HACKNET : SPICE Netlist Generator

A netlist generator manual

David Fang

This manual describes the usage and operation of HACKT's `hacknet` spice netlist generator.
This document can also be found online at http://www.csl.cornell.edu/~fang/hackt/hacknet.
The main project home page is http://www.csl.cornell.edu/~fang/hackt/.

Short Contents

1 Introduction 1
2 Usage 3
Concept Index 9

Table of Contents

1 Introduction 1

2 Usage 3
2.1 Option Summary 3
2.2 Configuration Options 4

Concept Index 9

1 Introduction

`hacknet` is a SPICE netlist generator for the HACKT suite. The input is a HAC file containing production rules, usually connected through instance hierarchy. The output is a hierarchical netlist with devices, subcircuits, and instances. The output should be suitable for SPICE-like circuit simulators.

How it works...

2 Usage

This chapter describes `hacknet`'s command-line options, and configuration file options.

Usage: '`hacknet [options] obj-file`'

The resulting netlist is printed to '`standard-out`', so it is common practice to redirect it to a file. Diagnostic messages will appear in '`standard-error`'.

2.1 Option Summary

For options that take an argument, the space between the flag and the argument is optional.

`-c` *file* [User Option]
: Parse configuration options from *file*. Options are of the form `key=values` with no space characters separating the `=`. Values may be singleton, comma-separated, or omitted. The same options can also be passed in through the command-line via '`-f`'. This option is repeatable and cumulative. See Section 2.2 [Configuration Options], page 4.

`-C` *file* [User Option]
: Backwards compatibility option. Parse configuration options from *file*. Options are of the form `type key value`. *type* may be '`int`', '`real`', or '`string`'. Values are only singleton. The same options can also be passed in through the command-line via '`-F`'. This option is repeatable and cumulative. See Section 2.2 [Configuration Options], page 4.

`-d` [User Option]
: Print the values of all configuration values to '`stdout`' and exits.

`-f` *options...* [User Option]
: Parse configuration options from *options*. Options are of the same key-value format as in the configuration file, see option '`-c`'. Options are space-separated instead of newline-separated. This option is repeatable and cumulative. See Section 2.2 [Configuration Options], page 4.

`-F` *option* [User Option]
: Backwards compatibility option. Parse configuration options from *options*. Options are of the same key-value format as in the configuration file, see option '`-C`'. Unlike '`-f`' option, only one parameter can be specified at a time. This option is repeatable and cumulative. See Section 2.2 [Configuration Options], page 4.

`-h` [User Option]
: Help. Print usage and exit.

`-H` [User Option]
: Describe all configuration options with default values and exit. See Section 2.2 [Configuration Options], page 4. See also the installed documentation for '`hacknet.info,html,pdf`'.

`-I` *path* [User Option]
: Append *path* to the list of paths to search for including and referencing other config files. See also the '`config_path`' configuration option.

`-t` *type* [User Option]

Instead of using the top-level instances in the source file, instantiate one instance of the named *type*, propagating its ports as top-level globals. In other words, use the referenced type as the top-level scope, ignoring the source's top-level instances. Convenient takes place of copy-propagating a single instance's ports.

2.2 Configuration Options

There are two ways to pass configuration options to `hacknet`. One is through the '`-f`' option on the command-line, the other way is to pass them in through a configuration file with the '`-c`' option. The option value specifications share the same syntax: *key=values* where *values* can be blank, a single value, or a comma-separated list of values. A key-value specifier is not permitted to have spaces in the string! The values themselves cannot contain comma characters.

In a configuration file, blank lines are ignored, as well as lines that begin with `#` (pound). Where boolean values are expected, pass 0 for *false*, or 1 for *true*.

The following parameters are used to manage distributed configuration files. This allows one to create incrementally different configurations.

`config_path` *paths* [User Option]

Append to list of paths for searching for configuration files, exactly like the '`-I`' command-line option. Reminder: paths are comma-separated.

`config_file` *files* [User Option]
`config_file_compat` *files* [User Option]

Import other configuration file(s), exactly like the '`-c`' and '`-C`' command-line options. File are searched using the configuration search path. The '`_compat`' variation processes old-style configuration files.

The following parameters affect emitted device sizes and units.

`lambda` (*real*) [User Option]

Technology-dependent scaling factor for device lengths and widths, the multiplier factor applied to lengths and widths specified in PRS. Default: 1.0

`length_unit` (*string*) [User Option]

Suffix-string to append to emitted length and width parameters. Can be a unit such as "u" or "n", or exponent such as "e-6" or "e-9". Default: u (micron)

`area_unit` (*string*) [User Option]

Suffix-string to append to emitted area valiues. Can be a unit such as "p" (for pico), or exponent such as "e-6" or "e-12". **Alert:** this must be set consistently with respect to *length_unit*. Default: p (pico, micron-squared)

`std_n_width` (*real*) [User Option]

Default width (in lambda) for NFETs used in logic, where unspecified.

`std_p_width` (*real*) [User Option]

Default width (in lambda) for PFETs used in logic, where unspecified.

