Yasmeen Osama

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
```

'as'

```
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
```

function at return value;

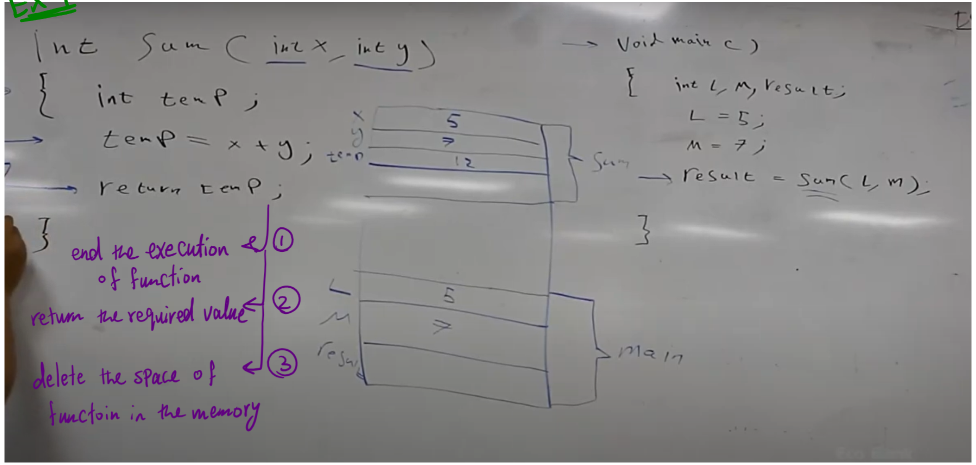
1) end the execution of function

2) return the required value

3) automatic : delete the space of function in the memory

static : does not delete the function space in the memory

Ex 1



```
int sum ( int x, int y )
{   int temp ;
    temp = x + y ;
    return temp ;
}
```

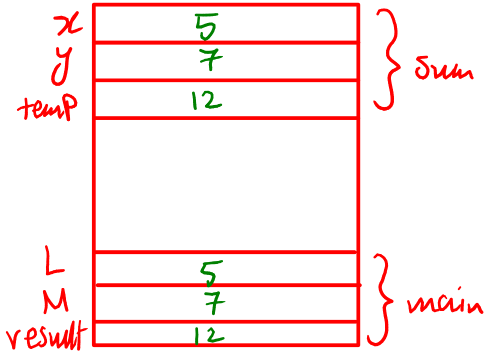
```
void main ( )
{   int L, M, result;
    L = 5;
    M = 7;
    result = sum( L, M );
}
```

end the execution of function ①

return the required value ②

delete the space of functoin in the memory ③

if x and y are inputs



$\boxed{1}$ $x = L = 5$

$y = M = 7$

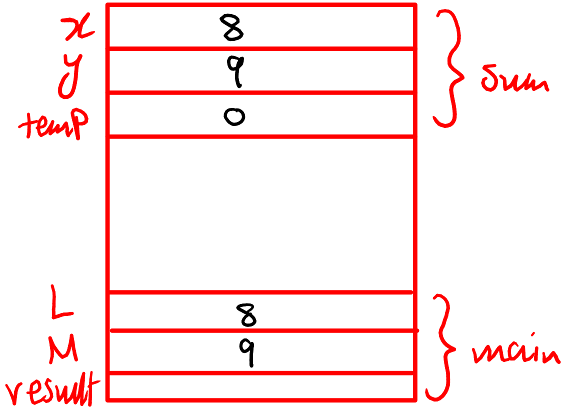
$temp = x + y = 12$

$\boxed{2}$ return the temp value in result

$\therefore$ result = temp = 12

Ex2

