SMEIL Language Reference

Grammar

⟨*module*⟩	::=	{ ⟨*import-stm*⟩ }
		{ ⟨*module-decl*⟩ }
		{ ⟨*entity*⟩ }
⟨*import-stm*⟩	::=	'import' ⟨*import-name*⟩ [⟨*qualified-specifier*⟩] ';'
	|	'from' ⟨*import-name*⟩
		'import' ⟨*ident*⟩ { ',' ⟨*ident*⟩ } [⟨*qualified-specifier*⟩]
		';'
⟨*import-name*⟩	::=	⟨*ident*⟩ { '.' ⟨*ident*⟩ }
⟨*qualified-specifier*⟩	::=	'as' ⟨*ident*⟩
⟨*module-decl*⟩	::=	⟨*type-def*⟩
	|	⟨*const-decl*⟩
	|	⟨*enum-decl*⟩
	|	⟨*func-decl*⟩
⟨*type-def*⟩	::=	'type' ⟨*ident*⟩ ':'
		⟨*type-name*⟩ (type alias)
		| ⟨*bus-signal-decls*⟩ (bus definition)
		';'
⟨*entity*⟩	::=	⟨*network*⟩
	|	⟨*process*⟩
⟨*network*⟩	::=	'network' ⟨*ident*⟩ '(' [⟨*params*⟩] ')'
		'{' { ⟨*network-decl*⟩ } '}'
⟨*process*⟩	::=	['clocked'] 'proc' ⟨*ident*⟩
		'(' [⟨*params*⟩] ')' { ⟨*process-decl*⟩ }
		'{' { ⟨*statement*⟩ } '}'
⟨*network-decl*⟩	::=	⟨*inst-decl*⟩
	|	⟨*bus-decl*⟩

$$\begin{array}{lll}
& | & \langle \textit{const-decl} \rangle \\
& | & \langle \textit{gen-decl} \rangle \\
& | & \langle \textit{connect-decl} \rangle
\end{array}$$

$$\begin{array}{lll}
\langle \textit{process-decl} \rangle & ::= & \langle \textit{var-decl} \rangle \\
& | & \langle \textit{const-decl} \rangle \\
& | & \langle \textit{bus-decl} \rangle \\
& | & \langle \textit{enum-decl} \rangle \\
& | & \langle \textit{func-decl} \rangle \\
& | & \langle \textit{inst-decl} \rangle \\
& | & \langle \textit{gen-decl} \rangle
\end{array}$$

$$\begin{array}{lll}
\langle \textit{func-decls} \rangle & ::= & \langle \textit{var-decl} \rangle \\
& | & \langle \textit{const-decl} \rangle \\
& | & \langle \textit{enum-decl} \rangle
\end{array}$$

$$\langle \textit{params} \rangle \quad ::= \quad \langle \textit{param} \rangle \ \{ \ , \ \langle \textit{param} \rangle \ \}$$

$$\langle \textit{param} \rangle \quad ::= \quad \langle \textit{direction} \rangle \ \langle \textit{ident} \rangle \ [\ `:' \ \langle \textit{type-name} \rangle \ ]$$

$$\begin{array}{lll}
\langle \textit{direction} \rangle & ::= & \text{`\texttt{in}'} \ (\text{input signal}) \\
& | & \text{`\texttt{out}'} \ (\text{output signal}) \\
& | & \text{`\texttt{const}'} \ (\text{constant input value})
\end{array}$$

$$\begin{array}{lll}
\langle \textit{signal\_direction} \rangle & ::= & \text{`\texttt{normal}'} \\
& | & \text{`\texttt{inverse}'}
\end{array}$$

$$\begin{array}{lll}
\langle \textit{var-decl} \rangle & ::= & \text{`\texttt{var}'} \ \langle \textit{ident} \rangle \ `:' \\
& & \langle \textit{type-name} \rangle \ [\ `=' \ \langle \textit{expression} \rangle \ ] \ `;'
\end{array}$$

