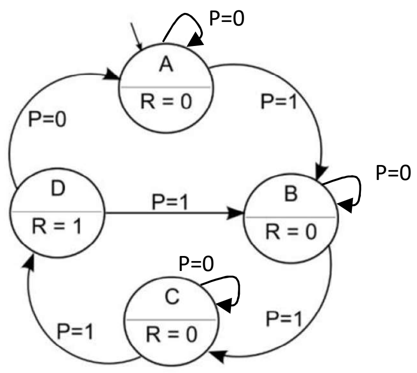
Lab session #5

Activity #1: FSM #1

A **FSM** has 1 input line P and 1 Moore output R. Its state diagram appears in the following figure:



Design the FSM resorting to the VHDL behavioural description style with:

- 2 processes
- 1 process.

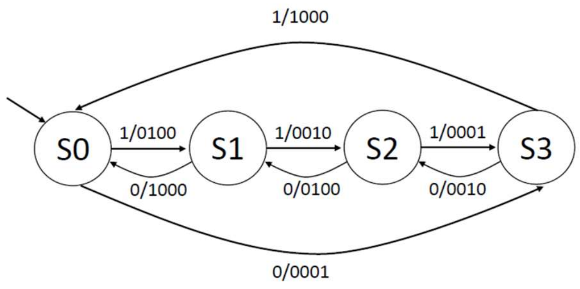
Compare the timing in the 2 cases. Reset is asynchronous. Synthesize the design, encoding states with:

- minimum length encoding
- one-hot encoding

Create a suitable testbench to test the FSM on relevant values reading test vectors from file. Read expected outputs from file as well and automatically check if the UUT behaves as expected.

Activity #2: FSM #2

A **FSM** has 1 input line UpDw and 1 4-bit Mealy output count. Its state diagram appears in the following figure:



It acts as a 1-hot encoded up/down counter: when UpDw=0 the counting sequence starting from S0 is 0001, 0010, 0100, 1000 and back to 0001, when UpDw=1 the counting sequence starting from S0 is 0100, 0010, 0001, 1000 and back to 0100. Design the FSM resorting to the VHDL behavioural description style with 2 processes. Reset is asynchronous.

Create a suitable testbench to test the FSM on relevant values reading test vectors from file.

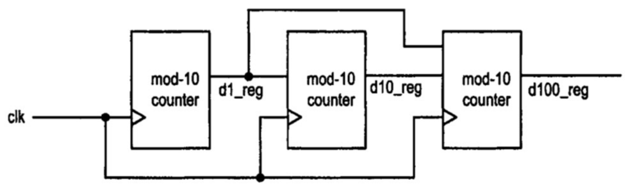
Activity #3: Gray counter

Design an N-bit **Gray counter** with enable. This kind of counter circulates through all 2^N states and its counting sequence follows the Gray code sequence, in which only one bit is changed between successive code words (cfr the example of Gray code sequence on 4 bits in Lab. #3). Hint: consider the Gray code incrementer designed in Lab. #3 to force the Gray counting sequence. Reset is synchronous. Resort to a VHDL behavioral description style (process for state machine, concurrent assignments for next state computation). Create a suitable testbench to test the counter on relevant values for N=4.

Activity #4: Decimal counter

A **decimal counter** circulates the patterns in binary-coded decimal (BCD) format. One possible way to construct a decimal counter is to design a BCD incrementer for the next-state logic, just like a regular incrementer in a binary counter. Because of the cumbersome implementation of the BCD incrementer, this method is not efficient. A better alternative is to divide the counter into stages of decade counters and use special enable logic to control the increment of the individual decade counters.

Design a **3-digit (12-bit) decimal counter** that counts from 000 to 999 and then repeats. Reset is asynchronous. Provide an enable signal as well. Hint: implement it by cascading three special decade counters, as shown in the following figure:



The leftmost decade counter represents the least significant decimal digit. It is a regular mod-10 counter that counts from 0 to 9 (i.e., from "0000" to "1001") and repeats. The middle decade counter is a mod-10 counter with a special enable circuit. It increments only when the least significant decimal digit reaches 9. The rightmost decade counter represents the most significant decimal digit, and it increments only when the two least significant decimal digits are equal to 99. Note that when the counter reaches 999, it must return to 000 at the next rising edge of the clock.

Create a suitable testbench and verify that the output sequence is as expected.