Verilog HDL: Task and Functions

Pravin Zode

Outline

- Tasks
- Functions
- Syntax
- Difference
- Summary

Verilog Task

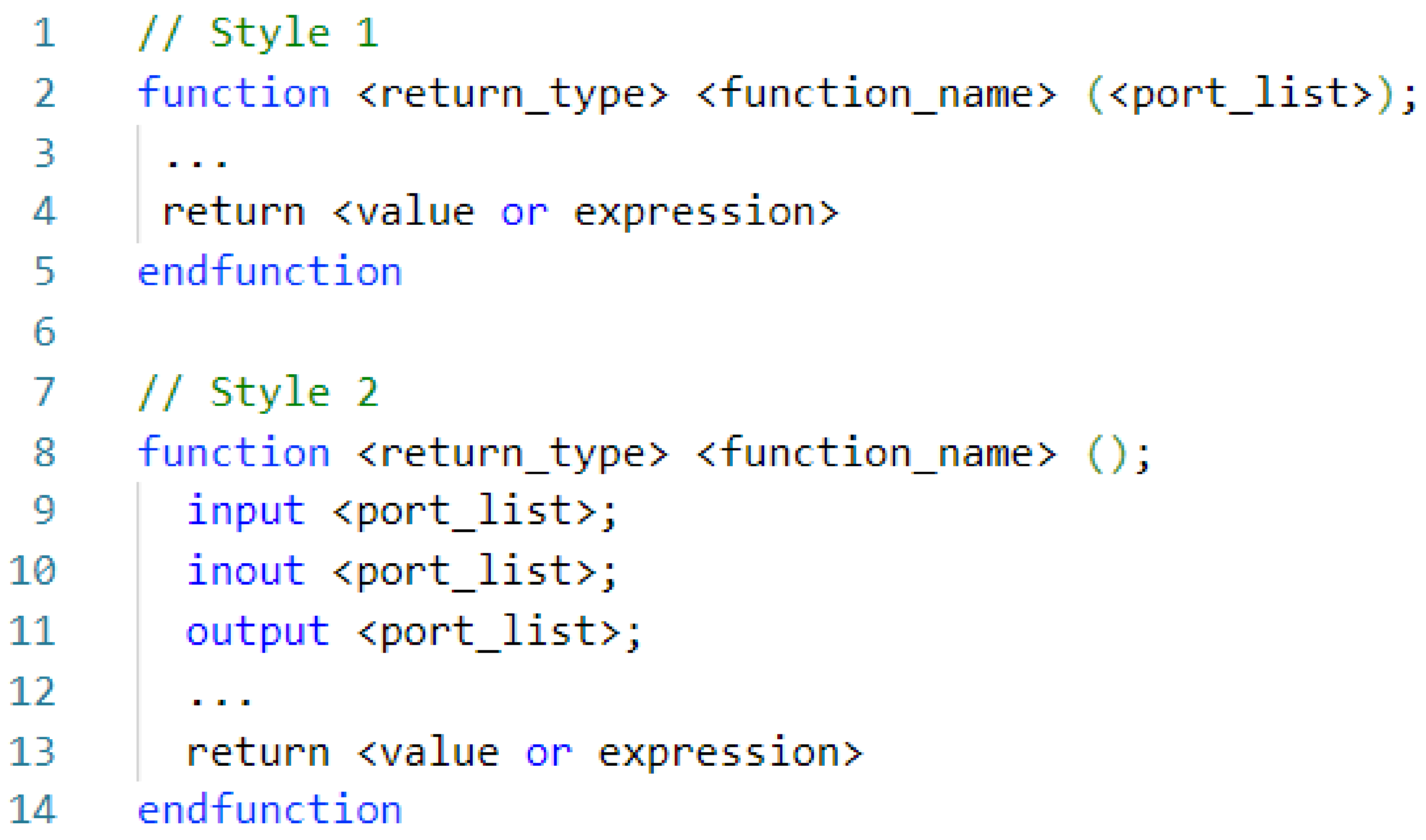
- More flexible than functions
- Can return multiple values using output or inout arguments
- Can contain delays (#), event controls (@), and non-blocking assignments (<=)
- Useful for complex operations that require timing control

Verilog Functions

- Functions are designed to process inputs and return a single value
- Cannot contain delays (#), event controls (@), or non-blocking assignments (<=)
- Execute in zero simulation time
- Useful for simple calculations.

