

21. Importing *e* files 1

Import statements (**import** and **verilog import**) load a given *e* file or Verilog file. This clause describes the **import** statement. *See also:* 2.1.1, 25.1.3, and Clause 20. 5

**Keyword issues (verilog import)??

21.1 import 10

Purpose	Load other <i>e</i> modules	
Category	Statement	
Syntax	import <i>file-name</i> , ... (<i>file-name</i> , ...)	15
Parameters	<p><i>file-name</i>, ... The names of files, separated by commas (,), that contain <i>e</i> modules to be imported. If no extension is given for a file name, an .<i>e</i> extension is assumed.</p> <p>The (<i>file-name</i>, ...) syntax is for cyclic importing, in which one module references a field in a second module, and the second module references a field in the first module.</p> <p>File names can contain references to environment variables using the UNIX notation $\\$name$ or $\{name\}$.</p> <p>The relative path indicators ./ and ../ can be used in filenames.</p>	20 25

This loads additional *e* modules before continuing to load the current file. If a specified module has already been loaded or compiled, the statement is ignored. For modules not already loaded or compiled, the search sequence is: 30

- a) directories specified by the PATH environment variable
- b) the current directory
- c) the directory in which the importing module resides.

Syntax example: 35

```
import test_drv.e;
```

21.2 Cyclic referencing and importing 40

When something in struct A needs to refer to a struct member of struct B, and something in struct B needs to refer to a struct member of struct A, this is called *cyclic referencing*. The **import** statement can handle cyclic references under the following conditions (this is called *implicit cyclic importing*). 45

- a) Before the definition of struct A in module A, add an **import** of module B in which struct B is defined.
- b) Before the definition of struct B in module B, add an **import** of module A.

Another way to do a cyclic import is to use the **import** (*file-name*, ...) syntax, which resolves cycles by loading the two or more modules as one. In this case, the following rules also apply. 50

- When multiple modules are loaded together, the behavior is as if the files are concatenated.
- No module is imported more than once.
- If an **import** statement includes a module that has already been loaded, that module is not imported. 55

21.3 Additional restrictions and considerations

- Within a given *e* module, **import** statements shall appear before any other statements except preprocessor directives (e.g., **#ifdef**), **define** statements, **verilog import** statements, and **package** statements. (**package** statements shall always precede any other statements.) Any other type of statement preceding an **import** statement shall cause a load-time or compile-time error.
- Modules that reside in different directories which use the same base names cannot be imported, even if they have different extensions. ~~This is because the *e* program internally uses only the base name, without its extension.~~
- A minimum of one file name needs to be used after the **import** keyword; otherwise, a load or compile-time error shall be issued.
- The case of an **import** followed by an **#ifdef** which, in turn, imports another module shall cause a load error if the second imported module has a statement other than a macro, preprocessor directive, or another **import** preceding one of those types of statements.
This is a special case which causes a violation of the statement order rule given in 2.2.1.
- A struct also cannot be referenced or extended during any type of import.
- Cyclic importing requires more memory to load multiple modules together than it takes to import modules singly, and it takes longer, which delays the automatic consistency checking of the *e* code.
- Cyclic importing can also mask problems with the ability of a module to be used in stand-alone mode in future applications, due to one module relying on another module being loaded.