Section 12.11

Changes (change in red):

12.11 Randomization of scope variables - std::randomize()

The built-in class randomize method operates exclusively on class member variables. Using classes to model the data to be randomized is a powerful mechanism that enables the creation of generic, reusable objects containing random variables and constraints that can be later extended, inherited, constrained, overridden, enabled, disabled, merged with or separated from other objects. The ease with which classes and their associated random variables and constraints can be manipulated make classes an ideal vehicle for describing and manipulating random data and constraints. However, some less-demanding problems that do not require the full flexibility of classes, can use a simpler mechanism to randomize data that does not belong to a class. The scope randomize method, std`::randomize()`, enables users to randomize data in the current scope, without the need to define a class or instantiate a class object.

Change Syntax box (change in red):

scope_randomize ::= [std::] **randomize** **(** [variable_identifier_list] **)** [**with** **{** constraint_block **}**]

Changes (change in red):

For example:

```
module stim;
       bit[15:0] addr;
       bit[31:0] data;

       function bit gen_stim();
              bit success, rd_wr;

              success = ~~randomize( addr, data, rd_wr );    // call
              std::randomize
              return rd_wr ;
       endfunction

       ...
endmodule
```

The function *gen_stim* calls std`::randomize()` with three variables as arguments: *addr*, *data*, and *rd_wr*. Thus, std::randomize() assigns new random variables to those variables that are visible in the scope of the *gen_stim* function. Note that *addr* and *data* have module scope, whereas *rd_wr* has scope local to the function. The above example can also be written using a class:

Section 13.1

Changes (change in red):

Semaphores and mailboxes are built-in types, nonetheless, they are classes, and can be used as base classes for deriving additional higher level classes. These ~~Built~~ built-in classes reside in the built-in ~~namespace~~ std package (see Section 7.10.1), thus, they may be re-defined by user code in any other scope.

Section Index

Changes (change in red):

built-in ~~namespace~~ package **50**

Section Annex H (New)

Changes (change in red):

Annex C
Std Package

(Informative)

The standard package contains system types (see Section 7.10.1). The following types are provided by the std build-in package. The descriptions of the semantics of these types are defined in the indicated sections.

C.1 Semaphore

The semaphore class is described in Section 13.2 and its prototype is:

```
class semaphore;
       function new(int keyCount = 0 );
       task put(int keyCount = 1);
       task get(int keyCount = 1);
       function int try_get(int keyCount = 1);
endclass
```

C.2 Mailbox

The mailbox class is described in Section 13.3 and its prototype is:

Note: *dynamic_singular_type* below represents a special type that enables run-time type-checking.

```
class mailbox #(type T = dynamic_singular_type) ;
       function new(int bound = 0);
       function int num();
       task put( T message);
       function int try_put( T message);
       task get( ref T message );
       function int try_get( ref T message );
       task peek( ref T message );
       function int try_peek( ref T message );
endclass
```

C.3 Randomize

The randomize function is described in Section 12.11 and its prototype is:

```
function int randomize( ... );
```

The syntax for the randomize function is:

randomize(variable_identifier {, variable_identifier } **)** [**with** constraint_block]**;**

C.4 Process

The process class is described in Section 9.9 and its prototype is:

```
class process;
       enum state { FINISHED, RUNNING, WAITING, SUSPENDED, KILLED };
```

```
        static function process self();
        function state status();
        task kill();
        task await();
        task suspend();
        task resume();
endclass
```