Tutorial 7

CSE313

Functions

```
function mem_mode return_type function_name (direction_of_arg arg_data_type arg_name,...);
  begin
    //code


    function_name= returned_value;
    or
    return returned_value;

  end
endfunction
```

Function name is treated as a temp variable

So the function body will be exectued and the return value will be independent of it

```
mem_mode: static or automatic (default static)
return_type: type+length, can't be wire,can be void, default is logic
function_name: same variable naming rules
direction_of_arg : input, output, inout
arg_data_type:type+length, can't be wire, default is logic
arg_name: same variable naming rules

example:
function static int sum (input int x,input int y);
  begin
    automatic int result
    result = x+y;


    sum = result;

  end
endfunction
```

Same as return result

Passing by reference and default values

```
function  void print (const ref int x[10]);
  begin
    foreach (x[i]) display(x[i]);

  end
endfunction
```

```
function static int sum (input int x=5,input int y=8);
  begin
    automatic int result
    result = x+y;


    sum = result;

  end
endfunction

initial begin
  sum(10,3)//13
  sum(,3)//8
  sum(10,)//18
  sum(.y(10),.x(7)); //17
end
```

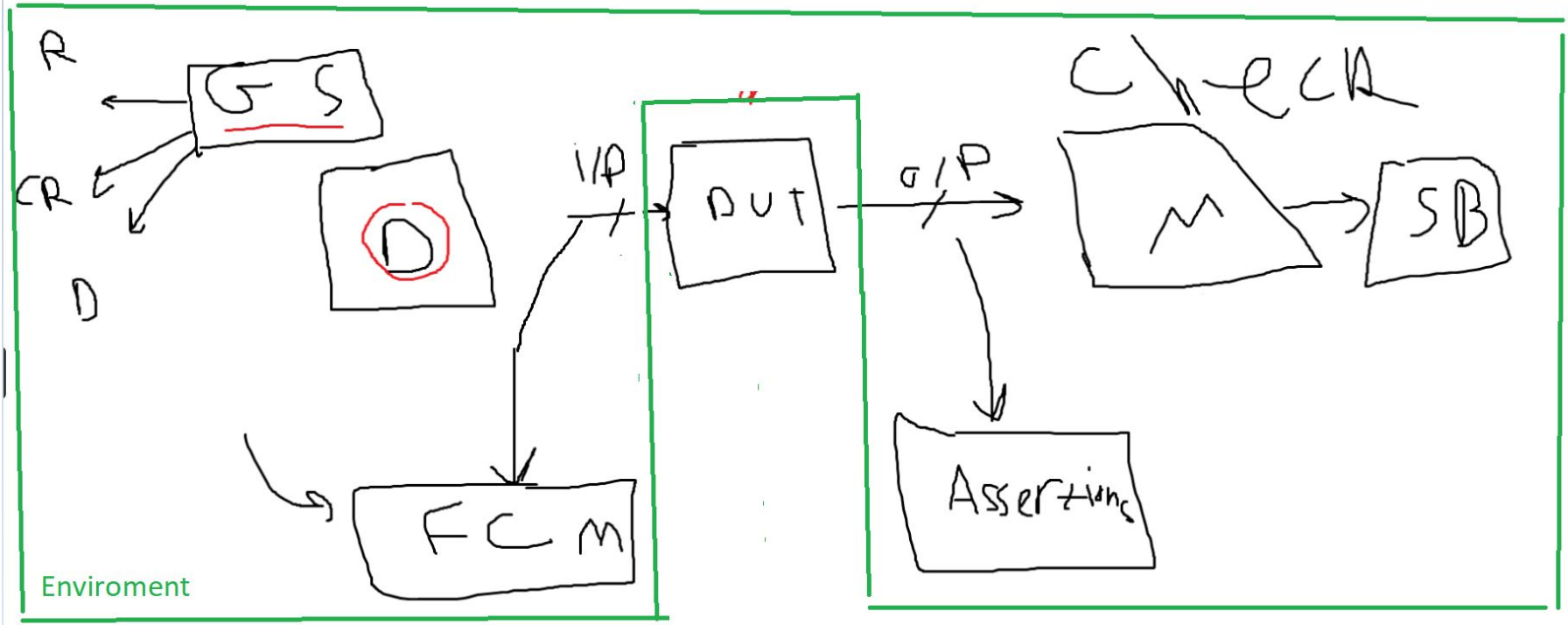
Tasks

- Have no return (but can use return keyword)
- Can use time consuming statements
- Can functions call tasks?
- Can we have recursion ?

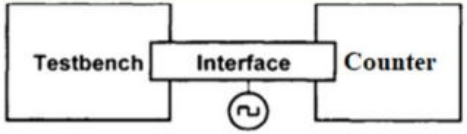
Sheet

1. What are void functions?
2. Explain about pass by ref and pass by value?
3. What is the concept of a "ref' and "const ref' argument in System Verilog function or task?
4. Is it possible for a function to return an array?
5. How to make sure that a function argument passed as ref is not changed by the function?

interfaces



Modports




```
1  interface count_ifc (input bit CLK);
2    logic [3:0] Q,P;
3    logic Load, Enable, MR;
4    modport driver (output P,Load, Enable, MR, input Q);
5    modport dut (input P,Load, Enable, MR, output Q);
6  endinterface
```

Interface

```
2  module decade_counter (count_ifc y);
3    always @(y.MR) begin
4      if (y.MR)
5        y.Q <= 4'b0000;
6    end
7
8    always @(posedge y.CLK) begin
9      if (!y.MR)
10       if (y.Load)
11         y.Q <= y.P;
12     else if(y.Enable)
13       y.Q <= (y.Q+1) % 10;
14   end
15 endmodule
```

