x7segl.vhd : Afficheur 7 segements

```
entity x7seg is
   Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
          sevenseg : out STD_LOGIC_VECTOR (6 downto 0));
end x7seg;

architecture Behavioral of x7seg is
begin
with sw select
   sevenseg <= "1000000" when "0000",
               "1111001" when "0001",
               "0100100" when "0010",
               "0110000" when "0011",
               "0011001" when "0100",
               "0010010" when "0101",
               "0000010" when "0110",
               "1111000" when "0111",
               "0000000" when "1000",
               "0010000" when "1001",
               "0001000" when "1010",
               "0000011" when "1011",
               "1000110" when "1100",
               "0100001" when "1101",
               "0000110" when "1110",
               "0001110" when others;
end Behavioral;
```

toplevel.vhd : Afficher sur les leds, la valeur inverse des switch bit à bit

```
entity toplevel is
   Port ( sw : in STD_LOGIC_VECTOR (15 downto 0);
          led : out STD_LOGIC_VECTOR (15 downto 0));
end toplevel;

architecture Behavioral of toplevel is
begin
           led <= not sw;
end Behavioral;
```

my_add.vhd : Additionneur 4 bits

Un mot de 4 bits est saisi sur les 4 switches de droite, le second sur les 4 de gauche, l'affichage du résultat se fera sur les 5 leds de droites (retenue comprise)

```
entity my_add is
   Port ( sw : in STD_LOGIC_VECTOR (15 downto 0);
          led : out STD_LOGIC_VECTOR (4 downto 0));
end my_add;

architecture Behavioral of my_add is
begin
 led <= ('0' & sw(3 downto 0)) + ('0' & sw(15 downto 12));
end Behavioral;
```

x7seg.vhd (que pour ce TP!!!!!)

```
entity x7seg is
   Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
          p : in STD_LOGIC_VECTOR (1 downto 0);
          sevenseg : out STD_LOGIC_VECTOR (6 downto 0));
end x7seg;

architecture Behavioral of x7seg is
begin
   PROCESS(sw, p)
   BEGIN
      IF (p = "00") THEN
         CASE sw IS
           when "0000" => sevenseg <= "1000000" ;
           when "0001" => sevenseg <= "1111001" ;
           when "0010" => sevenseg <= "0100100" ;
           when "0011" => sevenseg <= "0110000" ;
           when "0100" => sevenseg <= "0011001" ;
           when "0101" => sevenseg <= "0010010" ;
           when "0110" => sevenseg <= "0000010" ;
           when "0111" => sevenseg <= "1111000" ;
           when "1000" => sevenseg <= "0000000" ;
           when "1001" => sevenseg <= "0010000" ;
           when "1010" => sevenseg <= "0001000" ;
           when "1011" => sevenseg <= "0000011" ;
           when "1100" => sevenseg <= "1000110" ;
           when "1101" => sevenseg <= "0100001" ;
           when "1110" => sevenseg <= "0000110" ;
           when "1111" => sevenseg <= "0001110" ;
           when others => sevenseg <= "0000000" ;
         END CASE;

       ELSIF (p = "01") THEN
       -- comparaison
          CASE sw IS
             when "0000" => sevenseg <= "0001110";
             when others => sevenseg <= "0000111";
          END CASE;

       ELSIF (p = "10") THEN
       -- parity
          CASE sw IS
             when "0000" => sevenseg <= "0001100";
             when others => sevenseg <= "1001111";
          END CASE;

       ELSE
          sevenseg <= "0000000";
       END IF;
    END PROCESS;
end Behavioral;
```

comp.vhd

```
entity comp is
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
           b : in STD_LOGIC_VECTOR (7 downto 0);
           res : out STD_LOGIC);
end comp;

architecture Behavioral of comp is
begin
    PROCESS(a,b)
    BEGIN
        IF a=b THEN res <= '1';
        ELSE res <= '0';
        END IF;
    END PROCESS;
end Behavioral;
```

count1.vhd

```
entity count1 is
    Port ( a : in STD_LOGIC_VECTOR (15 downto 0);
           res : out STD_LOGIC_VECTOR (4 downto 0));
end count1;

architecture Behavioral of count1 is
begin
    PROCESS(a)
    VARIABLE nombre_1 : INTEGER;
    BEGIN
        nombre_1 := 0;
        FOR I IN 0 TO 15 LOOP
            IF a(I) = '1' THEN
                nombre_1 := nombre_1 + 1;
            END IF;
        END LOOP;
        res <= conv_std_logic_vector(nombre_1, 5);
      END PROCESS;
end Behavioral;
```

