CSE 313s
Selected Topics in Computer Engineering
Sheet 3

1. What is coverage?

Coverage is the metric of completeness of verification:

الcoverage هو قياس لأكتمال الverification بتاعي, بمعنى إني عندي features عايز اعملها test والdesign فيه سطور كتير, انا محتاج ابقى عارف الtestbench بتاعي ده غطى قد ايه من الfeatures اللي الdesign بيعملها, هل قدر اعمل test لكل الfeatures ولا لا.. وهل قدرت بالtest بتاعي اعدي على كل الconditions والstatements اللي في الdesign ولا لا..
كل ده بعرفه عن طريق الcoverage.

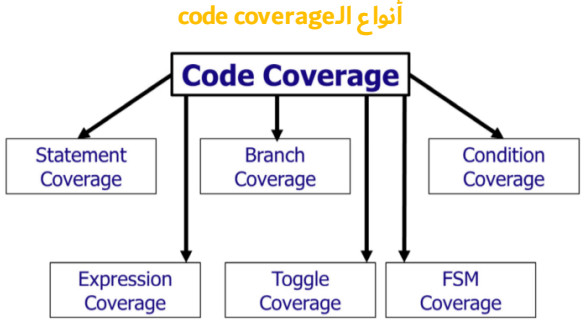
---

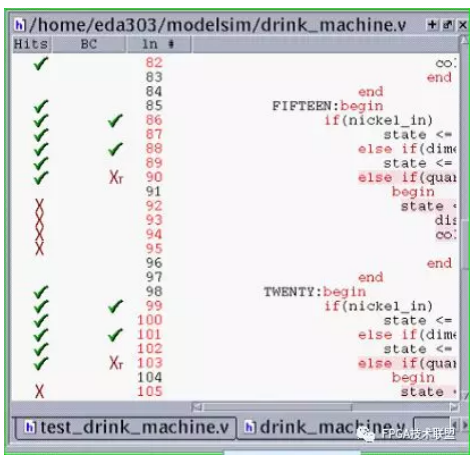
2. What is the meaning of code coverage?

Code coverage is a metric of the code covered during simulation.

الcode coverage هو أحد نوعين الcoverage, هو مهتم بقياس انت قد تغطي قد ايه من الstatements, branches, conditions وهكذا من الdesign code
بمعنى تاني, انت عايز تتأكد إن من خلال الtestbench بتاعك ده, قدرت تdrive stimulus مختلفة تحقق كل جوانب الdesign ولكن كcode مش كfunctionality

انواع الcode coverage





هنا مثال على الcode coverage, اللي جمبه علامة الصح
معناه إن الtestbench بتاعي خلاه الdesign يمر بالسطر ده
واللي جمبه علامة غلط معناه إن الtestbench مخلاش الdesign
يعدي على السطور دي

---

3. What is a coverage driven verification?

Coverage-driven verification (CDV) is a powerful methodology that can help you achieve your verification goals faster and more efficiently.
CDV combines the use of coverage metrics, constrained-random stimuli, and functional coverage models to guide and measure the verification process.

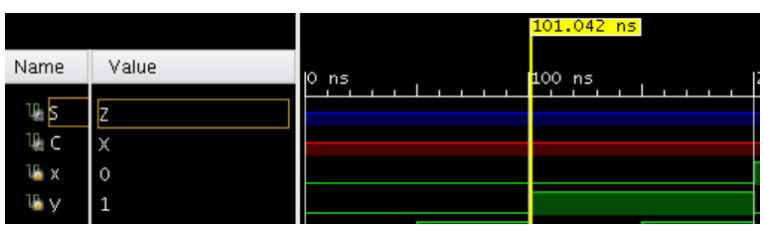
سريعًا كده, الCDV هي عبارة عن مهنجية او اسلوب في كتابة الtestbench بشكل معين من خلاله تقدر تعمل verification بكافئة اكبر وتحقق coverage اعلى وبشكل standardized

---

4. What is the difference between X and Z in verilog?

X: represents an unknown logic value (Either 0 or 1) معناها إن الsignal في الوقت الحالي غير معروف قيمتها, ممكن تكون uninitialized مثلًا

Z: represents a high-impedance state i.e., not connected to '0' or '1' left floating (Neither 0 nor 1) معناها إن الsignal مش متوصلة اصلًا



---

5. Which coverage has more importance, code coverage or functional coverage?

Both of them have equal importance in the verification. 100% functional coverage does not mean that the DUT is completly exercised and vice-versa.
Verification engineers will consider both coverages to measure the verifcation progress.

