Proposal to fix overly verbose package imports in port lists

Background (not part of actual proposal)

In order to use data types or constants declared within a package in module port declarations, one must either:

— Import the package into the $unit compilation unit space.

— Use the scope resolution operator for each module port or parameter that references the package.

Importing packages into the compilation unit space could conflict with other declarations in that space. In addition, some companies may impose modeling guidelines prohibiting declarations in the compilation unit space.

Using the scope resolution operator is verbose. If multiple ports reference types or parameters in a package, using the scope resolution operator can also become repetitious. Even the following short example shows how using the scope resolution operator in port lists becomes verbose and repetitive:

```
package shared_decls;
  parameter WIDTH = 32;

  typedef struct {
    bit [ 7:0] opcode;
    bit [23:0] addr;
  } instruction_t;
endpackage

module (input  [shared_decls::WIDTH-1:0] data,
        input   shared_decls::instruction_t a,
        output [shared_decls::WIDTH-1:0] result
       );
  ...
endmodule
```

The following proposal provides a simple way to reference package items in module, interface or program port declarations. It avoids the use of the compilation unit space, and eliminates the verbosity and receptiveness of using the scope resolution operator.

Proposal

1). Modify Annex A.1.4 as follows (add “package_import_declaration”).

parameter_port_declaration ::=
    parameter_declaration
  | data_type list_of_param_assignments
  | **type** list_of_type_assignments
  | package_import_declaration

2). Add new subclause 18.2.2, as follows (renumber current subclause 18.2.2 to 18.2.3).

18.2.2 Using packages with port declarations

Declarations such as typedefs and constants declared in packages can be referenced in module, interface or program port declarations by importing the package as part of the module, interface or program declaration. The syntax is:

```
module_nonansi_header ::=                                                         // from Annex A.1.3
        { attribute_instance } module_keyword [ lifetime ] module_identifier [ parameter_port_list ]
            list_of_ports ;
module_ansi_header ::=
        { attribute_instance } module_keyword [ lifetime ] module_identifier [ parameter_port_list ]
            [ list_of_port_declarations ] ;
interface_nonansi_header ::=
        { attribute_instance } interface [ lifetime ] interface_identifier
            [ parameter_port_list ] list_of_ports ;
interface_ansi_header ::=
        {attribute_instance } interface [ lifetime ] interface_identifier
            [ parameter_port_list ] [ list_of_port_declarations ] ;
program_nonansi_header ::=
        { attribute_instance } program [ lifetime ] program_identifier
            [ parameter_port_list ] list_of_ports ;
program_ansi_header ::=
        {attribute_instance } program [ lifetime ] program_identifier
            [ parameter_port_list ] [ list_of_port_declarations ] ;
parameter_port_list ::=                                                           // from Annex A.1.4
        # ( list_of_param_assignments { , parameter_port_declaration } )
      | # ( parameter_port_declaration { , parameter_port_declaration } )
parameter_port_declaration ::=
        parameter_declaration
      | data_type list_of_param_assignments
      | type list_of_type_assignments
      | package_import_declaration
```

Syntax 18-3—port declaration syntax (excerpt from Annex A)

Declarations that are imported as part of a module, interface or program declaration are visible throughout the module, interface or program. These imported declarations can be used as part of the port declarations. For example:

```
package shared_decls;
  parameter WIDTH = 32;

  typedef struct {
    bit [ 7:0] opcode;
    bit [23:0] addr;
  } instruction_t;
endpackage

module #(import shared_decls::*, parameter SIZE = 2048)
        (input  [WIDTH-1:0] data,
         input  instruction_t a,
         output [WIDTH-1:0] result
        );
  ...
endmodule
```