parity.vhd

```
entity parity is
    Port ( a : in STD_LOGIC_VECTOR (15 downto 0);
           res : out STD_LOGIC);
end parity;

architecture Behavioral of parity is
begin
    PROCESS(a)
    VARIABLE nombre_1: STD_LOGIC;
    BEGIN
       nombre_1 := '0';
       FOR I IN 0 TO 15 LOOP
          nombre_1 := nombre_1 XOR a(I);
       END LOOP;
    res <= nombre_1;
    END PROCESS;
end Behavioral;
```

calc.vhd : Calculatrice

BTN 4	BTN 3	BTN 2	BTN 1	BTN 0	Action
0	0	0	0	0	addition 4 bits
0	0	0	0	1	and 4 bits
0	0	0	1	0	or 4 bits
0	0	0	1	1	xor 4 bits
0	0	1	x	x	comparateur 4 bits
0	1	0	x	x	parité 8 bits
1	0	0	x	x	count1 8 bits

```
entity calc is
    Port ( sw : in STD_LOGIC_VECTOR (15 downto 0);
           btnR : in STD_LOGIC;
           btnL : in STD_LOGIC;
           btnC : in STD_LOGIC;
           btnU : in STD_LOGIC;
           btnD : in STD_LOGIC;
           led : out STD_LOGIC_VECTOR (15 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end calc;

architecture Behavioral of calc is

component add4
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (3 downto 0);
           sum : out STD_LOGIC_VECTOR (4 downto 0));
end component;
```

```
component x7seg
   Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
          p : in STD_LOGIC_VECTOR (1 downto 0);
          sevenseg : out STD_LOGIC_VECTOR (6 downto 0));
end component;


component comp
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          res : out STD_LOGIC);
end component;


component parity
   Port ( a : in STD_LOGIC_VECTOR (15 downto 0);
          res : out STD_LOGIC);
end component;


component count1
   Port ( a : in STD_LOGIC_VECTOR (15 downto 0);
          res : out STD_LOGIC_VECTOR (4 downto 0));
end component;



signal sig : std_logic_vector (4 downto 0);
signal s_add, s_and, s_or, s_xor : std_logic_vector(4 downto 0);
signal s_comp, s_parity : std_logic;
signal s_count1 : STD_LOGIC_VECTOR (4 downto 0);
signal btn : std_logic_vector (4 downto 0);
signal combi : STD_LOGIC_VECTOR (7 downto 0);
signal aff : STD_LOGIC_VECTOR (1 downto 0);

begin

   cmp_add : add4 port map (a=>sw(15 downto 12), b=>sw(3 downto 0), sum=>s_add);
   s_and <= ('0' & sw(15 downto 12)) AND ('0' & sw(3 downto 0));
   s_or <= ('0' & sw(15 downto 12)) OR ('0' & sw(3 downto 0));
   s_xor <= ('0' & sw(15 downto 12)) XOR ('0' & sw(3 downto 0));
   cmp_comp : comp port map (a=>sw(15 downto 8), b=>sw(7 downto 0), res=>s_comp);
   cmp_parity : parity port map (a=>sw, res=>s_parity);
   cmp_count1 : count1 port map (a=>sw, res=>s_count1);

   btn <= btnU & btnC & btnD & btnL & btnR;
```

```
    with btn select
        sig <= s_add when "00000",
               s_and when "00001",
               s_or when "00010",
               s_xor when "00011",

               "0000" & s_comp when "00100",
               "0000" & s_comp when "00101",
               "0000" & s_comp when "00110",
               "0000" & s_comp when "00111",

               "0000" & s_parity when "01000",
               "0000" & s_parity when "01001",
               "0000" & s_parity when "01010",
               "0000" & s_parity when "01011",

               s_count1 when "10000",
               s_count1 when "10001",
               s_count1 when "10010",
               s_count1 when "10011",

               "00000" when others;

    with btn select
        aff <= "01" when "00100",
               "01" when "00101",
               "01" when "00110",
               "01" when "00111",

               "10" when "01000",
               "10" when "01001",
               "10" when "01010",
               "10" when "01011",

               "00" when "00000",
               "00" when "00001",
               "00" when "00010",
               "00" when "00011",

               "00" when "10000",
               "00" when "10001",
               "00" when "10010",
               "00" when "10011",

               "11" when others;

    cmp_x7seg : x7seg port map (sw=>sig(3 downto 0), p=>aff, sevenseg=>seg);
    led <= "00000000000" & sig;
    an <= x"e";

end Behavioral;
```

