Homework 7 Solutions

1. **Blocking assignment:** This is denoted by ‘=’ symbol while assigning a value to a certain variable. It works as expected of a general assignment, evaluates the right-hand side and assigns it to the left-hand side variable.

This is used when we want to create a combinational logic.

Example: a = 2;

Non-blocking assignment: This is denoted by ‘<=’ symbol while assigning a value to a certain variable. It evaluates the right-hand side and assigns it to the left-hand side variable after some delay (usually happens at the end of the time step in simulation). So, until that time, if any other statements get executed, they will use the previous value of the left-hand side variable only.

This is used when we want to create a sequential logic.

Example: a<=3;

2.

(a) 6’b010000

(b) 6’b000011

(c) 7’b1111100

(d) 5’b10000

3.

(a) Concatenation Operator

(b) out1 = in*4 (Multiplication by 4)

(c) out2 = in/4 (Division by 4)

4. Answer (c) - The last value assigned will be the final value of the signal.

5.

Behavioral Model:

```
module circuit (input panic, input enable, input exiting, input window, input door, input garage, output alarm);

reg secure;

reg temp;
```

```
always@(window, panic, enable, exiting, door, garage) begin

secure = window & door & garage; //should not be using <= here unless there is a need of
sequential operation

temp = ~secure & ~exiting & enable;

alarm = temp | panic;

end

endmodule
```

Dataflow Model:

```
module circuit_d (input panic, input enable, input exiting, input window, input door, input
garage, output alarm);

reg secure;

reg temp;

assign secure = window & door & garage;

assign temp = ~secure & ~exiting & enable;

assign alarm = temp | panic;

endmodule
```

6.

```
// Let signal ‘a’ be the enable signal for buffer, ‘c’ be the input signal to buffer and ‘y’ be the
// output signal from the tristate buffer
```

Dataflow Model:

```
module tristate_buffer(a,c,y);

        input a,c;

        output y;

        assign y = c ? a : 1'bz;

endmodule
```

Behavioral Model:

```
Module tristate(a,c,y);

input a, c ;

output y;

reg  y;
```

```
always@ (a,c) begin
if(c)
y = a;
else
y = 1’bz;
end
endmodule
```

7.

From the given function, we can get the Sum of Product expression as C = AB + A’B’ as we can observe that C = 1 when (A = 0 = B) or (A = 1 = B).

Now, we can write the structural model for this function Y as below.

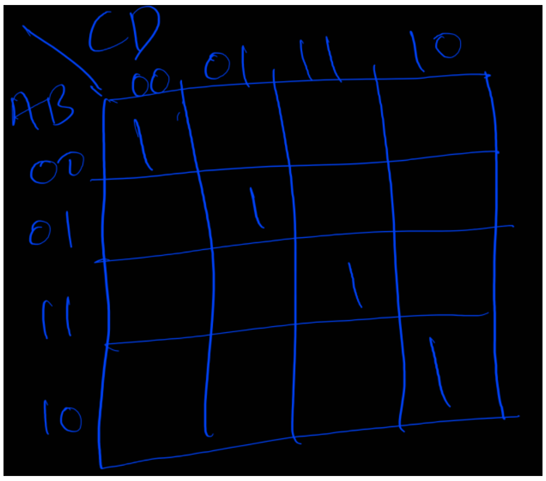
```
module fun  (A, B, C);
 input A, B;
 output C;
 wire X, Y, A_NOT, B_NOT;
  not A_INV (A_NOT, A);
  not B_INV (B_NOT, B);
  and A1 (X, A, B);
  and A2 (Y, A_NOT, B_NOT);
  or OUT (C, X, Y);
endmodule
```

8.

The output is ‘0’, as they take ‘X’ as the unknown state rather than 1/0 and they can’t compare x without === and anything other than sel = 1 will be thrown into ‘else’ condition giving output as 0.

9.

The lock value can be any value between 0(4’b0000) - 15(4’b1111). The requirement is that the first two digits are equal to the next two digits. Such cases are 0000, 0101, 1010, 1111 - 4 cases are possible for the lock to open. So, we can take a K-Map to implement this logic.



Place 1's as the output wherever these 4 cases occur, simplify and get a Sum of Products expression, as below.

Y = A'B'C'D' + AB'CD' + A'BC'D + ABCD -> Logic to help John open the lock.

10. Verilog module that implements Y.

```
module lock (A, B, C, D, Y);
input A, B, C, D;
output Y;
always@(A, B, C, D) begin
if({A,B} = {C,D}) begin
Y = 1;
end
end
endmodule
```

-> There can be many other ways to code the same problem, any **correct** solution will be rewarded.