```
int sum (output int x, output int y)
{ int temp;
    x = 8;
    y = 9;
    return; }
```



$\boxed{1}$ $x = 0$, $y = 0$, $temp = 0$ } default value

$\boxed{2}$ $x = 8$

$y = 9$

$temp = 0$ still default

$\boxed{3}$ return x in L

$\therefore L = x = 8$

and y in M

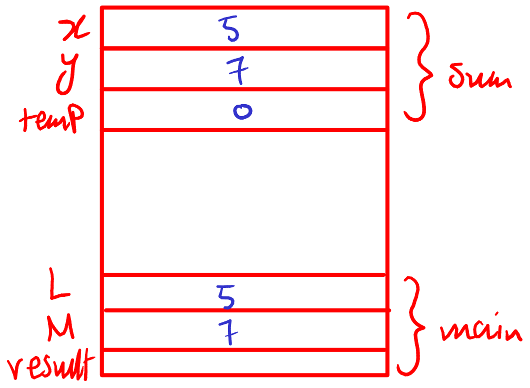
$\therefore M = y = 9$

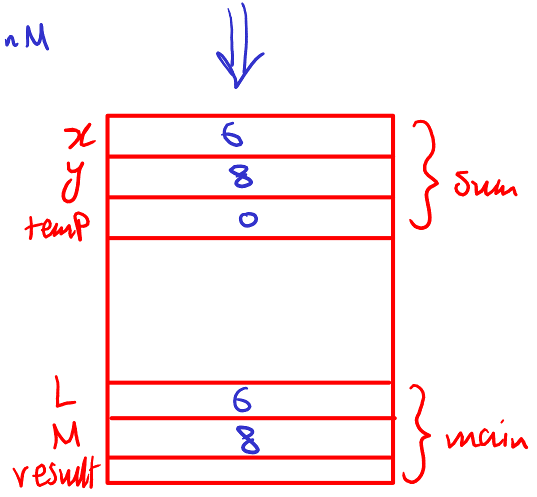
Ex 3 int sum (inout int x, inout int y)
```
{ int temp;
    x++;
    y++;
    return; }
```
[1] x = L = 5
y = M = 7
temp = 0

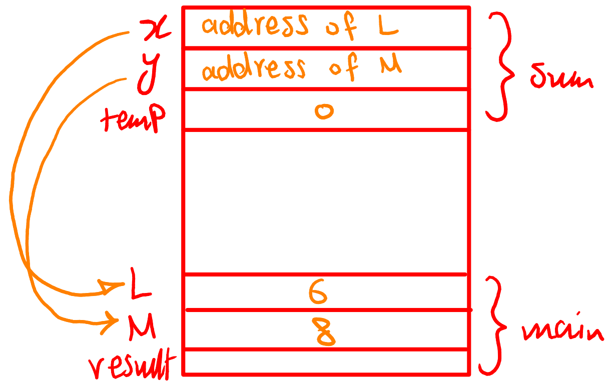
[2] x = 6
y = 8
temp = 0

[3] return x in L and y in M

L = x = 6
M = y = 8





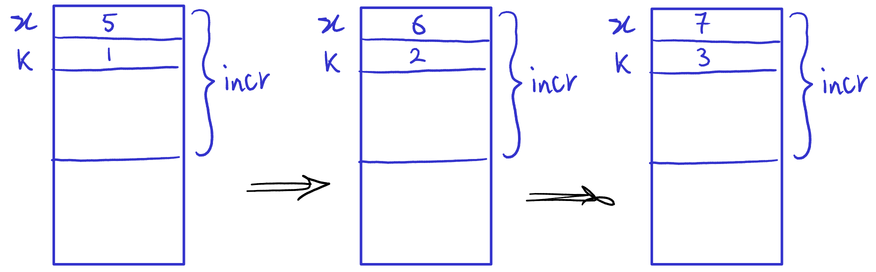
Ex 4 int sum (ref int x, ref int y)
```
{ int temp;
    x ++
    y ++
    return; }
```


→ The changes will be in L and M directly

→ x and y are Pointers of L and M

→ Calling by ref. take time less than Calling by value.

```
function static incr (int x);
      int k;
       k++;
 end function

initial begin
  incr(5);  → K=1
  incr(6);  → K=2
  incr(7);  → K=3
```

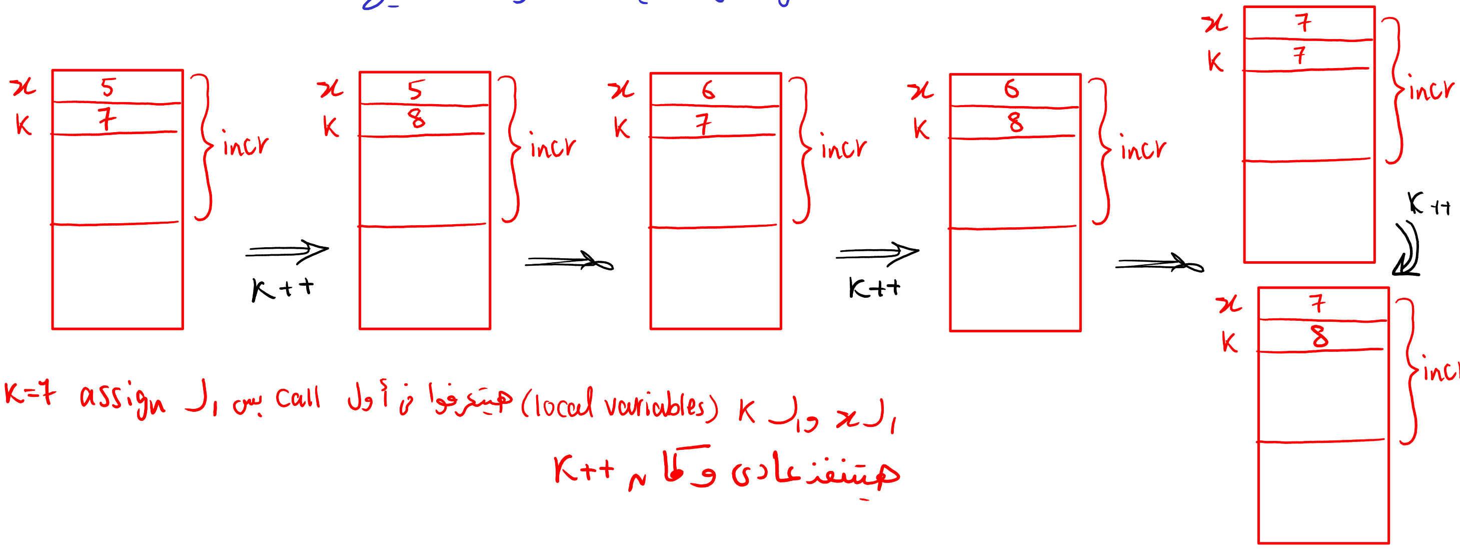


ال x وال K (local variables) هيتعرفوا ف أول call بس بالتالى ف تانى مرة

int k; , int x مش هيتنفذوا و K هيتفضل 1 مش 0 و K++ هيخليه 2

تالت مرة نفس الحكاية K=2 و K++ هيخليه 3

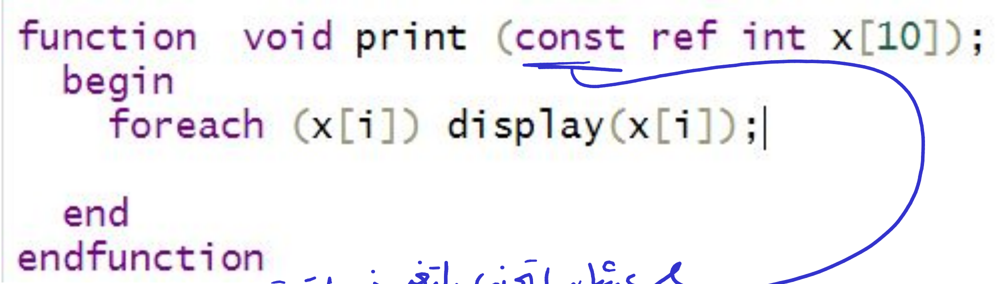
```
function static incr (int x);
      int k = 7;
       k++;
 end function

