Qflow 1.4 Digital Synthesis Flow Reference Page

Table of Contents

Qflow GUI

About the Qflow GUI

Qflow has a GUI interface to simplify the whole process of running synthesis, place, and route, and all of the other related processes. The GUI was introduced in qflow version 1.2.22 (April 29, 2018). The qflow GUI is written in python3, so it will be necessary to install python3 on your system if it is not already installed.

Starting the Qflow GUI

Command-line Syntax

qflow gui [**-T** *technology source_path*]

When invoking qflow with the GUI, it is assumed that the current working directory will be the project directory.

If qflow has been invoked previously and the project directory was set up and seeded with information about the technology and source file(s), then it is only necessary to run "**qflow gui**", as all other information will be ascertained from existing project files.

If running the qflow GUI for the first time in a new project directory, then it is preferable to give the technology and source filename on the command line. The project directory does not need to be set up

with standard subdirectories and such beforehand, as the first step in the GUI flow will take care of this automatically.

technology is the name of a technology installed with qflow, or the full path to a directory containing at least a file "*technology*.**sh**" and assorted other technology-dependent files used by qflow. If there is a subdirectory called "tech/" in the current working directory, then it is expected to be a directory containing the technology information, or a symbolic link to such a directory. If directory or symbolic linke "tech/" exists, then the technology found there will be used for the project.

source_path may be the full path of a verilog file if the project directory is not set up with a subdirectory "source/" before calling qflow. During the project preparation, qflow will create a symbolic link to this file from the qflow source. subdirectory.

All information below refers to the non-GUI invocation of qflow. Any project directory that has been set up in the manner described below with "source", "synthesis", and "layout" subdirectories may be used with or without the GUI interchangeably. A project that has been created without the GUI in a flat file space will not work with the GUI.

For more information about the operation of the GUI, see the GUI tutorial. In general, the steps provided by the GUI using push-button operation are the same steps as listed under "Actions" below for the command-line use of qflow, with the addition of several layout editor-related steps such as DRC, LVS, and generation of GDS format output.

Qflow Usage

Setup

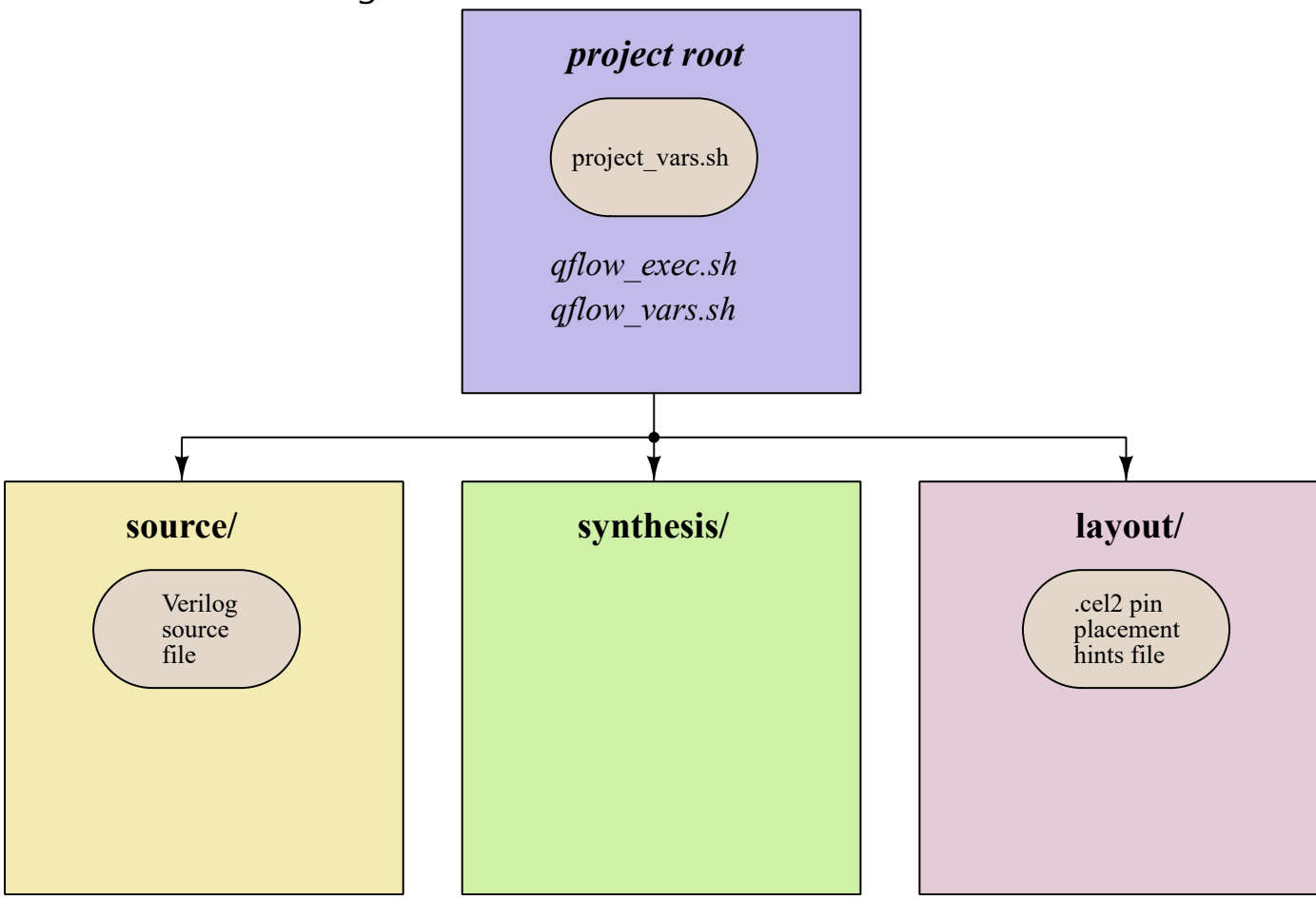
The first step in synthesizing a circuit is to have a **project directory** which will be the workspace for qflow. The project directory can be named whatever you want. The project directory should have three subdirectories, named

source
: Contains the verilog source code

synthesis, and
: Contains working files and RTL verilog output

layout.
: Contains working files and DEF file output

The **verilog source** file (**.v** extension) should be placed in the **source** subdirectory. This is mostly for convenience in organizing files. If you specify a flat directory structure, qflow will work with it, but you will have a large number of files all in the same directory, which can be confusing.