كلاهما مهم ومفيش واحد يغني عن التاني لأن كل واحد بيقيس حاجة مختلفة يعتبر, خصوصًا إني بعمل اللي بيكتب الtestbench ده مهندس مختلف عن اللي بيكتب الdesign
فممكن يكون الdesigner عامل design غلط او ممكن يكون الverification engineer هو اللي عامل الtestbench غلط.
فعن طريق الأتنين بنقدر نشوف الصورة كاملة وممكن نقيم الموقف بشكل سليم.

	RTL	Verification Environment
1) Code coverage = 50 Functional coverage = 100	Useless code	Features missing from verification plan
2) Code coverage = 100 Functional coverage = 50	Missing features	Need more testing

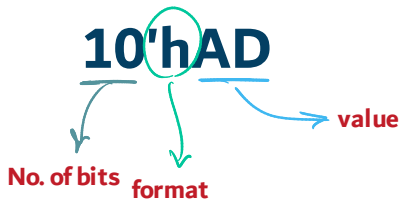
6. What is the difference between the following two declarations?

```
wire [7:0] w1;
wire [0:7] w2;
```

الفرق بينهم يعتبر نفس فكرة الendiannessلو فاكر معناها, وهو ترتيب الbits من اليمين للشمال ولا من الشمال لليمين, بمعنى آخر, اين تقع الMSB؟
بالنسبة لw1 فالMSB تقع على الشمال خالص في الleft-most وبالنسبة لw2 فالLSB تقع على اليمين خالص في الright-most

---

7. What is 10'had?



AD is a hexadecimal number represented in 10 bits.

- Since AD can be represented only in 8 bits, then the remaining 2 bits will be padded with zeores.
- Consider a case if the value where 10'hADE, which's represented in 12-bits, so the extra 2 bits will be truncated.

10'hAD = 1010 1101 = 0010101101

10'hADE = 1010 1101 1110 = 1011011110

---

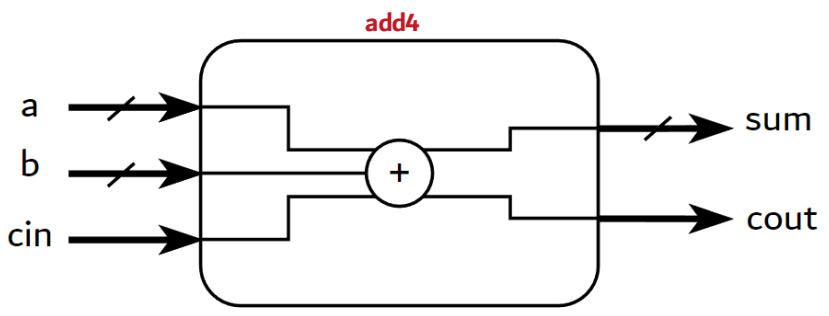
behavioral and structural

8. Write a Verilog ~~behavioral~~ description of a four-bit adder module. The adder should have three inputs, a, b, and c_{in}, and two outputs, sum and cout. Ports c_{in} and c_{out} are one bit, the other ports are four bits each.

Case (1): Behavioral Description

الطريقة دي في كتابة الcode, انا بوصف الcircuit بتاعتي من الbehavior او الfunctionality بصرف النظر عن الgates اللي بتكون الcircuit بتاعتي.
وبعدين بقى الsynthesis tool هي اللي تبقى تgenerate الgates وتعمل optimization لو فيه وكده

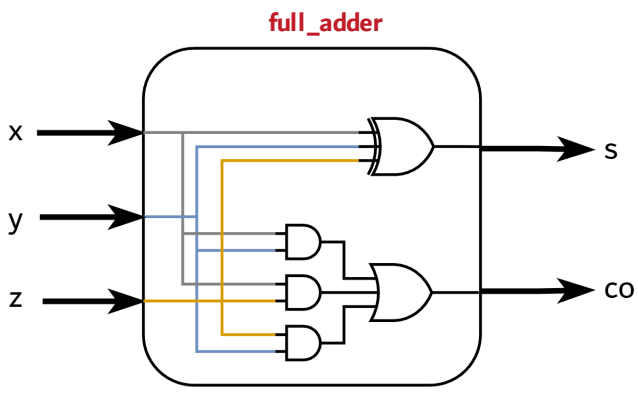
```
module add4 ( output  wire  [3:0] sum,
              output  wire        cout,
              input   wire  [3:0] a,
              input   wire  [3:0] b,
              input   wire        cin );

  assign {cout,sum} = a + b + cin;

endmodule
```