initial begin
  incr(5);  → K= 8
  incr(6);  → K= 8
  incr(7);  → K= 8
```



ال x وال K (local variables) هيتعرفوا ف أول call بس ال assign K=7

هيتنفذ عادى وكما ن K++

Passing by reference and default values



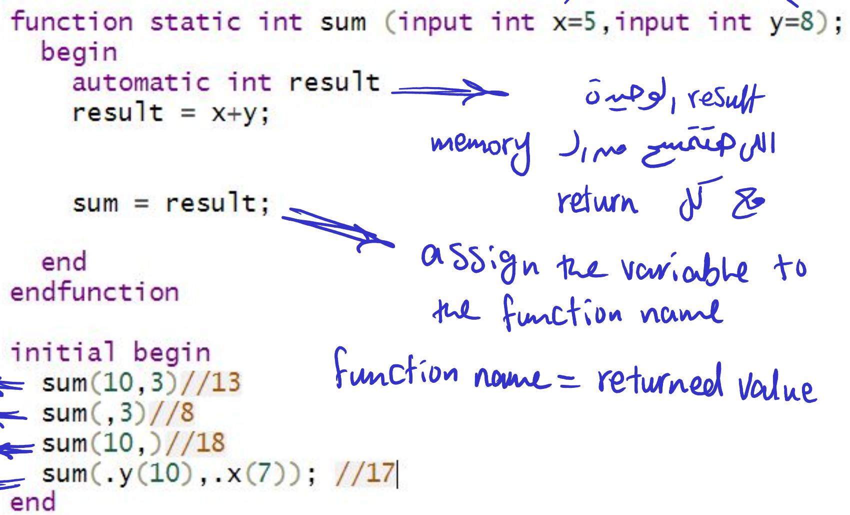
```
function  void print (const ref int x[10]);
  begin
    foreach (x[i]) display(x[i]);

  end
endfunction
```

عشان اتجنب التغير في القيمة الحقيقية

الي x Point ليه

Compilation error ولو غيرت فيه هيدي

default values



