Finite-State Machine (FSM)

Task:

Implement a **Mealy** type finite-state machine (ask lab assistants for the individual assignment). The task is divided in two main steps:

- Describe FSM in VHDL. Try to experiment with various state encodings that reduce the switching activity in the state register in order to find the best implementation in terms of power consumption (refer to "Low Power Design" for theoretical background on this topic). Report the resultant dynamic power (as estimated by "Report Power" in Vivado), resource utilization and performance. Optionally, try the decomposition as well (computational kernel extraction).
- After completing the previous step, add a debounce circuit for the push button. For implementation use one of the Push Buttons to generate a clock signal for the FSM, switches to set the input values and LEDs to observe the output values.

Simulate and implement the project on FPGA development board.

Moore FSM Example

Externally, the FSM is defined by its primary inputs, outputs and the clock signal. The clock signal determines when the inputs are sampled and outputs get their new values. It means that internally machine stores a state which is updated at each tick of the clock.

Example finite-state machine algorithm is presented in Figure 1. It will be implemented as a Moore type machine. FSM has three inputs marked as X1, X2, X3 and eight outputs. Output values are shown in hexadecimal format. Example Moore FSM has five states, corresponding to the number of output vertices of the graph. Let's name them State 0, State 1, State 2, State 3 and State 4. It is now possible to draw a state transition table (Table 1).

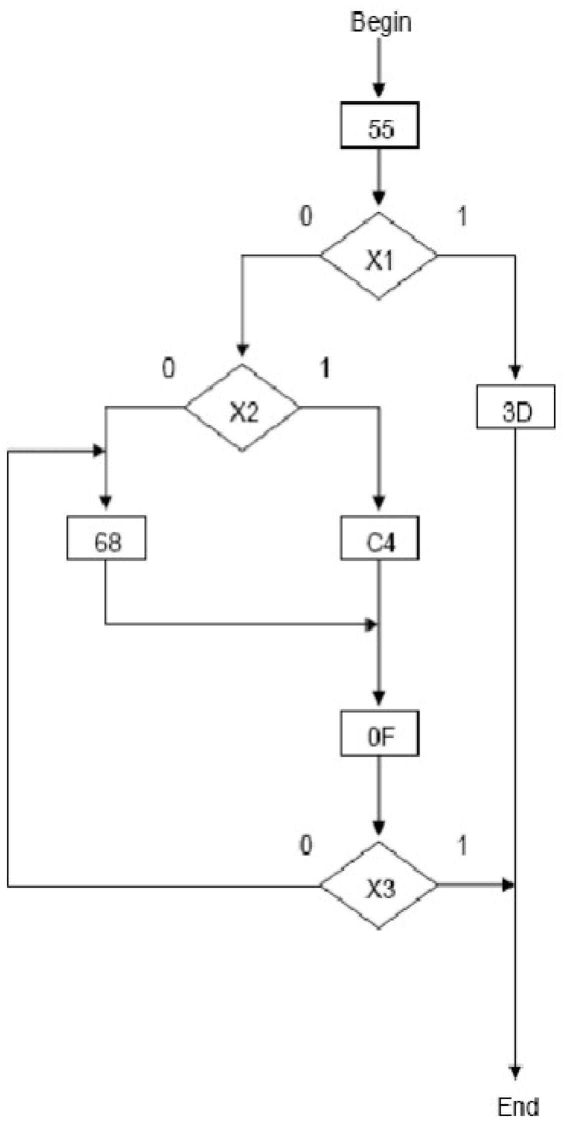

Figure 1: Example Finite-State Machine Algorithm

In order to model the states of the FSM in VHDL an enumerated type “State” is created. Signals, which hold the values of the current state and the next state, are of this type. States are encoded automatically during synthesis. Type and signals are declared as follows:

```
type State is (State_0, State_1, State_2, State_3, State_4);
signal current_state, next_state: State;
```

Table 1: Example FSM State Transition Table

Current State	X1	X2	X3	Next State	Output
State 0	1	-	-	State 1	X”55”
	0	0	-	State 2	
	0	1	-	State 3	
State 1	-	-	-	State 0	X”3D”
State 2	-	-	-	State 4	X”68”
State 3	-	-	-	State 4	X”C4”
State 4	-	-	0	State 2	X”0F”
	-	-	1	State 0	

An FSM is characterized by two functions – the next state function and the output function. The next state function depends on current state and inputs. The output function is defined by the FSM type. For Moore machine it depends on the current state only. VHDL description of these functions is presented in Listing 1 and Listing 2.

