1 Netlist grammar

Devices in SUGAR are described by netlists analogous to those used in packages like SPICE (for circuit simulation) or FEAP (for structural simulation). This document describes the syntax for netlists in SUGAR 2.0, both informally and in terms of a formal grammar.

1.1 Lexical notes

SUGAR 2.0 netlists are "free form"; that is, white space characters like tabs and carriage returns are not significant. For example, the netlists

```
uses mumps.net
beam2d p1
  [A B]
  [l = 100u
   w = 100u
  ]
```

and

```
uses mumps.net
beam2d p1 [A B] [l=100u w=100u]
```

are equivalent.

A comment in a netlist begins with `%` and extends to the end of the line.

A SUGAR identifier (like a C identifier) consists of a letter followed by a string of letters, numbers, and underscores. The keywords `subnet`, `param`, `process`, and `for` are reserved, and cannot be used as identifiers.

All integers are base 10, and are written as one would expect. Floating point numbers consist of a decimal number optionally followed by an exponent; for example 1e-6 or 1.5e6 are valid floating point constants. Floating point numbers can also have a standard suffix to indicate a power of 10. The standard suffixes are:

a	atto	$10^{-}18$
f	femto	$10^{-}15$
p	pico	$10^{-}12$
n	nano	$10^{-}9$
u	micro	$10^{-}6$
m	milli	$10^{-}3$
c	centi	$10^{-}2$
d	deci	$10^{-}1$
h	hecto	10^{2}
k	kilo	10^{3}
M	mega	10^{6}
G	giga	10^{9}
T	tera	$10^{1}2$
P	peta	$10^{1}5$
E	exa	$10^{1}8$

Since SUGAR uses MKS units, these suffixes are particularly useful for specifying lengths (100u is 100 microns), pressures (150G is 150 gigapascals), etc. Suffixes may be used with exponents; for example, 1.5e-3u is a valid float constant equal to 1.5n.

1.2 uses statements

In order to make it cleaner to define common files which include process parameters, libraries of often-used subnets, etc., SUGAR netlists can have `uses` statements. A `uses` statement has the form

```
uses filename
```

For example, to use the data for MUMPS process layers defined in `mumps.net`, you might write

```
uses mumps.net
```

uses statements differ from C-style `#include` statements in that a file can only be included once via uses. For example, if the file `subnets.net` started with

```
uses mumps.net
```

and a test netlist called `test.net` started with

```
uses subnets.net
uses mumps.net
```

then `test.net` would only include `mumps.net` once, and would not complain about the contents of `mumps.net` being defined multiple times.

1.3 Element lines

The basic unit of a SUGAR netlist is an element line. For example,

```
crossbeam beam2d p1 [A B] [l=100u w=2u]
```

is an element line describing a beam. This line consists of several fields:

- The first field, which is optional, is the name of the element; in this case, the element is named `crossbeam`.
- The second field is the name of the model for the element; in this case, it is the two-dimensional beam model `beam2d`. There are models for beams, anchors, electrical devices, etc.; a complete list of models, along with information on how to build new models, can be found in other SUGAR documentation.

- The third field is the name of the process parameter structure; in this case the beam is fabricated in the first layer of polysilicon in a MUMPS process, named `p1`. By specifying the process layer `p1`, a user informs the model function of common material parameters such as the Young's modulus for polysilicon and the thickness of the deposited layer. For models which require no process information, such as models for external forces, the process field may be set to `*`.
- After the process field comes a list of nodes, surrounded in brackets. In this case, the specified beam connects nodes `A` and `B`. Elements are connected together by sharing a common node. For instance, to attach a wider 100 micron beam to the `B` end of the beam above, we might write

  ```
  beam2d p1 [B C] [l=100u w=5u]
  ```

- After the list of nodes comes a list of model parameters. In the example above, the model parameters consisted of the length and width of a beam; other models will require other parameters. A parameter specification always has the form `identifier = expression`.

1.4 Expressions

Expressions in SUGAR are much like expressions in languages like C or Pascal. A SUGAR expression consists of sums, products, differences, quotients, and functions of integers, floating point numbers, and variables. The order of operations and use of parentheses is standard. More sophisticated operators, such as equality, inequality tests, and mod are not provided.