```
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
```

result لوحدة الي هتتمسح من memory مع كل return

assign the variable to the function name

function name = returned value

10+3

5+3

10+8

7+10

Tasks

task int sum ( ); ✗

no return data type

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

[1] end the execution of function
[2] delete the space of task in the memory
[3] does not return anything

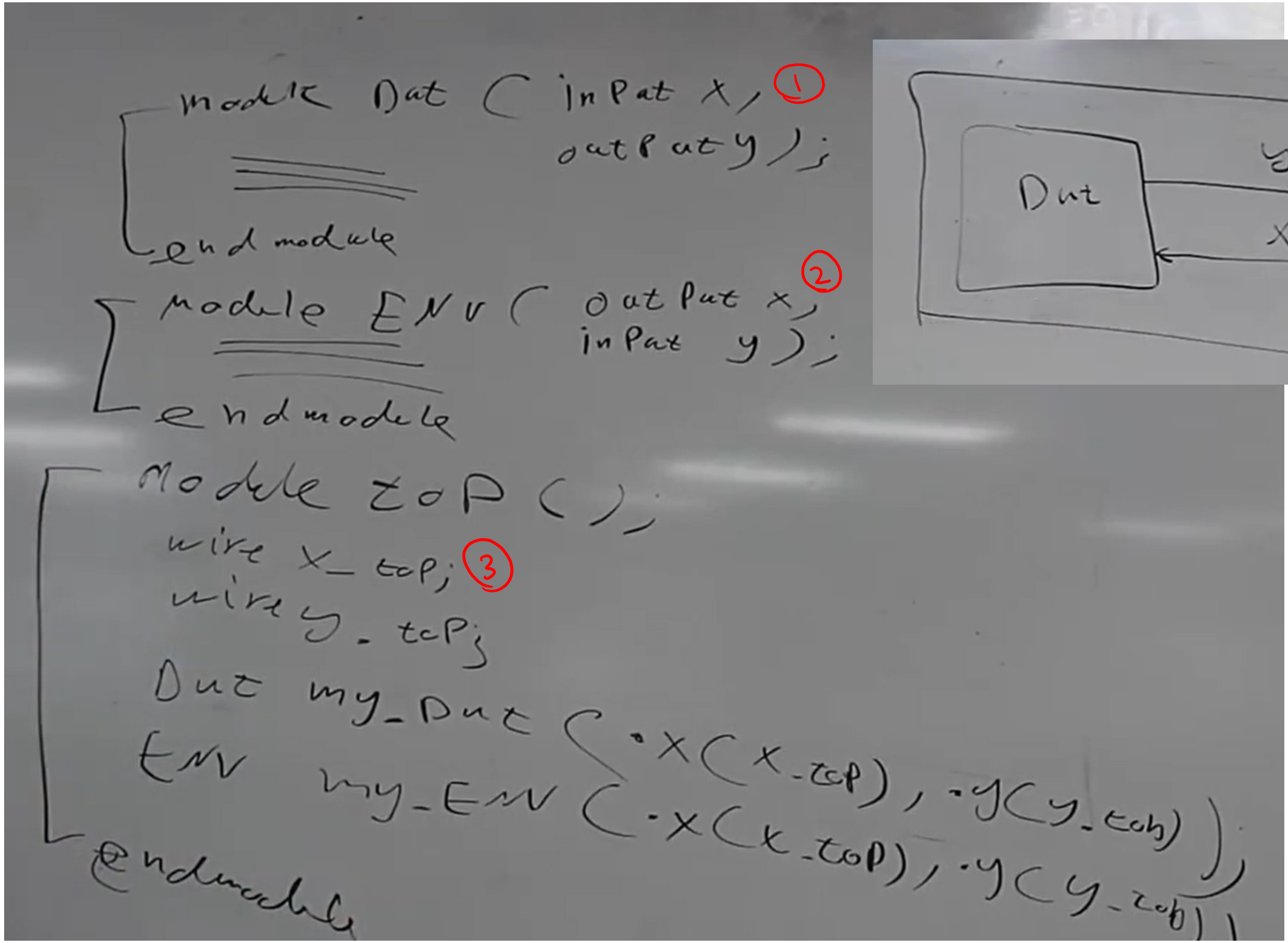
→ function can call another function, but cannot call task
→ Task can call function or another task

Sheet

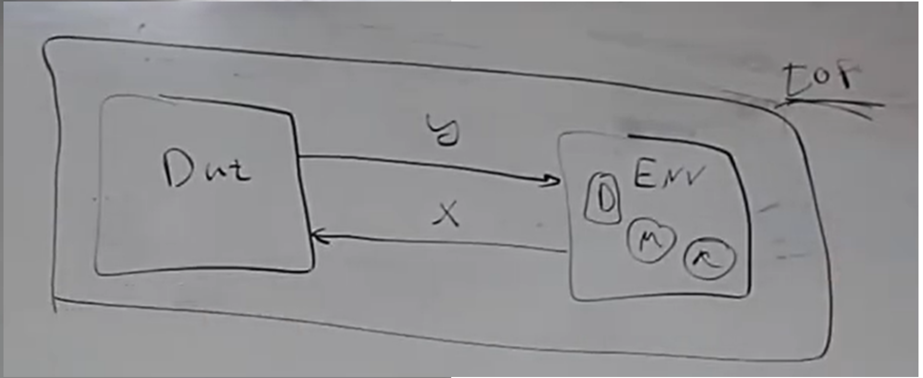
1. What are void functions? The function that does not return any value.
2. Explain about pass by ref and pass by value?
3. What is the concept of a "ref" and "const ref" argument in System Verilog function or task?
   "ref": allowing the function to modify the value of the argument
   "const ref": A function is not allowed to modify the value of the argument
4. Is it possible for a function to return an array? No, cannot directly return an array, but can define array type then can return this type
   ex
   typdef int x[5] arr_5;
   function arr_5 sum();
5. How to make sure that a function argument passed as ref is not changed by the function? Use "const ref"

2. Pass by ref : 1- Passing the memory address of the actual Parameter to the function
   2 - Changes made to the Parameter inside the function directly affect the original value

   Pass by value : 1- making a copy of the actual Parameter's value and Passing it to the function
   2- Changes made to the Parameter inside the function do not affect the original value



