DAY-63
#100DAYSRTL

"System Verilog: Functions & Task "

"Introduction":-

Often we find certain pieces of code to be repetitive and called multiple times within the RTL. They mostly do not consume simulation time and might involve complex calculations that need to be done with different data values. In such cases, we can declare a function/task place the repetitive code inside the function/task, and allow it to return the result. This will reduce the number of lines in the RTL drastically since all you need to do now is to do a function call and pass the data on which the computation needs to be performed.

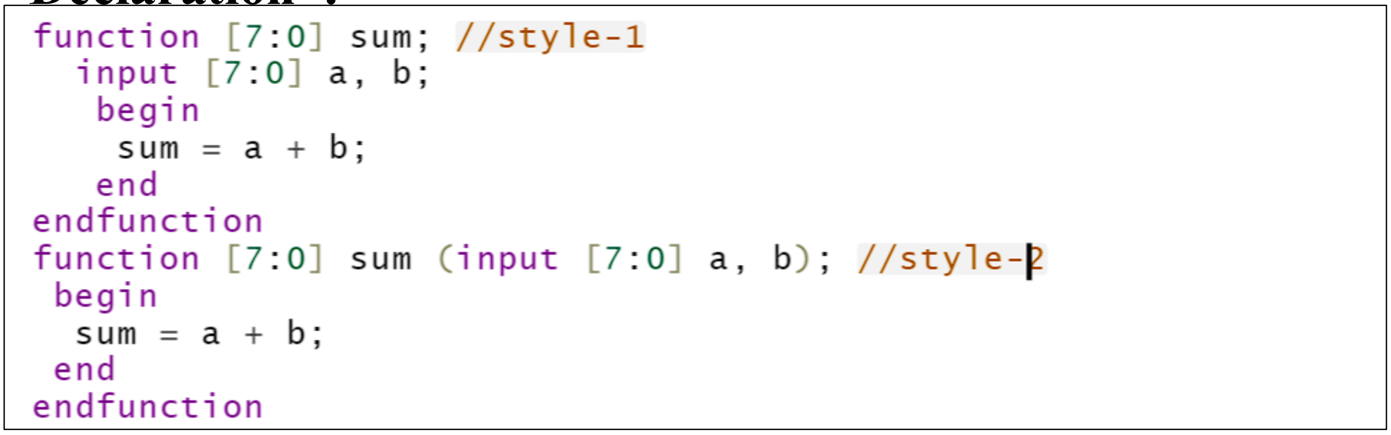
"Function":-

The purpose of a function is to return a value that is to be used in an expression. A function definition always starts with the keyword function followed by the return type, name, and port list enclosed in parentheses. Verilog knows that a function definition is over when it finds the end function keyword. Note that a function shall have at least one input declared and the return type will be void if the function does not return anything.

"Automatic Function":-

The keyword automatic will make the function reuse and items declared within the task are dynamically allocated rather than shared between different invocations of the task.This will be useful for recursive functions and when the same function is executed concurrently by N process when forked.

“Declaration”:-

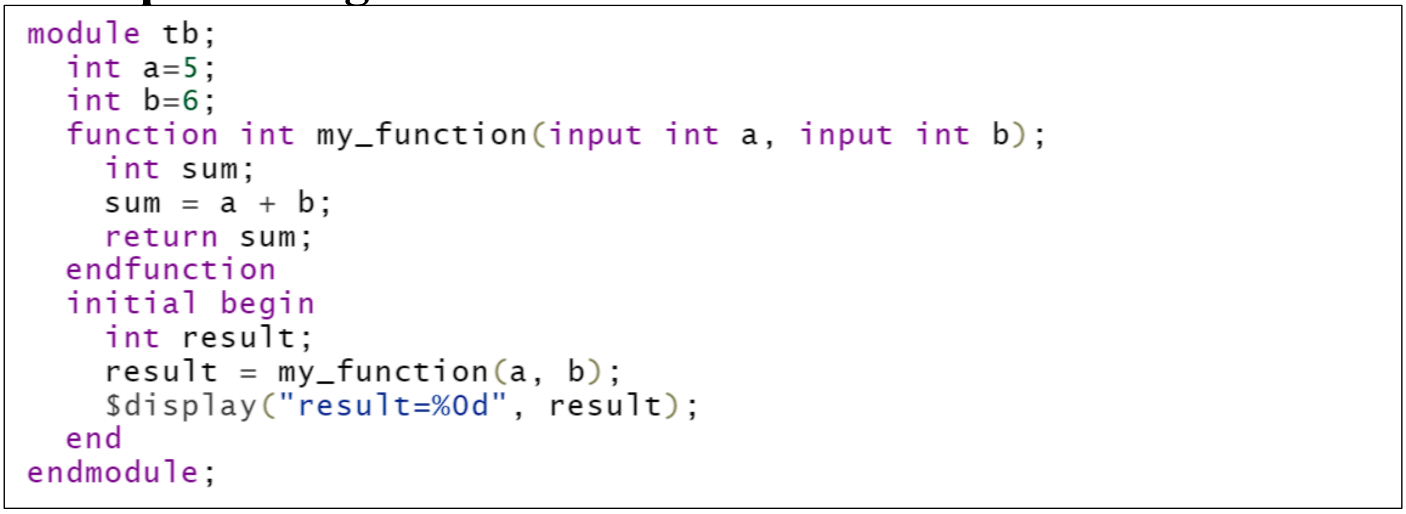


```
function [7:0] sum; //style-1
  input [7:0] a, b;
   begin
    sum = a + b;
   end
endfunction
function [7:0] sum (input [7:0] a, b); //style-2
 begin
  sum = a + b;
 end
endfunction
```