Execution

From the project directory, do:

qflow *source_name*

That is, if you have a verilog source file named "circuit.v", you should run the command "qflow circuit". For a complete list of options, see the section below. The result of this step is the creation of three files:

qflow_vars.sh
: Contains project-dependent variables needed by all the tools, identifying the project paths, qflow executable paths, and technology used.

qflow_exec.sh
: Contains the sequence of commands needed to complete the synthesis flow.

project_vars.sh
: This is an empty file, but may be edited for control over the command line options passed to various tools in the tool chain.

This file will never be overwritten, so user options can be preserved.

When **qflow** is given no options, the file **qflow_exec.sh** will have a list of commented commands. This allows the user to uncomment the tool chain commands one at a time and run each tool independently. In most cases, the design cycle will require checking the result of each step and making adjustments as necessary. Eventually, the flow will be stable and all steps can be run in batch.

Stages of the Flow

Here is the typical sequence of commands found in the **qflow_exec.sh** file for qflow 1.1:

synthesize.sh *options*
: Logic synthesis, standard-cell mapping, and load balancing, fanout reduction, and decongestion.

vesta.sh *options*
: Static timing analysis (without back-annotated parasitics).

placement.sh
: Placement of standard cells with buffer tree optimization

router.sh
: Detail routing

vesta.sh -d *options*
: Static timing analysis (with back-annotated parasitics).

migrate.sh

cleanup.sh
: Removal of temporary working files

display.sh
: Display of the final layout

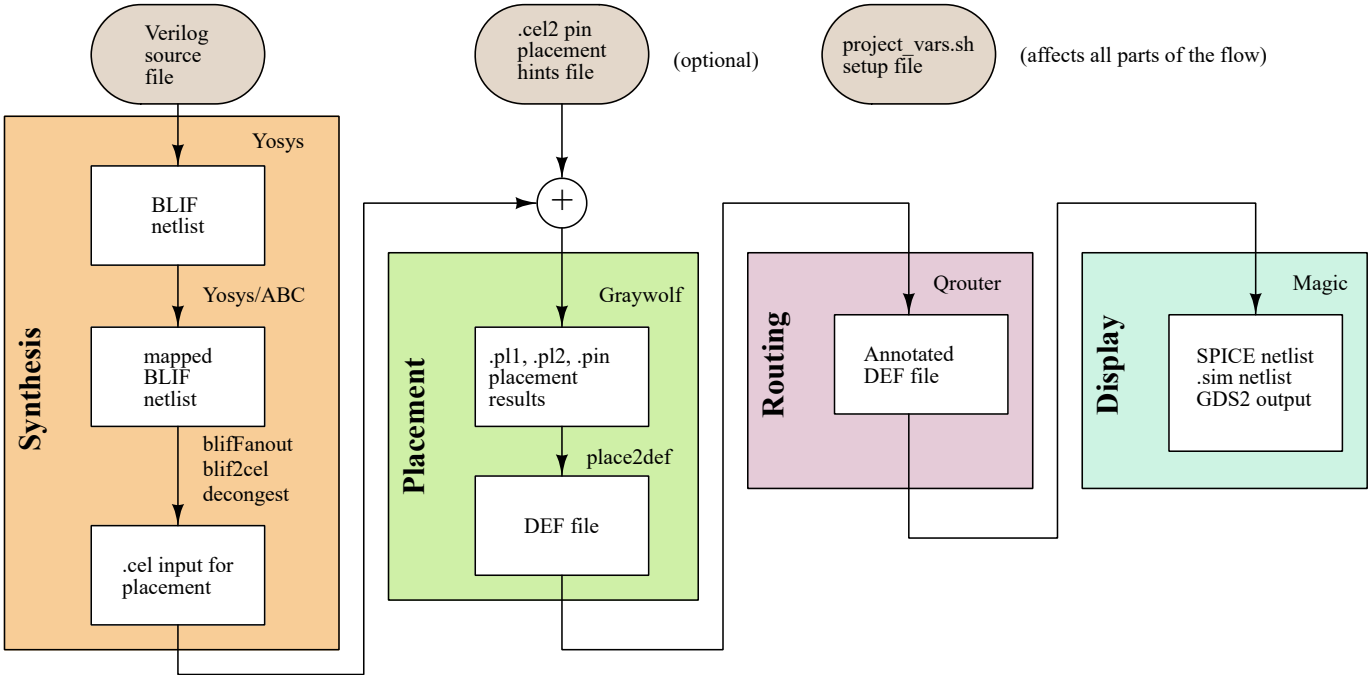
These additional sequences are available in qflow 1.2:

migrate.sh *options*
: Generation of layout and abstract views, and netlist extraction.

drc.sh *options*
: DRC (Layout design rule check) analysis

lvs.sh *options*
: LVS (Layout vs. schematic) analysis

gdsii.sh *options*
: Generation of GDS mask data



Qflow version 1.0 has an additional level of resynthesis followed by a second placement stage, which was improved and merged into a single step in qflow version 1.1:

synthesize.sh *options*
: Logic synthesis, standard-cell mapping, and load balancing.

vesta.sh *options*
: Static timing analysis (without back-annotated parasitics).

placement.sh
: Placement of standard cells

resynthesize.sh
: Placement-based fanout reduction, and load balancing

placement.sh
: Placement using modified netlist from resynthesis

router.sh
: Detail routing

cleanup.sh
: Removal of temporary working files

display.sh
: Display of the final layout

Qflow Syntax and Options

Qflow Command Syntax

qflow [*options*] [*actions*] *source_name*

Where *options* and *actions* are listed below. *source_name* is the name of the verilog source file to synthesize (with or without the ".v" extension).