```
module Dut ( input x,   1
             output y);
endmodule

module ENV ( output x,   2
             input y);
endmodule

module top ();
  wire x_top;   3
  wire y_top;
  Dut my_Dut (.x(x_top), .y(y_top));
  ENV   my_ENV (.x(x_top), .y(y_top));
endmodule
```

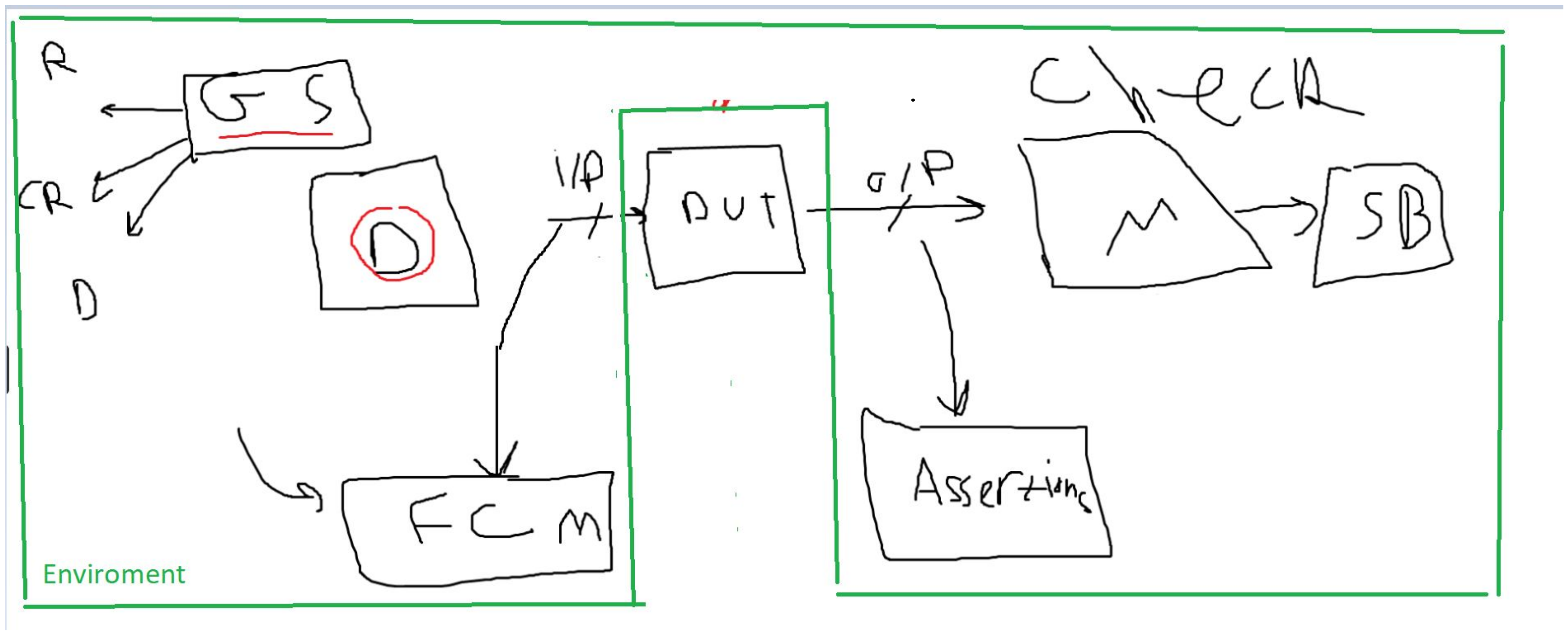


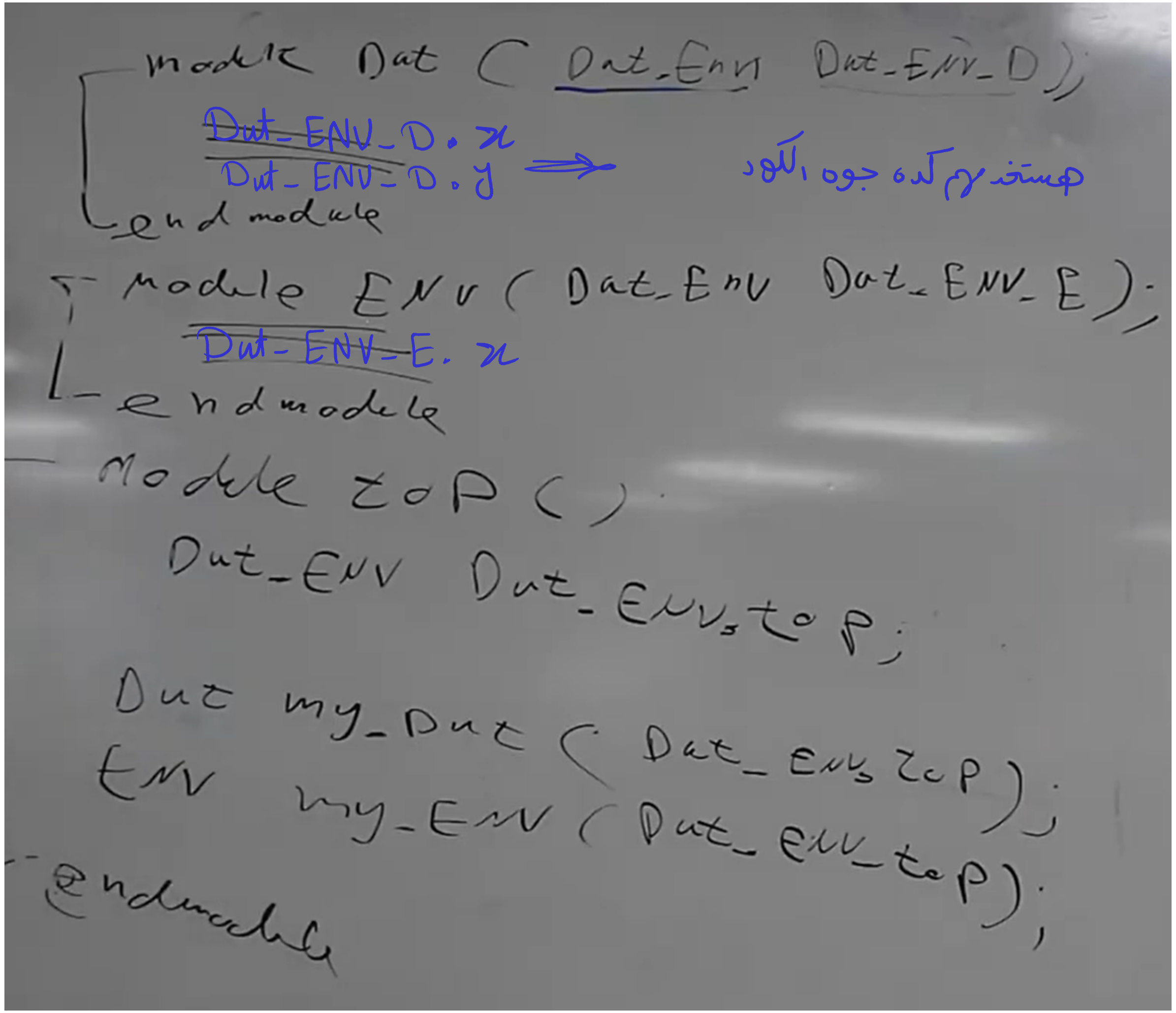
مشكلة الطريقة دى فى الربط
بينهم إن لو قررت أغير حاجة
لازم أغيرها ٣ مرات
مثلاً لو حبيت أغير الـ x
لازم أغيرها في كل module
وده صعب بسبب إن عدد
الـ signals كبير والمتغيرات كتير
الحل ← نعمل interface

interfaces

هدفه ياخد الـConnection المتكررة ٣ مرات و نشيلها ٣ مرات

