Fair Arbiter Design

In a system where two clients share one server, an **arbiter** decides which client gets served when both request access at the same time. **Design and implement** a *fair arbiter* that gives both clients equal opportunity to use the server.

Problem Description
Two clients, **A** and **B**, can independently request service through signals `req_a` and `req_b`.
The arbiter issues corresponding grant signals `grant_a` and `grant_b` to indicate which client gets served in each clock cycle.
Rules:

- Only one client can be served per cycle → at most one grant signal may be 1.
- If neither client requests service, both grant signals must remain 0.
- The arbiter must be **fair**: whichever client received the grant most recently should have **lower priority** in the next cycle.
- The server completes one service per clock cycle.
- Use a **synchronous reset**. After reset, if both clients request service in the same clock, client A should receive the first grant.

Tasks
(a) Draw the **state transition diagram** for this fair arbiter, modeled as a **Mealy finite-state machine**.

- Label each transition with:
    - **Input:** 2-bit `{req_a, req_b}` (MSB = `req_a`)
    - **Output:** 2-bit `{grant_a, grant_b}` (MSB = `grant_a`)
- Use `X` to mark don't-care bits on inputs when possible.
- Show all possible transitions that maintain fairness.

(b) Write **Verilog code** for your FSM from part (a).
Use the given code header and complete the design using a synchronous reset and Mealy-style output logic.

```
module arb (clk, reset, req_a, req_b, grant_a, grant_b);

  input clk, reset;
  input req_a, req_b;
  output grant_a, grant_b;

// Add code here

endmodule
```

Example of operation:

clk #	0	1	2	3	4	5	6	7	8
reset	1	0	0	0	0	0	0	0	0
{req_a,req_b}	xx	00	11	11	01	10	11	10	11
{grant_a,grant_b}	xx	00	10	01	01	10	01	10	01