Options

-T *technology*
--tech *technology*
: Use the technology *technology* for synthesis. If not specified, then the directory is searched for an existing file **qflow_vars.sh**. If it exists, then it is parsed to see what technology was previously specified for the project, and selects that to be the technology. Otherwise, the default technology (**osu035**) is used. If the technology is specified by name, it should be a named directory that can be found in the qflow install location under subdirectory "**tech**" (normally, "**/usr/local/share/qflow/tech/**"), or in the current working directory, or in the directory indicated by environment variable **QFLOW_TECH_DIR**. In the technology directory must be a file named *technology***.sh**, containing all of the configuration settings used by qflow (see section on adding new technologies, below).

-p *project*
--project *project*
: This option may be supplied if **qflow** is not being run in the top-level project directory. The value of *project* is the project directory where **qflow** should dump its files, and all pointers in those files will refer to the indicated project directory *project*.

-t *toolname*
--tool *toolname*
: This option specifies a non-default tool to be used by the synthesis flow. At present, only one option is available, which is **odin**, to use the Odin-II synthesis frontend instead of the default **yosys** synthesis frontend, in qflow version 1.0. In qflow version 1.1, this option is no longer available.

-h
--help
: Generate some help text, and exit.

-v
--version
: Print the qflow version number, and exit.

Actions

All actions are represented in the output file **qflow_exec.sh**. However, if specific actions are called out on the command line, those commands related to the actions will be left uncommented in the file. When **qflow** sources the command file, these actions will be executed in sequence.

There are two actions that are easy to remember and encompass the most important steps, so you don't have to remember all the names of all the steps to add to the command line:

build
: Run these steps in sequence: synthesize, place, buffer, route (see below)

all
: Run these steps in sequence: synthesize, place, buffer, route, clean, display (see below)

Individual synthesis steps are as follows:

synthesize
: Run the verilog synthesis (yosys, or, in qflow-1.0 only, Odin-II and ABC) and load balancing (blifFanout) and node fanout reduction (blifFanout, in qflow-1.1 only)

sta
: Run static timing analysis (vesta)

place
: Run the placement (graywolf)

buffer
(qflow 1.0 only)
: Perform fanout reduction, and rerun the load balancing and placement (clocktree, blifFanout, and graywolf). Note that qflow 1.1 does this in one pass during the "synthesize" stage.

route
: Run the detail router (qrouter)

backanno
: Run static timing analysis (vesta) with back-annotated delays

migrate
: Create layout and abstract views, and extract netlist from layout (qflow 1.2 and higher).

drc
: Run design rule checks on the layout (qflow 1.2 and higher).

lvs
: Run layout vs. schematic check (qflow 1.2 and higher).

gdsii
: Generate GDS mask layer output (qflow 1.2 and higher).

clean
: Clean up temporary files in all subdirectories.

display
: Display the layout result (magic)

Environment Variables

The following environment variables are meaningful to qflow:

QFLOW_TECH_DIR
: Use the variable definition as the location where the technology directory can be found. This environment variable overrides the default location in the **qflow** install path. It can be overridden on the command line with the "**-T**" option.

QFLOW_TECH
: Use the variable definition as the name of the directory where technology information can be found for the target technology for the synthesis. This environment variable overrides the default technology that is provided with the **qflow** distribution, which is "**osu035**". It can be overridden on the command line with the "**-T**" option.

QFLOW_PROJECT_ROOT
: Use the variable definition as the name of the top-level project directory. This environment variable overrides the default project location, which is the current working directory ("**.**"). It can be overridden on the command line with the "**-p**" option.

User-defined options

For most projects, the user will want to have control over many aspects of the synthesis process. The file "**project_vars.sh**" is the primary location of such user options. Most options are declared by setting a shell variable in the file. The following shell variables are recognized by qflow. This list is frequently updated. More options are available than are listed here; only those option settings of interest or help to the user are described here:

The first set of options (from qflow version 1.4) selects which tool to use for each step. The "tools" are shell scripts that are called from the qflow_exec.sh file. The script must have the name *tool_name***.sh** and should be installed in the qflow installation directory for script (by default, `/usr/local/share/qflow/scripts/`). In most cases, the tool name is just the name of the application that provides the flow step, but in cases where one tool has multiple uses (like magic), the script name needs to be unique (e.g., magic_drc and magic_gds).

To add a new tool to qflow, a shell script wrapper needs to be written that calls the new tools with the appropriate options and arguments. New tools are added as they become available and are integrated into qflow.

synthesis_tool=*tool_name*
Currently supported tools: **yosys**

placement_tool=*tool_name*
Currently supported tools: **graywolf** and **replace**

sta_tool=*tool_name*
Currently supported tools: **vesta**, **opensta**, and **opentimer**

router_tool=*tool_name*
Currently supported tools: **qrouter**

migrate_tool=*tool_name*
Currently supported tools: **magic_db**

lvs_tool=*tool_name*
Currently supported tools: **netgen_lvs**

drc_tool=*tool_name*
Currently supported tools: **magic_drc**

gds_tool=*tool_name*
Currently supported tools: **magic_gds**

display_tool=*tool_name*
Currently supported tools: **magic_view**

The second set of options applies to each of the flow steps:

source_file_list=*filename*
Qflow has yosys auto-detect all of the modules in the design and create a list of files to include to read all of the modules, in case the verilog source does not explicitly include each file. However, if files are not in the source directory then they won't be found. If for that reason or any other, the synthesis script fails to find all needed source files, a list of files can be put in a file, one source file per line, and this file passed as the value *filename* for **source_file_list**.

is_system_verilog
Qflow will auto-detect system verilog for files that have the extension "`.sv`". If files are not being properly detected as system verilog, then enable this option.

hard_macros="*macro1 macro2 ...*"

Qflow generally works on designs with standard cells. However, designs may contain instances of other macros that are not part of the standard cell set. To enable the synthesis flow to use hard macros, all such macros need to be listed as a space-separated list of names, enclosed in quotes.

yosys_options="*text*"

Options to pass to yosys for synthesis. In cases where more control is needed over the processing done by yosys, this option is the best one to use, namely for passing an alternative script to yosys through the "-s" option. Most other command-line options to yosys will not be of use to the qflow user.