“Function Rules”:-

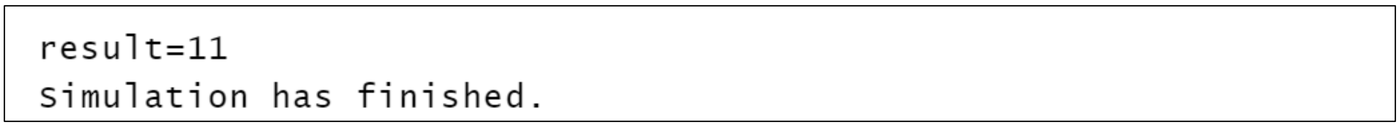
- A function cannot contain any time-controlled statements like #, @, wait, posedge, negedge
- A function cannot start a task because it may consume simulation time, but can call other functions
- A function should have at least one input
- A function cannot have non-blocking assignments or force-release or assign-design
- A function cannot have any triggers
- A function cannot have an output or input

Code practicing:-



```
module tb;
  int a=5;
  int b=6;
  function int my_function(input int a, input int b);
    int sum;
    sum = a + b;
    return sum;
  endfunction
  initial begin
    int result;
    result = my_function(a, b);
    $display("result=%0d", result);
  end
endmodule;
```

Result:-



```
result=11
Simulation has finished.
```

“Recursive Functions”:-

Functions that call itself is called recursive function

Code practicing:-

```
module tb;
  function automatic int factorial (input int a);
     if (a) factorial = a * factorial(a - 1);
     else factorial = 1;
  endfunction : factorial
  integer result;
  initial begin
    for(int i=0;i<11;i++) begin
      result = factorial(i);
      $display("factorial(%d) = %0d",i ,result);#10;
    end
  end
endmodule
```

Result:-

```
factorial(          0) = 1
factorial(          1) = 1
factorial(          2) = 2
factorial(          3) = 6
factorial(          4) = 24
factorial(          5) = 120
factorial(          6) = 720
factorial(          7) = 5040
factorial(          8) = 40320
factorial(          9) = 362880
factorial(         10) = 3628800
```

“Task”:-

A function is meant to do some processing on the input and return a single value, whereas a task is more general and can calculate multiple result values and return them using the output and inout type arguments. Tasks can contain simulation time-consuming elements such as @, posedge, and others.

“Declarations”:-

```
task [name];// Style 1
input [port_list]3;inout [port_list]3;output [port_list];
//[statements]
endtask
task [name] (input [port_list], inout [port_list], output [port_list]);// Style 2
//[statements]
endtask
// Empty port list
task [name] ();
//[statements]
endtask
```

“Automatic Task”:-

The keyword automatic will make the task reentrant, otherwise, it will be static by default. All items inside automatic tasks are allocated dynamically for each invocation and not shared between invocations of the same task running concurrently. Note that automatic task items cannot be accessed by hierarchical references.