$$\begin{array}{lll}
\langle \textit{enum-decl} \rangle & ::= & \text{`\texttt{enum}'} \ \langle \textit{ident} \rangle \\
& & \text{`\texttt{\{}'} \ \langle \textit{enum-field} \rangle \ \{ \ `,' \ \langle \textit{enum-field} \rangle \ \} \ \text{`\texttt{\}}'} \ `;'
\end{array}$$

$$\langle \textit{enum-field} \rangle \quad ::= \quad \langle \textit{ident} \rangle \ [\ `=' \ \langle \textit{integer} \rangle \ ]$$

$$\langle \textit{const-decl} \rangle \quad ::= \quad \text{`\texttt{const}'} \ \langle \textit{ident} \rangle \ `:' \ \langle \textit{type-name} \rangle \ `=' \ \langle \textit{expression} \rangle \ `;'$$

$$\langle \textit{bus-decl} \rangle \quad ::= \quad [\ \text{`\texttt{clocked}'} \ ] \ \text{`\texttt{bus}'} \ \langle \textit{ident} \rangle \ \langle \textit{bus-decl-content} \rangle \ `;'$$

$$\begin{array}{lll}
\langle \textit{func-decl} \rangle & ::= & \text{`\texttt{function}'} \ \langle \textit{ident} \rangle \ `(' \ \langle \textit{params} \rangle \ `)' \ \{ \ \langle \textit{func-decls} \rangle \ \} \ \text{`\texttt{\{}'} \\
& & \{ \ \langle \textit{statement} \rangle \ \} \ \text{`\texttt{\}}'}
\end{array}$$

$$\begin{array}{lll}
\langle \textit{bus-decl-content} \rangle & ::= & \text{`\texttt{\{}'} \ \langle \textit{bus-signal-decls} \rangle \ \text{`\texttt{\}}'} \\
& | & \langle \textit{type-name} \rangle
\end{array}$$

$$\langle \textit{bus-signal-decls} \rangle \quad ::= \quad \langle \textit{bus-signal-decl} \rangle \ \{ \ \langle \textit{bus-signal-decl} \rangle \ \}$$

$\langle \textit{bus-signal-decl} \rangle$ ::= $\langle \textit{ident} \rangle$ ':' $\langle \textit{type-name} \rangle$ ['=' $\langle \textit{expression} \rangle$] [',' $\langle \textit{signal\_direction} \rangle$] ';'

$\langle \textit{connect-entry} \rangle$ ::= $\langle \textit{name} \rangle$ '->' $\langle \textit{name} \rangle$

$\langle \textit{connect-decl} \rangle$ ::=   connect $\langle \textit{connect-entry} \rangle$ { ',' $\langle \textit{connect-entry} \rangle$ } ';'

$\langle \textit{inst-decl} \rangle$ ::=   'instance' $\langle \textit{instance-name} \rangle$ 'of' $\langle \textit{ident} \rangle$
     '(' [$\langle \textit{param-map} \rangle$ { ',' $\langle \textit{param-map} \rangle$ }] ')' ';'

$\langle \textit{instance-name} \rangle$ ::= $\langle \textit{ident} \rangle$ '[' $\langle \textit{expression} \rangle$ ']' (indexed instance)
     | $\langle \textit{ident} \rangle$ (named instance)
     | '_' (anonymous instance)

$\langle \textit{param-map} \rangle$ ::=   [$\langle \textit{ident} \rangle$ ':'] $\langle \textit{expression} \rangle$

$\langle \textit{gen-decl} \rangle$ ::=   'generate' $\langle \textit{ident} \rangle$ '=' $\langle \textit{expression} \rangle$ 'to' $\langle \textit{expression} \rangle$
     '{' { $\langle \textit{network-decl} \rangle$ } '}'