The preferred procedure is to run qflow with standard options to yosys; this will create a file "*modulename***.ys**" containing a usable, but not necessarily optimal, processing script. It will, however, contain all the correct commands for reading the verilog file or files, mapping to the technology specified for qflow, and writing a BLIF format file needed for the rest of the synthesis flow. This file will be regenerated and overwritten on each run of qflow. Copy this file to another name. Then edit it for any extra processing that is needed (e.g., "iopadmap -inpad ..." to add buffers to input signals), and set **yosys_options** to "**-s** *newscript***.ys**".

yosys_script=*filename*

The *filename* indicates the name of a file containing commands to pass to yosys to perform the main part of the synthesis. The contents of the file are inserted into the normal synthesis script between the commands to load the necessary verilog source files and the commands to map the synthesis result to the standard cell set. It therefore differs from passing "**-s** *filename*" in **yosys_options** (see above), which is a complete replacement for the entire synthesis script. The content of *filename* is a direct replacement for the following standard synthesis commands used by qflow:

```
proc; memory; opt; fsm; opt
```

yosys_debug

If this variable is defined, then the "-purge" option is not passed to yosys during optimization, so that all internal node names are retained. This results in a layout that is much bigger, with a number of non-functional buffers inserted to maintain nets with specific signal names (particularly prevalent for layouts derived from hierarchical verilog sources). The result allows all signal

names to be probed during simulation and debugging, but is generally not what one would want in an implementation.

nobuffers

If this variable is defined, then the module outputs will be unbuffered; that is, module output pins will be connected directly to register outputs, which may also be internally connected to other inputs. The default behavior is to buffer register outputs, so that output pin loads cannot change the internal timing characteristics of the circuit. Buffering is highly recommended. The typical reason for not buffering is to synthesize a small subcircuit, such as a decoder driving an analog circuit, where all output loads are small, and circuit area is restricted.

inbuffers

Specifies that all module inputs should be buffered. Buffering inputs is helpful to keep input timing accurate, since a signal with a slow rise- and/or fall-time may arrive at significantly different times on input gates throughout a digital layout. The buffering ensures that the arriving signal is repeated with a sharp edge, and the synthesis is better able to control the timing.

abc_script="*string*"

(Qflow version 1.1 only)

The *string* represents either the name of a file containing ABC commands, or, if it starts with the character "**+**", a set of semicolon-separated commands passed to the ABC logic optimizer (see ABC documentation for information about valid commands that can be passed to it; see yosys documentation for the specific syntax of the "**abc -script**" command in yosys). This option was added in qflow-1.1.13 when it was noted that a non-default script could improve synthesis results. The default script was updated to the improved one, but the option was added to allow the qflow end-user to experiment with other scripts, should they wish to do so.

Note that when this option is in the form of a single unquoted string starting with "**+**", ABC commands must be strung together without spaces or linefeeds, separated only by semicolons.

postproc_options="*text*"

These options are passed to the `vlog2Verilog` tool for manipulating netlists after synthesis. The most common use is to set *value* to "**-anchors**", which tells qflow to add antenna tie-

downs to all module inputs (assuming that the standard cell set has an antenna tie-down cell).

xspice_options="*text*"

This option provides values to pass to the `spi2xspice.py` script. In the current version (qflow-1.4), `spi2xspice.py` does not provide exact timing in the standard cells, so average delay values are used for every cell. Each value (options are shown below) is given as a SPICE-type value, *i.e.*, using a number and metric prefix (*e.g.*, "`10n`" for 10 nanoseconds):

-io_time=*value*

Rise and fall time for signals in and out of the digital block

-time=*value*

Rise and fall time of gate outputs

-idelay=*value*

Input delay at gate inputs

-odelay=*value*

Throughput delay of the gate

-cload=*value*

Gate output load capacitance

vesta_options="*text*"

One of the more important options to Vesta, the static timing analyzer, is "**--period** *value*", to specify an expected minimum clock period for circuit operations. The clock period is given in units of picoseconds (ps). The option "**-l** *load*" specifies a typical output pin load capacitance of the circuit, when integrated into and operating in a larger context.

opentimer_options="*text*"

List of options to pass to OpenTimer (see OpenTimer documentation).

opensta_options="*text*"

List of options to pass to OpenSTA (see OpenSTA documentation).

fill_ratios=*list*

A comma-separated list of the relative ratios of different types of fill cells to use for fill in the design (for graywolf placement only). There are four types of fill used: Plain fill, decap fill, antenna tie-downs, and substrate/well tie-downs. The ratios are given in the order listed (plain, decap, antenna, substrate), must

have no whitespace, and the values must sum to a total of 100 (e.g., "0,85,15,0"). Any zero value may be omitted but the commas must be present (e.g., ",85,15,"). Note that values are only relevant if the chosen standard cell set contains fill cells of the specified types (the OSU standard cell sets distributed with qflow contain only plain fill cells, for example). If not specified, the default behavior is to enumerate the types of fill cell present in the standard cell set and distribute them equally as fill.

nofanout

If **nofanout** is set, then the `vlogFanout` tool is not run. This prevents gate resizing and buffering of large fanout nets, and generation of clock trees. It is recommended only to use this option if the input source is a netlist already containing sized gates, or if the synthesis tool (or placement tool) does clock tree insertion and/or gate resizing.

fanout_options="*text*"

blifFanout has only a few options that are not already used by qflow: "-l " specifies a time value, in ps, that is the target maximum time through any gate in the circuit. "-c " specifies the maximum capacitive load, in fF, expected on any output pin of the circuit. Default values are 75ps latency, and 25fF capacitance.

