Problem

a) Design and implement a state machine to detect blocks of consecutive 2 or more ones, after a reset. The output z is a one when 2 or more consecutive ones had appeared and zero when consecutiveness disappeared. Use the single-step mechanism with buttons b0 and b1. Connect the input x and reset to switches.

Entity bca is
port (x, reset, ck, b0, b1 : in std_logic;
z : out std_logic
);
end bca;

b) Add another output called “done” to Part A. The output done is a one when the number of times blocks of 2 or more consecutive ones, appeared more than 5. Once the output done goes one it remains one until reset. The output z is the same as in Part A, it indicates an occurrence of a block.

Entity bc is
port (x, reset, ck, b0, b1 : in std_logic;
z, done : out std_logic
);
end bc;

Solution

Part b

```
----------------------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- State machine whose output is a one when a block
-- 2 or more consecutive ones appeared more than
-- 5 times. The output done remains one until reset.
----------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bc is
port (x,reset,ck,btn0,btn1 : in  std_logic;
                  z, done : out std_logic
     );
end bc;

architecture Behavioral of bc is
type my_state is (s0, s1, chill);
signal n_s : my_state;
signal en: std_logic;
begin

process(en)
subtype my_integer is integer range 0 to 7;
variable count : my_integer;
begin

if en = '1' and en'event then
  if reset = '1' then
  n_s <= s0; count := 0; z <= '0'; done <= '0';
  else
```

```
   case n_s is

    when s0 =>
       if x = '1' then
         n_s <= s1; -- 1 one
         z <= '0';
         done <= '0';
       else
         z <= '0';
         done <= '0';
       end if;

    when s1 =>
       if x = '1' then
       count := count + 1;
          if count > 5 then
            n_s <= chill;
            done <= '1';
          else
            n_s <= s1; -- 2 or more ones
            z <= '1';
            done <= '0';
          end if;
       else
       n_s <= s0; -- no longer consecutive
       z <= '0';
       done <= '0';
       end if;

    when chill =>
       done <= '1'; z <= '0';
    when others => null;
   end case;
   end if;
end if;
end process;
```

```
-- 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;
```