Problem

Design and implement a synchronous circuit which after a reset user serially applies input x and y (a pair at a time), after 3 single-steps, the output z is a 2-bit number representing the number times the input x and y are different. The circuit does not work until reset. The output z ranges from 0 to 3 that is “00” to “11”.

```
entity s_in_cnt is
port (x, y : in  std_logic;
          z :out std_logic_vector(1 downto 0);
  reset, ck, b0, b1: in std_logic
      );
end s_in_cnt;
```

```
-- single step, debounce (db)
-- btn0 to enter and btn1 to reset
process(ck)
type db_state is (not_rdy, rdy, pulse);
variable db_ns: db_state;
begin
if ck='1' and ck'event then
  case db_ns is
   when not_rdy => en <= '0';
   if btn1 = '1' then db_ns := rdy; end if;
   when rdy => en <= '0';
   if btn0 = '1' then db_ns := pulse; end if;
   when pulse => en <= '1';
   db_ns := not_rdy;
   when others => null;
  end case;
end if;
end process;
end Behavioral;
```

```
----------------------------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- Synchronous circuit which after a reset user
-- serially applies input x and y (a pair at a time),
-- after 3 single steps, the output z is a 2-bit number
-- representing the number times the input x and y are
-- different. The circuit does not work until reset.
----------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity s_in_cnt is
port (x, y : in  std_logic;
         z :out std_logic_vector(1 downto 0);
   reset, ck, btn0, btn1: in std_logic
        );
end s_in_cnt;

architecture Behavioral of s_in_cnt is
signal en : std_logic;
type my_state is (inputs, chill);
signal n_s : my_state;


begin
process(en)
subtype my_int is integer range 0 to 3;
variable count, t : my_int;
begin
if en='1' and en'event then
  if reset = '1' then count := 0; t := 0; n_s <= inputs; else
    case n_s is
      when inputs =>
        t := t+1; --
        if not (x = y) then count := count + 1; end if;
        if t = 3 then n_s <= chill; end if;
      when chill => null;
      when others => null;
    end case;
  end if;
end if;
-- decoding count to vector z
case count is
  when 0 => z <= "00";
  when 1 => z <= "01";
  when 2 => z <= "10";
  when 3 => z <= "11";
  when others => null;
end case;
end process;
```