addspacers_options="*options*"

Options "**-stripe** *width spacing pattern* [**-nostretch**]" will configure the addspacers tool to automatically add power and ground stripes from top to bottom of the layout, connecting together all rows' power and ground. *width* and *spacing* are given in microns, while *pattern* is a character string with some combination of "**P**" (power), "**G**" (ground), or "**0**" (none). These are applied to stripes in sequence from left to right, repeating as needed. So pattern "**PG**" would be a set of stripes alternating between power and ground (the most common pattern). To avoid specific routing difficulties or pin collisions, one may wish to apply a different pattern. The option "**-nostretch**" tells addspacers to place the power buses on top of the existing layout. Without this option, the cell will be stretched with fill cells under each power stripe. Stretching the cell may or may not help avoid routing issues around the power stripes.

initial_density=*value*

(Available from qflow version 1.0.95) This option sets an initial density to use, where the density *value* is defined as the fraction of the layout area comprising actively routed (that is, not fill or decap) cells. For small cells (< 1000 gates), it should not be

necessary to specify a density (the default is 1, or no filler cells except as needed to straighten up the edges of the layout area). For large cells, and especially for cells with areas of dense combinatorial logic (such as multipliers), specifying an initial density is critical to getting the design to route.

graywolf_options="*text*"

List of options to pass to graywolf (see graywolf documentation).

qrouter_options="*text*"

The useful qrouter options are "-k *tries*", "-f", and "-v *value*". Qrouter evaluates its progress, and in the case of a difficult route---such as too much congestion with too few route layers available---it may determine that it is stuck. Occasionally its evaluation is wrong and it exists just prior to getting a valid solution. It is rare, but just in case, the option "-k *tries*" tells qrouter to "keep going" *tries* more times, after things start looking hopeless. Beware---this can cause ridiculously long runtimes, and qrouter may never find a solution (prior to qrouter version 1.3.18, the option was simply "-k" with no addition value, and would keep trying indefinitely). The option "-f" tells qrouter to force a connection to a pin, even if qrouter determines that all tap points of a pin are unroutable due to obstructions. The "-v" sets the level of verbose output, which can produce a huge amount of output, so use it carefully.

The option "-r *value*" was useful for versions of qrouter prior to 1.3.42, and indicates the scalefactor of the output precision compared to centimicrons. Thus, if the manufacturing grid is 5nm, then "**-r 2**" should be used. From version 1.3.42, qrouter automatically gets the manufacturing grid precision from the technology LEF file.

route_show

If set, then the router is run with a graphic view. If not, the router is run in batch mode and only text output is shown.

route_layers=*value*

Set the number of layers to use for routing to *value*. This can be used to reserve upper metal layers for power only, and not to use those layers as route layers, even if the technology file defines them as routing layers.

via_use=*via_name_list*

This values is passed to qrouter from the ".cfg" router configuration file. It indicates a list of via names from the

technology LEF file that are allowed to be used. The restriction can be used to enforce via orientation, or double vias, etc.

via_stacks=*num_vias*

This value is passed to qrouter from the ".cfg" router configuration file. It indicates how many vias may be stacked on top of one another. Usually, this value will be set in the technology script, but the user may want to make the stacking more restrictive.

qrouter_nocleanup

Setting this value causes qrouter to not run the third stage "cleanup" routing. The third stage can take a long time to run, and although recommended for final layout, it may be omitted for speed when iterating routing to determine the best density at which a design is routable.

migrate_options="*text*"

The value of *text* is passed directly to the command line of Magic when processing the migration script.

lef_options="*text*"

The value of *text* is passed to the end of the "lef write" command in Magic when running the migration script. The main options in magic are "**-tech**", which writes technology information at the top of the macro file, and "**-hide**", which removes interior detail and replaces it with large blocks of obstruction layers. The default is to list all metal layers internal to the layout as obstructions or pins. For large designs, this can result in a very large LEF file.

drc_gdsview

Enable this option to use the full GDS database view of standard cells when running DRC. Otherwise, the abstract view will be used (such that only back-end metal stack rules will be checked). The most common reason to use this option is when a standard cell set has abstract views that violate DRC rules, leading to false positive DRC counts throughout a design.

drc_options="*text*"

The value of *text* is passed directly to the command line of Magic when processing the DRC script.

magic_display=*display_type*

Use the value *display_type* for the graphics interface when running Magic for the layout display. The display script will look for Cairo graphics support (the "better" graphics, in magic-8.2) but will (default to X11 if not found. Because OpenGL (the

"better" graphics in magic-8.1 and earlier) has problems on many systems, it is not supported by default, but if it works, it can be specified using **set magic_display = OGL** in the project_vars.sh file.

The Digital Synthesis Flow

Synthesis

- **yosys** frontend
    - **yosys** reads the verilog source file or files, performs synthesis, optimizations, and writes a netlist in BLIF format.
    - **ypostproc** cleans up some of the syntax from yosys output as needed by tools further downstream in the flow.
- **Odin-II** frontend (qflow version 1.0 only)
    - **vpreproc** reads the verilog source file and separates it into multiple source files and other informational files used by the synthesis process. For example, it pulls out all register reset values, which cannot be synthesized by Odin-II or mapped by ABC, and uses the reset values to substitute the proper set or reset flip-flops into the netlist.
    - **odin_ii** reads the revised verilog source file, checks syntax, and produces a logic format file (**BLIF**).
    - **abc** reads the logic format file, runs logic optimization, then maps the resulting logic to the standard cell set. It produces an output netlist file in the **BLIF** netlist format, which enumerates all of the mapped standard cells except flip-flops, which are left as general-purpose "latch" statements.
    - A post-processing script "**postproc**" replaces registers in the netlist file with set or reset flops, according to information provided by the preprocessor.
    - Another simple post-processing "sed" script resolves some of the ungainly syntax produced by Odin-II and ABC.
- **blifFanout** analyzes the netlist for the load on each gate output, and attempts to keep the latency of each output less than the default value of 75ps by making use of logic cells with higher drive strength. The "**-l** *latency*" option specifies the

maximum output latency, in ps. Nets with high fanout can be reduced by buffering using the "**-c** *cap*" option to specify a maximum load capacitance on an output, in fF, but fanout reduction is better handled by the resynthesis stage (see below), which takes placement into account when breaking a network into groups.