$\langle \textit{statement} \rangle$ ::= $\langle \textit{name} \rangle$ '=' $\langle \textit{expression} \rangle$ ';' (assignment)
     | $\langle \textit{name} \rangle$ '(' $\langle \textit{param-map} \rangle$ ')' ';' (function call)
     | 'if' $\langle \textit{expression} \rangle$ '{' { $\langle \textit{statement} \rangle$ } '}'
     { $\langle \textit{elif-block} \rangle$ } [$\langle \textit{else-block} \rangle$]
     | 'for' $\langle \textit{ident} \rangle$ '=' $\langle \textit{expression} \rangle$ 'to' $\langle \textit{expression} \rangle$
     '{' { $\langle \textit{statement} \rangle$ } '}'
     | 'switch' $\langle \textit{simple-expression} \rangle$
     '{' $\langle \textit{switch-case} \rangle$ { $\langle \textit{switch-case} \rangle$ } ['default' '{' $\langle \textit{statement} \rangle$
     { $\langle \textit{statement} \rangle$ } '}'] '}'
     | 'trace' '(' $\langle \textit{format-string} \rangle$ { ',' $\langle \textit{expression} \rangle$ } ')' ';'
     | 'assert' '(' $\langle \textit{expression} \rangle$ [',' $\langle \textit{string-literal} \rangle$] ')' ';'
     | 'break' ';'

$\langle \textit{switch-case} \rangle$ ::=   'case' $\langle \textit{simple-expression} \rangle$ '{' { $\langle \textit{statement} \rangle$ } '}'

$\langle \textit{elif-block} \rangle$ ::=   'elif <expression> '{' { $\langle \textit{statement} \rangle$ } '}'

$\langle \textit{else-block} \rangle$ ::=   'else' '{' { $\langle \textit{statement} \rangle$ } '}'

$\langle \textit{format-string} \rangle$ ::=   '"' { $\langle \textit{format-string-part} \rangle$ } '"'

$\langle \textit{format-string-part} \rangle$ ::=   '{}' (placeholder string)
     | $\langle \textit{string-char} \rangle$

$\langle \textit{simple-expression} \rangle$ ::= $\langle \textit{literal} \rangle$
     | $\langle \textit{name} \rangle$

⟨*expression*⟩   ::= ⟨*simple-expression*⟩
          | ⟨*expression*⟩ ⟨*bin-op*⟩ ⟨*expression*⟩
          | ⟨*un-op*⟩ ⟨*expression*⟩
          | '(' ⟨*expression*⟩ ')'
          | '(' ⟨*type-name*⟩ ')' ⟨*expression*⟩ (type cast)

⟨*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-literal*⟩
          | '[' ⟨*integer*⟩ { ',' ⟨*integer*⟩ } ']' (Array literal)
          | 'true'
          | 'false'
          | ''U' (Undefined value)

⟨*string-literal*⟩ ::= '"'{ ⟨*string-char*⟩ }'"'

⟨*intrinsic-type*⟩ ::= 'i' ⟨*integer*⟩ (signed integer)
          | 'int' (arbitrary-width signed integer)
          | 'u' ⟨*integer*⟩ (unsigned integer)
          | 'uint' (arbitrary-width unsigned integer)
          | 'float' (arbitrary-width floating point)
          | 'f8' (8 bit floating point)

| 'f16' (16 bit floating point)
| 'f32' (single-precision floating point)
| 'f64' (double-precision floating point)
| 'bool' (boolean value)

⟨*type-name*⟩   ::= ⟨*intrinsic-type*⟩
         | ⟨*name*⟩ (type definition)
         | ⟨*type-name*⟩ '[' [⟨*expression*⟩] ']' (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
- await
- barrier
- break
- bus
- case
- const
- connect
- clocked
- default
- elif
- else
- enum
- exposed
- for
- from
- function
- generate
- if
- import
- in
- instance
- inverse
- network
- normal
- of
- out
- proc
- return
- switch
- sync
- to
- unique
- var
- wait
- where