```
interface Dut_Env
  wire x;
  wire y;
end interface
```





```
module Dut ( Dut_Env Dut_ENV_D );
    ~~Dut_ENV_D . x~~
    ~~Dut_ENV_D . y~~  →  هستخدم كده جوه الكود
endmodule

module ENV ( Dut_Env Dut_ENV_E );
    ~~Dut_ENV_E . x~~
endmodule

module top ( );
    Dut_ENV Dut_ENV_top;

    Dut my_Dut ( Dut_ENV_top );
    ENV    my_ENV ( Dut_ENV_top );
endmodule
```

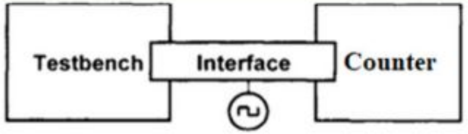
تعالى نقوله عندي كذا interface
وكلهم فيهم نفس اسم ال variable

interface_name . variable_name

كده حددت عايز أنهي interface
وأنهي variable فيه

Modports

DUT
بحدد directions من وجه نظر الميزان



Interface

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

Testbench

```
module test(count_ifc x);
  initial begin
    x.P <= 4'b0111;
    x.MR <= 1'b0;
    x.Enable <= 1'b1;
    x.Load <= 1'b0;
    #3 x.MR <= 1'b1;
    #6 x.MR <= 1'b0;
    #43 x.Enable <= 1'b0;
    #15 x.Enable <= 1'b1;
    #16 x.Load <= 1'b1;
    #9 x.Load <= 1'b0;
  end
endmodule
```

Design

