SMEIL Language Reference

Grammar

⟨*module*⟩ ::= { ⟨*import-stm*⟩ } ⟨*entity*⟩
{ ⟨*entity*⟩ }

⟨*import-stm*⟩ ::= '`import`' ⟨*import-name*⟩ ⟨*qualified-specifier*⟩ '`;`'
| '`from`' ⟨*import-name*⟩
'`import`' ⟨*ident*⟩ { '`,`' ⟨*ident*⟩ } ⟨*qualified-specifier*⟩ '`;`'

⟨*import-name*⟩ ::= ⟨*ident*⟩ { '`.`' ⟨*ident*⟩ }

⟨*qualified-specifier*⟩ ::= { '`as`' ⟨*ident*⟩ }

⟨*entity*⟩ ::= ⟨*network*⟩
| ⟨*process*⟩

⟨*network*⟩ ::= '`network`' ⟨*ident*⟩ '`(`' [⟨*params*⟩] '`)`'
'`{`' ⟨*network-decl*⟩ '`}`'

⟨*process*⟩ ::= ['`sync`' | '`async`'] '`proc`' ⟨*ident*⟩
'`(`' [⟨*params*⟩] '`)`' { ⟨*declaration*⟩ }
'`{`' { ⟨*statement*⟩ } '`}`'

⟨*network-decl*⟩ ::= ⟨*instance*⟩
| ⟨*bus-decl*⟩
| ⟨*const-decl*⟩
| ⟨*gen-decl*⟩

⟨*declaration*⟩ ::= ⟨*var-decl*⟩
| ⟨*const-decl*⟩
| ⟨*bus-decl*⟩
| ⟨*enum-decl*⟩
| ⟨*inst-decl*⟩
| ⟨*gen-decl*⟩

⟨*params*⟩ ::= ⟨*param*⟩ { , ⟨*param*⟩ }

⟨*param*⟩ ::= ['`[`' [⟨*integer*⟩] '`]`'] ⟨*direction*⟩ ⟨*ident*⟩

⟨*direction*⟩ ::= '`in`' (input signal)
| '`out`' (output signal)
| '`const`' (constant input value)

⟨*var-decl*⟩ ::= '`var`' ⟨*ident*⟩ '`:`'
⟨*type-name*⟩ ['`=`' ⟨*expression*⟩] [⟨*range*⟩] '`;`'

⟨*range*⟩ ::= '`range`' ⟨*expression*⟩ '`to`' ⟨*expression*⟩

⟨*enum*⟩ ::= '`enum`' ⟨*ident*⟩
'`{`' ⟨*enum-field*⟩ { '`,`' ⟨*enum-field*⟩ } '`}`' '`;`'

⟨*enum-field*⟩ ::= ⟨*ident*⟩ ['`=`' ⟨*integer*⟩]

⟨*const-decl*⟩ ::= '`const`' ⟨*ident*⟩ '`:`' ⟨*type-name*⟩ '`=`' ⟨*expression*⟩ '`;`'

⟨*bus-decl*⟩ ::= ['`exposed`'] '`bus`' ⟨*ident*⟩
'`{`' ⟨*bus-signal-decls*⟩ '`}`' '`;`'

⟨*bus-signal-decls*⟩ ::= ⟨*bus-signal-decl*⟩ { ⟨*bus-signal-decl*⟩ }

⟨*bus-signal-decl*⟩ ::= ⟨*ident*⟩ '`:`' ⟨*type*⟩ ['`=`' ⟨*expression*⟩] [⟨*range*⟩] '`;`'

⟨*inst-decl*⟩ ::= '`instance`' ⟨*instance-name*⟩ '`of`' ⟨*ident*⟩
'`(`' [⟨*param-map*⟩ { '`,`' ⟨*param-map*⟩ }] '`)`' '`;`'

⟨*instance-name*⟩ ::= ⟨*ident*⟩ '`[`' ⟨*expression*⟩ '`]`' (indexed instance)
| ⟨*ident*⟩ (named instance)
| '`_`' (anonymous instance)

⟨*param-map*⟩ ::= [⟨*ident*⟩ '`:`'] ⟨*expression*⟩

⟨*gen-decl*⟩ ::= '`generate`' ⟨*ident*⟩ '`=`' ⟨*expression*⟩ '`to`' ⟨*expression*⟩
'`{`' { ⟨*network-decl*⟩ } '`}`'

⟨*statement*⟩ ::= ⟨*name*⟩ '`=`' ⟨*expression*⟩ '`;`' (assignment)
| '`if`' '`(`' ⟨*expression*⟩ '`)`' '`{`' { ⟨*statement*⟩ } '`}`'
{ ⟨*elif-block*⟩ } [⟨*else-block*⟩]
| '`for`' ⟨*ident*⟩ '`=`' ⟨*expression*⟩ '`to`' ⟨*expression*⟩
'`{`' { ⟨*statement*⟩ } '`}`'
| '`switch`' ⟨*expression*⟩
'`{`' ⟨*switch-case*⟩ { ⟨*switch-case*⟩ } ['`default`' '`{`' ⟨*statement*⟩
{ ⟨*statement*⟩ } '`}`'] '`}`'
| '`trace`' '`(`' ⟨*format-string*⟩ { '`,`' ⟨*expression*⟩ })'`;`'

| ‘`assert`’ ‘`(`’ ⟨*expression*⟩ [‘`,`’ ⟨*string*⟩] ‘`)`’‘`;`’
| ‘`break`’ ‘`;`’

