Problem

a) Design and implement an 8-bit register with load enable ld, and clear enable clr. Connect ld and clr to buttons.

```
entity reg8 is
port (x: in std_logic_vector(7 downto 0);
      ck, ld, clr: in std_logic;
       z: out std_logic_vector(7 downto 0)
      );
end reg8;
```

b) Design and implement an 8-bit register with a load or swap select ld_swp. If ld_swp = '0' the input x is loaded into the register, else if ld_swp = '1' the even position bits swap with their higher adjacent odd position bits, that is, the 0^{th} bit swap with the 1^{st} bit, …, the 6^{th} bit swaps with the 7^{th} bit. Connect ld_swp to a switch. Use single-step mechanism and connect b0 and b1 to buttons

```
entity reg_swp is
port (x: in std_logic_vector(7 downto 0);
      ck, ld_swp, b0, b1: in std_logic;
       z: out std_logic_vector(7 downto 0)
      );
end reg_swp;
```

Solution

b) reg_swp

```
----------------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- Register with a load or swap select ld_swp.
-- If ld_swp = '0' load the input x
-- else if ld_swp = '1' swap the even position bit
-- with their higher adjacent odd position bits
----------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg_swp is
port (x: in std_logic_vector(7 downto 0);
      ck, ld_swp, btn0, btn1: in std_logic;
      z: out std_logic_vector(7 downto 0)
        );
end reg_swp;

architecture Behavioral of reg_swp is
signal temp : std_logic_vector(7 downto 0);
signal en : std_logic;
begin
process(en)
begin
if en = '1' and en'event then
  case ld_swp is
   when '0' => temp <= x;
   when '1' =>
     for i in 0 to 3 loop
        temp(2*i) <= temp(2*i+1);
        temp(2*i+1) <= temp(2*i);
     end loop;
   when others => null;
  end case;
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;
```