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

Table of Contents

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 Tutorial

This chapter demonstrates the basic usage of `hacknet` for generating SPICE-netlists. A complete description of features and options follows in the next chapters.

2.1 Top-level circuits

Write the following HAC file, '`inv-top.hac`':

```
bool x, y;
prs {
  x => y-
}
```

Compile the source to an object file:

```
$ haco inv-top.hac inv-top.haco
$ haccreate inv-top.haco inv-top.haco-c
```

Generate a netlist:

```
$ hacknet inv-top.haco-c > inv-top.spice
```

The resulting output should look like:

```
My:dn:0 !GND x y !GND nch W=5u L=2u
My:up:0 !Vdd x y !Vdd pch W=5u L=2u
```

The resulting netlist produces a set of transistors at the top-level. The `!Vdd` and `!GND` nodes are implicit power supplies. The default device types are `nch` for NFETs and `pch` for PFETs. Default widths and lengths were chosen because none were specified.

2.2 Subcircuits

Process definitions are emitted as subcircuit definitions, and can be instantiated with the SPICE `X` card.

Write the following HAC file, '`inv-def.hac`':

```
defproc inv(bool x, y) {
prs {
  x => y-
}
}
```

Compile the source to an object file as before. Since there are no top-level instances, you'll need to tell `hacknet` what type to emit as the top subcircuit.

```
$ hacknet -T 'inv' inv-def.haco-c
```

should produce:

```
.subckt inv<> !GND !Vdd x y
My:dn:0 !GND x y !GND nch W=5u L=2u
My:up:0 !Vdd x y !Vdd pch W=5u L=2u
.ends
```

Use of single-quotes around the type argument is encouraged (in fact, only required for template parameters to protect the `<>` characters from being interpreted by the shell).

Finally, entire hierarchal netlists can be produced by instantiating subcircuit definitions. Write the following file '`invs.hac`', and compile it into '`invs.haco-c`':

```
import "inv-def.hac";

// pair of inverters
defproc foo(bool a, b, c) {
inv p(a, b), q(b, c);
}

foo bar;
```

Run `hacknet`:

```
$ hacknet invs.haco-c > invs.spice
```

to produce the following hierarchical netlist.

```
.subckt inv<> !GND !Vdd x y
My:dn:0 !GND x y !GND nch W=5u L=2u
My:up:0 !Vdd x y !Vdd pch W=5u L=2u
.ends

.subckt foo<> !GND !Vdd a b c
xp !GND !Vdd a b inv<>
xq !GND !Vdd b c inv<>
.ends

xbar !GND !Vdd bar.a bar.b bar.c foo<>
```

In this example, there is a top-level instance of type `foo` named `bar`.

2.3 Configuration help

A quick way to list all of the known configuration options with their default values is:

```
$ hacknet -d
```

The output can be pasted into a file for modification. To use a configuration file, pass the '`-c`' option:

```
$ hacknet -c my.hacknet-conf inv.haco-c > inv.spice
```

3 Language Features

Some features of the HAC language were added specifically for the sake of netlist generation. We briefly describe them here, but the reader is referred to the full HAC language documentation, specifically chapters "Connections" and "PRS".

Recall that every process definition has two implicit ports for the power supply, `!GND` and `!Vdd`. When unspecified, the default behavior automatically connects transistors and subcircuits to these *primary supplies*, hierarchically and recursively. There are two places where we can override these supplies.

3.1 PRS Supply Override

To override the supplies use to drive production rules (pull-up and pull-down) and pass-gates, we can write:

```
prs <myVdd> {
... // These rules use myVdd to pull-up, and !GND for pull-down
}
prs <myVdd, myGND> {
... // Use myVdd to pull-up, myGND to pull-down
}
prs < , myGND> {
... // Use !Vdd to pull-up, myGND to pull-down
}
```

Rules that balong to a supply domain will use the specified supply node for the drain of the foot transistor(s), and for the substrate (body) terminal.

```
prs <myVdd,myGND> {
  x & y -> z-
}
// begets:
M0 myGND x #0 myGND nch W=5u L=2u
M1 #0 y z myGND nch W=5u L=2u
```

Normally, single-supply domain circuits need not be written with explicit production rule supply overrides. Instead, they can be connected externally to different supplies when instantiated, which is described in the next section.

