HW8

1. Blocking vs non-blocking assignments
   - **Blocking assignment** (=) executes in series because a blocking assignment blocks execution of next statement until it completes. Therefore the results of the next statement may depend on the 1st one being completed.
     + This means evaluation and updating immediately
     + This concept is ideal for **combinational logic circuit** (assigned in always @(*))

   - **Non blocking assignment** (<=) executes in parallel because it describes assignments that all occur at the same time. The result of a statement on 2nd line will not depend on the results of statement on the 1st line. Instead, the 2nd line will execute as the 1st line had not happened yet. This concept will execute inexorably if there is no delay in the program
     + This means evaluating immediately and postpones the update until other planned evaluations in the same time step have been completed.
     + Assigned in always@(posedge clk) used in **sequential circuit**.

2. Case, casex and casez in behavioral model
   The case statement checks if the given expression one among the other expressions inside the list and branches. It's typically accustomed implement a device.
   Case
   A Verilog case statement starts with the case keyword and ends with the endcase keyword.
   The expression within parentheses area unit aiming to be evaluated specifically once and is compared with the list of alternatives inside the order they are written.
   If none of the case things match the given expression, statements within the default item unit of measurement dead. The default statement is non mandatory, and there's only 1 default statement throughout a case statement. Case statements are nested. Execution will exit the case block whereas not doing 1 thing if none of the items match the expression, and a default statement is not given.
   Casex
   In Verilog, there is a casex statement, a variation of the case statement that enables "z", "? " and "x" values to be treated throughout comparison as "don't care" values
   "z", "? " and "x" unit of statement treated as a don't care if they are inside the case expression or if they are inside the case item
   Casez
   In Verilog, there is a casez statement, a variation of the case statement that enables "z" and "?" values to be treated throughout case-comparison as "don't care" values.
   "Z" and "?" unit of measurement treated as a don't care if they are inside the case expression or if they are inside the case item
   When secret writing a case statement with "don't care", use a casez statement and use "?" characters instead of "z" characters inside the case things to purpose "don't care" bits.

3. Sequential circuit with mix blocking and non-blocking assignment
   Based on the example we can separate memory and the combinational circuit and derive the 2-segment code in the below example:

Listing 7.5 Two-segment implementation

```
module ab_ff_2seg
    (
     input wire clk,
     input wire a, b,
     output reg q
    );

    reg q_next;

    // D FF
    always @(posedge clk)
       q <= q_next;

    // combinational circuit
    always @*
       q_next = a & b;

endmodule
```

Alternatively, we can combine the 2 segments and describe the circuit in a single always block. On this example they use 6 attempts with various combinations of blocking and nonblocking assignments

block. Six attempts, with various combinations of blocking and nonblocking assignments, are made in Listing 7.6.

Listing 7.6 Mixed assignment example

```
module ab_ff_all
   (
    input wire clk,
    input wire a, b,
    output reg q0, q1, q2, q3, q4, q5
   );

   reg ab0, ab1, ab2, ab3, ab4, ab5;

   // attempt 0
   always @(posedge clk)
   begin
      ab0 = a & b;
      q0 <= ab0;
   end

   // attempt 1
   always @(posedge clk)
   begin                  // ab1_entry = ab1; q1_entry = q1;
      ab1 <= a & b;       // ab1_exit = a & b
      q1 <= ab1;          // q1_exit = ab1_entry
   end                    // ab1 = ab1_exit; q1 = q1_exit

   // attempt 2
   always @(posedge clk)
   begin
      ab2 = a & b;
      q2 = ab2;

   end

   // attempt 3 (switch the order of attempt 0)
   always @(posedge clk)
   begin
      q3 <= ab3;
      ab3 = a & b;
   end

   // attempt 4 (switch the order of attempt 1)
   always @(posedge clk)
   begin                  // ab4_entry = ab4; q4_entry = q4;
      q4 <= ab4;          // q4_exit = ab4_entry
      ab4 <= a & b;       // ab4_exit = a & b
   end                    // ab4 = ab4_exit; q4 = q4_exit

   // attempt 5 (switch the order of attempt 2)
   always @(posedge clk)
   begin
```

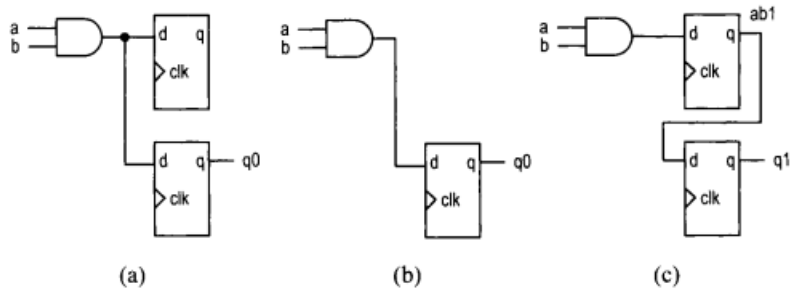


```
      q5 = ab5;
      ab5 = a & b;
   end

endmodule
```

Analyzation process:

- In an attempt 0, assignments to ab0 and q0 infer 2 registers initially, 1 to store the registered ab0 and 1 to store the registered q0



-
  As ab0 is updated immediately by the blocking assignment, q0 gets the value of a & b. The corresponding circuit diagram (no a). Since ab0 is not used outside the always block, the registered ab0 output is not needed and thus the corresponding register can be removed. The resulting diagram is shown has demonstrated the desired circuit (no b)

- In attempt 1, a blocking assignment is used for ab1. The corresponding interpretation in shown in the comments. The ab1 entry is the prev stored value of ab1 and corresponds to the registered output. The corresponding diagram in shown in c. An unintended input buffer is inferred and the storage of a &b is delayed by 1 clock cycle.
- In attempt 2, blocking assignments are used for both ab2 and q2. The circuit inferred is identical to that in attempt 0 as shown in a and b. Since using blocking assignment to infer FFs may introduce a race condition.
- Attempts 3 4 5 is used to examine what will happen after switching the order of the assignments of attempts. In attempt 3, ab3 is used before assigning a new value. Thus, q3 get the “prev value ” from the earlier activation. The value is stored in a register and corresponds to the registered a &b -> c inferior circuit. In attempt 4, switching the order has no effect on the code, which is identical to the code in attempt 1. In attempt 5, ab5 is used before it is assigned a new value and thus q5 gets the registered a &b. It infers a circuit identical in attemp3.
- In conclusion, only the code in attempt 0 describes the desired circuit correctly and reliably