In the following example of instantiating a voltage-controlled oscillator, the parameters are specified on a named-association basis much as they are for ports.

```
module n(lo_out, rf_in):
electrical lo_out, rf_in;

   //create an instance and set parameters
   vco #(.centerFreq(5000), .convGain(1000)) vco1(lo_out, rf_in);
endmodule
```

Here, the name of the instantiated `vco` module is `vco1`. The `centerFreq` parameter is passed a value of 5000 and the `convGain` parameter is passed a value of 1000. The positional assignment mechanism for ports assigns `lo_out` as the first port and `rf_in` as the second port of `vco1`.

7.2.4 Parameter dependence

A parameter (for example, `gate_cap`) can be defined with an expression containing another parameter (for example, `gate_width` or `gate_length`). Since `gate_cap` depends on the value of `gate_width` and `gate_length`, a modification of `gate_width` or `gate_length` changes the value of `gate_cap`.

Examples:

In the following parameter declaration, an update of `gate_width`, whether by a *defparam* statement or in an instantiation statement for the module which defined these parameters, automatically updates `gate_cap`.

```
parameter
    gate_width = 0.3e-6,
    gate_length = 4.0e-6,
    gate_cap = gate_length * gate_width * ‘COX;
```

7.2.5 Detecting parameter overrides

In some cases, it is important to be able to determine whether a parameter value was obtained from the default value in its declaration statement or if that value was overridden. In such a case, the **$param_given()** function described in Section 10.2 can be used.

7.3 Paramsets

A paramset definition is enclosed between the keywords **paramset** and **endparamset**, as shown in Syntax 7-5. The first identifier following the keyword **paramset** is the name of the paramset being defined. The second identifier will usually be the name of a module with which the paramset is associated. The second identifier can also be the name of a second paramset; a chain of paramsets may be defined in this way, but the last paramset in the chain shall reference a module.

```
paramset_declaration ::=
    {attribute_instance} paramset paramset_identifier module_or_paramset_identifier ;
    paramset_item_declaration {paramset_item_declaration}
    paramset_statement { paramset_statement }
    endparamset
paramset_item_declaration ::=
     {attribute_instance} parameter_declaration
    | {attribute_instance} local_parameter_declaration
    | {attribute_instance} string_parameter_declaration
    | {attribute_instance} local_string_parameter_declaration
    | aliasparam_declaration
    | {attribute_instance} integer_declaration
    | {attribute_instance} real_declaration

paramset_statement ::=
     .module_parameter_identifier = expression ;
    | statement
    | paramset_seq_block

paramset_seq_block ::=
    begin
     { paramset_statement }
    end
```

Syntax 7-5—Syntax for paramset

The paramset itself contains no behavioral code; all of the behavior is determined by the associated module. The restrictions on statements in the paramset are described in Section 7.3.1.

The paramset provides a convenient way to collect parameter values for a particular module, such that an instance need only provide overrides for a smaller number of parameters. A simulator can use this information to optimize data storage for the instances: multiple instances may share a paramset, and the simulator can share storage of parameters of the underlying module. The shared storage of paramsets makes them similar to the SPICE model card. Also like the SPICE model card, paramsets may be overloaded, as described in Section 7.3.2.

A paramset can have a description attribute, which shall be used by the simulator when generating help messages for the paramset.

7.3.1 Paramset statements

The restrictions on statements or assignments allowed in a paramset are similar to the restrictions for analog functions. Specifically, a paramset:

- can use any statements available for conditional execution (see Section 6.1);

- shall not use access functions;
- shall not use contribution statements or event control statements; and
- shall not use named blocks.

The special syntax

```
.module_parameter_identifier = expression ;
```

is used to assign values to the parameters of the associated module. The expression on the right-hand side can be composed of numbers and parameters, local parameters, and variables declared in the paramset, as well as out-of-module references to parameters of a different module.

7.3.2 Paramset overloading

Paramset identifiers need not be unique: multiple paramsets can be declared using the same *paramset_*identifier, and they may refer to different modules. During elaboration, the simulator shall choose an appropriate paramset from the set that shares a given name for every instance that references that name.

When choosing an appropriate paramset, the following rules shall be enforced:

- All parameters overridden on the instance shall be parameters of the paramset
- The parameters of the paramset, with overrides and defaults, shall be all within the allowed ranges.
- The local parameters of the paramset, computed from parameters, shall be within the allowed ranges.

The rules above may not be sufficient for the simulator to pick a unique paramset, in which case the following rules shall be applied in order until a unique paramset has been selected:

- The paramset with the fewest number of un-overridden parameters shall be selected.
- The paramset with the greatest number of local parameters with specified ranges shall be selected.

It shall be an error if there are still more than one applicable paramset for an instance after application of these rules.

7.3.3 Paramset output variables

As with modules, integer or real variables in the paramset that are declared with descriptions are considered output variables; see Section 3.1.1. A few special rules apply to paramset output variables and output variables of modules referenced by a paramset:

- If a paramset output variable has the same name as an output variable of the module, the value of the paramset output variable is the value reported for any instance that uses the paramset.
- If a paramset variable without a description has the same name as an output variable of the module, the output variable of that name shall not be available for instances that use the paramset.
- A paramset output variable's value may be computed from values of any output parameters of the module by using the special syntax
  .*module_output_variable_*identifier
  However, any paramset variable that depends on a module output variable shall not be used in the assignment of module parameters.

7.4 Ports

Ports provide a means of interconnecting instances of modules. For example, if a module A instantiates module B, the ports of module B are associated with either the ports or the internal nets of module A.

7.4.1 Port association

The syntax for a port association is shown in Syntax 7-6.

```
port ::=
     port_expression
   | . port_identifier ( [ port_expression ] )

port_expression ::=
     port_identifier
   | port_identifier [ constant_expression ]
   | port_identifier [ constant_range ]

constant_range ::=
   msb_constant_expression : lsb_constant_expression
```

Syntax 7-6—Syntax for port

The port expression in the port definition can be one of the following:

- a simple net identifier