```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity num_to_DISPLAY is
  port ( NUM     : in  std_logic_vector;
         DISPLAY : out std_logic_vector(6 downto 0));
end num_to_DISPLAY;

architecture mio of num_to_DISPLAY is

begin
  process(NUM)
  begin
    case NUM is
      when "0000" => DISPLAY <= "0000010"; --0
      when "0001" => DISPLAY <= "0101111"; --1
      when "0010" => DISPLAY <= "0110000"; --2
      when "0011" => DISPLAY <= "0101000"; --3
      when "0100" => DISPLAY <= "0101101"; --4
      when "0101" => DISPLAY <= "1001000"; --5
      when "0110" => DISPLAY <= "1000000"; --6
      when "0111" => DISPLAY <= "0101011"; --7
      when "1000" => DISPLAY <= "0000000"; --8
      when "1001" => DISPLAY <= "0001000"; --9
      when others => DISPLAY <= "1111111";
    end case;
  end process;
end mio;


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity esameaAPRILE is
  port(CLK       : in std_logic;
       RESET      : in std_logic;
       KM_IN      : in std_logic_vector(7 downto 0);
       MONEY_IN   : in std_logic;
       MONEY_DATA : in std_logic_vector(7 downto 0);

       DISPLAY_0  : out std_logic_vector(6 downto 0);
       DISPLAY_1  : out std_logic_vector(6 downto 0);
       DISPLAY_2  : out std_logic_vector(6 downto 0);
       RESTO      : out std_logic_vector(7 downto 0);
       LOCK       : out std_logic
       );
end esameaAPRILE;

architecture esame of esameaAPRILE is
  component num_to_DISPLAY
    port (
      NUM     : in  std_logic_vector(3 downto 0);
      DISPLAY : out std_logic_vector(6 downto 0));
  end component;
  type stato is (idle,prezzo0,prezzo1,prezzo2,verifica,sblocca);
  signal cs,ns : stato;
  signal km_diff_ck,km_diff : std_logic_vector(7 downto 0);
  signal reset_km,enable_km,reset_fsm : std_logic;
```

```
  signal km_diff_int: integer;
  signal num0,num1,num2: std_logic_vector(3 downto 0);
  signal num0_ck,num1_ck,num2_ck : std_logic_vector(3 downto 0);
  signal en_num0,en_num1,en_num2: std_logic;
  signal money_ck,money_temp : std_logic_vector(7 downto 0);
  signal timerout,reset_counter : std_logic;
  signal count, next_count: std_logic_vector(3 downto 0);

begin

  -- parte sequenziale della FSM
  process(clk)
  begin
    if CLK'event and CLK = '1' then
      if RESET = '1' then
        cs <= idle;
      else
        cs <= ns;
      end if;
    end if;
  end process;

  -- registro per (250 - KM_IN) e relativa logica di controllo
  reset_km <= RESET or reset_fsm;
  enable_km <= '0' when KM_IN = "00000000"
               else '1';
  km_diff <=  conv_std_logic_vector(250,8) - KM_IN;
  km_diff_int <= conv_integer(km_diff_ck);

  process(clk)
  begin
    if CLK'event and CLK = '1' then
      if (reset_km = '1') then
        km_diff_ck <= (others => '0');
      elsif (enable_km = '1') then
        km_diff_ck <= km_diff;
      end if;
    end if;
  end process;

  -- registri per il controllo dei display
  process(clk)
  begin
    if CLK'event and CLK = '1' then
      if (reset_km = '1') then
        num2_ck <= (others => '0');
      elsif (en_num2 = '1') then
        num2_ck <= num2;
      end if;
    end if;
  end process;

  process(clk)
  begin
    if CLK'event and CLK = '1' then
      if (reset_km = '1') then
        num1_ck <= (others => '0');
      elsif (en_num1 = '1') then
        num1_ck <= num1;
      end if;
    end if;
```

```
  end process;

  process(clk)
  begin
    if CLK'event and CLK = '1' then
      if (reset_km = '1') then
        num0_ck <= (others => '0');
      elsif (en_num0 = '1') then
        num0_ck <= num0;
      end if;
    end if;
  end process;

  -- convertitori per l'accensione dei display
  dis0: num_to_DISPLAY
    port map(
      NUM => num0_ck,
      DISPLAY => DISPLAY_0);

  dis1: num_to_DISPLAY
    port map(
      NUM => num1_ck,
      DISPLAY => DISPLAY_1);

  dis2: num_to_DISPLAY
    port map(
      NUM => num2_ck,
      DISPLAY => DISPLAY_2);

  -- contatore del denaro
  process(clk)
  begin
    if CLK'event and CLK = '1' then
      if reset_km = '1' then
        money_ck <= (others => '0');
      elsif(MONEY_IN = '1') then
        money_ck <= money_temp;
      end if;
    end if;
  end process;

  money_temp <= money_ck + MONEY_DATA;

  -- contatore 10 cicli di clock
  process(clk)
  begin
    if CLK'event and CLK = '1' then
      if reset_counter = '1' then
        count <= (others => '0');
      else
        count <= next_count;
      end if;
    end if;
  end process;

  next_count <= count +1;
  reset_counter <= RESET or not(reset_fsm);
  timerout <= '1' when count = conv_std_logic_vector(9,4) else '0';

  -- processo combinatorio, calcolo uscite e stato successivo
  process(cs,enable_km,km_diff_int,num2_ck, num1_ck,money_ck,km_diff_ck,timerout)
```

```
  begin

    RESTO <= (others => '0');
    LOCK  <= '1';

    reset_fsm <= '0';
    en_num2 <= '0';
    en_num1 <= '0';
    en_num0 <= '0';

    num2<= (others =>'0');
    num1<= (others =>'0');
    num0<= (others =>'0');

    case cs is
      when idle =>
        if (enable_km ='1') then
          ns <= prezzo2;
        else
          ns <= idle;
        end if;

      when prezzo2 => num2 <= conv_std_logic_vector(km_diff_int/100,4);
                      en_num2 <= '1';
                      ns <= prezzo1;

      when prezzo1 => num1 <= conv_std_logic_vector((km_diff_int - (conv_integer
(num2_ck) * 100))/10,4);
                      en_num1 <= '1';
                      ns <= prezzo0;

      when prezzo0 => num0 <= conv_std_logic_vector((km_diff_int - (conv_integer
(num2_ck) * 100) - (conv_integer(num1_ck) *10)),4);
                      en_num0 <= '1';
                      ns <= verifica;

      when verifica =>
        if (money_ck >= km_diff_ck) then
          RESTO <= money_ck - km_diff_ck;
          ns <= sblocca;
        else
          ns <= verifica;
        end if;

      when sblocca =>
        reset_fsm <= '1';
        LOCK <= '0';
        if (timerout = '1') then
          ns <= idle;
        else
          ns <= sblocca;
        end if;

      when others => null;
    end case;

  end process;
end esame;
```