3.2 Instance Supply Override

All processes have implicit primary supply ports `!Vdd` and `!GND`, that are accessed differently than normal ports. By default, a process instance will connect its primary supply ports to those of its immediate parent process. To specify a different supply, one can write:

```
bool myVdd, myGND, x, y;
inverter foo;
foo $(myGND, myVdd);
foo (x, y);

inverter yoyo(x, y);
```

```
yoyo $(myGND, myVdd);

inverter bar $(myGND, myVdd);
bar (x, y);

inverter bq $(myGND, myVdd) (x, y);
```

Either supply port may be ommitted to use the corresponding default primary supply. **Alert:** note that the supply ordering for instances is `GND, Vdd`, whereas for production rules, it is `Vdd, GND`.

4 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`'.

4.1 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 4.2 [Configuration Options], page 8.

`-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 4.2 [Configuration Options], page 8.

`-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 4.2 [Configuration Options], page 8.

`-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 4.2 [Configuration Options], page 8.

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

`-H` [User Option]
: Describe all configuration options with default values and exit. See Section 4.2 [Configuration Options], page 8. 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. See also the '`-T`' option.

`-T` *type* [User Option]
: Instead of using the top-level instances in the source file, instantiate one lone instance of the named *type*, at the top-level, ignoring the source's top-level instances. Unlike the '`-t`' option, the contents of named type *type* do not get unrolled directly into the top-level scope. This instance's ports will be unconnected, and the instance's name is unspecified (because you shouldn't care!). This option also implies '`emit_top=0`'. Current limitation: can only specify one type for now.

`-v` [User Option]
: Print version and build information and exit.

4.2 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 values. 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)

`capacitance_unit` (*string*) [User Option]
Suffix-string to capacitance values. Can be a unit such as `"p"` (for pico), or exponent such as `"e-6"` or `"e-12"`. Default: (blank)

`resistance_unit` (*string*) [User Option]
Suffix-string to resistance values. Default: (blank)

`inductance_unit` (*string*) [User Option]
Suffix-string to inductance values. Default: (blank)

`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.

`nfet_svt` (*string*) [User Option]
`nfet_lvt` (*string*) [User Option]
`nfet_hvt` (*string*) [User Option]
`pfet_svt` (*string*) [User Option]
`pfet_lvt` (*string*) [User Option]
`pfet_hvt` (*string*) [User Option]
Override the default device type names for NFETs and PFETS along with their non-standard threshold voltage variants.

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_perimeter_gate_edge` (*bool*) [User Option]
When estimating parasitics, when this is true, include the gate edge of the rectangle when estimating perimeter values for parasitic capacitances. Including the gate edge length results in increased capacitance, which can be pessimistic. Default: 1

`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. Typically, this value comes from the minimum diffusion overhang rule in your process's DRC rule deck. Default: 6.0

`fet_spacing_diffonly` (*real*) [User Option]
When computing parasitics, this is the length of diffusion between adjoining transistors, in lambda. For the purpose of computing parasitic capacitances on shared nodes, this value should actually be *half* of the minimum spacing between stacked transistors according to your technology's DRC rule deck. This effectively assigns half of the diffusion area to the device on either side. Default: 4.0

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

`print` (*bool*) [User Option]
If set to 0, suppress normal netlist output. Default: 1

`output_format` *style* [User Option]
Pseudo-option: preset bundle of options for formatting output. Valid choices of *style* are: `spice`, `spectre`.

`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

`subcircuit_definition_style` *style* [User Option]
- `spice` - `.subckt CKT ... .ends`
- `spectre` - `subckt CKT ... ends CKT`

`instance_port_style` *style* [User Option]
- `spice` - `inst ports ... type`
- `spectre` - `inst (ports ...) type`

`node_ports` (*bool*) [User Option]
: If set to 1, include bools (nodes, wires) in port lists for subcircuit definitions and instances. This is enabled with '`output_format=spice,spectre`'. Default: 1

`struct_ports` (*bool*) [User Option]
: If set to 1, include user-defined structs and channels in port lists for subcircuit definitions and instances. This is enabled with '`output_format=verilog`'. 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

`named_port_connections` (*bool*) [User Option]
: If set to 1, emit instances' port connections using `.port(local)` syntax, otherwise emit port connections positionally. Most spice-like formats support only positional ports, but the Verilog language supports both. Named port connections makes the output more verbose, but less prone to positional connection errors. Default: 0

`prefer_port_aliases` (*bool*) [User Option]
: Instead of using default heuristic for choosing shallowest or shortest canonical name, prefer any equivalent port name, if applicable. This can make netlists and simulation results easier to grok. Default: 0

