Section 10

Tasks and Functions

Tasks, functions and analog functions provide the ability to execute common procedures from several different places in a description. They also provide a means of breaking up large procedures into smaller ones to make it easier to read and debug the source descriptions. The clause discusses the differences between tasks and functions, the differences between functions and analog functions, describes how to define and invoke task, functions and analog functions, and presents examples of each.

10.1 Distinctions between tasks, functions and analog functions

Refer to *IEEE 1364-2001 Verilog HDL* for differences between discrete tasks and functions. The following rules distinguish functions from analog functions:

- Functions execute within one discrete simulation time unit while analog functions execute within one analog timestep iteration.
- Functions disallows output and inout arguments, but analog function allows them.
- When the return type is not specified, functions return a one bit reg value but analog functions return a real value.
- Functions can be called within called within tasks, initial blocks, always blocks and other functions; but analog functions can be called from analog blocks and other analog functions.
- Functions can specify time and realtime return values, while analog functions cannot.
- Functions can specify a return value range, while analog functions cannot.
- Functions can declare registers, named events, local parameters, realtime and time arguments and local variables, while analog functions cannot.
- Functions can optionally specify a port list, while analog functions cannot.
- The return value from a function is mandatory, but the return value from an analog function is optional.

10.2 Tasks and task enabling

Refer to *IEEE 1364-2001 Verilog HDL* for declaration and usage. Calling analog functions within tasks is not permitted.

10.3 Functions and function calling

Refer to *IEEE 1364-2001 Verilog HDL* for declaration and usage. Calling analog functions within functions is not permitted.

10.4 Analog functions and analog function calling

The purpose of an analog function is to return a value that is to be used within an analog expression. ~~Each function can be an analog function or a digital function (as defined in *IEEE 1364-2001 Verilog HDL*).~~

10.4.1 Analog function declarations

The syntax for defining an analog function is given in Syntax 0-1.

```
analog_function_declaration ::= (From Annex A - A.6.6)
    analog function [ analog_function_type ] analog_function_identifier ;
    analog_function_item_declaration { analog_function_item_declaration }
    analog_function_statement
    endfunction
analog_function_type ::= integer | real
analog_function_item_declaration ::=
    analog_block_item_declaration
    | analog_function_input_declaration
    | analog_function_output_declaration
    | analog_function_inout_declaration
analog_function_input_declaration ::= input [ range ] list_of_port_identifiers ;
analog_function_output_declaration ::= output [ range ] list_of_port_identifiers ;
analog_function_inout_declaration ::= inout [ range ] list_of_port_identifiers ;
analog_block_item_declaration ::= (From Annex A - A.6.8)
     { attribute_instance } parameter_declaration
    | { attribute_instance } integer_declaration
    | { attribute_instance } real_declaration
analog_function_statement ::= (From Annex A - A.10.4)
    { attribute_instance } analog_function_case_statement
    | { attribute_instance } analog_function_conditional_statement
    | { attribute_instance } analog_function_loop_statement
    | { attribute_instance } analog_function_seq_block
    | { attribute_instance } analog_procedural_assignment
    | { attribute_instance } analog_system_task_enable
```

Syntax 0-1—Syntax for an analog function declaration

An analog function declaration shall begin with the keywords **analog function**, optionally followed by the type of the return value from the function, then the name of the function and a semicolon, and ending with the keyword **endfunction**. When specified, the type of the return value can be either real or integer. An analog function specified without a type, returns a real value.

Examples:

The following example defines an analog function called `maxValue`, which returns the potential of whichever signal is larger.

```
analog function real maxValue;
input n1, n2 ;
real n1, n2 ;
begin
   // code to compare potential of two signal
   maxValue = (n1 > n2) ? n1 : n2 ;
end
endfunction
```

The next example defines an analog function called geomcalc, which returns both the area and perimeter of a rectangle.

```
analog function real geomcalc;
input l, w;
output area, perim;
real l, w, area, perim;
begin
   area = l * w ;
   perim = 2 * ( l + w ) ;
end
endfunction
```

10.4.2 Returning a value from an analog function

The analog function declaration implicitly declares a scalar variable, internal to the analog function, with the same name as the analog function. This variable either defaults to real or is the same type as the type specified in the analog function declaration. The analog function definition initializes the return value from the analog function by assigning the analog function result to the internal variable with the same name as the analog function. This variable can be read and assigned within the flow; its last assigned value is passed back on the return call.

It is illegal to declare another object with the same name as the analog function in the scope where the analog function is declared. Inside the function, there is an implied variable with the name of the analog function, which may be used in expressions within the function. It is, therefore, also illegal to declare another object with the same name as the analog function inside the function scope.

Example:

The following line (from the example in 10.4.1) illustrates this concept:

```
maxValue = (n1 > n2) ? n1 : n2 ;
```

When an analog function does not assign its internal variable, the function shall return zero (`0`).

When an analog function computes more than one value, output and inout arguments can be used to pass these values back to the calling context. These variables can be read and assigned within the flow; the last value assigned during function evaluation is then assigned to the corresponding expression in the argument list for the function, which shall be an integer or real identifier or an element of an integer or real array.

Example:

The following lines (from the example in 10.4.1) illustrates this concept:

```
area = l * w ;
perim = 2 * ( l + w ) ;
```

10.4.3 Calling an analog function

An analog function call is an operand within an analog expression. The analog function call has the syntax given in _____.

```
analog_function_call ::= (From Annex A - A.8.2)
    analog_function_identifier { attribute_instance } ( analog_expression { , analog_expression } )
```

Syntax 0-2—Syntax for analog function call

The order of evaluation of the arguments to an analog function call is undefined.

Example:

The following example uses the `maxValue` function defined in 10.4.1.

```
V(out) <+ maxValue(val1, val2) ;
```

10.4.4 Function Rules

Analog functions have a number of constraints on the declaration and usage. An analog function shall:

- not access analog nets and branch values.
- not access digital nets and register values.
- declare at least one input argument of a specific type.
- declare the type of each local variable.
- not declare parameters, localparams or aliasparams.
- not use named sequential block, contribution and analog event control statements.
- only reference local variables or variables passed as arguments.
- only be called within an analog block or other analog function declarations.
- not (directly or indirectly) recursively call itself.
- be able to be called outside of scope that is declared in.
- not enable tasks or call functions.