Listing 1: Next State Function of Example Moore FSM

```
process (current_state, X1, X2, X3)
begin

case current_state is
  when State_0 => if X1 = '1' then
                    next_state <= State_1;
                  elsif X2 = '1' then
                    next_state <= State_3;
                  else
                    next_state <= State_2;
                  end if;
  when State_1 => next_state <= State_0;
  when State_2 => next_state <= State_4;
  when State_3 => next_state <= State_4;
  when State_4 => if X3 = '1' then
                    next_state <= State_0;
                  else
                    next_state <= State_2;
                  end if;
end case;

end process;
```

Listing 2: Output Function of Example Moore FSM

```
process (current_state)
begin

case current_state is
  when State_0 => Outputs <= X"55";        -- same as "01010101"
  when State_1 => Outputs <= X"3D";        -- same as "00111101"
  when State_2 => Outputs <= X"68";        -- same as "01101000"
  when State_3 => Outputs <= X"C4";        -- same as "11000100"
  when State_4 => Outputs <= X"0F";        -- same as "00001111"
end case;

end process;
```

Next state and output functions form a combinational part of the FSM. Both functions are described using a separate process statements. Sensitivity list of each process features all signals, which serve as inputs to the combinational portion of the design represented by this process. Just like in a real combinational circuit, whenever any of these signals changes a value, the process responds by reevaluating the outputs. Thus, output values for every input combination should be defined. Note, that outputs of the example FSM are represented by an 8-bit vector.

VHDL description of the sequential logic that stores the state of the example FSM is presented in Listing 3. Signal "current_state" infers a register during synthesis, as it is assigned a value inside an edge sensitive **if** statement. The state change occurs on the positive edge of the clock. The control signals (e.g. reset and enable) are omitted from the description for the purpose of simplification, but generally should also be present (as described in the previous lab).

Listing 3: Sequential Logic for Storing State of the Example FSM

```
process (clock)
begin

if clock'event and clock = '1' then
  current_state <= next_state;
end if;

end process;
```

The block diagram of the entire example Moore FSM is presented on Figure 2. It consists of three main blocks: “Next State Function”, “Output Function” and “State Register”. Each block is described with a corresponding process from Listing 1, Listing 2 and Listing 3 respectively. The “cloud” blocks represent combinational logic, the rectangle block – sequential logic.

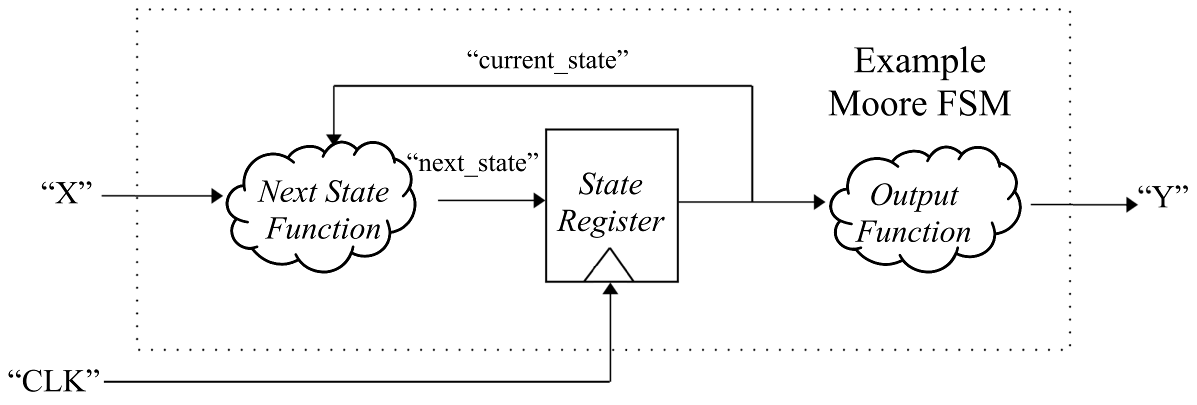

Figure 2: Block Diagram of Example Moore FSM

Push Button Debounce Circuit

In order to use a Push Button to generate clock signal pulses it has to be debounced. The problem is that Push Button contains a metal spring and it actually makes contact several times before stabilizing. The high-speed logic circuits may react to the contact bounce as if the Push Button has been pressed several times, because some pulses may have a sufficient voltage and long enough duration. Thus, for the hardware implementation to work correctly, contact bounce must be filtered.

A 4-bit shift register, which is clocked at 100Hz, can do the trick. With each clock tick it shifts in Push Button's output. When all four bits of the shift register are High, its output goes also High. This will delay the Low to High change until the contact bounce stops.