1.5 Parameters and definitions

In order to allow users to experiment with variations on a simulation, or to do parameter sweeps, SUGAR supports named parameters. An example might be

```
param nfingers, l=10u
```

A parameter definition consists of the `param` keyword, followed by a comma-separated list of parameter names and default values. Default values are optional; however, it is an error to use the netlist without setting any parameters without defaults. Parameter defaults may be expressions depending on previously defined variables or parameters.

SUGAR netlists may also include definitions, such as

```
long_length = 200u
short_length = 100u
avg_length = (long_length + short_length)/2
```

Netlist variables are scoped, so that a definition made inside a subnet (see below) will not affect top-level element statements.

1.6 Process parameter structures

Physical parameters associated with a particular layer of a particular material are process parameters. An example of the baseline process information for the polysilicon layers in MUMPS (`default`) is provided below

```
process default = [
    Poisson = 0.3                 %Poisson's Ratio = 0.3
    thermcond = 2.33              %Thermal conductivity Si = 2.33e-6/C
    viscosity = 1.78e-5           %Viscosity (of air) = 1,78e-5
    fluid = 2e-6                  % Between the device and the substrate.
    density = 2300                %Material density = 2300 kg/m^3
    Youngsmodulus = 165e9         %Young's modulus = 1.65e11 N/m^2
    permittivity = 8.854e-12      %permittivity: C^2/(uN.um^2)=(C.s)^2/kg.um^3
    sheetresistance = 20          %Poly-Si sheet resistance [ohm/square]
]
```

In general a process definition has the form `process name = [...]`, where `process` is a keyword, `name` is the name to be given to the process information, and process definitions are given between the square brackets.

Process parameter structures may be derived from other process parameter structures. For example, a 2 micron poly layer named `p1` might be written

```
process p1 : default = [
    h = 2u
]
```

This layer automatically includes all the definitions made in the default process parameter structure.

1.7 Subnets

A subnet is analogous to a SPICE subcircuit, or to a function in C. Subnets provide users with a means to extend the set of available models without leaving SUGAR. An example subnet for a single unit of a serpentine structure taking up area is shown below:

```
subnet serpent [A E] [unitwid=* unitlen=* beamw=2u]
[
  len2 = unitlen/2
  beam2d parent [A b] [l=unitwid w=beamw oz=-90]
  beam2d parent [b c] [l=len2    w=beamw]
  beam2d parent [c d] [l=unitwid w=beamw oz=90]
  beam2d parent [d E] [l=len2    w=beamw]
]
```

Element lines using subnets are invoked in the same manner as element lines using model functions built in Matlab

```
serp1 serpent p1 [x y] [unitwid=10u unitlen=10u]
serpent p1 [y z] [unitwid=10u unitlen=10u w=3u]
```

The `parent` process for a subnet is the process specified in creating an instance of that subnet. In the above example, the `p1` process information would be used for the beams in the serpent subnet.

In general, a subnet definition consists of the keyword `subnet`, a name for the subnet model, a bracketed list of node names, a bracketed list of parameters, and a code block. The code block is enclosed in square brackets, and may include definitions, element lines, and array structures (see below).

Sometimes it may be necessary to access variables attached to nodes internal to a netlist. For example, in the above example we might be interested in the version of node `b` for subnet instance `serp1`. In the analysis functions, that node would be referred to as `serp1.b`. It would not be valid to refer to node `x` as `serp1.A`, since `x` already has a name defined outside the subnet.

Subnet instances that are not explicitly named, like the second `serpent` element in the example above, are assigned names consisting of `anon` followed by some number. It is possible to use a name like `anon1.b` to refer to the `b` node in the second line, but it is not recommended since the internal naming schemes for anonymous elements are subject to future change.

1.8 Arrays

SUGAR provides syntactic support for arrays of structures. For example, the following code fragment creates a spring composed of twenty of the serpentine units from the subnet example and anchors it at one end:

```
a1 anchor p1 [x(1)] []
for k = 1:20
[
  link(k) serpent p1 [x(k) x(k+1)] [unitwid=10u unitlen=10u]
]
```

Note that both element names and node names may be indexed. It is possible to have names with multiple indices as well (eg `link(i,j)`). The index variable is only valid within the loop body.

The general syntax of a for loop is

```
for index = lowerbound : upperbound
[
  ... code lines ...
]
```

where `index` is the name of the index variable, `lowerbound` is an expression for the lower bound of the loop, and `upperbound` is an espression for the upper bound of a loop.