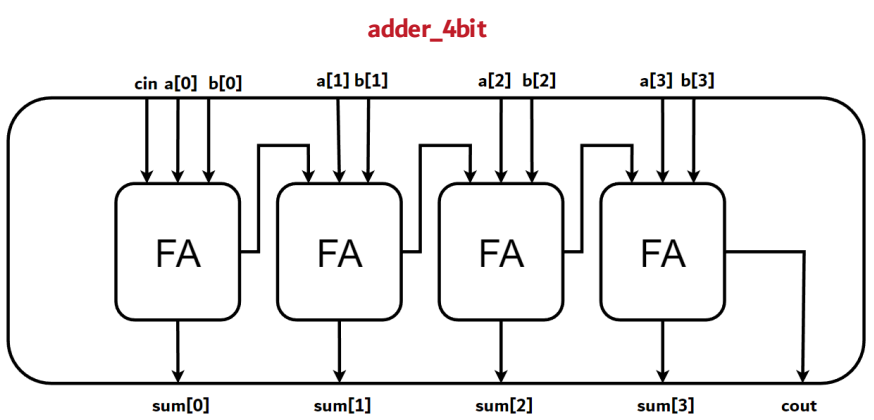


Case (2): Structural Description

الطريقة دي في كتابة الcode, انا بوصف الcircuit بتاعتي من حيث الgates المكونة ليها

```
module full_adder ( output  wire  s,
                    output  wire  co,
                    input   wire  x,
                    input   wire  y,
                    input   wire  z );

  assign s  = x ^ y ^ z;
  assign co = (x&y) | (x&z) | (y&z);

endmodule

//=========================================

module adder_4bit ( output  wire  [3:0] sum,
                    output  wire        cout,
                    input   wire  [3:0] a,
                    input   wire  [3:0] b,
                    input   wire  [3:0] cin );

wire [2:0] carry;

full_adder U0 (sum[0], carry[0], a[0], b[0], cin);
full_adder U1 (sum[1], carry[1], a[1], b[1], carry[0]);
full_adder U2 (sum[2], carry[2], a[2], b[2], carry[1]);
full_adder U3 (sum[3], cout    , a[3], b[3], carry[2]);

endmodule
```





9. Write a Verilog structural description of an eight-bit adder that uses two of the four-bit adders of the previous problem. (That is, instantiate the modules designed in the previous problem)

```
module add8 ( output  wire  [7:0] sum,
              output  wire        cout,
              input   wire  [7:0] a,
              input   wire  [7:0] b,
              input   wire        cin );

wire carry;

add4 U1 (sum[3:0], carry, a[3:0], b[3:0], cin);
add4 U2 (sum[4:7], cout , a[4:7], b[4:7], carry);

endmodule
```

---

10. An accumulator has three inputs, *amt*, *reset*, and *clk*, and an output, *sum*. The accumulator has an internal 32-bit register which is updated as follows:
    - On a positive edge of *clk* it adds *amt*, a 32-bit integer, to the register;
    - On the negative edge of *clk* it places the new *sum* on its outputs (until the next negative edge).
    - Whenever *reset* is high the register is set to zero and the output changes immediately.

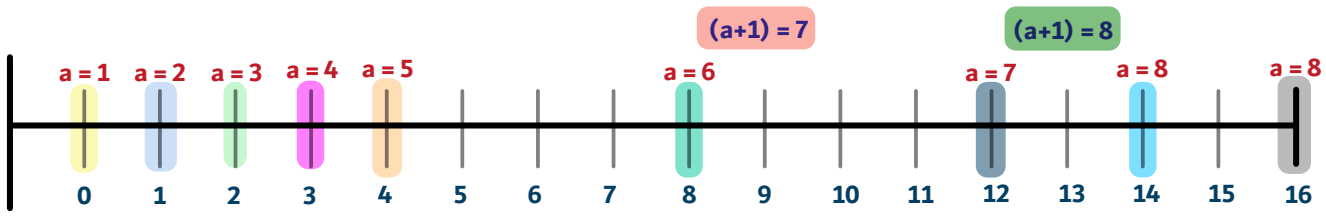
    Write a Verilog behavioral description of this module.

```
module accumulator (  input   wire [31:0] data_input,
                      input   wire        clk,
                      input   wire        rst,
                      output  reg [31:0]  accumulated_sum );

reg [31:0] internal_accumulator;

always @(posedge clk or posedge rst)
begin
    if (rst)
        internal_accumulator <= 32'b0;
    else
        internal_accumulator <= internal_accumulator + data_input;
end

always @(negedge clk or posedge rst)
begin
    if (rst)
        accumulated_sum <= 32'b0;
    else
        accumulated_sum <= internal_accumulator;
end

endmodule
```

