch synthesize.tcl

# If your Quartus II project exists already, you can just
# recompile the design.
# You can also use the script described in a later example to
# create a new project from scratch
quartus_sh --flow compile myproject

# Use the quartus_sta executable to do fast and slow-model
# timing analysis
quartus_sta myproject --model=slow
quartus_sta myproject --model=fast

# Use the quartus_eda executable to write out a gate-level
# Verilog simulation netlist for ModelSim
quartus_eda my_project --simulation --tool=modelsim --format=verilog
```

Example 2–5. Script for End-to-End Flow (Part 2 of 2)

```
# Perform the simulation with the ModelSim software
vlib cycloneii_ver
vlog -work cycloneii_ver /opt/quartusii/eda/sim_lib/cycloneii_atoms.v
vlib work
vlog -work work my_project.vo
vsim -L cycloneii_ver -t 1ps work.my_project
```

Makefile Implementation

You can use the Quartus II command-line executables in conjunction with the `make` utility to automatically update files when other files they depend on change. The file dependencies and commands used to update files are specified in a text file called a makefile.

To facilitate easier development of efficient makefiles, the following “smart action” scripting command is provided with the Quartus II software:

```
quartus_sh --determine_smart_action ↵
```

Because assignments for a Quartus II project are stored in the Quartus II Settings File (**.qsf**), including it in every rule results in unnecessary processing steps. For example, updating a setting related to programming file generation (which requires re-running only `quartus_asm`) modifies the Quartus II Settings File, requiring a complete recompilation if the Quartus II Settings File is included in every rule.

The smart action command determines the earliest command-line executable in the compilation flow that must be run based on the current Quartus II Settings File, and generates a change file corresponding to that executable. For a given command-line executable named `quartus_`<*executable*>, the change file is named with the format <*executable*>**.chg**. For example, if `quartus_map` must be re-run, the smart action command creates or updates a file named **map.chg**. Thus, rather than including the Quartus II Settings File in each makefile rule, include only the appropriate change file.

Example 2–6 uses change files and the smart action command. You can copy and modify it for your own use. A copy of this example is included in the help for the makefile option, which is available by typing:

```
quartus_sh --help=makefiles ↵
```

Example 2–6. Sample Makefile (Part 1 of 2)

```
###################################################################
# Project Configuration:
#
# Specify the name of the design (project), the Quartus II Settings
# File (.qsf), and the list of source files used.
###################################################################

PROJECT = chiptrip
SOURCE_FILES = auto_max.v chiptrip.v speed_ch.v tick_cnt.v time_cnt.v
ASSIGNMENT_FILES = chiptrip.qpf chiptrip.qsf

###################################################################
# Main Targets
#
# all: build everything
# clean: remove output files and database
###################################################################
all: smart.log $(PROJECT).asm.rpt $(PROJECT).sta.rpt

clean:
   rm -rf *.rpt *.chg smart.log *.htm *.eqn *.pin *.sof *.pof db

map: smart.log $(PROJECT).map.rpt
fit: smart.log $(PROJECT).fit.rpt
asm: smart.log $(PROJECT).asm.rpt
sta: smart.log $(PROJECT).sta.rpt
smart: smart.log
###################################################################
# Executable Configuration
###################################################################

MAP_ARGS = --family=Stratix
FIT_ARGS = --part=EP1S20F484C6
ASM_ARGS =
STA_ARGS =

###################################################################
# Target implementations
###################################################################

STAMP = echo done >

$(PROJECT).map.rpt: map.chg $(SOURCE_FILES)
   quartus_map $(MAP_ARGS) $(PROJECT)
   $(STAMP) fit.chg

$(PROJECT).fit.rpt: fit.chg $(PROJECT).map.rpt
   quartus_fit $(FIT_ARGS) $(PROJECT)
   $(STAMP) asm.chg
   $(STAMP) sta.chg

$(PROJECT).asm.rpt: asm.chg $(PROJECT).fit.rpt
   quartus_asm $(ASM_ARGS) $(PROJECT)

$(PROJECT).sta.rpt: sta.chg $(PROJECT).fit.rpt
   quartus_sta $(STA_ARGS) $(PROJECT)
smart.log: $(ASSIGNMENT_FILES)
   quartus_sh --determine_smart_action $(PROJECT) > smart.log
```

Example 2–6. Sample Makefile (Part 2 of 2)