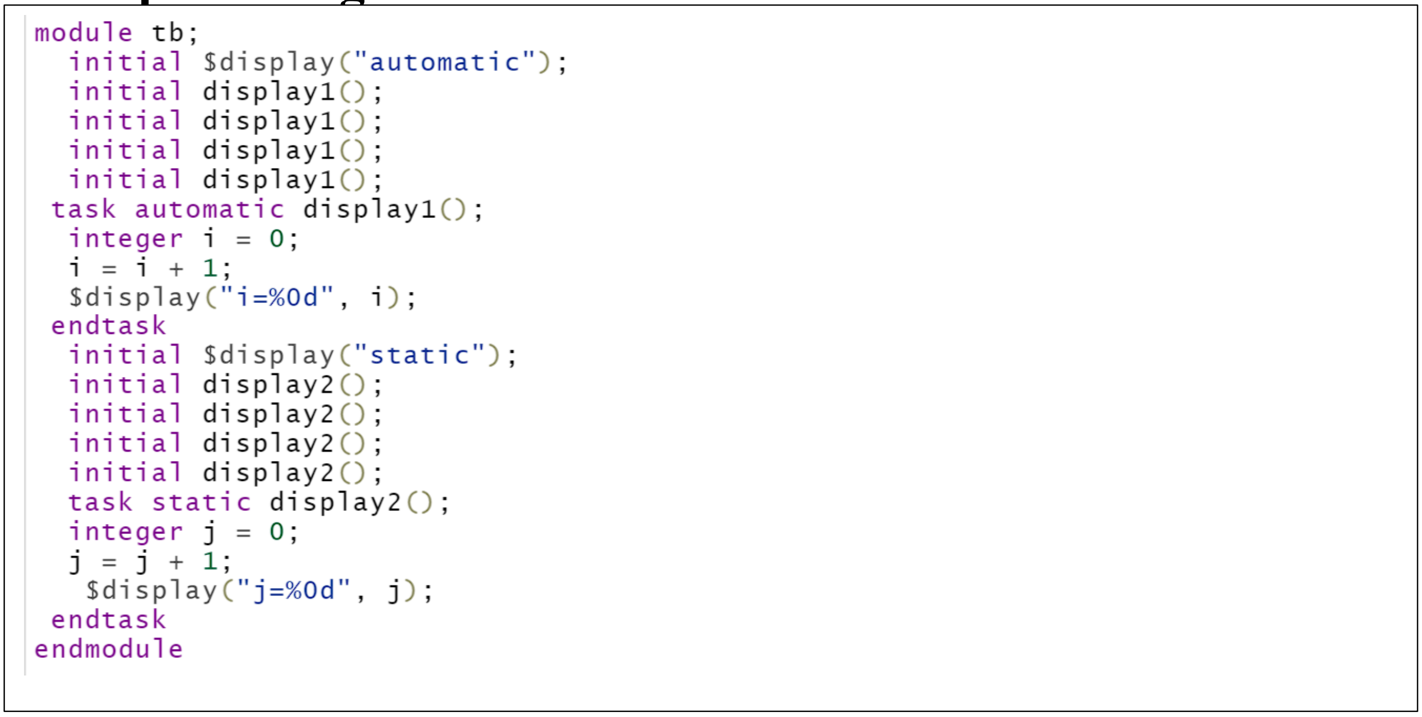
“Static Task”:-

If a task is static, then all its member variables will be shared across different invocations of the same task that has been launched to run concurrently

“Static vs Automatic”:-

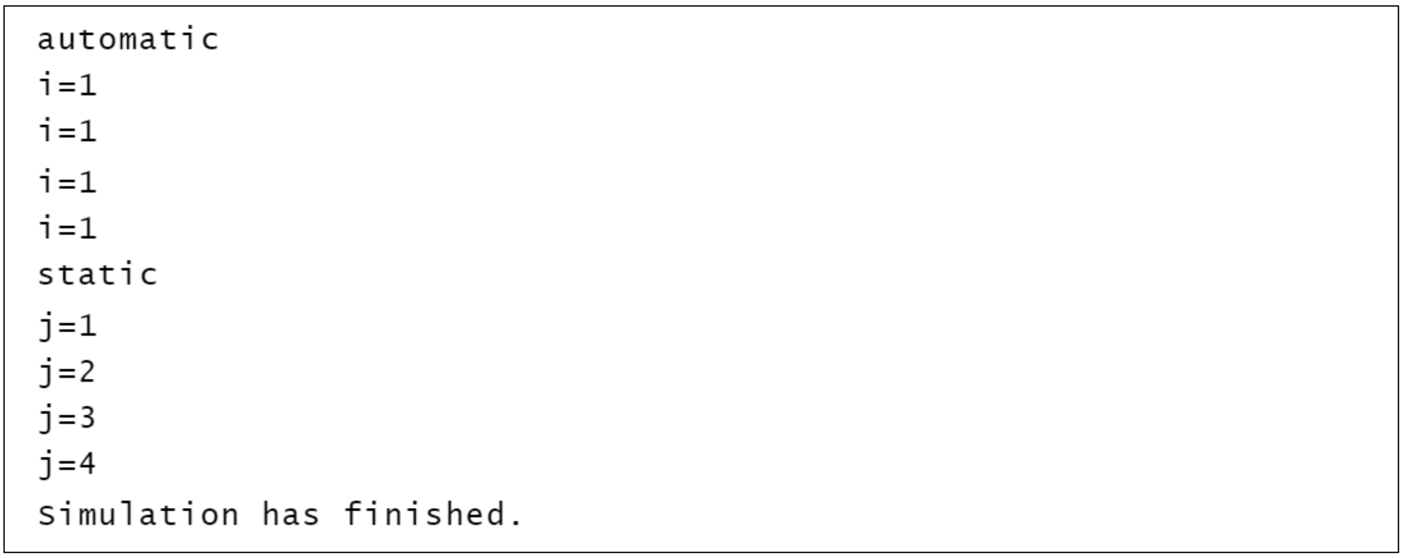
- If the task is made automatic, each invocation of the task is allocated a different space in simulation memory and behaves differently.
- If the task is made static, each invocation of the task is allocated a same space in simulation memory and behaves identically