Qflow version 1.1 uses **blifFanout** to buffer nodes with large fanout, replacing the **clocktree** tool from qflow version 1.0. The method using **blifFanout** is a single-pass method, not requiring a pre-placement.

- The **blif2cel** script takes the final netlist file and turns it into an input file for the graywolf placement tool. In addition, it takes a template parameter file for graywolf "*techname***.par**", and places it in the layout subdirectory for use by graywolf, and renames it "*source_name***.par**"

Placement

- The first task of placement is to look for a file named *source_name***.cel2**. This file is appended to *source_name***.cel** in the layout subdirectory, and provides the information about where pins should be placed. See below under "Additional Information" for the format of the **.cel2** file. Pins can be fixed, or they can be restricted to specific sides of the layout, or they can be put in specific order or allowed to float to the most convenient place.

- The technology parameter file "*source_name***.par**" contains information about the technology that is used by graywolf to estimate the routing resources. Generally the "RULES" section of this file is unimportant, as graywolf will not attempt a detailed route. There are, however, a few lines that may be user-edited. Note that once the **.par** file exists in the layout directory, it will not be overwritten, so changes made to the file will be retained.

  ***rowSep :** *relative absolute*

  This line forces the row spacing. Normally, a standard cell set will have borders that abut between rows, and so the values for *relative* and *absolute* should be left at their default values of zero. The *relative* value is multiplied by the row height, so that logic cell rows can be spaced apart by, say, one-half of the row height. The *absolute* value is an absolute measurement. The units are the same as used by the **.cel** and should normally be in centimicrons.

  GENR*numrows : *value*

Graywolf will make a layout that has *value* rows of standard cells. This value can be used to control the aspect ratio of the layout. If the line is commented out or deleted, graywolf will attempt to create a square layout.

GENR*flip_alternate_rows : *value*

where *value* is either 1 (true) or 0 (false). If true, then every other row is flipped upside-down. This is the normal way to abut standard cells, so that pairs of rows share power busses. The value will be set to match the technology, so it should not be changed unless there is a need to orient the cell rows differently. Then it is the responsibility of the user to ensure that the pins remain on route tracks.

- **Graywolf** is executed, and generates a placement for all of the digital standard cell instances in the design. The run-time of the placement stage is approximately proportional to the number of standard cell instances in the design. The output of the placement stage is three files in a format used only by graywolf and the qflow scripts that convert its output.

- The tool **place2def** converts the graywolf output into a standard DEF file containing all placement and netlist information, but with the NETS section containing no routing information. In addition, it writes out a **.cfg** file that has additional information needed by the router. Specifically, the **.cfg** file has the final say about the number of layers used for routing. By default, it will use as many layers as are defined for routing by the technology. It has a few additional options to define whether or not there are restrictions on the stacking of vias (this is technology-dependent), and defines obstructions to prevent routing above and below pins. The user may use this file to restrict the number of routing layers used, and to specify additional obstructions for routing.

Fanout Reduction

- The tool **blifFanout** breaks up high-fanout nodes into buffered trees to reduce total latency on the node. This method connects destination nodes to buffers arbitrarily, and depends on methods used by the **graywolf** placement tool to automatically adjust these connections for optimal routing. This is a single-pass method done during the initial synthesis step. The adjusted connections are back-annotated into the netlist after placement.

Routing

- The tool **qrouter** performs the detail route based solely on the technology information about route layers and the unrouted DEF file. It produces an output file which is an annotated DEF file with the detail route information added to it. The output-buffered netlist input to qrouter is named *source_name*.**def**; the router script renames the input file to *source_name*_**unrouted.def** and the router output filename becomes *source_name*.**def**, and is the final physical circuit. Layout tools that are able to read DEF format files can read this file directly.

Display

- The tool **magic** (version 8.0) reads the LEF file describing the routing layers for the technology, then reads the final DEF file, and displays the result in the layout viewer.

Additional Information

1. There are several switches you can pass to the `synthesis.sh` script to have some meager amount of control over the post-synthesis process. The two parsed options are "`-x`" and "`-c` *file*". If "`-x`" is selected, all inputs are latched by inserting a flip-flop after each input signal. The "`-c`" option is similar, except that *file* points to a file containing a list of input signals to be latched. If there is a clock signal that is passed to flip-flops inside the cell, it is used as the clock signal for the input latches. If all the internal logic is combinatorial, then an extra signal called "**clock**" is added to the circuit. (NOTE: This item refers to netlist manipulation performed by AddIOtoBDnet, which is currently removed from the flow)

   You can also change the "*flopcell*" or "*bufcell*" entries in the technology shell script to change the cell used for the input latches or the output buffers, respectively. For example, you might want the inputs to be latched with a transparent latch instead of a flip-flop, or you might want all outputs to be buffered with 4x strength buffers.

2. The `.cel` file contains the list of standard-cell subcircuits used to implement the circuit, and the network of connections between them. While the netlist is automatically generated and should not be messed with, the script that generates it appends a file of the same name with the extension `.cel2` to the end of the `.cel` file. The `.cel2` file is a convenient place to declare all the input and output signals as part of `padgroups` and give detailed information about where they should be placed around the edges of the layout.

For a complete description of the `.cel` file format, see the documentation on **graywolf**. In summary, pins are grouped into "padgroups" with the following syntax:

```
padgroup group_name [permute| nopermute]
twpin_pinname [fixed|nonfixed ]
...
[restrict side sides]
[sidespace frac-low frac-high]
```

where *sides* may be one or more of **T**, **B**, **L**, or **R**, concatentated (with no spaces or other puntuation in between), which tells graywolf to which side or sides those signals are restricted. If not restricted, graywolf will place the pin on the side of the cell that minimizes the net length.
The `sidespace` statement restricts placement between limits specified as a fraction of the total side length, between zero and one inclusive.

