Design a Counter Using a Finite State Machine and Programming a FPGA

Background:

A Finite State Machine (FSM) is a digital circuit whose state changes based on both the current state (of the FSM) and the current inputs. Many processes in digital electronics follow predefined sequence of steps initiated by a series of clock pulses. These processes can be driven by a single clock input and have one or more outputs that response in a particular order at each clock input pulse. The sequence of events can be implemented in a State Machine. State machine FSM can be implemented by VHDL by defining the correct sequence of output states and then stepping through the states in numerical order. A finite state machine (FSM) is a model that can be implemented as a digital system, all states must be represented as patterns using a fixed number of bits, all inputs must be translated into bits, and all outputs must be translated into bits.
The table shows the binary codes need used to drive a counter to counting and counting down.

Table I

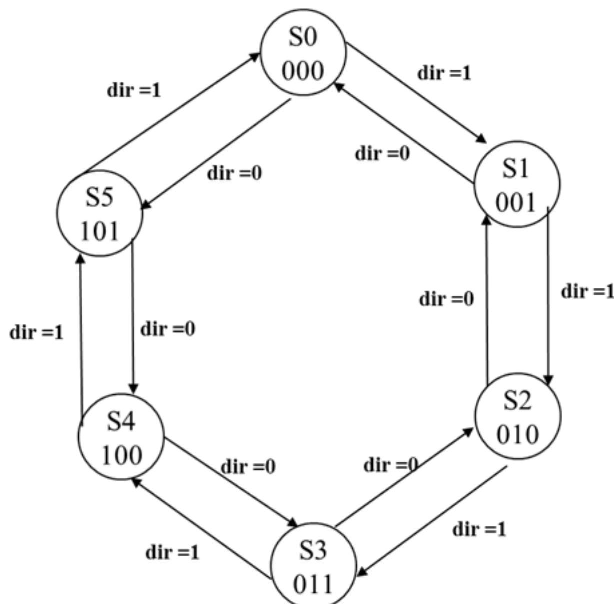
Binary Codes for Counting Up and Counting Down	
UP (Counting Up)	DOWN(Counting Down)
0 0 0 (state s0)	101 (state s5)
0 0 1 (state s1)	1 0 0 (state s4)
0 1 0 (state s2)	0 1 1 (state s3)
0 1 1 (state s3)	0 1 0 (state s2)
1 0 0 (state s4)	0 0 1 (state s1)
1 0 1 (state s5)	0 0 0 (state s0)
0 0 0 (state s0)	101 (state s5)
0 0 1 (state s1)	1 0 0 (state s4)
0 1 0 (state s2)	0 1 1 (state s3)
…	…

In state machine design, we need to refer to the state as present-state and next-state. Table II shows state changes for count up/down.

Table II

State Changes for Counting Up and Counting Down Rotation			
Counting Up rotation		Counting Down rotation	
Present sate	Next state	Present sate	Next state
0 0 0 (state s0)	0 0 1 (state s1)	101 (state s5)	1 0 0 (state s4)
0 0 1 (state s1)	0 1 0 (state s2)	1 0 0 (state s4)	0 1 1 (state s3)
0 1 0 (state s2)	0 1 1 (state s3)	0 1 1 (state s3)	0 1 0 (state s2)
0 1 1 (state s3)	1 0 0 (state s4)	0 1 0 (state s2)	0 0 1 (state s1)
1 0 0 (state s4)	101 (state s5)	0 0 1 (state s1)	0 0 0 (state s0)
101 (state s5)	0 0 0 (state s0)	0 0 0 (state s0)	101 (state s5)
0 0 0 (state s0)	0 0 1 (state s1)	101 (state s5)	1 0 0 (state s4)
0 0 1 (state s1)	0 1 0 (state s2)	1 0 0 (state s4)	0 1 1 (state s3)
So on		So on	

A state transition diagram (or transition diagram, or state diagram), as shown in Fig 1, illustrates the contents of the next-state table graphically, with each state drawn in a circle, and arcs between states labeled with the input combinations that cause these transitions from one state to another. With dir =1, a counter will count up. With dir=0, a counter will count down.



