```vhdl
-----------------------------------------
-- Block code:  fsm_exercise.vhd
-- History:     21.Apr.2014 - 1st version (dqtm)
-- Function:    fsm exercise for mid-sem-exam
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

 ENTITY fsm_exercise IS
    PORT (
        clock   : in std_logic;
        reset_n : in std_logic;
        take1   : in std_logic;
        take2   : in std_logic;
        acknow  : in std_logic;
        count_o : out std_logic_vector(3 downto 0)
        );
END fsm_exercise;

ARCHITECTURE rtl OF fsm_exercise IS
    -- Signals & Types Declaration
    TYPE fsm_state_typ IS (idle, check_interval, count_down, wait_ack);
    SIGNAL state, next_state    : fsm_state_typ ;
    SIGNAL count, next_count    : unsigned(3 downto 0);

BEGIN
    -----------------------------------------
    -- Process for combinational logic
    -----------------------------------------
    comb_logic: PROCESS(take1,take2,acknow,state,count)
    BEGIN
    -- default statements
    next_state <= state;
    next_count <= count;

    -- state transitions
    CASE state IS
        WHEN idle =>
            -- check take1
            IF take1='1' THEN
                next_state <= check_interval;
                next_count <= to_unsigned(7,4);
            END IF;

        WHEN check_interval =>
            -- check & decrement counter
            IF count>0 THEN
                next_count <= count-1;
                --check take2
                IF take2='1' THEN
                    next_state <= count_down;
                    next_count <= to_unsigned(15,4);
                END IF;
            ELSE  -- count=0
                next_state <= idle;
            END IF;

        WHEN count_down =>
            --check counter
            IF count>0 THEN
                next_count <= count-1;
            ELSE
                next_state <= wait_ack;
            END IF;

        WHEN wait_ack =>
            --check acknowledge input
            IF acknow='1' THEN
                next_state <= idle;
            END IF;
```

```
72
73          WHEN OTHERS =>
74              next_state <= idle;
75              next_count <= to_unsigned(0,4);
76          END CASE;
77      END PROCESS comb_logic;
78
79
80      -------------------------------------------
81      -- Process for registers (flip-flops)
82      -------------------------------------------
83      flip_flops : PROCESS(clock, reset_n)
84      BEGIN
85          IF reset_n = '0' THEN
86              state <= idle;
87              count <= to_unsigned(0,4);
88          ELSIF RISING_EDGE(clock) THEN
89              state <= next_state;
90              count <= next_count;
91          END IF;
92      END PROCESS flip_flops;
93
94      -------------------------------------------
95      -- Concurrent Assignements
96      -- e.g. Assign outputs from intermediate signals
97      -------------------------------------------
98      count_o <= std_logic_vector(count);
99
100 END rtl;
101
102
103
104
105
106
107
```