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

entity controllore is
  port (
    CLK      : in  std_logic;
    RESET    : in  std_logic;
    PROGR    : in  std_logic;
    BUTTON   : in  std_logic;
    POS_REF  : in  unsigned(7 downto 0);
    POS_CORR : in  unsigned(7 downto 0);
    ERROR    : out unsigned(15 downto 0);
    PRENDI   : out std_logic;
    MON      : out std_logic);

end controllore;


architecture A of controllore is

  signal POS_REF_CK,error_CK, error_int : unsigned(7 downto 0);
  type stato is (idle,leggi,compara,attiva10,attiva8,attiva6,attiva4,fine);
  type stato2 is (uscita1,uscita2);
  signal cs,ns : stato;
  signal ns2,cs2 : stato2;
  signal enable_reg,reset_timer,en_timer,RESET_t, RESET_fsm : std_logic;
  signal n_count,c_count : unsigned(13 downto 0);
  signal sec10,sec8,sec6,sec4 : std_logic;
  signal n_somma,somma  : unsigned(15 downto 0);


begin  -- A

-- prima parte

  -- registro per memorizzare posizione desiderata
  process (CLK, RESET)
  begin  -- process
    if RESET = '1' then
      POS_REF_CK <= (others => '0');
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      if PROGR = '1' then
        POS_REF_CK <= POS_REF;
      end if;
    end if;
  end process;

  -- registro per memorizzare ERROR
  process (CLK, RESET)
  begin  -- process
    if RESET = '1' then
      error_CK <= (others => '0');
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      if enable_reg = '1' then
        error_CK <= error_int;
      end if;
    end if;
  end process;
```

```
  -- contatore dei secondi (per semplicità 1 sec = 1 clock in simulazione)
  -- 10 sec = 10000 cicli => 14 bit
  process (CLK, reset_t)
  begin  -- process
    if RESET_t = '1' then
      c_count <= (others => '0');
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      if en_timer = '1' then
        c_count <= n_count;
      end if;
    end if;
  end process;

  n_count <= c_count + 1;
  RESET_t <= RESET or reset_timer;

  sec10 <= '1' when c_count = 9 else '0';
  sec8  <= '1' when c_count = 7 else '0';
  sec6  <= '1' when c_count = 5 else '0';
  sec4  <= '1' when c_count = 3 else '0';
--   sec10 <= '1' when c_count = 9999 else '0';
--   sec8  <= '1' when c_count = 7999 else '0';
--   sec6  <= '1' when c_count = 5999 else '0';
--   sec4  <= '1' when c_count = 3999 else '0';


  -- FSM
  process (CLK, RESET)
  begin  -- process
    if RESET = '1' then
      cs <= idle;
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      cs <= ns;
    end if;
  end process;

  process (cs,PROGR,POS_REF_CK,POS_CORR,error_CK,sec10,sec8,sec6,sec4)
  begin  -- process

    MON <= '0';
    PRENDI <= '0';
    error_int <= conv_unsigned(0,8);
    enable_reg <= '0';
    reset_timer <= '0';
    en_timer <= '0';

    case cs is

      when idle => if PROGR = '1' then
                     ns <= leggi;
                   else
                     ns <= cs;
                   end if;

      when leggi => error_int <= POS_REF_CK - POS_CORR;
                    enable_reg <= '1';
                    ns <= compara;

      when compara => reset_timer <= '1';
                      if error_CK = 0  then
                        ns <= fine;
```

```
                    else

                      if error_CK >= 196 then
                        ns <= attiva10;
                      elsif error_CK >= 128 then
                        ns <= attiva8;
                      elsif error_CK >= 64 then
                        ns <= attiva6;
                      else
                        ns <= attiva4;
                      end if;

                    end if;

      when attiva10 => en_timer <= '1';
                       MON <= '1';
                       if sec10 = '1' then
                         ns <= leggi;
                       else
                         ns <= cs;
                       end if;

      when attiva8  => en_timer <= '1';
                       MON <= '1';
                       if sec8  = '1' then
                         ns <= leggi;
                       else
                         ns <= cs;
                       end if;

      when attiva6  => en_timer <= '1';
                       MON <= '1';
                       if sec6  = '1' then
                         ns <= leggi;
                       else
                         ns <= cs;
                       end if;

      when attiva4  => en_timer <= '1';
                       MON <= '1';
                       if sec4  = '1' then
                         ns <= leggi;
                       else
                         ns <= cs;
                       end if;

      when fine => PRENDI <= '1';
                   ns <= idle;

      when others => null;
    end case;

  end process;


-- parte seconda

  -- FSM
  process (CLK, RESET_fsm)
  begin  -- process
    if RESET_fsm = '1' then
```

```
      cs2 <= uscita1;
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      cs2 <= ns2;
    end if;
  end process;

  RESET_fsm <= RESET or PROGR;

  process(cs2,BUTTON,error_CK,somma)
  begin  -- process

    case cs2 is
      when uscita1 => ERROR <= conv_unsigned(error_CK,16);
                      if BUTTON = '1' then
                        ns2 <= uscita2;
                      else
                        ns2 <= cs2;
                      end if;

      when uscita2 => ERROR <= somma;
                      if BUTTON = '1' then
                        ns2 <= uscita1;
                      else
                        ns2 <= cs2;
                      end if;

      when others => null;
    end case;

  end process;

-- accumulatore della somma errori
  process (CLK, RESET_fsm)
  begin  -- process
    if RESET_fsm = '1' then
      somma <= (others => '0');
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      if enable_reg = '1' then
        somma <= n_somma;
      end if;
    end if;
  end process;

  n_somma <= somma + conv_unsigned(error_int,16);


end A;
```