State diagram for Mod-6 3-bit counter with count up/down control

You can follow the VHDL program to implement a 3-bit counter sequence with up/down control. The input are *clk* and *dir* and the output is a 3-bit vector name *Q*. The *state_machine* is defined

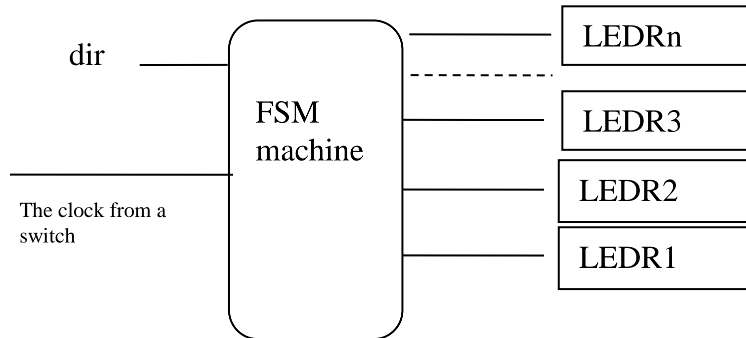
with six values: *S0, S1, S2, S3, S4, S5*. The internal signal is declared as TYPE: *state_machine* and will be assigned values in the CASE assignment group.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY machine IS
        PORT (  clk, dir: IN STD_LOGIC;
                 Q : OUT STD_LOGIC_VECTOR (2 downto 0));
END     machine;

ARCHITECTURE arc of machine IS
        TYPE state_machine IS (S0, S1, S2, S3, S4, S5); --user defined type
        SIGNAL state: state_machine; --(internal signal, wire)

BEGIN
        PROCESS(clk)
        BEGIN
                IF clk'EVENT and clk ='1' THEN
                        IF dir ='1' THEN
                                CASE state IS
                                        WHEN S0 => state <= S1; --state increment
                                        WHEN S1 => state <= S2;
                                        WHEN S2 => state <= S3;
                                        WHEN S3 => state <= S4;
                                        WHEN S4 => state <= S5;
                                        WHEN S5 => state <= S0;
                                END CASE;
                        ELSE
                                CASE state IS
                                        WHEN S0 => state <= S5; --state decrement
                                        WHEN S1 => state <= S0;
                                        WHEN S2 => state <= S1;
                                        WHEN S3 => state <= S2;
                                        WHEN S4 => state <= S3;
                                        WHEN S5 => state <= S4;
                                END CASE;
                        END IF;
                END IF;
        END PROCESS;

        WITH state SELECT
                Q   <=    "000" WHEN S0,
                           "001" WHEN S1,
                           "010" WHEN S2,
                           "011" WHEN S3;
                           "100" WHEN S4;
                           "101" WHEN S5;
END arc;
```



Design and implement your circuit for LED rotation control.

1. Design the FSM table to count even number (2,4,6,8,10,12,16) with the choice of up/down in two directions with control of the ***dir.***
2. Create a new project for the CounterFSM.
3. Include in the project your VHDL file that uses the style of code above.
4. Use the toggle switch SW0 on the DE2 board as a *dir* input for the FSM, and the pushbutton KEY0 (or another switch from GPIO) as the clock input which is applied manually (or the input can come from function generator. You can choose the frequency up to **~50Hz**. Use the onboard green light LEDG0 as the output to show the status of *dir=1 or dir=0,* and assign the state outputs to the red lights LEDn, … , LEDR2 to LEDR0. Assign the pins on the FPGA to connect to the switches and the LEDs.
5. To examine the circuit produced by Quartus II open the RTL Viewer tool.
6. Simulate the behavior of your circuit.
7. Once you are confident that the circuit works properly as a result of your simulation, download the circuit into the FPGA chip. Test the functionality of your design by applying the input sequences and observing the output LEDs. Make sure that the FSM properly transitions between states as displayed on the red LEDs,
8. Assign the state outputs to the GPIO pin. And assign the pins on the FPGA to connect to the external LEDs and observe the binary Mod-6 counter.