Also look at this example .cel2 file that corresponds to the qflow tutorial example.

3. The `.par` file contains the parameters passed to the graywolf place and route program. For a complete list of options and their meanings, you should consult the graywolf documentation in the distribution. Most of the values in this file are determined by the fabrication process and the properties of the standard cells. The one value you will want to pay attention to is "GENR*numrows", which is the number of rows of standard cells that will be used. If you remove this line, graywolf will attempt to create a square (1:1 aspect ratio) placement. Often you will have a specific space in a chip design to fill, and can divide this space by the standard cell height to arrive at the number or rows of logic to use.

   Graywolf is *not* a sea-of-gates global router (it makes reference to one called SGGR in the distribution, but SGGR happens to be missing from the distribution. . .). By specifying a large number of implicit pass-through connections in each standard cell, the result produced by graywolf is close to the result it would get for a sea-of-gates or over-the-cell router.

4. To get the final GDS format file, run Magic either with the GDS view of the standard cells saved to a directory and accessed with the "addpath" command, or read on the fly, which is the method I show here:

   ```
   magic -d OGL
   ```

   And on the command line:

   ```
   gds readonly true
   gds rescale false
   gds read osu035_stdcells.gds2
   load map9v3
   gds write map9v3
   ```

The end result of this step is the finished GDS file of the design. . . you're done!

5. The Magic layout editor is also a good resource for extracting a netlist of the layout based entirely on the physical layout, to get either a SPICE or sim netlist. The SPICE netlist can be used in conjunction with **netgen** to verify the layout against the original synthesized netlist. The sim netlist can be used in conjunction with **IRSIM** to perform a functional verification of the system. For details, see the tutorial page.

Creating a new Digital Flow

The following are the steps needed to create a digital flow for an arbitrary technology, given the presence of the components outlined in the first section at the top of this web page.

Most of these files have simple syntax that can be copied from an existing technology, like the OSU035 technology that is packaged with qflow, and modified as appropriate for the new technology.

The list of files and what they are used for is given below, followed by a description of the syntax of each file format used.

.sh Main variable definition file

This is the most important file of all, because it is used to gather information from different sources into a single file of variable names that can be used by each of the qflow scripts.

The file consists entirely of "**set**" lines in tcsh syntax (**set** *varname*=*value_string*). If "*value_string*" contains spaces then it should be in quotes. Many scripts source this file, so the entire file must contain valid tcsh syntax.

The variable names in the technology .sh file that are recognized by qflow are listed below in four groups. The first group is critical and points to all of the standard format files needed for routing:

set leffile = *path*/*file*.**lef**

Indicates the path to the LEF file defininig the standard cell macros.

set techleffile = *path*/*file*.**lef**

Indicates the path to the LEF file defining route layer width, pitch, spacing, etc. Sometimes this information is in the cell macro LEF file, in which case this variable may be omitted.

set libertyfile = *path*/*file*.**lib**

Indicates the path to the liberty format description of the standard cell functions and pin timing.

The second group is files that are not in the critical path of the synthesis flow, but are used to generate various files for simulation or display.

set spicefile = *path*/*file*.**cdl**
: Indicates the path to the file containing SPICE subcircuits of the standard cells. This is used to create a SPICE netlist of the design. File may be in SPICE or CDL format.

set gdsfile = *path*/*file*.**gds**
: Indicates the path to the GDS mask-layer definition of the standard cells. This is only needed for full extraction and transistor-level verification.

set techfile = *path*/*file*.**tech**
: Indicates the path to the techfile for the Magic layout editor. Magic is used for displaying the cell or doing extraction from layout, post-synthesis.

The third group gives pointers to qflow on various names of cells to prefer for use in different situations, default global net names, and such.

set tiehi = "*logic1_gate*"
: The name of the cell that outputs a constant logic 1 value.

set tiehipin_out = "*logic1_pin_name*"
: The name of the output pin of the logic 1 cell.

set tielo = "*logic0_gate*"
: The name of the cell that outputs a constant logic 0 value.

set tielopin_out = "*logic0_pin_name*"
: The name of the output pin of the logic 0 cell.

set fillcell = "*fill_cell_name*"
: The name of the cell to use as a fill cell. This may be a decap cell, or just an empty placeholder with power and ground rails. If there are multiple filler cells, specify the common part of the cell name.

set gndnet = "*ground_net_name*"
: The preferred name of the standard cell ground net.

set vddnet = "*power_net_name*"
: The preferred name of the standard cell power net.

set separator = "*separator_string*"
: The character or characters that separate a cell name in the standard cell set from (usually) a number or string indicating the cell's drive strength. For example, if there are buffer cells "BUFX2", "BUFX4", etc., then the base cell name is "BUF" and the separator is "X". Occasionally standard cell sets don't use a separator character, in which case this is an empty string.

set bufcell = "*buffer_cell_name*"
: The name of the cell to use for buffering fanout trees. Standard cell sets often have normal, balanced, and/or

clock buffers. Usually a clock buffer will be most appropriate here.

Some additional, similar variables are related to earlier versions of qflow and are no longer needed. These include the pin names to cells (qflow will now look them up in the liberty format file), nand, nor, invert, and flip-flop cell names (not needed).

The last group is a list of default values that will be copied into the appropriate variables in the **project_vars.sh** file in the project. Any variable that shows up in **project_vars.sh** (see "User Defined Options", above) may be given a default in the technology .sh file by adding "**_default**" to the end of the variable name. Most of the **project_vars.sh** variables are strictly for the user to choose appropriately for each project. Sometimes, however, there are preferred definitions that are technology-dependent or site-dependent. The most likely variables to need such defaults in the technology .sh file are listed below.

set fanout_options_default = "*default_string***"**
Best for setting the "-l" and "-c" options for typical load and latency. This is usually process-feature-size dependent.

set via_use_default = "*default_string***"**
Best for limiting which vias to allow qrouter to use for routing the design.

set via_stacks_default = "*default_string***"**
For technologies requiring a minimum metal area that is larger than a single contact, the via stacks need to be limited to 2 to prevent minimum area violations.

set route_layers_default = "*default_string***"**
A typical site definition may be not to use the top layer of metal for routing but reserve it for the power supply.

set vesta_options_default = "*default_string***"**
A typical site definition may be to specify the long format.

