Action Item 46

Change to use static by default, reference class members, provide example of automatic usage.

Section 10.4

Change as shown in red:

In Verilog-2001, the default lifetime for tasks and functions is **`static`**. Automatic tasks and functions must be explicitly declared, using the **`automatic`** keyword.

SystemVerilog adds an optional qualifier to specify the default lifetime of all tasks and functions declared within a module, interface, or program (see Section 15). The lifetime qualifier is **`automatic`** or **`static`**. The default lifetime is **`static`** ~~for modules, and **`automatic`** for the program block (see section 15)~~.

```
program automatic test ;

        task foo( int a ); ... endtask      // arguments and variables in foo are automatic

endmodule
```

Class methods are by default **`automatic`**, regardless of the lifetime attribute of the scope in which they are declared. Classes are discussed in section 11.

Annex A

module_declaration ::=
{ attribute_instance } module_keyword [lifetime] module_identifier [parameter_port_list]
list_of_ports **;** [timeunits_declaration] { module_item }
endmodule
| { attribute_instance } module_keyword [lifetime] module_identifier [parameter_port_list]
[list_of_port_declarations] **;** [timeunits_declaration] { non_port_module_item }
endmodule

lifetime ::= **static** | **automatic** *// This production already exists*

Editors Note: This qualifier alters the BNF rules as follows:

Action Item 47

Add const (plus example), include comment on simulation semantic for var updates, add restriction on wire usage.

Section 10.5.2

Change as shown in red:

> When the argument is passed by reference, both the caller and the subroutine share the same representation of the argument, so any changes made to the argument either within the caller or the subroutine will be visible to each other. The semantics of assignments to variables passed by reference are the same as for static objects; that is, changes are seen outside the subroutine immediately (before the subroutine returns). Only variables, not wires, can be passed by reference.

Add the following paragraph at the end of the section.

> To protect arguments passed by reference from being modified by a subroutine, the **`const`** qualifier can be used together with **`ref`** to indicate that the argument, although passed by reference, is a read-only variable.

```
task show( const ref char [] data );
  for( int j = 0; j < data.size ; j++ )
      $display( data[j] );     // data can be read but not written
endtask
```

> When the formal argument is declared as a **const ref**, the subroutine cannot alter the variable, and an attempt to do so will generate compiler error.

Action Item 48

Add clarification about mixed named and default argument usage.

Section 10.5.4

Add paragraph at end of section as shown in red:

> If the arguments have default values, they are treated like parameters to module instances. If the arguments do not have a default, then they must be given or the compiler shall issue an error.
>
> If both positional and named arguments are specified in a single subroutine call then all the positional arguments must come before the named arguments. Then, using the same example as above:

```
fun( .s("yes"), 2 );          // illegal
fun( 2, .s("yes") );          // OK
```

Action Item 70

Clarify order that semaphores and mailboxes do something

Section 12.2.3

Change as shown in red:

> The semaphore waiting queue is First-In First-Out (FIFO). This does not guarantee the order in which processes will arrive at the queue, only that their arrival order will be preserved by the semaphore.

Section 12.4.5

Change as shown in red:

> The mailbox waiting queue is FIFO. This does not guarantee the order in which processes will arrive at the queue, only that their arrival order will be preserved by the mailbox.