Modify 6.18 (User-defined types):

A forward typedef shall be considered incomplete prior to the final type definition. While incomplete forward types, ~~type parameters (see 6.20.3),~~ and types defined by an interface based typedef may resolve to class types, use of the class scope resolution operator (see 8.23) to select a type with such a prefix shall be restricted to ~~a~~ typedef declarations and type parameter assignments. It shall be an error if the prefix does not resolve to a class.

Append 6.20.3 (Type parameters):

A parameter constant can also specify a data type, allowing modules, interfaces, or programs to have ports and data objects whose type is set for each instance.

```
module ma #( parameter p1 = 1, parameter type p2 = shortint )
           (input logic [p1:0] i, output logic [p1:0] o);
   p2 j = 0; // type of j is set by a parameter, (shortint unless redefined)
   always @(i) begin
      o = i;
      j++;
   end
endmodule

module mb;
   logic [3:0] i,o;
   ma #(.p1(3), .p2(int)) u1(i,o); //redefines p2 to a type of int
endmodule
```

In an assignment to, or override of, a type parameter, the right-hand expression shall represent a data type.

A data-type parameter (**`parameter type`**) can only be set to a data type. Package references are allowed. Hierarchical names are not allowed.

It shall be illegal to override a type parameter with a **`defparam`** statement.

Similar to forward typedefs (see 6.18), type parameters may restrict their valid types by including the basic data types **`enum`**, **`struct`**, **`union`**, **`class`**, and **`interface class`** before the type parameter identifier:

```
type enum parameter_identifier
type struct parameter_identifier
type union parameter_identifier
type class parameter_identifier
type interface class parameter_identifier
```

It shall be an error if the type parameter is assigned a type definition that does not conform to the specified basic data type.

While type parameters may resolve to class types, use of the class scope resolution operator (see 8.23) to select a type with such a prefix shall be restricted to typedef declarations and type parameter assignments. It shall be an error if the prefix does not resolve to a class.

Example:

```
class P#(type C);
  C::T x;                     // Illegal, C is an incomplete type
  localparam type C_t = C::T; // Legal, reference to C::T is made
  C_t y;                      // by parameter assignment
endclass : P
```

```
class X;
  typedef int T;
endclass : X
typedef P#(X) P_X;
```

Modify 8.23 (class scope resolution operator)

Original

The left operand of the scope resolution operator **::** shall be a class type name, package name (see 26.2), **covergroup** type name, **coverpoint** name, **cross** name (see 19.5, 19.6), **typedef** name, or type parameter name. When a type name is used, the name shall resolve to a class or covergroup type after elaboration.

Modified

The left operand of the scope resolution operator **::** shall be a class type name, package name (see 26.2), **covergroup** type name, **coverpoint** name, **cross** name (see 19.5, 19.6), **typedef** name, or type parameter name. When a type name is used, the name shall resolve to a class or covergroup type after elaboration. While incomplete forward types, types defined by an interface-based typedef (see 6.18), and type parameters (see 6.20.3) may resolve to class types, use of the class scope resolution operator to select a type with such a prefix shall be restricted to typedef declarations and type parameter assignments.

Modify A.1.3 and all references

Original

parameter_port_list ::=
**(** list_of_param_assignments { **,** parameter_port_declaration } **)**
| **#** **(** parameter_port_declaration { **,** parameter_port_declaration } **)**
| **#( )**
parameter_port_declaration ::=
parameter_declaration
| local_parameter_declaration
| data_type list_of_param_assignments
| **type** list_of_type_assignments

Modified

forward_type ::= **enum** | **struct** | **union** | **class** | **interface class**
parameter_port_list ::=
**(** list_of_param_assignments { **,** parameter_port_declaration } **)**
| **#** **(** parameter_port_declaration { **,** parameter_port_declaration } **)**
| **#( )**
parameter_port_declaration ::=
parameter_declaration
| local_parameter_declaration
| data_type list_of_param_assignments
| type_parameter_declaration
type_parameter_declaration ::= **type** [forward_type] list_of_type_assignments

Modify A.2.1.3 and all references

Original

type_declaration ::=
typedef data_type type_identifier { variable_dimension } ;
| **typedef** interface_instance_identifier constant_bit_select . type_identifier type_identifier ;
| **typedef** [**enum** | **struct** | **union** | **class** | **interface class**] type_identifier ;

Modified

type_declaration ::=
typedef data_type_or_incomplete_class_scoped_type type_identifier { variable_dimension } ;
| **typedef** interface_instance_identifier constant_bit_select . type_identifier type_identifier ;
| **typedef** [forward_type] type_identifier ;

Add to A.2.2.1 and all references

type_identifier_or_class_type ::= type_identifier | class_type
incomplete_class_scoped_type :: =
  type_identifier **::** type_identifier_or_class_type
  | incomplete_class_scoped_type **::** type_identifier_or_class_type
data_type_or_incomplete_class_scoped_type ::= data_type | incomplete_class_scoped_type

Modify A.2.4 and all references

Original

type_assignment :: =
  type_identifier [= data_type] [18]

Modified

type_assignment :: =
  type_identifier [= data_type_or_incomplete_class_scoped_type] [18]

Modify A.2.11 and all references

Original

local_parameter_declaration ::=
localparam data_type_or_implicit list_of_param_assignments
| **localparam type** list_of_type_assignments
parameter_declaration ::=
parameter data_type_or_implicit list_of_param_assignments
| **parameter type** list_of_type_assignments

Modified

local_parameter_declaration ::=
localparam data_type_or_implicit list_of_param_assignments
| **localparam** type_parameter_declaration

parameter_declaration ::=
parameter data_type_or_implicit list_of_param_assignments
| **parameter** type_parameter_declaration