.lib (Liberty) file
The **.lib** file contains detailed timing information for all standard cells, as well as information about the standard cell function, and a number of other properties. This is an open standards format and is usually included with standard cell libraries. It is required when using the **yosys** synthesis frontend with qflow, and it is required for the qflow static timing analysis (vesta).

.genlib file (qflow version 1.0 only)
The .genlib file is deprecated with the use of the **yosys** synthesis frontend, since yosys reads a liberty format timing file and automatically generates a **.genlib** file for use in calling **abc** for combinatorial optimization.
A **.genlib** file is needed in the tech library directory only for use with the optional Odin-II/ABC synthesis frontend.

The genlib file tells ABC what standard cells to use for synthesis, and provides a functional description of each. Genlib files for ABC are difficult to find, but easy to create. It just takes a little time to work through the documentation of a standard cell library. The format describes the logic function and physical size of each cell. There's a lot of timing information in the file, too, but left with more or less random values, ABC will still generate a reasonably good synthesis.

Qflow comes with a tool called "**liberty2tech**", which is compiled and placed in the qflow binaries install target directory. If the standard-cell vendor supplies a timing file in the Liberty text format, then this tool will generate a genlib file and a gate.cfg file (see below) automatically from the contents of this file. For standard cell sets with many entries, this tool avoids a lot of hand-editing. Note that the liberty2tech tool will generally produce more cells than are needed by the synthesis flow. Remove any entries that are not required. Note that the ABC mapper does NOT like to see DFF entries in the file. Flip-flops are inserted by replacement into the netlist file after ABC has run.

Usage:

liberty2tech *liberty_file genlib_file gate_cfg_file* [*pattern*]

where:

liberty_file
: is the name of the Liberty format file

genlib_file
: is the name of the output genlib file

gate_cfg_file
: should be "**gate.cfg**" (output file)

pattern
: determines what cells go into the genlib file. Pattern matching is exact except for characters "^" indicating beginning-of-string, and "$" indicating end-of-string. For example, if you want all cells ending with "X1", the pattern string would be "X1$".

As noted, Qflow version 1.1 does not use the **.genlib** file.

gate.cfg file (qflow version 1.0 only)

This file is used by blifFanout to figure out what variants of standard cells exist with what drive strength. To perform the load balancing, it is necessary for this file to record the propagation delay from input to output per fF of load, and the input capacitance of each of its inputs. Usually this information is provided in documentation by the standard cell vendor.

See also comments above about the use of the **liberty2tech** tool for automatically generating this file.

As noted, Qflow version 1.1 does not use the **gate.cfg** file.

.super file (qflow version 1.0 only)

This file is also deprecated when used with the **yosys** synthesis frontend (see **.genlib** files, above). This file is an auxilliary file used by ABC. ABC is able to create this file itself, so you don't need to. However, ABC is not very smart about where to put the file when it creates it. To get this file, run ABC manually. First, run the synthesis flow through the "synthesis" step. ABC will fail. However, in the first synthesis step, Odin-II will produce a ".blif" file. Copy this .blif file to the tech directory. Then run ABC and give it the following commands:

read_blif *filename***.blif**
read_library *techname***.genlib**
map

ABC will generate the "*techname***.super**" file and write it into the tech directory. Delete the ".blif" file, and you're done.

As noted, Qflow version 1.1 does not use the **.super** file.

.par file

This file contains a set of parameters passed to graywolf, which itself runs numerous sub-programs and needs a complicated setup. Mostly this file just tells graywolf basic information about the number of metal layers used for routing. The file is copied from the technology directory to the layout directory, where it can be customized for a particular synthesis run.

LEF file(s)

The LEF file contains technology information useful to the placement and routing tools. Its primary function is to provide a description of each cell in the standard cell set, including the cell boundary and position and geometry of pins and other routing obstructions, but without any functional description, or details of transistors (or anything that is not on a routing layer). Another important function is to specify routing layer width, pitch, direction, and minimum separation information for use by the detail router. These two functions are sometimes found in the same LEF file, and sometimes they are separated into more than one file. Qflow accepts any number of LEF files, which are specified in the technology script (see "Main variable definition file", above). Often a LEF file will be provided with a standard cell set. If not, the layout tool Magic can generate a LEF macro library from a GDS file of the standard cell set using the "**lef write**" or "**lef writeall**" commands. See the documentation for Magic for details.

Standard cell SPICE file

This is the SPICE library file containing a ".subckt" entry for each of the standard cells.

magic techfile (**.tech** file)

If Magic is to be used for circuit extraction, then the technology file needs to be specified, preferably with GDS input/output sections, DRC rules, device extraction information, and LEF input/output information, all corresponding to the technology used by the standard cells.

.magicrc file

In most cases, a few lines of commands suffices to tell Magic what technology file to load on startup, and additionally (optionally) refine the grid spacing, turn DRC on or off, etc.

Standard cell GDS file

This is the physical layout library containing the GDS cell view for every standard cell in the technology.

.prm file

This is a simple parameter file for use by IRSIM which tells IRSIM very basic modeling information about transistors in the process so that IRSIM can make a decent approximation of the behavior of the transistor-level circuit in the digital domain. The information includes gate capacitance, and resistance of devices when active or saturated.

Standard cell verilog file

This is the functional simulation library containing the verilog modules for every standard cell in the technology. It is not mandatory for qflow, but can be used to simulate the gate-level netlist using a verilog simulator such as Icarus verilog.

email: tim@opencircuitdesign.com

Last updated: November 1, 2019 at 10:13pm