Finall 22

A. State briefly what is the difference between _testing_ and _verification_?
B. Make a comparison between _code coverage_ and _functional coverage_.
C. Comment briefly on the following two statements (agree or disagree and why):
- 100% functional coverage ensures that the design can have no bugs.
- 100% code coverage guarantees that the test bench is very well written.

Ⓐ • Testing: occurs after fabrication on all chips

• verification: occurs during the implementation Process after each stage to make sure that the design is a bug-free.

Ⓑ • Code coverage: It's a metric of the code covered during simulation

① Statement ② Branch ③ Condition
④ expression ⑤ Toggle ⑥ FSM

• functional coverage: It covers the functionality of the design based on design specification

Ⓒ - No, if there are some code lines that are not tested then it may exist some features were not included in the verification Plan. So, we may have a bug-free design but not Completed yet.

- No, there are some features need to be tested in a specific Path, for example: for loop must be tested when it executes certain number of iterations, however only one iteration will achieve statement coverage.

D. One of the practical problems in simulation-based verification is the large input space. State briefly how the problem is tackled in real life.
E. State the differences between *formal verification* and *simulation-based verification*
F. What are the two main types of formal verification?
G. What is the difference between *rand* and *randc* in System Verilog?

(D) we make a randomization inputs To try covering all Possibilities of inPuts.
and then make a directed tests to cover values that were not covered.

(E)
- Formal verification : Method to Prove or disprove a design implementation against a formal specification or Property.
  uses mathematical reasoning
- simulation-based : generate InPut sequence, then simulate the DUT to generate outPuts and verify by compare the DUT outPuts against the Golden outPut.

(F)
① Equivalence checking + شوية شرح
② Model checking + شوية شرح

(G) rand : العادية بتاعتنا

randc : (cyclic) after randomization, the same values will be generated again but only after all other values have been generated.

Question 2: [ILO: c1, c2] [10 marks]

1. Logic variables can have multiple drivers
   a. True [✓ b. False]

2. Target to the completion of Verification process is
   [✓ a.] Functional coverage 100% and code coverage is 100%
   b. If all the test cases ran
   c. Functional coverage 100% and code coverage is not considered
   d. Code coverage should be 100% and functional coverage is not considered

3. *array [4][8]*. This array declaration type is:
   [✓ a.] Compact declaration
   b. Verbose declaration
   c. Single dimension array declaration
   d. None of the above

   array $\underbrace{[3:0][7:0]}_{\text{Verbose}}$

4. *modport* is used to
   a. declare input and output skew
   [✓ b.] declare direction of signals
   c. synchronize signals
   d. make interface complex

C. How many bins are created in following example for coverpoint cp_x ?

```
bit[3:0] var_x;
covergroup test_cg @(posedge clk);
   cp_x : coverpoint var_x {
   bins low_bins[] = {[0:3]};     → 4
   bins med_bins = {[4:12]};     → 1
}
```

total = 5 bins

D. What are the transitions covered by following coverpoint?

```
coverpoint my_variable {
  bins trans_bin[] = ( a,b,c => x, y);
}
```

a ⇒ x   b ⇒ x   c ⇒ x
a ⇒ y   b ⇒ y   c ⇒ y

E. Is this a correct code? State the reason.

not correct

```
module test();
  enum {a=0, b=7, c, d=8} alphabet;
endmodule
```

Typedef enum{a=0, b=7, c, d=8}

c & d have the same value

c=8
because b=7

Question 4: **[ILO: b1, b2, c1, c2] [20 marks]**

A. Write *concurrent* SVA to express the following:

① • Every time a request signal *req* is high (at the rising edge of the clock), an acknowledgement signal *ack* becomes high exactly 3 clocks later.

② • Every time (at the rising edge of the clock) the valid signal *vld* is high, the *cnt* signal is incremented by 1.

①
```
assert property (@(posedge clk)
        req |-> ##3 ack ) ;
```

②
```
assert property (@(posedge clk)
        vld |-> (cnt == $past(cnt) + 1) ) ;
```

B. Two signal lines *r1* and *r2* are mutually exclusive (i,e both of them cannot be high at the same time). The following assertion is used to take care of that. State whether this assertion is right or wrong with justification.

```
assert (tr.r1 ^ tr.r2 );
```

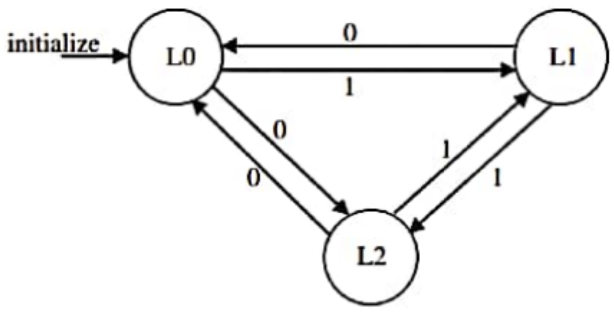
r1	xor	r2	out	
0		0	0	✗
0		1	1	✓
1		0	1	✓
1		1	0	✓

ال case دي بالذات
ال assert غلط

```
assert(not(tr.r1 && tr.r2));
```

r1	and	r2	not(out)	
0		0	1	✓
0		1	1	✓
1		0	1	✓
1		1	0	✓

C. A finite state machine is described by the following state diagram. It has one input *pi*, a clock signal *clk*, and one signal called *initialize*, which puts the machine in the state L0 whenever it is active. Write a coverage group to check whether all the transitions have been exercised.



```
covergroup   CovPort;
   covePoint   ifc.state
      { bins   L0_trans [ ] = (L0 => L1 , L2);
        bins   L1_trans [ ] = (L1 => L0, L2 );
        bins   L2_trans [ ] = (L2 => L0 , L1 );
      }
```

D. What is the maximum coverage you can get from the following coverage group? How to increase the maximum coverage in this case?

```
class Transaction;
  rand bit [2:0] p;
  rand bit [3:0] k;
endclass

Transaction tr;

covergroup CovPort;
  CP1 : coverpoint (tr.p + tr.k);
endgroup
```

→ 3 bits → max 7

→ 4 bits → max 15

3-bits + 4-bits → 4-bits

max 15

max 7+15 = 22

I have only <u>15 bins</u>, but I can get values up to <u>22</u>.

then I can get 100 % coverage but values from 16 to 22 may be not generated.

So, the real coverage I can see is only
16 bins from 23 bins : $\frac{16}{23} * 100\ \%$

To handle this :

```
coverPoint (tr.P + tr.K + 5'b 00000)
    { bins  my_bin [] = { [0 : 22] } ;
     ignore_bins  x_bin  = { [23 : 32] } ;
    }
```

ومش لازم اكتب ديه

بس كده