```
###################################################################
# Project initialization
###################################################################

$(ASSIGNMENT_FILES):
   quartus_sh --prepare $(PROJECT)

map.chg:
   $(STAMP) map.chg
fit.chg:
   $(STAMP) fit.chg
sta.chg:
   $(STAMP) sta.chg
asm.chg:
   $(STAMP) asm.chg
```

A Tcl script is provided with the Quartus II software to create or modify files that are specified as dependencies in the make rules, assisting you in makefile development. Complete information about this Tcl script and how to integrate it with makefiles is available by running the following command:

```
quartus_sh --help=determine_smart_action ↵
```

The MegaWizard Plug-In Manager

The MegaWizard™ Plug-In Manager provides a GUI-based flow to configure megafunction and IP variation files. However, you can use command-line options for the **qmegawiz** executable to modify, update, or create variation files without using the GUI. This capability is useful in a fully scripted design flow, or in cases where you want to generate variation files without using the wizard GUI flow.

The MegaWizard Plug-In Manager has three functions:

- Providing an interface for you to select the output file or files
- Running a specific MegaWizard Plug-In
- Creating output files (such as variation files, symbol files, and simulation netlist files)

Each MegaWizard Plug-In provides a user interface for configuring the variation, and performs validation and error checking of your selected ports and parameters. When you create or update a variation with the GUI, the parameters and values are entered through the GUI provided by the Plug-In. When you create a Plug-In variation with the command line, you provide the parameters and values as command-line options.

Example 2–7 shows how to create a new variation file at a system command prompt.

Example 2–7. MegaWizard Plug-In Manager Command-Line Executable

```
qmegawiz [options] [module=<module name>|wizard=<wizard name>] [<param>=<value> ...
    <port>=<used|unused> ...] [OPTIONAL_FILES=<optional files>] <variation file name>
```

When you use `qmegawiz` to update an existing variation file, the module or wizard name is not required.

If a megafunction changes between software versions, the variation files must be regenerated. To do this, use `qmegawiz -silent <variation file name>`. For example, if your design contains a variation file called `myrom.v`, type the following command:

```
qmegawiz -silent myrom.v ↵
```

For more information on updating megafunction variation files as part of a scripted flow, refer to "Regenerating Megafunctions After Updating the Quartus II Software" on page 2–23.

Table 2–3 describes the supported options.

Table 2–3. qmegawiz Options

Option	Description
`-silent`	Run the MegaWizard Plug-In Manager in command-line mode, without displaying the GUI.
`-f:<`**param file**`>`	A file that contains all options for the `qmegawiz` command. Refer to "Parameter File" on page 2–16.
`-p:<`**working directory**`>`	Sets the default working directory. Refer to"Working Directory" on page 2–17.

For information about specifying the module name or wizard name, refer to "Module and Wizard Names" on page 2–13.

For information about specifying ports and parameters, refer to "Ports and Parameters" on page 2–14.

For information about generating optional files, refer to "Optional Files" on page 2–15.

For information about specifying the variation file name, refer to "Variation File Name" on page 2–17.

Command-Line Support

Only the MegaWizard Plug-Ins listed in Table 2–4 support creation and update in command-line mode. For Plug-Ins not listed in the table, you must use the MegaWizard Plug-In Manager GUI for creation and updates.

Table 2–4. MegaWizard Plug-Ins with Command Line Support (Part 1 of 2)

MegaWizard Plug-In	Wizard Name	Module Name
alt2gxb	ALT2GXB	alt2gxb
alt4gxb	ALTGX	alt4gxb
altasmi_parallel	ALTASMI_PARALLEL	altasmi_parallel
altclkctrl	ALTCLKCTRL	altclkctrl
altddio_bidir	ALTDDIO_BIDIR	altddio_bidir
altddio_in	ALTDDIO_IN	altddio_in
altddio_out	ALTDDIO_OUT	altddio_out
altecc_decoder	ALTECC	altecc_decoder
altecc_encoder		altecc_encoder
altfp_abs	ALTFP_ABS	altfp_abs

Table 2–4. MegaWizard Plug-Ins with Command Line Support (Part 2 of 2)

MegaWizard Plug-In	Wizard Name	Module Name
altfp_add_sub	ALTFP_ADD_SUB	altfp_add_sub
altfp_compare	ALTFP_COMPARE	altfp_compare
altfp_convert	ALTFP_CONVERT	altfp_convert
altfp_div	ALTFP_DIV	altfp_div
altfp_exp	ALTFP_EXP	altfp_exp
altfp_inv_sqrt	ALTFP_INV_SQRT	altfp_inv_sqrt
altfp_inv	ALTFP_INV	altfp_inv
altfp_log	ALTFP_LOG	altfp_log
altfp_matrix_inv	ALTFP_MATRIX_INV	altfp_matrix_inv
altfp_matrix_mult	ALTFP_MATRIX_MULT	altfp_matrix_mult
altfp_mult	ALTFP_MULT	altfp_mult
altfp_sincos	ALTFP_SINCOS	altfp_sincos
altfp_sqrt	ALTFP_SQRT	altfp_sqrt
altiobuf_bidir	ALTIOBUF	altiobuf_bidir
altiobuf_in		altiobuf_in
altiobuf_out		altiobuf_out
altlvds_rx	ALTLVDS	altlvds_rx
altlvds_tx		altlvds_tx
altmult_accum	ALTMULT_ACCUM (MAC)	altmult_accum
altmult_complex	ALTMULT_COMPLEX	altmult_complex
altotp	ALTOTP	altotp
altpll_reconfig	ALTPLL_RECONFIG	altpll_reconfig
altpll	ALTPLL	altpll
altremote_update	ALTREMOTE_UPDATE	altremote_update
altshift_taps	ALTSHIFT_TAPS	altshift_taps
altsyncram	RAM: 2-PORT	altsyncram
	RAM: 1-PORT	
	ROM: 1-PORT	
alttemp_sense	ALTTEMP_SENSE	alttemp_sense
alt_c3gxb	ALT_C3GXB	alt_c3gxb
dcfifo	FIFO	dcfifo
scfifo		scfifo

Module and Wizard Names

You must specify the wizard or module name, shown in Table 2–4, as a command-line option when you create a variation file. Use the option `module=<module name>` to specify the module, or use the option `wizard=<wizard name>` to specify the wizard. If there are spaces in the wizard or module name, enclose the name in double quotes, for example:

```
wizard="RAM: 2-PORT"
```

When there is a one-to-one mapping between the MegaWizard Plug-In and the wizard name and the module name, you can use either the wizard option or the module option.

When there are multiple wizard names that correspond to one module name, you should use the wizard option to specify one wizard.

When there are multiple module names that correspond to one wizard name, you should use the module option to specify one module. For example, use the module option if you create a FIFO because one wizard is common to both modules. However, you should use the wizard option if you create a RAM, because one module is common to three wizards.

If you edit or update an existing variation file, the wizard or module option is not necessary, because information about the wizard or module is already in the variation file.

Ports and Parameters

Ports and parameters for each MegaWizard Plug-In are described in Quartus II Help, and in the Megafunction User Guides on the Altera website. You should use these references to determine appropriate values for each port and parameter required for a particular variation configuration. Refer to "Strategies to Determine Port and Parameter Values" for more information. You do not have to specify every port and parameter supported by a Plug-In. The MegaWizard Plug-In Manager uses default values for any port or parameter you do not specify.

Specify ports as used or unused, for example:

```
<port>=used
<port>=unused
```

You can specify port names in any order. Grouping does not matter. Separate port configuration options from each other with spaces.

Specify a value for a parameter with the equal sign, for example:

```
<parameter>=<value>
```

You can specify parameters in any order. Grouping does not matter. Separate parameter configuration options from each other with spaces. You can specify port names and parameter names in upper or lower case; case does not matter.

All MegaWizard Plug-Ins allow you to specify the target device family with the `INTENDED_DEVICE_FAMILY` parameter, as shown in the following example:

```
qmegawiz wizard=<wizard> INTENDED_DEVICE_FAMILY="Cyclone III" <file>
```

You must specify enough ports and parameters to create a legal configuration of the Plug-In. When you use the GUI flow, each MegaWizard Plug-In performs validation and error checking for the particular ports and parameters you choose. When you use command-line options to specify ports and parameters, you must ensure that the ports and parameters you use are complete and valid for your particular configuration.

For example, when you use a RAM Plug-In to configure a RAM to be 32 words deep, the Plug-In automatically configures an address port that is five bits wide. If you use the command-line flow to configure a RAM that is 32 words deep, you must use one option to specify the depth of the RAM, then calculate the width of the address port and specify that width with another option.

Invalid Configurations

If the combination of default and specified ports and parameters is not complete to create a legal configuration of the Plug-In, `qmegawiz` generates an error message that indicates what is missing and what values are supported. If the combination of default and specified ports and parameters results in an illegal configuration of the Plug-In, `qmegawiz` generates an error message that indicates what is illegal, and displays the legal values.

Strategies to Determine Port and Parameter Values

For simple Plug-In variations, it is often easy to determine appropriate port and parameter values with the information in Quartus II Help and other megafunction documentation. For example, determining that a 32-word-deep RAM requires an address port that is five bits wide is straightforward. For complex Plug-In variations, an option in the GUI might affect multiple port and parameter settings, so it can be difficult to determine a complete set of ports and parameters. In this case, you should use the GUI to generate a variation file that includes the ports and parameters for your desired configuration. Open the variation file in a text editor and use the port and parameter values in the variation file as command-line options.

Optional Files

In addition to the variation file, the MegaWizard Plug-In Manager can generate other files, such as instantiation templates, simulation netlists, and symbols for graphic design entry. Use the `OPTIONAL_FILES` parameter to control whether the MegaWizard Plug-In Manager generates optional files. Table 2–5 lists valid arguments for the `OPTIONAL_FILES` parameter.

Table 2–5. Arguments for the OPTIONAL_FILES Parameter

Argument	Description
`INST`	Controls the generation of the **<variation>_inst.v** file.
`INC`	Controls the generation of the **<variation>.inc** file.
`CMP`	Controls the generation of the **<variation>.cmp** file.
`BSF`	Controls the generation of the **<variation>.bsf** file.
`BB`	Controls the generation of the **<variation>_bb.v** file.
`SIM_NETLIST`	Controls the generation of the simulation netlist file, wherever there is wizard support.
`SYNTH_NETLIST`	Controls the generation of the synthesis netlist file, wherever there is wizard support.
`ALL`	Generates all applicable optional files.
`NONE`	Disables the generation of all optional files.

Specify a single optional file, for example:

```
OPTIONAL_FILES=<argument>
```

Specify multiple optional files separated by a vertical bar character, for example:

```
OPTIONAL_FILES=<argument 1>|...|<argument n>
```

If you prefix an argument with a dash (for example, `-BB`), it is excluded from the generated optional files. If any of the optional files exist when you run `qmegawiz` and they are excluded in the `OPTIONAL_FILES` parameter (with the `NONE` argument, or prefixed with a dash), they are deleted.

You can combine the `ALL` argument with other excluded arguments to generate "all files except <*excluded files*>." You can combine the `NONE` argument with other included arguments to generate "no files except <*files*>."

When you combine multiple arguments, they are processed from left to right, and arguments evaluated later have precedence over arguments evaluated earlier. Therefore, the `ALL` or `NONE` argument should be the first in a combination of multiple arguments. When `ALL` is the first argument, all optional files are generated before exclusions are processed (deleted). When `NONE` is the first argument, none of the optional files are generated (in other words, any that exist are deleted), then any files you subsequently specify are generated.

Table 2–6 shows examples for the `OPTIONAL_FILES` parameter and describes the result of each example.

Table 2–6. Examples of Different Optional File Arguments

Example Values for OPTIONAL_FILES	Description
`BB`	The optional file **<variation>_bb.v** is generated, and no optional files are deleted
`BB|INST`	The optional file **<variation>_bb.v** is generated, then the optional file **<variation>_inst.v** is generated, and no optional files are deleted.
`NONE`	No optional files are generated, and any existing optional files are deleted.
`NONE|INC|BSF`	Any existing optional files are deleted, then the optional file **<variation>.inc** is generated, then the optional file **<variation>.bsf** is generated.
`ALL|-INST`	All optional files are generated, then **<variation>_inst.v** is deleted if it exists.
`-BB`	The optional file **<variation>_bb.v** is deleted if it exists.
`-BB|INST`	The optional file **<variation>_bb.v** is deleted if it exists, then the optional file **<variation>_inst.v** is generated.

The `qmegawiz` command accepts the `ALL` argument combined with other included file arguments, for example, `ALL|BB`, but that combination is equivalent to `ALL` because first all optional files are generated, then the file <*variation*>**_bb.v** is generated (again). Additionally, the software accepts the `NONE` argument combined with other excluded file arguments, for example, `NONE|-BB`, but that combination is equivalent to `NONE` because no optional files are generated (any that exist are deleted), then the file <*variation*>**_bb.v** is deleted if it exists.

Parameter File

You can put all parameter values and port values in a file, and pass the file name as an argument to `qmegawiz` with the `-f:`<*parameter file*> option. For example, the following command specifies a parameter file named **rom_params.txt**:

```
qmegawiz -silent module=altsyncram -f:rom_params.txt myrom.v ↵
```

The **rom_params.txt** parameter file can include options similar to the following:

```
RAM_BLOCK_TYPE=M4K DEVICE_FAMILY=Stratix WIDTH_A=5 WIDTHAD_A=5
    NUMWORDS_A=32 INIT_FILE=rom.hex OPERATION_MODE=ROM
```

☞ If you use the `-f` option and the `-p` option together, the MegaWizard Plug-In Manager sources the parameter file in a directory specified with the `-p` option, or in a directory relative to that directory. For example, if you specify **C:\project\work** with the `-p` option and **work\params.txt** with the `-f` option, the MegaWizard Plug-In Manager attempts to source the file **params.txt** in **C:\project\work\work**.

Working Directory

You can change the working directory that `qmegawiz` uses when it generates files. By default, the working directory is the current directory when you execute the `qmegawiz` command. Use the `-p` option to specify a different working directory, for example:

```
-p:<working directory>
```

You can specify the working directory with an absolute or relative path. Specify an alternative working directory any time you do not want files generated in the current directory. The alternative working directory can be useful if you generate multiple variations in a batch script, and keep generated files for the different Plug-In variations in separate directories.

Variation File Name

The language used for a variation file depends on the file extension of the variation file name. The MegaWizard Plug-In Manager creates HDL output files in a language based on the file name extension. Therefore, you must always specify a complete file name, including file extension, as the last argument to the `qmegawiz` command. Table 2–7 shows the file extension that corresponds to supported HDL types.

Table 2–7. Variation File Extensions

Variation File HDL Type	Required File Extension
Verilog HDL	**.v**
VHDL	**.vhd**
AHDL	**.tdf**

Command-Line Scripting Examples

This section presents various examples of command-line executable use.

Create a Project and Apply Constraints

The command-line executables include options for common global project settings and commands. To apply constraints such as pin locations and timing assignments, run a Tcl script with the constraints in it. You can write a Tcl constraint file yourself, or generate one for an existing project. From the Project menu, click **Generate Tcl File for Project**.

Example 2–8 creates a project with a Tcl script and applies project constraints using the tutorial design files in the <*Quartus II installation directory*>**/qdesigns/fir_filter/** directory.

Example 2–8. Tcl Script to Create Project and Apply Constraints

```
project_new filtref -overwrite
# Assign family, device, and top-level file
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C12F256C6
set_global_assignment -name BDF_FILE filtref.bdf
# Assign pins
set_location_assignment -to clk Pin_28
set_location_assignment -to clkx2 Pin_29
set_location_assignment -to d[0] Pin_139
set_location_assignment -to d[1] Pin_140
# Other assignments could follow
project_close
```

Save the script in a file called **setup_proj.tcl** and type the commands illustrated in Example 2–9 at a command prompt to create the design, apply constraints, compile the design, and perform fast-corner and slow-corner timing analysis. Timing analysis results are saved in two files, **filtref_sta_1.rpt** and **filtref_sta_2.rpt**.

Example 2–9. Script to Create and Compile a Project

```
quartus_sh -t setup_proj.tcl ↵
quartus_map filtref ↵
quartus_fit filtref ↵
quartus_asm filtref ↵
quartus_sta filtref --model=fast --export_settings=off ↵
mv filtref_sta.rpt filtref_sta_1.rpt ↵
quartus_sta filtref --export_settings=off ↵
mv filtref_sta.rpt filtref_sta_2.rpt ↵
```

Type the following commands to create the design, apply constraints, and compile the design, without performing timing analysis:

```
quartus_sh -t setup_proj.tcl ↵
quartus_sh --flow compile filtref ↵
```

The `quartus_sh --flow compile` command performs a full compilation, and is equivalent to clicking the **Start Compilation** button in the toolbar.

Check Design File Syntax

The UNIX shell script example shown in Example 2–10 assumes that the Quartus II software **fir_filter** tutorial project exists in the current directory. You can find the **fir_filter** project in the <*Quartus II directory*>**/qdesigns/fir_filter** directory unless the Quartus II software tutorial files are not installed.

The --analyze_file option performs a syntax check on each file. The script checks the exit code of the quartus_map executable to determine whether there is an error during the syntax check. Files with syntax errors are added to the FILES_WITH_ERRORS variable, and when all files are checked, the script prints a message indicating syntax errors. When options are not specified, the executable uses the project database values. If not specified in the project database, the executable uses the Quartus II software default values. For example, the **fir_filter** project is set to target the Cyclone device family, so it is not necessary to specify the --family option.

Example 2–10. Shell Script to Check Design File Syntax

```
#!/bin/sh
FILES_WITH_ERRORS=""
# Iterate over each file with a .bdf or .v extension
for filename in `ls *.bdf *.v`
do
# Perform a syntax check on the specified file
    quartus_map fir_filter --analyze_file=$filename
  # If the exit code is non-zero, the file has a syntax error
    if [ $? -ne 0 ]
    then
       FILES_WITH_ERRORS="$FILES_WITH_ERRORS $filename"
    fi
done
if [ -z "$FILES_WITH_ERRORS" ]
then
    echo "All files passed the syntax check"
    exit 0
else
    echo "There were syntax errors in the following file(s)"
    echo $FILES_WITH_ERRORS
    exit 1
fi
```

Create a Project and Synthesize a Netlist Using Netlist Optimizations

This example creates a new Quartus II project with a file **top.edf** as the top-level entity. The --enable_register_retiming=on and --enable_wysiwyg_resynthesis=on options allow the technology mapper to optimize the design using gate-level register retiming and technology remapping.

For more information about register retiming, WYSIWYG primitive resynthesis, and other netlist optimization options, refer to Quartus II Help.

The --part option tells the technology mapper to target an EP20K600EBC652-1X device. To create the project and synthesize it using the netlist optimizations described above, type the command shown in Example 2–11 at a command prompt.

Example 2–11. Creating a Project and Synthesizing a Netlist Using Netlist Optimizations

```
quartus_map top --source=top.edf --enable_register_retiming=on
   --enable_wysiwyg_resynthesis=on --part=EP20K600EBC652-1X ↵
```

Archive and Restore Projects

You can archive or restore a Quartus II project with a single command. This makes it easy to take snapshots of projects when you use batch files or shell scripts for compilation and project management. Use the `--archive` or `--restore` options for `quartus_sh` as appropriate. Type the command shown in Example 2–12 at a command prompt to archive your project.

Example 2–12. Archiving a Project

```
quartus_sh --archive <project name> ↵
```

The archive file is automatically named <*project name*>**.qar.** If you want to use a different name, rename the archive after it has been created. This command overwrites any existing archive with the same name.

To restore a project archive, type the command shown in Example 2–13 at a command prompt.

Example 2–13. Restoring a Project Archive

```
quartus_sh --restore <archive name> ↵
```

The command restores the project archive to the current directory and overwrites existing files.

For more information about archiving and restoring projects, refer to the *Managing Quartus II Projects* chapter in volume 2 of the *Quartus II Handbook*.

Perform I/O Assignment Analysis

You can perform I/O assignment analysis with a single command. I/O assignment analysis checks pin assignments to ensure they do not violate board layout guidelines. I/O assignment analysis does not require a complete place and route, so it is a quick way to ensure your pin assignments are correct. The command shown in Example 2–14 performs I/O assignment analysis for the specified project and revision.

Example 2–14. Performing I/O Assignment Analysis

```
quartus_fit --check_ios <project name> --rev=<revision name> ↵
```

Update Memory Contents Without Recompiling

You can use two commands to update the contents of memory blocks in your design without recompiling. Use the `quartus_cdb` executable with the `--update_mif` option to update memory contents from **.mif** or **.hexout** files. Then, rerun the assembler with the `quartus_asm` executable to regenerate the **.sof**, **.pof**, and any other programming files.

Example 2–15 shows these two commands.

Example 2–15. Commands to Update Memory Contents Without Recompiling

