Agenda

- Requirements
- Packages
- Compilation Unit
- $root
Requirements

- Provide globally available named scopes containing declarations
- Support separate compilation of Verilog source
- Simplify (eliminate?) $root
Packages

• Support for sharing:
  – nets
  – variables, types, package imports
  – tasks, functions, dpi_import_export
  – classes, extern constraints, extern methods
  – parameters, local parameters, spec params
  – properties, sequences
  – anonymous program

• Can contain timeunit and timeprecision

• Cannot contain hierarchical references
package ComplexPkg;
    typedef struct {
        float i, r;
    } Complex;

    function Complex add(Complex a, b)
        add.r = a.r + b.r;
        add.i = a.i + b.i;
    endfunction

    function Complex mul(Complex a, b)
        mul.r = (a.r * b.r) + (a.i * b.i);
        mul.i = (a.r * b.i) + (a.i * b.r);
    endfunction
endpackage : ComplexPkg
Using Packages

• Scope resolution operator ::
  ComplexPkg::Complex cout = ComplexPkg::mul(a, b);

• Explicit import
  import ComplexPkg::Complex;
  import ComplexPkg::add;

• Wildcard import
  import ComplexPkg::*;
Package import

- Import
  - provides direct visibility to identifiers within the package
  - does not inline declarations
  - hierarchical references to imported identifiers are allowed as if they are defined in the importing scope

- Explicit import
  - like a local declaration
  - multiple identical allowed
Wildcard import

- Identifier in package a candidate for import
- Imported if neither declared nor explicitly imported
- Overridden by subsequent declaration within scope
- Same identifier from two wildcard imported packages shall have the identifier undefined and generate an error
package p;
    typedef enum { FALSE, TRUE } BOOL;
    const BOOL c = FALSE;
endpackage;

package q;
    const int c = 0;
endpackage;

import p::*;
import q::*;
y = c;

import p::*;  
import q::*;
int c = 1;
y = c;

import p::*;  
import q::*;
int c = 1;
y = c;

import p::*;  
import q::*;
wire a = c;

import p::*;  
import q::*;
wire a = c;
Compilation Unit Definitions

- Verilog compiles files
- Compilation unit: a collection of one or more files compiled together
- Compilation-unit scope: scope local to the compilation unit. Contains declarations outside of any other scope.
- $unit: name used to explicitly access identifiers in compilation-unit scope
Compilation Unit

- Definition mechanism for mapping files to compilation units is tool specific
- Compliant tool shall provide a mechanism
- Two extremes
  - All files in design compiled as a single compilation unit
  - Each file in design compiled as a compilation unit
- Compiler directives do not cross compilation units
- Top-level of compilation unit can contain:
  - modules, macromodules, primitives, programs, interfaces, packages, bind, and compilation-unit scope items
Compilation-unit Scope

• Can contain any item that can be defined in a package (including import)
• Scope is local to compilation unit (cannot be referenced from outside compilation unit)
• Useful for:
  – simple declarations (when package is too much)
  – items private to compilation unit scope
  – importing declarations for use in port and parameter declarations on modules, programs, interfaces.
Name search rules

• First: the nested scope is searched (1364-2001 12.6)
• Next: the compilation-unit scope is searched
• Finally: the instance hierarchy is searched (1364-2001 12.5)
$unit

- Name used in explicit scope resolution of compilation-unit scope
- Unambiguous reference to declarations in compilation-unit scope

```verilog
bit b;
task foo;
int b;
b = 5 + $unit::b; // $unit::b is the one outside.
endtask
```
$root simplification

- $root declaration, instance, and statement support are removed
- $root is the name to unambiguously refer to the top-level instance (root of instantiation tree)

```
$root.A.B     // item B within top instance A
$root.A.B.C   // item C within instance B within instance A
```
Name spaces

• Reworked name space to be
  – 2 global (definitions and package)
  – 2 compilation unit (compilation-unit scope and text macro)
  – 4 local (module, block, port, and attribute)

• Updated definitions to include programs, interfaces, and packages