```
module decade_counter (count_ifc y);
  always @(y.MR) begin
    if (y.MR)
      y.Q <= 4'b0000;
  end

  always @(posedge y.CLK) begin
    if (!y.MR)
      if (y.Load)
        y.Q <= y.P;
    else if(y.Enable)
        y.Q <= (y.Q+1) % 10;
  end
endmodule
```

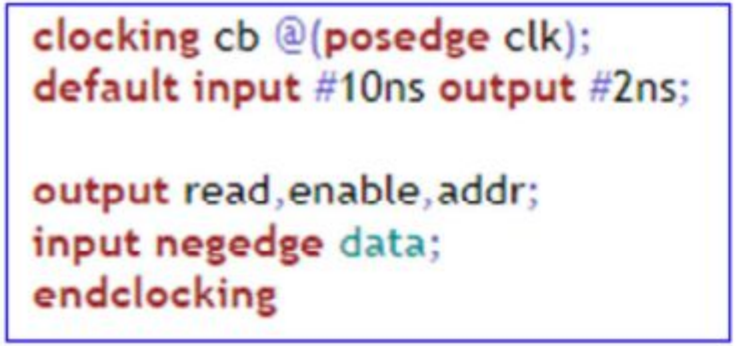
Top module

```
module top;
  bit clk;
  always #5 clk <= ~clk;
  count_ifc ifc(clk);
  decade_counter u1(ifc.dut);
  test u2(ifc.driver);

  initial begin
    $dumpfile("counter.vcd");
    $dumpvars;
    #200 $finish;
  end
endmodule
```

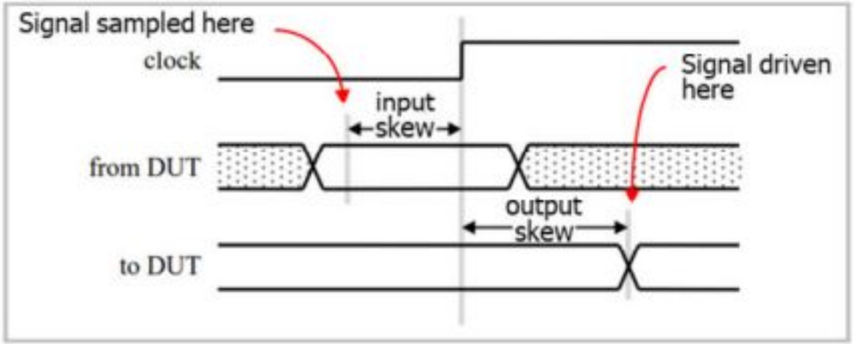
ENV
بحدد directions من وجهة نظر الـ Environment

Clocking blocks

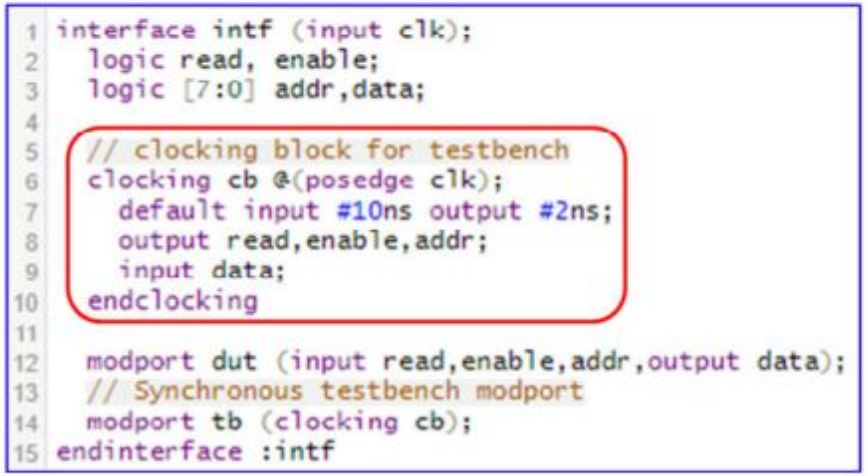
بديل للـ Synthesizer



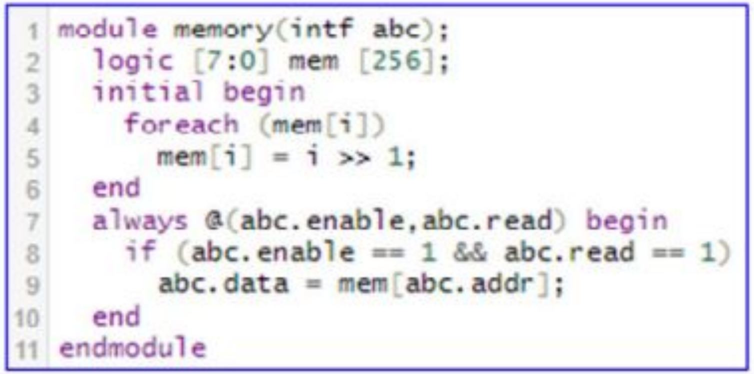
```
clocking cb @(posedge clk);
default input #10ns output #2ns;