⟨*switch-case*⟩ ::= ‘`case`’ ⟨*expression*⟩ ‘`{`’ { ⟨*statement*⟩ } ‘`}`’

⟨*elif-block*⟩ ::= ‘`elif ‘(`’ ⟨*expression*⟩ ‘`)`’ ‘`{`’ { ⟨*statement*⟩ } ‘`}`’

⟨*else-block*⟩ ::= ‘`else`’ ‘`{`’ { ⟨*statement*⟩ } ‘`}`’

⟨*format-string*⟩ ::= ‘`"`’ { ⟨*format-string-part*⟩ } ‘`"`’

⟨*format-string-part*⟩ ::= ‘`{}`’ (placeholder string)
| ⟨*string-char*⟩

⟨*expression*⟩ ::= ⟨*name*⟩
| ⟨*literal*⟩
| ⟨*expression*⟩ ⟨*bin-op*⟩ ⟨*expression*⟩
| ⟨*un-op*⟩ ⟨*expression*⟩
| ‘`(`’ ⟨*expression*⟩ ‘`)`’

⟨*bin-op*⟩ ::= ‘`+`’ (addition)
| ‘`-`’ (subtraction)
| ‘`*`’ (multiplication)
| ‘`/`’ (division)
| ‘`%`’ (modulo)
| ‘`==`’ (equal)
| ‘`!=`’ (not equal)
| ‘`<<`’ (shift left)
| ‘`>>`’ (shift right)
| ‘`<`’ (less than)
| ‘`>`’ (greater than)
| ‘`>=`’ (greater than or equal)
| ‘`<=`’ (less than or equal)
| ‘`&`’ (bitwise-and)
| ‘`|`’ (bitwise-or)
| ‘`^`’ (bitwise-xor)
| ‘`&&`’ (logical conjunction)
| ‘`||`’ (logical disjunction)

⟨*un-op*⟩ ::= ‘`-`’ (negation)
| ‘`+`’ (identity)
| ‘`!`’ (logical negation)
| ‘`~`’ (bitwise-not)

⟨*literal*⟩ ::= ⟨*integer*⟩
| ⟨*floating*⟩
| ‘`"`’{ ⟨*string-char*⟩ }‘`"`’ (String literal)

| ‘[’ ⟨*integer*⟩ { ‘,’ ⟨*integer*⟩ } ‘]’ (Array literal)
| ‘`true`’
| ‘`false`’
| ⟨*special-literal*⟩

⟨*special-literal*⟩ ::= ‘`'U`’ (Undefined value)

⟨*type*⟩ ::= ‘`i`’ ⟨*integer*⟩ (signed integer)
| ‘`int`’ (arbitrary-width signed integer)
| ‘`u`’ ⟨*integer*⟩ (unsigned integer)
| ‘`uint`’ (arbitrary-width unsigned integer)
| ‘`f32`’ (single-precision floating point)
| ‘`f64`’ (double-precision floating point)
| ‘`bool`’ (boolean value)
| ‘`[`’ [⟨*expression*⟩] ‘`]`’ ⟨*type*⟩ (array of type)

⟨*ident*⟩ ::= ⟨*letter*⟩ { ⟨*letter*⟩ | ⟨*number*⟩ | ‘`_`’ | ‘`-`’ } (identifier)

⟨*name*⟩ ::= ⟨*ident*⟩
| ⟨*name*⟩ ‘`.`’ ⟨*name*⟩ (hierarchical accessor)
| ⟨*name*⟩ ‘`[`’ ⟨*array-index*⟩ ‘`]`’ (array element access)

⟨*array-index*⟩ :: ‘`*`’ (wildcard)
| ⟨*expression*⟩ (element index)

⟨*integer*⟩ ::= ⟨*number*⟩ { ⟨*number*⟩ } (decimal number)
| ‘`0x`’ ⟨*hex-digit*⟩ { ⟨*hex-digit*⟩ } (hexadecimal number)
| ‘`0o`’ ⟨*octal-digit*⟩ { ⟨*octal-digit*⟩ } (octal number)

⟨*floating*⟩ ::= { ⟨*number*⟩ } ‘`.`’ ⟨*number*⟩ { ⟨*number*⟩ }

⟨*number*⟩ ::= ‘`0`’ - ‘`9`’

⟨*letter*⟩ ::= ‘`a`’ - ‘`z`’
| ‘`A`’ - ‘`Z`’

⟨*hex-digit*⟩ ::= ⟨*number*⟩
| ‘`a`’ - ‘`f`’
| ‘`A`’ - ‘`F`’

⟨*octal-digit*⟩ ::= ‘`0`’ - ’8’

⟨*string-char*⟩ ::= (ISO-8859-1 char with value > 26)

Operator precedence

Precedence	Operators
0	+ - ! ~ (unary)
1	* / %
2	+ -
3	<< >>
4	< > <= >=
5	== !=
6	& ^ |
7	&&
8	||

Keywords

- as
- async
- barrier
- break
- bus
- case
- const
- default
- elif
- else
- enum
- exposed
- for
- from
- func
- generate
- if
- import
- in
- instance
- network
- of
- out
- proc
- range
- return
- switch
- sync
- to
- unique
- var
- where