EXERCISE 2
DESIGN AND SIMULATION OF LOOP-BASED COMBINATIONAL LOGIC

In a competition consisting on an individual match between two contestants, A and B, three judges give 0 or 1 points to each of the contestants and a combinational system computes the winner. The score given by each of the judges can be one of four:

Table 1: Each judge votes for none, A, B, or both

Judge Votes	Description
00	No decision
01	1 point for A
10	1 point for B
11	1 point for both A and B

The result is given by the combinational system in a form of 4 possible outcomes:

Table 2: Winner output

Winner	Description
00	All judges gave 0 points to both A and B
01	A is the winner. A has more points than B.
10	B is the winner. B has more points than A.
11	Tie. A and B have equal number of points.

The votes from each of the judges are combined into two vectors, one containing the votes for contestant A and one for contestant B.

Table 3 includes a few examples. The first three columns include the vote of each of the three judges. The middle two columns show the two vectors containing the votes for contestants A and B, respectively, and the last column is the match winner.

Table 3: Examples

Judge 1	Judge 2	Judge 3	Votes A	Votes B	Winner
00	00	00	000	000	00
01	00	00	100	000	01
11	10	01	101	110	11
11	11	11	111	111	11
10	10	10	000	111	10

Part 1: Design a combinational vote calculator

Design entity tally takes the two vectors as inputs and computes the two-bit output named winner. Given the following entity declaration for tally:

```
entity tally is
     port (
        scoresA, scoresB : in  std_logic_vector(2 downto 0);
        winner           : out std_logic_vector(1 downto 0)
     );
end entity;
```

Write an architecture named loopy that computes the output winner. The computation to determine the winner must be performed using a loop statement. Use type integers in the intermediate computations for the final output value.

Part 2: Simulation and Verification

Write a testbench that provides exhaustive verification of the tally scoring system. The testbench must use a *loop* to generate *all possible input combinations*. **You should use *different* methods to calculate/obtain the winner in the module architecture and the testbench.** You can use, for example, table method as in the first exercise; array that contains the correct number of votes according to an input vector (array of integers). An input vector can be used as an index number; or some other method, you name it.

A function must be used to calculate the expected result for the tally. In other words, a function can only calculate the expected value or calculate the expected value and verify the correctness of the tally. For example, a function can be used in an assert statement as follows:

assert(winner = result(scoresA, scoresB))

where *result* is the name of the function, *scoresA* and *scoresB* are the input vectors to the tally, and *winner* the result calculated by your design. You may also do the comparison inside the body of the function:

assert result(scoresA, scoresB, winner)