Exercise 03: Sequential logic 2

Clock divider

A clock divider for generating a certain clock period will be shown in the lab (see also on LMS). The clock period of the clock divider can be set when instantiating the entity and it is fixed in the compiled design.

Write a new clock divider with frequency selection logic by modifying the code of the fixed one:

```vhdl
entity ex03_clkdiv_sel is
port (
    o_clk       : out   std_logic;
    i_sel       : in    std_logic_vector(3 downto 0) := (others => '0');
    i_reset_n   : in    std_logic;
    i_clk       : in    std_logic--;
);
end entity;
```

The output frequency (division factor) is selected with the `i_sel` input port (4 bits for a total of 16 possible values). Assume that this new clock divider is driven from a 62.5 ms clock `i_clk`. The following output clock periods should be available for selection: 0.125 s, 0.25 s, 0.375 s, 0.5 s, 0.625 s, 0.75 s, 0.875 s, 1 s, 1.5 s, 2 s, 2.5 s, 3 s, 3.5 s, 4 s, 4.5 s, 5 s.

Test the correct behavior in a simulation with the provided testbench.

Test your entity with the provide top file `ex03_top_clkdiv` where 4 switches are connected to the input port `i_sel`. Modify division factor for `clkdiv` entity such that it generate 62.5 ms. Observe different frequencies of the blinking LED.

Instructions:

In `ex03_clkdiv_sel.vhd`

- Include the package `ieee.numeric_std`
- Define in the architecture a new data type:

  `type array_positive_t is array(natural range <>) of positive;`
- Define a constant of this data type to store the needed divider values:

  `constant C_DIVIDER_VALUES : array_positive_t(0 to 15) := ( 1, ... );`

  (put a comma separated list of 16 divider values in the last parenthesis for achieving the requested clock periods)
- Define a signal `sel_uint` of the data type `natural`, which will hold the unsigned integer representation of the input port `i_sel`
- Convert the `std_logic_vector i_sel` to the `natural sel_uint`:

  `sel_uint <= to_integer(unsigned(i_sel));`
- Replace `N` in the comparison in the process with `C_DIVIDER_VALUES(sel_uint)`.

Linear-feedback shift register

A linear-feedback shift register (LFSR) is a register where the input is a linear combination of the previous state (feedback). LFSRs with a certain feedback logic show a periodic behavior, which can be used for pseudo-random number generators (PRNG) and fast counters.

The figure below shows as an example of LFSR's previous ('input') and next ('output') values for a 5-bit Fibonacci-LFSR that uses an XNOR of the 'taps' on bits 5 and 3.

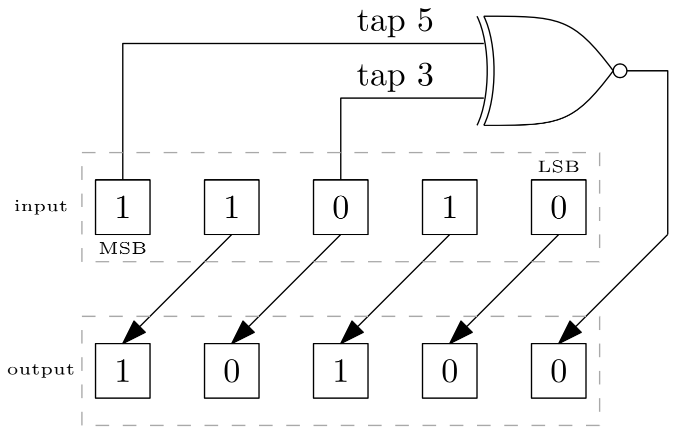

Figure 1: lfsr

Write an entity for an 8-bit Fibonacci-LFSR using `XNOR` gates:

```
entity ex03_lfsr8 is
generic (
    -- LFSR reset value
    g_SEED : std_logic_vector(7 downto 0) := (others => '0')
);
port (
    -- output LFSR value
    o_lfsr      : out   std_logic_vector(7 downto 0);
    -- output period of this LFSR
    o_period    : out   std_logic_vector(7 downto 0);

    -- input reset (active low)
    i_reset_n   : in    std_logic;
    -- input clock
    i_clk       : in    std_logic--;
);
end entity;
```

The generic `g_SEED` is the value to which the LFSR is reset when `i_reset_n` is low. The entity should include a counter to measure the period of the generated sequence (the number of different LFSR values from one appearance of `g_SEED` to the next time). The output port `o_period` should show the last measured sequence period, which means it should be updated when the period measurement is finished.

Instructions:

- To achieve maximum period, use the taps from https://docs.xilinx.com/v/u/en-US/xapp052 (page 5) (it's not mentioned in the document, but it is for Fibonacci-LFSRs). Also note, that the taps given there are counted from 1 (not 0).

- Define in the architecture an 8-bit signal `lfsr` representing the shift register. Decide yourself, if you want to count its bits from 1 or 0.
- Define a 1-bit signal `feedback` for the feedback logic.
- Write a clocked process, running with `i_clk`:
  - Reset `lfsr` to `g_SEED` when `i_reset_n` is low.
  - Otherwise assign to the lowest bit of `lfsr` the `feedback`.
  - Shift the other bits by assigning to the seven highest bits of `lfsr` the seven lowest bits.
- Outside of the process, assign the necessary logical combination of the taps of the LFSR to `feedback` (again, pay attention how to count the bits).
- For the period measurement, write a clocked process (also running with `i_clk`) for a binary counter `period_cntr`. When `lfsr` has the value `g_SEED`, then `period_cntr` shall be reset to zero and the current value of `period_cntr` shall be output to port `o_period`. When `lfsr` has a different value, `period_cntr` shall be incremented.

Test the entity behavior with the provided test bench and top file.

NOTE

- remember to set appropriate duration for simulation (usually 1 microsecond)
- to use `clkdiv_sel` it may be necessary to change to 'Sinplify Pro' synthesis tool (right click on implementation `impl1`, select 'Properties' and in 'Synthesis Tool' select 'Sinplify Pro'),

---

The labs take place in Staudingerweg 9 building: group 1 in room 4-516 on Thurday and group 2 in Newton room on Friday.

Send your solutions through MOODLE (https://moodle.uni-mainz.de).

The solution should include:

- Source code (only **your** `.vhd` files)
- Brief description if applicable, questions, etc.
- Screen shot of the simulation (use simulation range of 1 microsecond)
