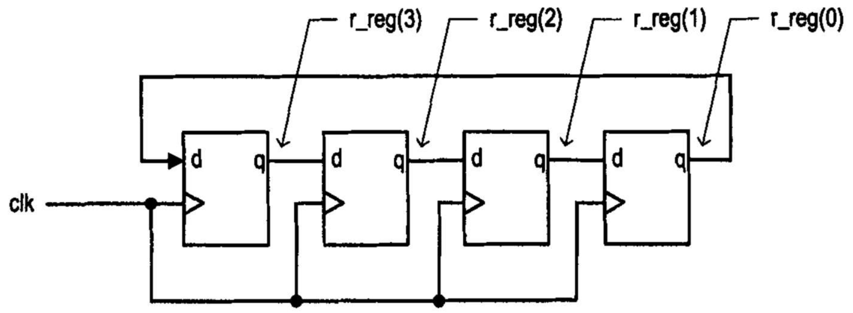
Lab session #6

Activity #1: Ring counter

An N-bit **Ring counter** circulates through N of its states a 1-hot pattern, i.e., all bits are set to 0 except one that is set to 1 and this 1 value shifts at each clock tick. For example, if pattern 0001 is loaded in a 4-bit ring counter, the counter circulates through the following states: "1000", "0100", "0010" and "0001" and then repeats. A simple implementation of a ring counter is a shift-register whose serial-out port is connected to its serial-in port, as shown in the following figure for N=4:



The ring counter works correctly only if a "seed" other than the all-0 pattern is preloaded in the shift register. Design an N-bit **Ring counter** with enable. 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.

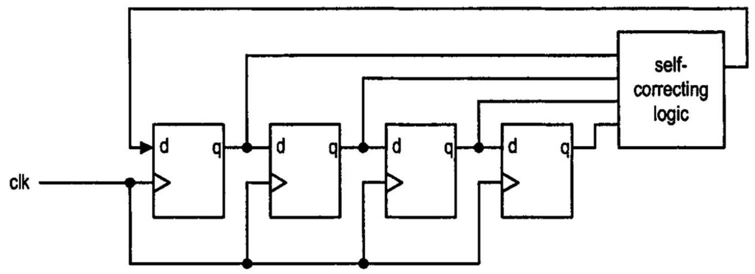
Activity #2: Self-correcting ring counter

The limit of the previous solution resides in the need to preload a suitable seed. An alternative implementation resorts to a self-correcting logic to feed the serial-in port with the correct pattern:

- the shift_in value is '1' only if the current N-1 MSBs of the register are all at '0'
- the shift_in value is '0' if any of the current N-1 MSBs of the register is at '0'.

This process continues until all the N-1 MSBs become '0' and a '1' is shifted in afterward, allowing operation to start.

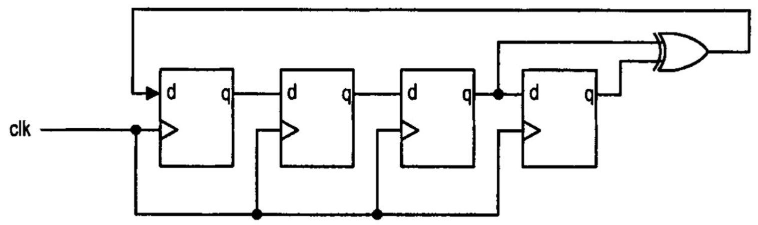
Note that this scheme works even when the register contains an invalid pattern initially. For example, if the initial value of the register is "1101", the logic will gradually shift in 0's and return to the normal circulating sequence. The following figure shows a block diagram of a 4-bit self-correcting ring counter:



Design an N-bit **self-correcting Ring counter** N-bit with enable. 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 #3: Linear Feedback Shift Register

A **Linear Feedback Shift Register** (LFSR) is a shift register with a special feedback circuit to generate the serial input value. This circuit XORs certain bits of the register. As a result an N-bit register cycles through a set of 2^N-1 unique states. As an example, consider the 4-bit LFSR of the following figure:



The two LSB signals of the register are XORed to generate the serial input. Assuming that the initial state is "1000", the circuit will circulate through the 1 states as follows: "1000", "0100", "0010", "1001", "1100", "0110", "1011", "0101", "1010", "1101", "1110", "1111", "011l", "0011", "0001". The only missing state is "0000". If the LFSR enters this state accidentally, it will be stuck in this state. The mathematics behind LFSRs is based on Galois fields and it's beyond the scope of this course. The main mathematical properties of LFSRs are:

1. An N-bit LFSR can cycle through up to 2^N - 1 states. The all-zero state is excluded
2. A feedback circuit to generate maximal number of states exists for any N.
3. The sequence generated by the feedback circuit is pseudorandom, which means that the sequence exhibits a certain statistical property and appears to be random.

Denoting with q_{N-1}, q_{N-2}, …, q_1, q_0 the outputs of the shift register, the expressions of the feedback circuit as a function of N are listed below in some significant cases:

N	Feedback circuit
2	$q_1 \oplus q_0$
3	$q_1 \oplus q_0$
4	$q_1 \oplus q_0$
5	$q_2 \oplus q_0$
6	$q_1 \oplus q_0$
7	$q_3 \oplus q_0$
8	$q_4 \oplus q_3 \oplus q_2 \oplus q_0$
16	$q_5 \oplus q_4 \oplus q_3 \oplus q_0$
32	$q_{22} \oplus q_2 \oplus q_1 \oplus q_0$
64	$q_4 \oplus q_3 \oplus q_1 \oplus q_0$
128	$q_{29} \oplus q_{17} \oplus q_2 \oplus q_0$

Note that the LFSR cannot be initialized with the all-zero pattern. In pseudo number generation, the initial value of the sequence is known as a seed.

Design an **8-bit LFSR**, initialized with a proper seed. Reset is asynchronous. Create a suitable testbench to test the LFSR on relevant values.

Activity #4: de Bruijn counter

It is possible to modify an N-bit LFSR so that it cycles through all 2^N states by modifying the feedback circuit. In an LFSR after the "00….01" state a '1' must be shifted in, as the all-zero state is forbidden. The next state thus becomes "10….00". The revised design will insert the all-zero state, "00….00", between the "00….01" and "10….00" states. Denoting with q_{N-1}, q_{N-2}, …, q_1, q_0 the outputs of the shift register and with f the original feedback signal of the LFSR, the modified feedback value f_{mod} is:

$$f_{mod} = f \oplus (q'_{N-1} \cdot q'_{N-2} \cdot \;\ldots\ldots\; \cdot q'_2 \cdot q'_1)\cdot$$

Note that:

- $q'_{N-1} \cdot q'_{N-2} \cdot \ldots\ldots \cdot q'_2 \cdot q'_1 = 1$ indicates that the N-1 MSBs are at 0, so the state of the LFSR is either "00….01" or "00….00"
- if the above condition is false, $f_{mod} = f \oplus 0 = f$, thus the circuit will operate as a basic LFSR
- if the state is "00….01", the above condition is true and f=1, so $f_{mod} = 1 \oplus 1 = 0$ and the next state will be "00….00"
- if the state is "00….00", the above condition is true and f=0, so $f_{mod} = 0 \oplus 1 = 1$ and the next state will be "10….00"
- once the state is "10….00", the above condition is false, $f_{mod} = f$, thus the circuit will operate as a basic LFSR.

Design an 8-bit de Bruijn counter. Create a suitable testbench to test the counter on relevant values, in particular prove that the all-zero seed can be loaded into the register during system initialization.