Design

```
6  module test(count_ifc x);
7    initial begin
8      x.P <= 4'b0111;
9      x.MR <= 1'b0;
10     x.Enable <= 1'b1;
11     x.Load <= 1'b0;
12     #3 x.MR <= 1'b1;
13     #6 x.MR <= 1'b0;
14     #43 x.Enable <= 1'b0;
15     #15 x.Enable <= 1'b1;
16     #16 x.Load <= 1'b1;
17     #9 x.Load <= 1'b0;
18   end
19 endmodule
```

Testbench

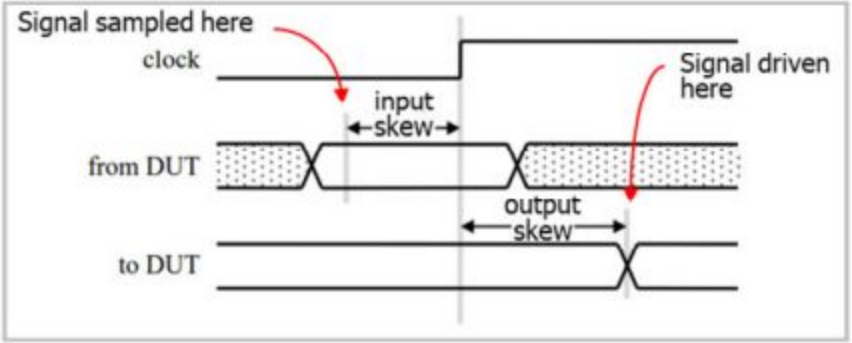
```
18 module top;
19   bit clk;
20   always #5 clk <= ~clk;
21   count_ifc ifc(clk);
22   decade_counter u1(ifc.dut);
23   test u2(ifc.driver);
24
25   initial begin
26     $dumpfile("counter.vcd");
27     $dumpvars;
28     #200 $finish;
29   end
30 endmodule
```

Top module

Clocking blocks

```
clocking cb @(posedge clk);
default input #10ns output #2ns;

output read,enable,addr;
input negedge data;
endclocking
```



Clocking blocks

```
interface intf (input clk);
  logic read, enable;
  logic [7:0] addr,data;

  // clocking block for testbench
  clocking cb @(posedge clk);
    default input #10ns output #2ns;
    output read,enable,addr;
    input data;
  endclocking

  modport dut (input read,enable,addr,output data);
  // Synchronous testbench modport
  modport tb (clocking cb);
endinterface :intf
```

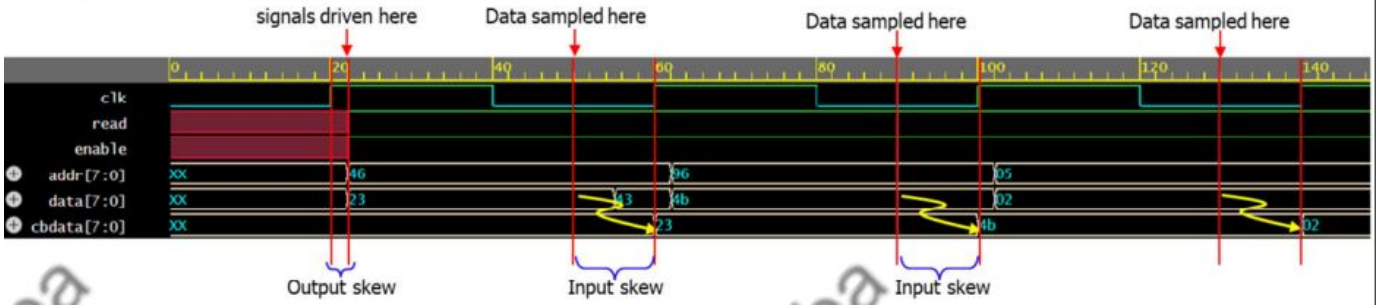
```
module memory(intf abc);
  logic [7:0] mem [256];
  initial begin
    foreach (mem[i])
      mem[i] = i >> 1;
  end
  always @(abc.enable,abc.read) begin
    if (abc.enable == 1 && abc.read == 1)
      abc.data = mem[abc.addr];
  end
endmodule
```

Clocking blocks

```
15 module testbench(intf xyz);
16   logic[7:0] cbdata;
17   initial begin
18     xyz.cb.read <= 1;    // driving a synchronous signal
19     xyz.cb.enable <= 1; // driving a synchronous signal
20     xyz.cb.addr <= 70;  // driving a synchronous signal
21     #30 xyz.cb.addr <= 150;
22     #25 xyz.data <= 67; // disturbing the DUT data
23     #40 xyz.cb.addr <= 5;
24   end
25   always @(xyz.cb)
26     cbdata = xyz.cb.data;// get the sampled data
27 endmodule
```

```
// clocking block for testbench
clocking cb @(posedge clk);
  default input #10ns output #2ns;
  output read,enable,addr;
  input data;
endclocking
```

```
26 module top;
27   bit clk = 0;
28   always #20 clk = ~clk;
29
30   intf i1(clk);
31   memory m1(i1.dut);
32   testbench t1(i1.tb);
33
34   initial begin
35     $dumpfile("uvm.vcd");
36     $dumpvars;
37     #200 $finish;
38   end
39 endmodule
```



Clocking blocks

```
clocking ck1 @ (posedge clk);
  default input #5ns output #2ns;

  input data, valid, ready;
  output x, y;

  output negedge grant;
  input #1step addr;
endclocking
```