fpd.vhd : Bascule D (utile pour mémoriser l'état précédent)

```
entity fpd is
GENERIC (init_value: STD_Logic  := '0');
   Port ( d : in  STD_LOGIC;
          q : out  STD_LOGIC:= init_value;
          clk : in  STD_LOGIC);
end fpd;

architecture Behavioral of fpd is
begin
          process (clk)
          begin
          if (clk'event and clk='1') then
                    q <= d;
          end if ;
          end process ;
end Behavioral;
```

shift.vhd : 1 fpd pour chaque led + relier les entrées et sorties - Bascule 0 démarre à 1 contrairement aux autres (0)

```
entity shift is
   Port ( btn : in STD_LOGIC_VECTOR (4 downto 0);
          sw : in STD_LOGIC_VECTOR (15 downto 0);
          led : out STD_LOGIC_VECTOR (15 downto 0);
          clk : in  STD_LOGIC);
end shift;

architecture Behavioral of shift is

signal Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15 : std_logic;

component fpd
   GENERIC (init_value: STD_Logic  := '0');
   Port ( d, clk : in  STD_LOGIC;
          q : out  STD_LOGIC:= init_value);
end component;

begin
   d0: fpd GENERIC MAP(init_value => '1') PORT MAP(d => Q15,q => Q0, clk => clk);
   d1: fpd PORT MAP(d=> Q0, q=>Q1, clk=>clk);
   d2: fpd PORT MAP(d=> Q1, q=>Q2, clk=>clk);
   d3: fpd PORT MAP(d=> Q2, q=>Q3, clk=>clk);
   d4: fpd PORT MAP(d=> Q3, q=>Q4, clk=>clk);
   d5: fpd PORT MAP(d=> Q4, q=>Q5, clk=>clk);
   d6: fpd PORT MAP(d=> Q5, q=>Q6, clk=>clk);
   d7: fpd PORT MAP(d=> Q6, q=>Q7, clk=>clk);
   d8: fpd PORT MAP(d=> Q7, q=>Q8, clk=>clk);
   d9: fpd PORT MAP(d=> Q8, q=>Q9, clk=>clk);
```

```
   d10: fpd PORT MAP(d=> Q9, q=>Q10, clk=>clk);
   d11: fpd PORT MAP(d=> Q10, q=>Q11, clk=>clk);
   d12: fpd PORT MAP(d=> Q11, q=>Q12, clk=>clk);
   d13: fpd PORT MAP(d=> Q12, q=>Q13, clk=>clk);
   d14: fpd PORT MAP(d=> Q13, q=>Q14, clk=>clk);
   d15: fpd PORT MAP(d=> Q14, q=>Q15, clk=>clk);

   led<= Q15&Q14&Q13&Q12&Q11&Q10&Q9&Q8&Q7&Q6&Q5&Q4&Q3&Q2&Q1&Q0;

end Behavioral;
```

shift_vector.vhd : Ping_pong avec rebond sur bords de table et raquettes (=switchs)

```
entity shift_vector is
   Port ( btnC, btnU, btnL, btnR, btnD : in STD_LOGIC;
          sw : in STD_LOGIC_VECTOR (15 downto 0);
          led : out STD_LOGIC_VECTOR (15 downto 0);
          clk : in  STD_LOGIC);
end shift_vector;

architecture Behavioral of shift_vector is
begin
   process(clk, sw)
      variable temp : STD_LOGIC_VECTOR (15 downto 0) := "0000000000000001";
      variable sens : STD_LOGIC := '0'; --'0' signifie de droite à gauche
      variable i : integer := 0;
   begin
      if (clk'event and clk='1') then
         if (sw(i)='1'or (temp(0)='1' and sens='1') or (temp(15)='1' and sens='0')) then
            sens := not sens;
         end if;
         if (sens='0') then
            temp := temp(14 downto 0) & temp(15);
            i := i+1;
         else
            temp := temp(0) & temp(15 downto 1);
            i := i-1;
         end if;
      end if;
      led <= temp;
   end process;
end Behavioral;
```

clk_div.vhd

```
entity clk_div is
   Port ( clk : in STD_LOGIC;
          clk_4hz : inout STD_LOGIC := '0');
end clk_div;
```

```
architecture Behavioral of clk_div is
begin
   process(clk)
   variable counter : unsigned(23 downto 0):= (others => '0');
   begin
      if (clk'event and clk='1') then
         counter := counter+1;
         if(counter=X"BEBC20") then
            counter := (others => '0');
            clk_4hz <= NOT clk_4hz;
         end if;
      end if;
   end process;
end Behavioral;
```