---

11. The code below starts executing a t = 0. Show all changes in a, include the time of the change and the new value.

```
integer a;
initial begin
  a = 1;
  #1;
  a = 2;
  #1;
  a = 3;
  #1;
  a <= 4;
  #1;
  a <= 5;
  #1;
  #3 a = a+1;
  #1;
  a = #3 a+1;
  #1;
  a <= #3 a+1;
  #1;
  a = a+1;
end
```

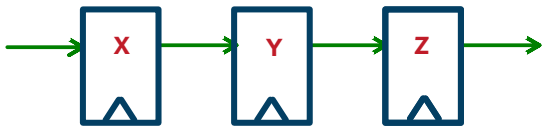


12. The programmer expected execution to exit the loop below when either i was 1000 or a[i] == c, but that's not what happened. What goes wrong and how can it be fixed? The loop must be exited using a *disable* statement.

```
//...
while (i < 1000) begin : LOOP
  i = i + 1; // Increment i first
  if (i >= 1000 || a[i] == c) disable LOOP; // Check if i is out of bounds or if a[i] is equal to c
end
```

---

13. For the Verilog code segment below, which of the lines implement a shift register?




```
1 always@ (posedge clk)
2   begin
3      z = y; y = x;
4      y1= x1; z1 = y1;
5      z2<= y2; y2 <= x2;
6      y3<= x3; z3 <= y3;
```

In order to implement a shift register, We need to assign the old value of Y to Z then the old value of X to Y

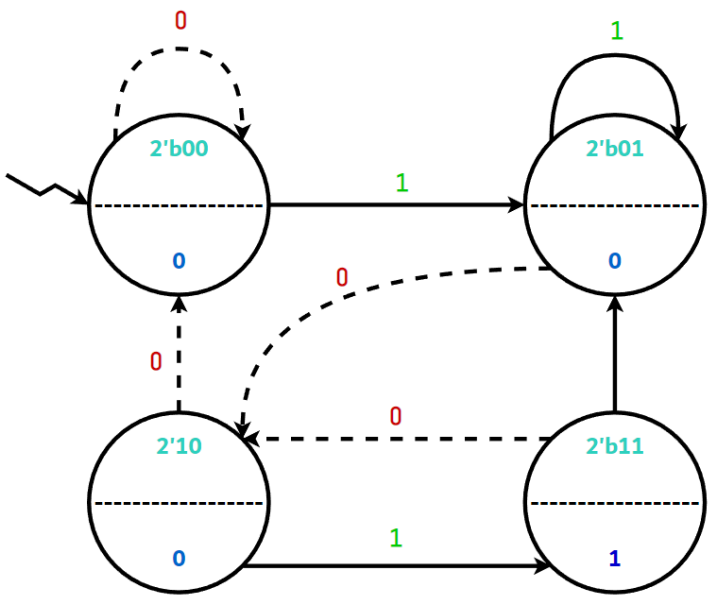
- in case of blocking assignment: Z should be updated before Y
- in case of non-blocking assignment: it doesn't matter

---

14. If a variable is not assigned in all possible executions of an always statement then:
    a. A don't care is inferred
    b. A latch is inferred
    c. The variable is set to 0
    d. The synthesis process will fail

---

16. A recognizer has one input "X" and one output "Y". At each clock cycle, the input "X" value is read. When a sequence of "101" is observed in the input sequence, the output, "Y", will become 1, otherwise it will be 0.
    a. Draw Moore state machine diagram with minimum number of states.
    b. Write the Verilog representation of your Moore state machine.