- Functions: Arithmetic operations like addition, multiplication.
- Tasks: Stimulus generation, waveform printing, complex testbench operations

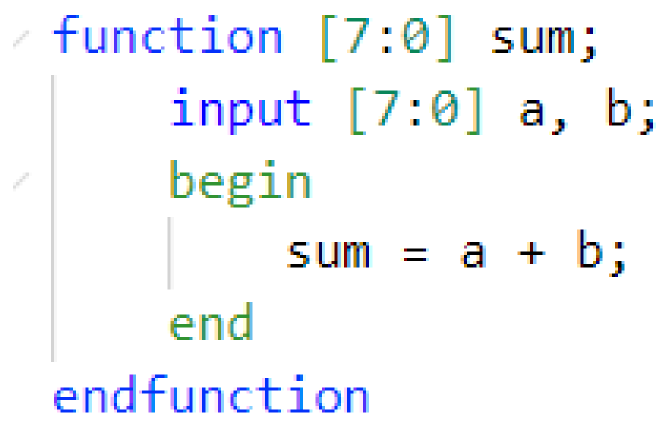
Verilog Function Syntax



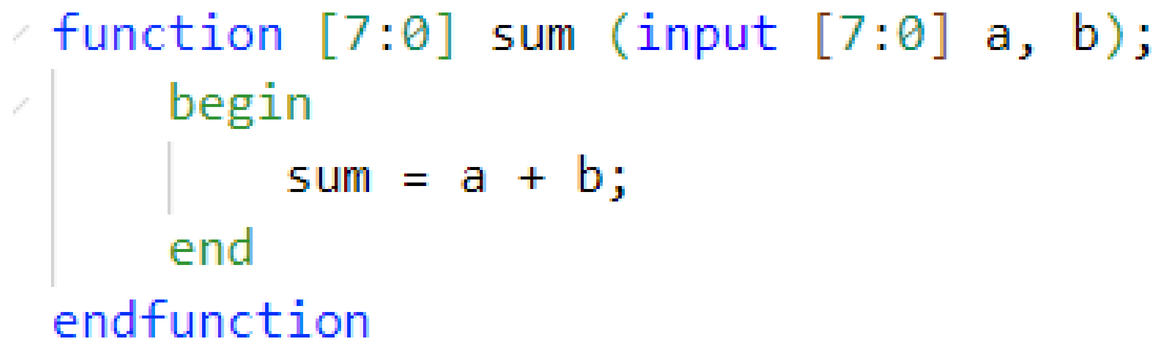
```
// Style 1
function <return_type> <function_name> (<port_list>);
 ...
 return <value or expression>
endfunction

// Style 2
function <return_type> <function_name> ();
  input <port_list>;
  inout <port_list>;
  output <port_list>;
  ...
  return <value or expression>
endfunction
```

Function Declaration

There are two ways to declare inputs to a function:



```
function [7:0] sum;
    input [7:0] a, b;
    begin
        sum = a + b;
    end
endfunction
```



```
function [7:0] sum (input [7:0] a, b);
    begin
        sum = a + b;
    end
endfunction
```

Function Call & Return

Function Return

```
sum = a + b;
```

Function Call

```
reg [7:0] result;
reg [7:0] a, b;

initial begin
    a = 4;
    b = 5;
    #10 result = sum (a, b);
end
```