`std_n_length` (*real*) [User Option]
Default length (in lambda) for NFETs used in logic, where unspecified.

`std_p_length` (*real*) [User Option]
Default length (in lambda) for PFETs used in logic, where unspecified.

`stat_n_width` (*real*) [User Option]
Default width (in lambda) for NFETs used in keepers (staticizers), where unspecified.

`stat_p_width` (*real*) [User Option]
Default width (in lambda) for PFETs used in keepers (staticizers), where unspecified.

`stat_n_length` (*real*) [User Option]
Default length (in lambda) for NFETs used in keepers (staticizers), where unspecified.

`stat_p_length` (*real*) [User Option]
Default length (in lambda) for PFETs used in keepers (staticizers), where unspecified.

`min_width` (*real*) [User Option]
Minimum transistor width in lambda.

`min_length` (*real*) [User Option]
Minimum transistor length in lambda.

`max_p_width` (*real*) [User Option]
Maximum PFET width.

`max_n_width` (*real*) [User Option]
Maximum NFET width.

The following options are related to transistor parasitics.

`emit_parasitics` (*bool*) [User Option]
If set to 1, include source and drain area and perimeter parameters for every transistor Default: 0

`fet_diff_overhang` (*real*) [User Option]
When computing parasitics, this is the length of diffusion overhang past the end of the drawn transistor, in lambda. Default: 6.0

`fet_spacing_diffonly` (*real*) [User Option]
When computing parasitics, this is the length of diffusion between adjoining transistors, in lambda. Default: 4.0

`hacknet` provides several options for formatting the emitted output (because not all SPICEs are created alike).

`emit_top` (*bool*) [User Option]
If set to 1, include the top-level instances in the netlist output. Setting this to 0 is useful for producing a library of subcircuit definitions for every type that was instantiated, recursively w.r.t dependencies. Default: 1

`nested_subcircuits` (*bool*) [User Option]
: If this option is set to 1, then emit local subcircuits as nested definitions within their used definitions. Not sure which variants of SPICE support this. Default: 0

`empty_subcircuits` (*bool*) [User Option]
: If this option is set to 1, then emit empty subcircuits, i.e. subcircuits with no devices. Probably want to force unused port nodes to be emitted in empty subcircuit definitions, option '`unused_ports`'. Default: 0

`unused_ports` (*bool*) [User Option]
: If this option is set to 1, then consider all ports used even if they are unconnected, for the purposes of emitting port lists. This is useful '`empty_subcircuits`', which would result in subcircuits with no ports. Default: 0

`pre_line_continue` (*string*) [User Option]
: String to print before emitting a continued line. Default: (none)

`post_line_continue` (*string*) [User Option]
: String to print after emitting a continued line. Default: `+`

The following options are used for name mangling type-names and instance-names. By default, no mangling is done to keep the output (more) human-readable.

`mangle_underscore` (*string*) [User Option]
: Substitute the '`_`' (underscore) character with another string, which may contain more underscores. **Alert:** It is essential to set this appropriately if underscores are to be used in other mangling replacement strings.

`mangle_process_member_separator` (*char*) [User Option]
: String used to separate members of process instances. e.g., the '`.`' in 'a.b' usually denotes that b is a member of typeof(a). Default: .

`mangle_struct_member_separator` (*char*) [User Option]
: String used to separate members of datatype and channel instances. Default: .

`mangle_array_index_open` (*string*) [User Option]
: Mangle the '`[`' character with a replacement string.

`mangle_array_index_close` (*string*) [User Option]
: Mangle the '`]`' character with a replacement string.

`mangle_template_open` (*string*) [User Option]
: Mangle the '`<`' character with a replacement string.

`mangle_template_close` (*string*) [User Option]
: Mangle the '`>`' character with a replacement string.

`mangle_template_empty` (*string*) [User Option]
: Mangle the '`<>`' sequence with a replacement string. This is applied *before* `<` and `>` are mangled.

`mangle_parameter_separator` (*string*) [User Option]
Mangle the '`,`' character with a replacement string.

`mangle_parameter_group_open` (*string*) [User Option]
Mangle the '`{`' character with a replacement string.

`mangle_parameter_group_close` (*string*) [User Option]
Mangle the '`}`' character with a replacement string.

`mangle_scope` (*string*) [User Option]
Mangle the '`::`' sequence with a replacement string. This is applied *before* `:` is mangled.

`mangle_colon` (*string*) [User Option]
Mangle the '`:`' character with a replacement string.

`mangle_internal_at` (*string*) [User Option]
Mangle the '`@`' character with a replacement string.

`mangle_auxiliary_pound` (*string*) [User Option]
Mangle the '`#`' character with a replacement string.

Concept Index

C

command-line . . . 3
configuration . . . 3, 4

F

flags . . . 3
formatting . . . 5

I

introduction . . . 1

L

lambda . . . 4

M

mangling . . . 6

N

name-mangling . . . 6

O

options . . . 3

P

parasitics . . . 5

S

SPICE-formatting . . . 5

U

usage . . . 3