Proposal to fix overly verbose package imports in port lists

Background (not part of actual proposal)

In order to use typedefs or constants declared within a package in module port declarations, one must use the scope resolution operator for each module port or parameter that references the package. Using the scope resolution operator is verbose. A real design could have dozens, even hundreds of ports. Repeating the package name for every port requires redundant typing and can lead to typographical errors. 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
        ... // dozens more ports that use the package items ...
       );
  ...
endmodule
```

The only work around to avoid this verbose and redundant typing is to import the package into the $unit compilation space. Importing packages into the compilation unit space could conflict with other declarations in that space. Importing packages $unit as a work around could also require adding extra code and conditional compilation flags in order to work with both separate file compilation and multi-file compilation.

Proposal

Change A.1.2 from:

module_nonansi_header ::=
{ 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] **;**

To:

module_nonansi_header ::=
{ attribute_instance } module_keyword [lifetime] module_identifier ~~[parameter_port_list]~~
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

module_ansi_header ::=
{ attribute_instance } module_keyword [lifetime] module_identifier ~~[parameter_port_list]~~
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

...

interface_nonansi_header ::=
{ attribute_instance } **interface** [lifetime] interface_identifier
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

interface_ansi_header ::=
{attribute_instance } **interface** [lifetime] interface_identifier
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

...

program_nonansi_header ::=
{ attribute_instance } **program** [lifetime] program_identifier
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

program_ansi_header ::=
{attribute_instance } **program** [lifetime] program_identifier
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

Add a new footnote to the current end of Annex A footnotes (editor to number footnote accordingly).

1) A package_import_declaration in a module_ansi_header, interface_ansi_header, or program_ansi_header shall be followed by a parameter_port_list or list_of_port_declarations, or both.

Add new subclause after 25.3, as follows (renumber subsequent subclauses).

25.4 Using packages in module headers

Package items can be referenced in module, interface or program parameter and port declarations by importing the package as part of the header to the module, interface or program declaration. The syntax is shown in Syntax 25-4: (Editor to adjust syntax number as needed for position in text)

---

module_nonansi_header ::=
{ attribute_instance } module_keyword [lifetime] module_identifier ~~[parameter_port_list]~~
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

module_ansi_header ::=
{ attribute_instance } module_keyword [lifetime] module_identifier ~~[parameter_port_list]~~
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

...

interface_nonansi_header ::=
{ attribute_instance } **interface** [lifetime] interface_identifier
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

interface_ansi_header ::=
{attribute_instance } **interface** [lifetime] interface_identifier
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

...

program_nonansi_header ::=

{ attribute_instance } **program** [lifetime] program_identifier
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

program_ansi_header ::=
{attribute_instance } **program** [lifetime] program_identifier
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

---

1) A package_import_declaration in a module_ansi_header, interface_ansi_header, or program_ansi_header shall be followed by a parameter_port_list or list_of_port_declarations, or both.

Syntax 25-4—Package import in module, interface, or program header syntax (excerpt from Annex A)

Package items that are imported as part of a module, interface or program header are visible throughout the module, interface or program, including in parameter and port declarations.

For example:

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

package B;
   typedef enum bit {FALSE, TRUE} boolean_t;
endpackage: B

module M import A::instruction_t, B::*;
         #(WIDTH = 32)
          (input  [WIDTH-1:0]    data,
           input  instruction_t  a,
           output [WIDTH-1:0]    result,
           output boolean_t      OK
          );
  ...
endmodule: M
```