FUNCTION AND TASK

FUNCTION VS TASK: Main difference betweeis.
Task: Consumes time, @(posedge), wait, #10, etc. can be used inside task.
Function: Will not consume time.

keywords and uses
ref
→ WHAT? Pass the reference rather than copying
→ WHY? While passing large array into a function, array is copied onto the stack, so passing larger array is expensive for memory. Thus, passing reference is beneficial.
→ WHERE? If u want to trigger something based on the change of data, then using reference of data instead of real data will be useful. Check my EDA example for
better understanding.

Automatic:
→ WHY? Unlike static, memory can be de-allocated in automatic.
→ WHAT? For a variable Automatic lifetime its memory will be de-allocated once execution of that method or block is over. Generally, it is stack storage of variable (for multiple entries to a task, function or block, it will have stack storage).

Const:
→ WHAT? Const keyword prevents changing the variable or array that being passed or referenced.

Void:
→ WHY? If a function does not return a value, then void is used.

Types of function and task:
It is mainly classified as static and automatic.

STATIC:
The static shares same memory space of memory for each call.

AUTOMATIC:
The automatic allocate unique storage for every call.
Another words you can call "mytask" and "myfunction" multiple times with diff arguments, without automatic modifier, if you call "mytask" for 2nd time while the first call is still waiting, the second would overwrite the previous arguments that are passed in the task. In order to observe clearly call same task 2 times and pass different variable inside fork join. This example has been discussed below.

Fig: Automatic vs static

BHUVANESH NOTES

```
//AUTOMATIC vs STATIC TASK

module tb;
  logic clk;

  initial begin:clock
    clk=0;
    while(1) #5 clk=~clk;
  end

  initial begin
      fork
        mytask(5,4);
        mytask(3,2);
        mytask(10,11);
        mytask_auto(5,4);
        mytask_auto(3,2);
        mytask_auto(10,12);
      join
    #100;
    disable clock;
  end

  task mytask(int a,b);
    @(posedge clk);
    $display("This task run is non automatic");
    $display("The val is a: %0d, b: %0d",a,b);
  endtask


  task automatic mytask_auto(int a,b);
    @(posedge clk);
    $display("This task run is automatic");
    $display("The val is a: %0d, b: %0d",a,b);
  endtask


endmodule
```

Output:

```
# KERNEL: This task run is non automatic
# KERNEL: The val is a: 10, b: 11
# KERNEL: This task run is non automatic
# KERNEL: The val is a: 10, b: 11
# KERNEL: This task run is non automatic
# KERNEL: The val is a: 10, b: 11
# KERNEL: This task run is automatic
# KERNEL: The val is a: 5, b: 4
# KERNEL: This task run is automatic
# KERNEL: The val is a: 3, b: 2
# KERNEL: This task run is automatic
# KERNEL: The val is a: 10, b: 12
```

Here in this above Fig 1: All the task and function has been made to wait until the posedge of clock thus the task without automatic modifier overwrites everytime when a new task is being called.

https://www.edaplayground.com/x/k9im - Link for the EDA page.

Practice Question: https://www.edaplayground.com/x/AJqQ
Think answer for the problem below.

BY NILESH NOTES

```
module tb;
  int val,a_val;
  initial begin

    sum();
    sum();

    a_sum();
    a_sum();
  end

  task sum();
    int i;
    i=i+4;
    $display("val: %0d",i);
  endtask

  task automatic a_sum();
    int i;
    i=i+4;
    $display("val: %0d",i);
  endtask


endmodule
```

You can go and run here: https://www.edaplayground.com/x/AJqQ

Why you need function?

If a portion of code being repeated in your design, we can make a function for that portion of code and return a value or data. That is if a portion code is being used multiple times, then converting that into a function will be easier

Note: If you have a system Verilog task that does not consume time, then declare that as void function.

FUNCTION:

Type 1:

In System Verilog (SV) if you want to ignore a return value use “void” type. Let’s see an example.

BHUVA SH NOTES

VOID, RETURN AND REFERENCE in function.

Here in the below code, there is 4 function example of how they are used as VOID, RETURNING VALUE, PASS BY REFRENCE, PASS BY VALUE.

EDA Link: https://www.edaplayground.com/x/YW9D

```
module tb;
  int ret_val,pass_val;
  initial begin
    disp();
    ret_val = return_function();
    $display("The returned value is : %d",ret_val);
    reference_function(ret_val);
    $display("Pass by referenced output is : %d",ret_val);
    pass_val = value_function(ret_val);
    $display("Pass by value output is : %d",pass_val);
    $display("********TEST ENDS*************");
  end
endmodule

//VOID TYPE
function void disp();
  $display("This is a void function");
endfunction

//WITH RETURN VALUE
function int return_function();
  int a=5;
  return a;
endfunction

//PASS BY REFRENCE
function automatic void reference_function(ref int ref_val);
  ref_val=ref_val*13;
endfunction

//PASS BY VALUE
function int value_function(int ref_val);
  return ref_val*13;
endfunction
```

OUTPUT:

```
# KERNEL: This is a void function
# KERNEL: The returned value is:          5
# KERNEL: Pass by referenced output is:         65
# KERNEL: Pass by value output is:        845
# KERNEL: ********TEST ENDS*************
```

TYPE 2: ARRAY RETURN

Passing and returning array, won't work in the same way as passing single data. This is happening because the return type for array is not available, so we should use typedef inorder to create return type. Or another way is passing ref of the array.

Code link: https://www.edaplayground.com/x/RMwg

```
//Function Returning Array
module tb;
  int ram[16];
  int ret_ram[16];
  typedef int dyn_arr[16];   // as there is no array return type,
                                we need to create the return type
  initial begin
    $display("The array is %0p",ram);
    arr_ram(ram);
    $display("The array is returned by reference %0p",ram);
    ram=arr_ram_ret(ram);
    $display("The array is returned value %0p",ram);
  end

  function automatic void arr_ram(ref int ram[16]);
    foreach(ram[i]) begin
      ram[i]=i*2;
    end
  endfunction

  function dyn_arr arr_ram_ret(int ram[16]);
    int dyn_array[16] = ram;
    foreach(ram[i]) begin
      dyn_array[i]=ram[i]*2;
    end
    return dyn_array;
  endfunction

endmodule
```

Output:

```
# KERNEL: The array is 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# KERNEL: The array is returned by reference 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
# KERNEL: The array is returned value 0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60
# KERNEL: Simulation has finished. There are no more test vectors to simulate.
```

Task:

Task and function are similar, however the task consumes time and in task the return is not available instead we will use output. Thus, the two main things

1. Why task? TASK can consume time
2. How to return in task? Instead RETURN use OUTPUT.

Practice : https://www.edaplayground.com/x/C3cz

```
// TASK BASICS
module tb;
  int ret_val,out_val;
  initial begin
    disp();
    return_task(ret_val);
    $display("value: %d",ret_val);
    in_out_function(3,8,out_val);
    $display("Output_val: %d",out_val);
    $display("*********TEST ENDS*************");
  end
endmodule

//SIMPLE TASK
task disp();
  $display("This is a simple display task");
endtask

//RETURN A VALUE
task return_task(output int x);
  x=5;
endtask

//INPUT AND OUTPUT VALUE
task in_out_function(input int x,y, output int
out_val);
  out_val=x*y;
endtask
```

OUTPUT:

```
# KERNEL: This is a simple display task
# KERNEL: value:           5
# KERNEL: Output_val:          24
# KERNEL: *********TEST ENDS*************
```