Code practising:-



```
module tb;
  initial $display("automatic");
  initial display1();
  initial display1();
  initial display1();
  initial display1();
 task automatic display1();
  integer i = 0;
  i = i + 1;
  $display("i=%0d", i);
 endtask
  initial $display("static");
  initial display2();
  initial display2();
  initial display2();
  initial display2();
  task static display2();
  integer j = 0;
  j = j + 1;
   $display("j=%0d", j);
 endtask
endmodule
```

Result:-

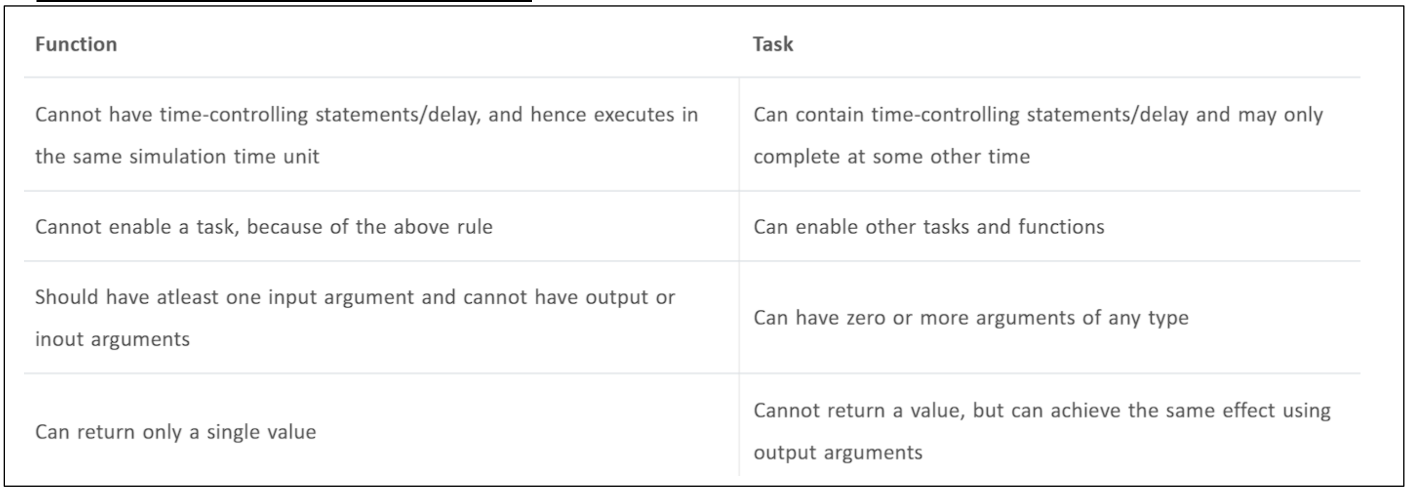


```
automatic
i=1
i=1
i=1
i=1
static
j=1
j=2
j=3
j=4
Simulation has finished.
```

"Task rules":-

- A task can contain time-controlled statements like #, @, wait, posedge, negedge.
- A task can start a task.
- A task can have any number of inputs, outputs, and inout ports.
- A task can have non-blocking assignments, force-release, and assign-design.
- A task can have any number of triggers.
- A task can have any number of outputs and inout ports**.**

"<u>Function Vs Task</u>":-



Function	Task
Cannot have time-controlling statements/delay, and hence executes in the same simulation time unit	Can contain time-controlling statements/delay and may only complete at some other time
Cannot enable a task, because of the above rule	Can enable other tasks and functions
Should have atleast one input argument and cannot have output or inout arguments	Can have zero or more arguments of any type
Can return only a single value	Cannot return a value, but can achieve the same effect using output arguments