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



Clocking blocks

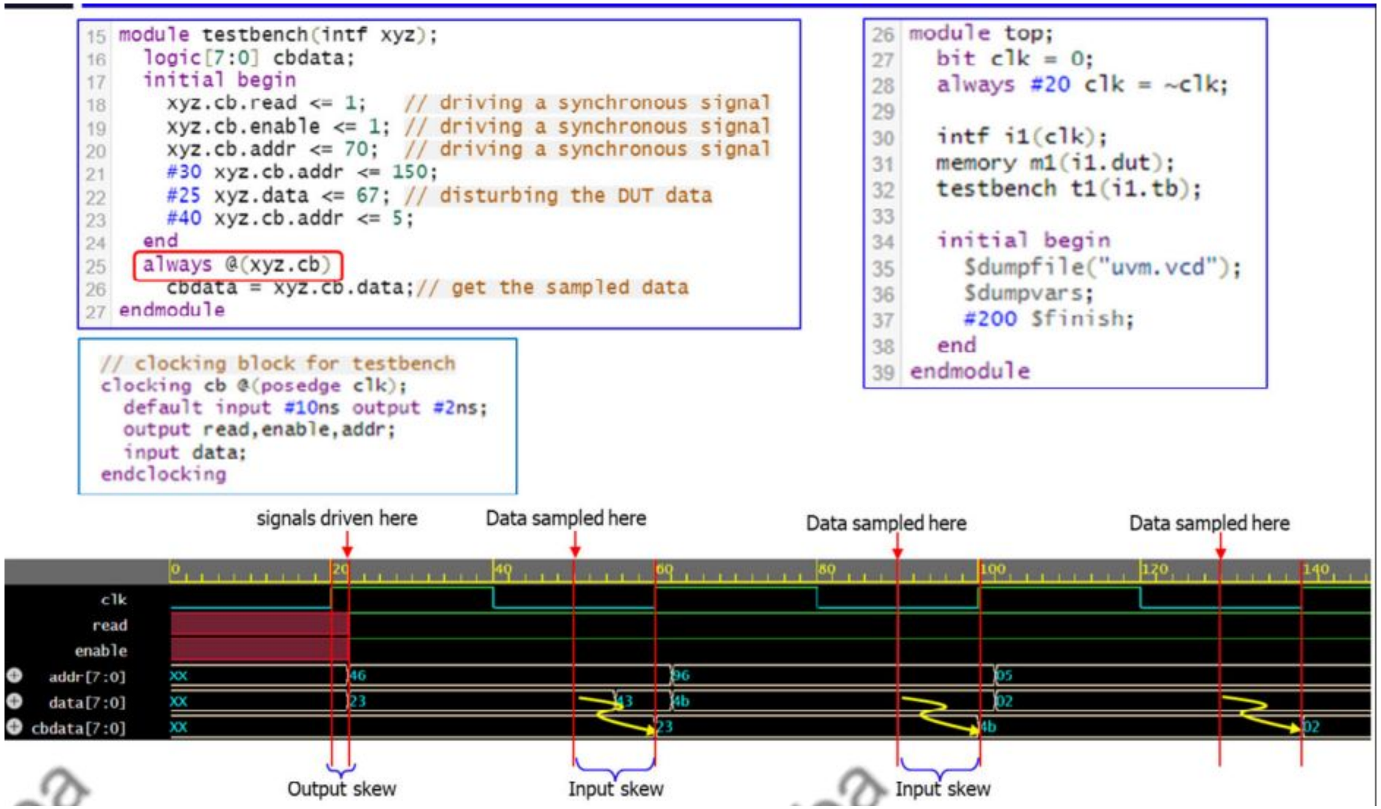


```
interface intf (input clk);
  logic read, enable;
  logic [7:0] addr,data;

  // 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

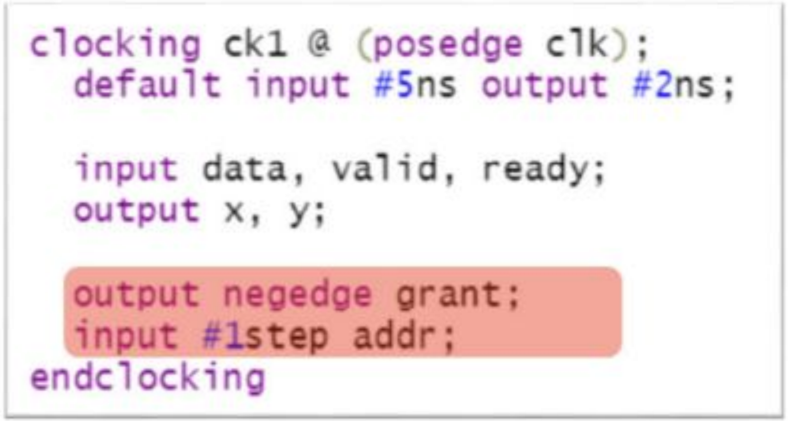


```
module testbench(intf xyz);
  logic[7:0] cbdata;
  initial begin
    xyz.cb.read <= 1;    // driving a synchronous signal
    xyz.cb.enable <= 1;  // driving a synchronous signal
    xyz.cb.addr <= 70;   // driving a synchronous signal
    #30 xyz.cb.addr <= 150;
    #25 xyz.data <= 67;  // disturbing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

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

```
module top;
  bit clk = 0;
  always #20 clk = ~clk;

  intf i1(clk);
  memory m1(i1.dut);
  testbench t1(i1.tb);

  initial begin
    $dumpfile("uvm.vcd");
    $dumpvars;
    #200 $finish;
  end
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
```