`preferred_names` (*strings*) [User Option]
: Declare local node names that should always take precedence over canonical shortest-names and preferred-port-aliases. This is useful for supply nodes that appear in structures, that are passed around globally. The names passed to this list should be pre-mangled, using the default struct-member-separator, `$`. Default: (blank)

`emit_port_summary` (*bool*) [User Option]
: If set to 1, prints out node port information, including signal direction. Default: 0

`emit_node_aliases` (*bool*) [User Option]
: If set to 1, print a sets of node aliases (equivalent names). Default: 0

`emit_node_caps` (*bool*) [User Option]
: If set to 1, print a cumulative capaticance components for every local node. Components include: total diffusion perimeter length, total diffusion area, total gate area, and total wire area. Default: 0

`emit_node_terminals` (*bool*) [User Option]
: If set to 1, print for every node, the terminals to which it connects on all devices. This is useful for debugging the redundant graphical representation used for graph traversals and analysis. Default: 0

`transistor_prefix` (*string*) [User Option]
: String to print at the beginning of a transistor instance. For spice, this is usually the `M` card. Default: M

`subckt_instance_prefix` (*string*) [User Option]
: String to print at the beginning of a subcircuit instance. For spice, this is usually the `x` card. Default: x

`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: `+`

`comment_prefix` (*string*) [User Option]
: String to print before whole-line comments. Default: `"* "`

`auto_wrap_length` (*int*) [User Option]
: If set to `>` 0, automatically wrap lines that would be longer than the given length. This is useful when there are external limits to line length that need to be accounted for. Default: 0 (no-wrap)

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 (designating named internal node) with a replacement string.

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

`mangle_implicit_bang` (*string*) [User Option]
Mangle the '`!`' character (designating implicit supply node) with a replacement string.

`mangle_double_quote` (*string*) [User Option]
Mangle the '`"`' character (from template parameter strings) with a replacement string.

`mangle_escaped_instance_identifiers` (*bool*) [User Option]
Verilog-style escaped identifiers. Wrap all instance names with slash and a space, e.g. `\foo[0]` . This is enabled for '`output_style=verilog`' by default. Default: 0

`mangle_escaped_type_identifiers` (*bool*) [User Option]
Verilog-style escaped identifiers. Wrap all type names with slash and a space, e.g. `\foo[0]` . This is disabled for '`output_style=verilog`' by default. Default: 0

`emit_mangle_map` (*bool*) [User Option]
If set to 1, print a list of manglings in comments. Default: 0

`reserved_names` (*strings*) [User Option]
Amend the set of names that should be rejected from normal use in the netlist because they have special meaning to other back-ends. Issuing a blank value will clear out all previous values. Comparison is done using the *post-mangled* names. Default: (blank)

Option policies control the behavior of certain diagnostics. Policy options accept the values "ignore", "warn", or "error".

`unknown_option` (*string*) [User Option]
: Set error-handling policy when encountering unknown configuration options. Default: warn

`internal_node_supply_mismatch` (*string*) [User Option]
: Set error-handling policy when an internal node is used in a different supply domain than the one it was defined in. Default: warn

`undriven_node` (*string*) [User Option]
: Set error handling policy for subcircuits that find nodes that are neither used (connected to source, gate terminals) but not driven (connected to drain terminal) nor coming from subcircuit port. This check is always skipped for top-level circuits. Default: warn

`case_collision` (*string*) [User Option]
: Set error handling policy for names that collide when case is ignored (insensitive). For this policy warnings are promoted to errors; use ignore to completely disable. If the value is anything but `ignore`, then the set of reserved names will also be transformed when detecting collisions. Default: warn

`non_CMOS_precharge` (*string*) [User Option]
: Set the error handling policy for precharge expressions that are written non-CMOS, i.e. not fully-restoring. (for example, using an NMOS expression to pull-up an internal node in an NMOS stack) Such nets will only charge or precharge to 1 Vt from the supply rail. Be very careful with this option, it allows you to write nonsense like '`a &{+x & ~y} b & c -> o-`'. Default: warn

`below_min_width` (*string*) [User Option]
: Set the error handling policy when a transistor width is clamped to the minimum width (from configuration). Default: warn

5 Algorithms

This chapter describes some of the internal algorithms for netlist generation by `hacknet`. Hackers may find this informative for extending the functionality of netlist generation.

Concept Index