Exercise 04: State machine

Coffee controller

Write a controller for a coffee machine that prepares coffee by performing a sequence of operations, such as grinding beans, adding water and adding milk. Each operation lasts for a certain amount of time. After finishing (normally or stopped), the machine performs a cleaning cycle and goes back into idle mode.

There are 3 types of coffee that this machine should produce: black coffee (3 units of beans and 8 units of water), espresso (5 units of beans and 2 units of water) and cappuccino (3 units of beans, 4 units of water and 4 units of milk). Each unit corresponds to 1 second of operation. The cleaning operation takes 4 seconds.

Use the following declaration for the coffee machine entity:

```vhdl
entity ex04_coffee_ctrl is
port (
    -- select coffee
    i_btn       : in    std_logic_vector(3 downto 0);
    -- current state of the machine
    o_state     : out   std_logic_vector(4 downto 0);
    i_reset_n   : in    std_logic;
    i_clk       : in    std_logic--;
);
end entity;
```

The coffee is selected by the `i_btn` input port: `i_btn(1)` - black coffee, `i_btn(2)` - espresso, `i_btn(3)` - cappuccino. These inputs are only accepted during idle state. The preparation of the coffee can be aborted at any point by `i_btn(0)` the input, which will send the machine into the cleaning state.

The current state of the machine is given by `o_state` port:

- `o_state(0)` - `IDLE`,
- `o_state(1)` - `BEAN_GRINDER`,
- `o_state(2)` - `WATER_PUMP`,
- `o_state(3)` - `MILK_PUMP`,
- `o_state(4)` - `CLEANING`.

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

Instructions:

Write a state machine that controls the state transitions (note that some states can be skipped):

`IDLE -> BEAN_GRINDER -> WATER_PUMP -> MILK_PUMP -> CLEANING -> IDLE`.

When in IDLE state, set counters for required steps (bean grinder, water pump, etc.) according to selected button and transition to next step (e.g. `BEAN_GRINDER`).

At each state decrement corresponding counter and transition to next state when counter reaches zero (handle situation when there is no transition to `MILK_PUMP` state).

Handle `i_btn(0)` press, which should force transition to `CLEANING` state,

Button decoder

Write a button decoder which detects single, double and triple presses, and long single presses of a button.

```
entity ex04_button_decoder is
generic (
    T_SHORT : positive := 4;
    T_LONG : positive := 8--;
);
port (
    i_btn       : in    std_logic;
    o_single    : out   std_logic;
    o_double    : out   std_logic;
    o_triple    : out   std_logic;
    o_long      : out   std_logic;
    i_reset_n   : in    std_logic := '1';
    i_clk       : in    std_logic--;
);
end entity;
```

The `T_SHORT` parameter controls the maximum number of cycles required between button presses for a consequent button press to be detected as a sequence.

The `T_LONG` parameter controls the minimum number of cycles required for an input signal to be considered a long press.

For single/double/triple presses the time difference should be counted between rising edges of the input button presses. For long press the time difference is between the rising and falling edges.

The output for each detected signal should be one clock cycle long. Invalid sequences (e.g. four presses) should not produce any output.

Test the entity with provided test bench and top file.

Instructions:

- Create two `edge_detector` entities do detect rising and falling edges of the `i_btn` input
- On each clock cycle increment the time counter `t`
- On detected rising edge of `i_btn` reset the time counter to zero and increment the button counter `n`
- When the time counter reaches `T_SHORT-1` generate pulse on `o_single/o_double/o_triple` output ports according to the value of button counter (`n = 1/2/3`) and reset the button counter to zero
- On detected falling edge of `i_btn`, if time counter is `T_LONG-1`, then generate signal on `o_long`

Note that above logic will generate both single/double/triple signal and long signal if last press is long. How would you modify the decoder to avoid such behavior?

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)