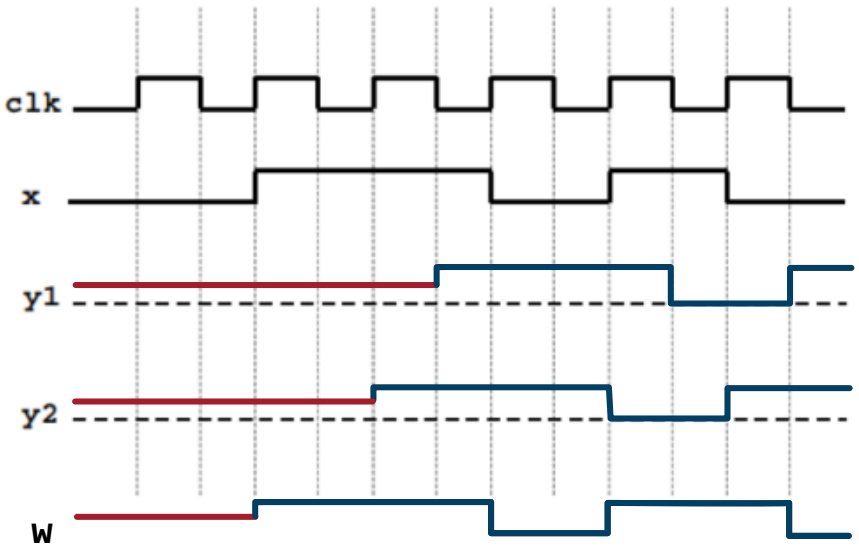



```
module seq_det (  output  reg   y,
                  input   wire  rst,
                  input   wire  clk,
                  input   wire  x);

reg [1:0] state;

always @(posedge clk or posedge rst) begin
  if (rst) begin
    state <= 2'b00;
  end
  else begin
    case({state,x})
    3'b000: state <= 2'b00;
    3'b001: state <= 2'b01;
    3'b010: state <= 2'b10;
    3'b011: state <= 2'b01;
    3'b100: state <= 2'b00;
    3'b101: state <= 2'b11;
    3'b110: state <= 2'b10;
    3'b111: state <= 2'b01;
    endcase
  end
end

always @(*) begin
  if(state == 2'b11)
    y = 1'b1;
  else
    y = 1'b0;
end

endmodule
```

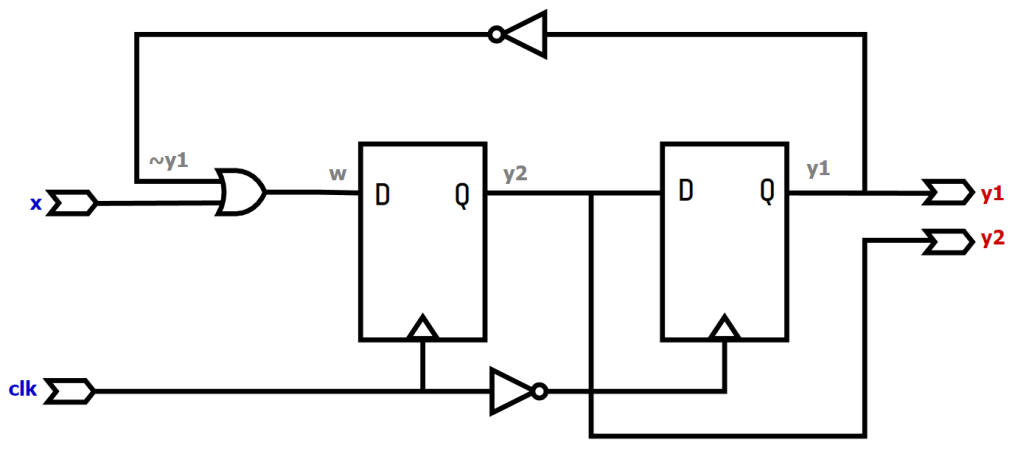
17. Use the following Verilog module to answer the question below:

```
1  module my_unit(clk, x, y1, y2);
2    input clk, x;
3    output y1, y2;
4    reg y1, y2;
5    wire w;
6
7    assign w = x|(~y1);
8    always@(posedge clk)
9      y2 <= w;
10
11   always@(negedge clk)
12     y1 <=y2;
13 endmodule
```

a. Draw the output waveforms for the specified inputs.



b. Draw the logic diagram for the circuit that represents **my_unit**



---

18. Your job is to design a 3-bit ALU for the specifications in the table below. This unit has a two bit control lines (P_1 P_0), to select the required operation, and 3-bit input data D[2:0]. The output lines are Q[2:0].

P_1 P_0	Operation
0 0	Q = D
0 1	Q = Q' (New Q is the complement of the current Q)
1 0	Q = 2Q (New Q is twice the current Q)
1 1	Q = 2Q + 1 (New Q is twice the current Q plus 1)

```
module alu  ( input   wire        clk,
              input   wire        rst,
              input   wire  [1:0] ctrl,
              input   wire  [2:0] d_in,
              output  reg   [2:0] q_out);

always @(posedge clk or posedge rst) begin
  if(rst)
    q_out <= 3'b000;
  else begin
    case(ctrl)
      2'b00:  q_out <= d_in;
      2'b01:  q_out <= ~q_out;
      2'b10:  q_out <= q_out<<1;
      2'b11:  q_out <= (q_out<<1)+1'b1;
    endcase
  end
end

endmodule
```

Juba