EXERCISE 1: Combinatorial circuits

Implement a combinatorial circuit that allows multiplexing an output from 4 input channels ch1_i - ch4_i by asserting 4 individual enable inputs sel1_i - sel4_i. Each input is a bus composed by 8 bits. If all selection input are 0 choose one arbitrary input or a default value of all zero.

What would you do to change priorities of inputs?

Hint 1: Interface of the circuit

```
entity exe_1_top is
port (
      ch1_i  : in  std_logic_vector(7 downto 0);
      ch2_i  : in  std_logic_vector(7 downto 0);
      ch3_i  : in  std_logic_vector(7 downto 0);
      ch4_i  : in  std_logic_vector(7 downto 0);
    sel1_i : in  std_logic;
    sel2_i : in  std_logic;
    sel3_i : in  std_logic;
    sel4_i : in  std_logic;
    out_o  : out std_logic_vector(7 downto 0);
);
end entity exe_1_top;
```

EXERCISE 2: Sequential circuits

Implement a rising_edge detector for an input signal (that we assume coming from another flip flop, hence only changing state at clock edges). The edge detector will issue an "HIGH" output when the input goes from 0 to 1 (at the rising edge of the clock). The circuit also should have a reset.

Hint 1: Interface of the circuit

```
entity exe_2_top is
port (
    clk_i         : in  std_logic;
    arst_i        : in  std_logic;
    in_signal_i   : in  std_logic;
    rising_edge_o : out std_logic;
);
end entity exe_2_top;
```

Hint 2:

The circuit will need to be sequential, with one flip flop keeping the last state of the input signal.

We will then compare the input signal with its last state to detect the presence of a rising edge.

We will need an internal signal to keep the last state.

EXERCISE 3: Counters

Implement a watchdog counter that counts for 16 clock cycles while an enable_i signal is high, generating an output strobe each time it reaches the count of 16. After reaching 16 the counter will rollover and repeat counting from the start, keeping generating a strobe on the count of 16. If the enable goes low the counter is reset.

Hint 1 : interface

```
entity exe_3_top is
    port (
        clk_i          : in  std_logic;
        arst_i         : in  std_logic;
        enable_i       : in  std_logic;
        watchdog_o     : out std_logic);
end entity exe_3_top;
```

Hint 2:

This is actually a quite easy behaviour. Once the count reaches 16, if the counter signal is adequately sized, it will just rollover and restart, generating a strobe at the needed rate. You will only need a condition to test when you have reached the desired value.

EXERCISE 4: Finite State Machines

Implement the control logic of an automatic gate, connected to an electric motor which has two input signals "open_o" and "close_o" (which are output from our module). The gate is activated (opened) by a std_logic signal "button_i" (input to our module). We will suppose that the action of the motor takes 16 clock cycles to complete (you can use the module from exercise 3 for this). Once the 16 clock cycles elapse the gate is open. After other 16 clock cycles (you can instantiate a different watchdog or re-use the one which times the opening - more complicated) it will start closing. If at any time during the closing phase an input "photocell_i" goes HIGH, it will restart the opening phase (as if it entered it from the IDLE state). The open_o and close_o signals to the motor stay HIGH during the entire opening and closing phase.

Hint 1:

This is implemented with a FSM with four states. IDLE, OPENING, OPENED, and CLOSING. The graph is

We can use the watchdog from exercise 3 to time the phases. Easiest approach is to use a watchdog for each state. The enable input will be issued by the process which implements the FSM whenever it is in the relevant state, and the strobe (watchdog) output will be used to trigger the state transition.

The interface of the module is

```
entity exe_4_top is
port (
    clk_i          : in  std_logic;
    arst_i         : in  std_logic;
    button_i       : in  std_logic;
    photocell_i    : in  std_logic;
    open_o         : out std_logic;
    close_o        : out std_logic);
end entity exe_4_top;
```

Hint 2: to connect the watchdogs, you will just need to wire the "enable" inputs of the watchdog to a signal that is high whenever the FSM is in a particular state.

In the FSM process you will then use the watchdog output to check if you have to transition to another state in the case statement.

EXERCISE 5: DSP (FIR FILTERING)

Implement a lowpass moving average filter with a kernel of 1/8*[1 2 2 2 1] Please Note the filter is normalized. The circuit shall get 1 input sample per clock cycle and give 1 output sample per clock cycle. Input and outputs are 8 bits wide. The circuit should have an asynchronous reset and an "enable_i" input which enables sequential elements.

Hint 1: Interface

```
entity exe_5_top is
port (
    clk_i        : in  std_logic;
    arst_i       : in  std_logic;
    in_signal_i  : in  std_logic_vector(7 downto 0);
    filtered_o   : out std_logic_vector(7 downto 0);
);
end entity exe_5_top;
```

Hint 2:

The circuit is composed of a shift register which accepts the input samples, a set of multipliers that multiply the filter tap coefficients for the samples in the shift registers and a summing tree to obtain the output sample.