```
quartus_cdb --update_mif <project name> [--rev=<revision name>]↵
quartus_asm <project name> [--rev=<revision name>]↵
```

Example 2–16 shows the commands for a DOS batch file for this example. You can paste the following lines into a DOS batch file called **update_memory.bat.**

Example 2–16. Batch file to Update Memory Contents Without Recompiling

```
quartus_cdb --update_mif %1 --rev=%2
quartus_asm %1 --rev=%2
```

Type the following command at a command prompt:

```
update_memory.bat <project name> <revision name> ↵
```

Create a Compressed Configuration File

You can create a compressed configuration file in two ways. The first way is to run `quartus_cpf` with an option file that turns on compression. The second way is to run `quartus_cpf` with a Conversion Setup File (**.cof**).

To create an option file that turns on compression, type the following command at a command prompt:

```
quartus_cpf -w <filename>.opt ↵
```

This interactive command guides you through some questions, then creates an option file based on your answers. Use the `--option` option to `quartus_cpf` to specify the option file you just created. For example, the following command creates a compressed **.pof** that targets an EPCS64 device:

```
quartus_cpf --convert --option=<filename>.opt --device=EPCS64 <file>.sof <file>.pof ↵
```

Alternatively, you can use the Convert Programming Files utility in the Quartus II software to create a conversion setup file. Configure any options you want, including compression, then save the conversion setup. Use the following command to run the conversion setup you specified.

```
quartus_cpf <file>.cof ↵
```

Fit a Design as Quickly as Possible

This example assumes that a project called **top** exists in the current directory, and that the name of the top-level entity is **top**. The `--effort=fast` option causes the Fitter to use the fast fit algorithm to increase compilation speed, possibly at the expense of reduced f_{MAX} performance. The `--one_fit_attempt=on` option restricts the Fitter to only one fitting attempt for the design.

To attempt to fit the project called **top** as quickly as possible, type the command shown in Example 2–17 at a command prompt.

Example 2–17. Fitting a Project Quickly

```
quartus_fit top --effort=fast --one_fit_attempt=on ↵
```

Fit a Design Using Multiple Seeds

This shell script example assumes that the Quartus II software tutorial project called **fir_filter** exists in the current directory (defined in the file **fir_filter.qpf**). If the tutorial files are installed on your system, this project exists in the <*Quartus II directory*>/qdesigns<*quartus_version_number*> /**fir_filter** directory. Because the top-level entity in the project does not have the same name as the project, you must specify the revision name for the top-level entity with the `--rev` option. The `--seed` option specifies the seeds to use for fitting.

A seed is a parameter that affects the random initial placement of the Quartus II Fitter. Varying the seed can result in better performance for some designs.

After each fitting attempt, the script creates new directories for the results of each fitting attempt and copies the complete project to the new directory so that the results are available for viewing and debugging after the script has completed.

Example 2–18 is designed for use on UNIX systems using **sh** (the shell).

Example 2–18. Shell Script to Fit a Design Using Multiple Seeds

```
#!/bin/sh
ERROR_SEEDS=""
quartus_map fir_filter --rev=filtref
# Iterate over a number of seeds
for seed in 1 2 3 4 5
do
echo "Starting fit with seed=$seed"
# Perform a fitting attempt with the specified seed
quartus_fit fir_filter --seed=$seed --rev=filtref
# If the exit-code is non-zero, the fitting attempt was
# successful, so copy the project to a new directory
if [ $? -eq 0 ]
then
         mkdir ../fir_filter-seed_$seed
         mkdir ../fir_filter-seed_$seed/db
         cp * ../fir_filter-seed_$seed
         cp db/* ../fir_filter-seed_$seed/db
else
         ERROR_SEEDS="$ERROR_SEEDS $seed"
fi
done
if [ -z "$ERROR_SEEDS" ]
then
echo "Seed sweeping was successful"
exit 0
else
echo "There were errors with the following seed(s)"
echo $ERROR_SEEDS
exit 1
fi
```

Use the Design Space Explorer (DSE) included with the Quartus II software script (by typing `quartus_sh --dse` ↵ at a command prompt) to improve design performance by performing automated seed sweeping.

For more information about the DSE, type `quartus_sh --help=dse` ↵ at a command prompt, or refer to *Design Space Explorer* in Quartus II Help.

Regenerating Megafunctions After Updating the Quartus II Software

Some megafunction variations may reqiure regeneration when you update your installation of the Quartus II software. Read the release notes for the Quartus II software and any new documentation for the IP functions used in your design to determine if regeneration is necessary.

If regeneration is necessary, you can use a Tcl script to run the **qmegawiz** executable to update each function, allowing you to avoid regenerating each function in the Megawizard Plug-In Manager GUI.

Wizard-generated files are identified in the Source Files Used report panel (contained in <*project name*>**.map.rpt**) in the File Type column as "Auto-Found Wizard-Generated File". In a Tcl script, use the commands in the **::quartus::report** package from the Quartus II Tcl API to recover the list of files. Use the `qexec` command in a loop to run **qmegawiz** for each variation file:

```
qexec "qmegawiz -silent <variation file name>"
```

For example, if your script determines that your design contains a variation file called `myrom.v`, in one iteration of the loop in your script, a combination of strings and variables passed to the `qexec` command would be equivalent to the following command:

```
qexec "qmegawiz -silent myrom.v"
```

If your design flow incorporates parameter files, those can be included in the qmegawiz call in the same way you would include them from a command prompt:

```
qexec "qmegawiz -silent -f:<parameter file>.txt <variation file name>"
```

For more information about the **::quartus::report** Tcl package, refer to *::quartus::report* in Quartus II Help.

For more information about the Quartus II Tcl scripting API, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*.

The QFlow Script

A Tcl/Tk-based graphical interface called QFlow is included with the command-line executables. You can use the QFlow interface to open projects, launch some of the command-line executables, view report files, and make some global project assignments. The QFlow interface can run the following command-line executables:

- `quartus_map` (Analysis and Synthesis)
- `quartus_fit` (Fitter)
- `quartus_sta` (TimeQuest timing analyzer)
- `quartus_asm` (Assembler)
- `quartus_eda` (EDA Netlist Writer)

To view floorplans or perform other GUI-intensive tasks, launch the Quartus II software.

Start QFlow by typing the following command at a command prompt:

```
quartus_sh -g ↵
```

☞ The QFlow script is located in the <*Quartus II directory*>**/common/tcl/apps/qflow/** directory.

Document Revision History

Table 2–8 shows the revision history for this chapter.

Table 2–8. Document Revision History

Date	Version	Changes
December 2012	10.1.0	Template update. Added section on using a script to regenerate megafunction variations. Removed references to the Classic Timing Analyzer (quartus_tan). Removed Qflow illustration.
July 2010	10.0.0	Updated script examples to use quartus_sta instead of quartus_tan, and other minor updates throughout document.
November 2009	9.1.0	Updated Table 2–1 to add quartus_jli and quartus_jbcc executables and descriptions, and other minor updates throughout document.
March 2009	9.0.0	No change to content.
November 2008	8.1.0	Added the following sections: ■ "The MegaWizard Plug-In Manager" on page 2–11 ■ "Command-Line Support" on page 2–12 ■ "Module and Wizard Names" on page 2–13 ■ "Ports and Parameters" on page 2–14 ■ "Invalid Configurations" on page 2–15 ■ "Strategies to Determine Port and Parameter Values" on page 2–15 ■ "Optional Files" on page 2–15 ■ "Parameter File" on page 2–16 ■ "Working Directory" on page 2–17 ■ "Variation File Name" on page 2–17 ■ "Create a Compressed Configuration File" on page 2–21 ■ Updated "Option Precedence" on page 2–5 to clarify how to control precedence ■ Corrected Example 2–5 on page 2–8 ■ Changed Example 2–1, Example 2–2, Example 2–4, and Example 2–7 to use the EP1C12F256C6 device ■ Minor editorial updates ■ Updated entire chapter using 8½" × 11" chapter template
May 2008	8.0.0	■ Updated "Referenced Documents" on page 2–20. ■ Updated references in document.

For previous versions of the *Quartus II Handbook*, refer to the Quartus II Handbook Archive.

Take an online survey to provide feedback about this handbook chapter.