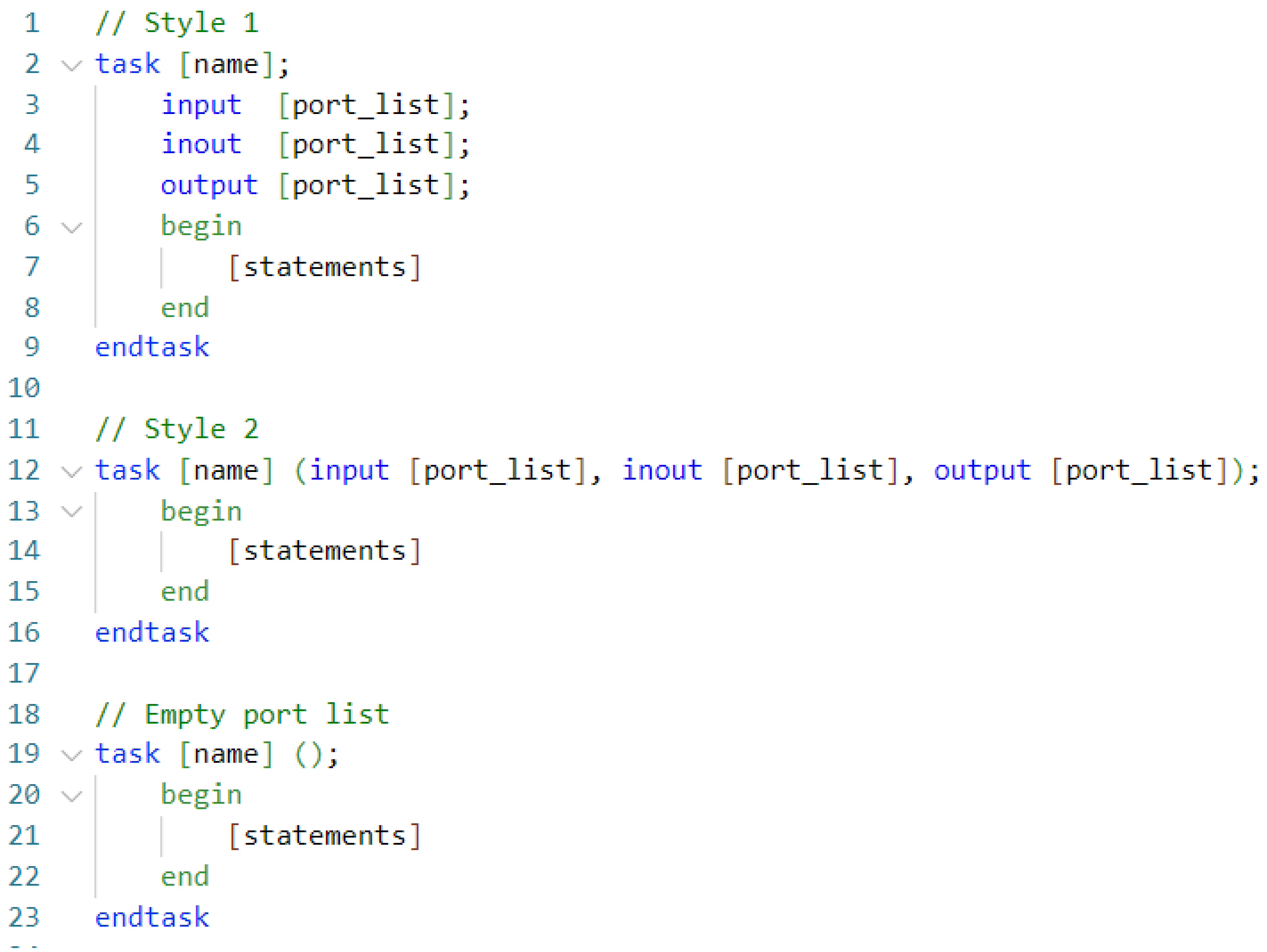
Verilog Function Syntax

```verilog
module function_example;

  function compare(input int a, b);
    if(a>b)
      $display("a is greater than b");
    else if(a<b)
      $display("a is less than b");
    else
      $display("a is equal to b");
    return 1; // Not mandatory to write
  endfunction

  initial begin
    compare(10,10);
    compare(5, 9);
    compare(9, 5);
  end
endmodule
```

Output

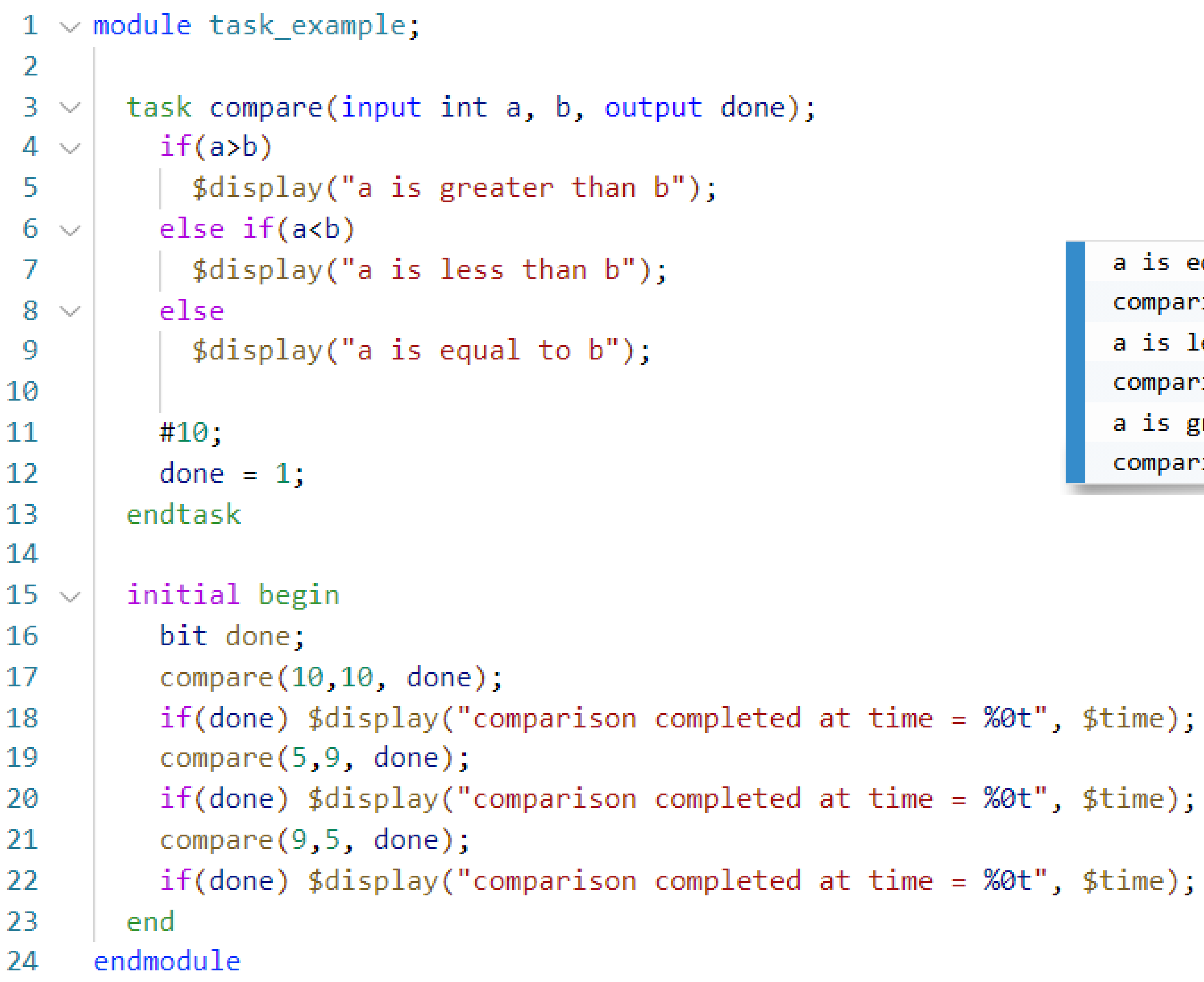
```
a is equal to b
a is less than b
a is greater than b
```

Verilog Task Syntax



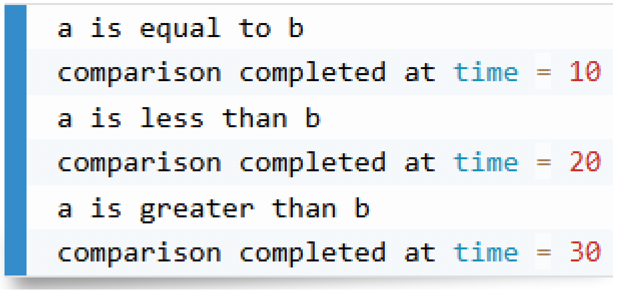
```
// Style 1
task [name];
    input  [port_list];
    inout  [port_list];
    output [port_list];
    begin
        [statements]
    end
endtask

// Style 2
task [name] (input [port_list], inout [port_list], output [port_list]);
    begin
        [statements]
    end
endtask

// Empty port list
task [name] ();
    begin
        [statements]
    end
endtask
```

Example : Verilog Task



```verilog
module task_example;

  task compare(input int a, b, output done);
    if(a>b)
      $display("a is greater than b");
    else if(a<b)
      $display("a is less than b");
    else
      $display("a is equal to b");

    #10;
    done = 1;
  endtask

  initial begin
    bit done;
    compare(10,10, done);
    if(done) $display("comparison completed at time = %0t", $time);
    compare(5,9, done);
    if(done) $display("comparison completed at time = %0t", $time);
    compare(9,5, done);
    if(done) $display("comparison completed at time = %0t", $time);
  end
endmodule
```

Output



```
a is equal to b
comparison completed at time = 10
a is less than b
comparison completed at time = 20
a is greater than b
comparison completed at time = 30
```

Difference Functions and Task

Function	Task
Function can enable another function but not another task	A task can enable other tasks and functions
Function always execute in 0 simulation time	Task may execute in non-zero simulation time
Functions must not contain any delay, event or timing control statements	Tasks may contain delay event or timing control statements
Functions must have atleast one input argument	Tasks may have zero or more arguments
Functions always return single value	Task do not return with a value
Functions cannot have output or inout arguments	Task can pass multiple values through output and inout arguments

Summary

Tasks

- Can have input, output, and inout arguments
- Can include delays, event controls, and timing constructs
- Used for complex operations or multi-step procedures
- Invoked like a procedure call: my_task(a, b);

Functions

- Only have input arguments.
- Cannot contain delays or event controls.
- Must execute in one simulation time unit.
- Return a single value: result = my_function(x);

Thank you !

Happy Learning