chenillard.vhd

```
entity chenillard is
   Port ( btnC, btnU, btnL, btnR, btnD : in STD_LOGIC;
          sw : in STD_LOGIC_VECTOR (15 downto 0);
          led : out STD_LOGIC_VECTOR (15 downto 0);
          clk : in  STD_LOGIC);
end chenillard;

architecture Behavioral of chenillard is

signal new_clk : std_logic;

component clk_div
   Port ( clk : in STD_LOGIC;
          clk_4hz : inout STD_LOGIC := '0');
end component;

component shift_vector
   Port ( btnC, btnU, btnL, btnR, btnD : in STD_LOGIC;
          sw : in STD_LOGIC_VECTOR (15 downto 0);
          led : out STD_LOGIC_VECTOR (15 downto 0);
          clk : in  STD_LOGIC);
end component;

begin
   clock : clk_div PORT MAP(clk=> clk, clk_4hz=>new_clk);
   shift_v : shift_vector PORT MAP(btnC=>btnC, btnU=>btnU, btnL=>btnL, btnR=>btnR,
                                   btnD=>btnD, sw=>sw, led=>led, clk=>new_clk);
end Behavioral;
```

btn_to_number.vhd

```
entity btn_to_number is
    Port ( btnR : in STD_LOGIC;
           btnD : in STD_LOGIC;
           btnL : in STD_LOGIC;
           btnU : in STD_LOGIC;
           num : out STD_LOGIC_VECTOR (1 downto 0);
           isP : OUT STD_LOGIC);
end btn_to_number;

architecture Behavioral of btn_to_number is

signal s : STD_LOGIC_VECTOR(2 downto 0);
signal btn : STD_LOGIC_VECTOR(3 downto 0);

begin
    btn <= btnR & btnD & btnL & btnU;
    s <= "100" when (btn = "1000") else
         "101" when (btn = "0100") else
         "110" when (btn = "0010") else
         "111" when (btn = "0001") else
         "000";
    num <= s(1 downto 0);
    isP <= s(2);
end Behavioral;
```

btn_pulse.vhd

```
entity btn_pulse is
    Port ( inp : in STD_LOGIC;
           E : in STD_LOGIC;
           clk : in STD_LOGIC;
           outp : out STD_LOGIC);
end btn_pulse;

architecture Behavioral of btn_pulse is

signal q0, q1, q2, q3, q4, q5, o1 : std_logic;

begin
process (clk)
           begin
           if (clk'event and clk ='1') then
                      if E ='1' then
                                 q0 <= inp;
                                 q1 <= q0;
                                 q2 <= q1;
                      end if;
                      q3 <= o1;
                      q4 <= q3;
                      q5 <= q4;
           end if;
end process;
```

```
o1 <= q0 and q1 and q2;
outp <= q3 and q4 and not(q5) ;
end Behavioral;
```

clkdiv.vhd (= Enable190 = clk190) => E190 qui produit une impulsion durant 10ns à la fréquence de 190Hz. Le clk190 est une clock de fréquence 190Hz.

```
entity clkdiv is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           E190, clk190 : out  STD_LOGIC);
end clkdiv;

architecture Behavioral of clkdiv is
signal clkin: std_logic :='0';
begin
    process(clk,reset)
    variable q: std_logic_vector(23 downto 0):= X"000000";
    begin
        if reset ='1' then
            q := X"000000";
            clkin <= '0';
        elsif clk'event and clk = '1' then
            q := q+1;
            if Q(18)='1' and clkin='0' then
                E190 <= '1' ;
            else
                E190 <= '0';
            end if;
        end if;
        clkin<= Q(18);
    end process;

    clk190 <= clkin;
end Behavioral;
```

digicode.vhd

```
entity digicode is
  Port (clk : IN STD_LOGIC;
        sw : IN STD_LOGIC_VECTOR(7 downto 0);
        btnR, btnD, btnL, btnU, btnC : IN STD_LOGIC;
        led : OUT STD_LOGIC_VECTOR(1 downto 0));
end digicode;

architecture Behavioral of digicode is

signal E190, clockdiv : STD_LOGIC;
signal outBtnR, outBtnD, outBtnL, outBtnU, outBtnC : STD_LOGIC;
```

```
component btn_pulse
    Port ( inp : in STD_LOGIC;
           E : in STD_LOGIC;
           clk : in STD_LOGIC;
           outp : out STD_LOGIC);
end component;

component clkdiv
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           E190, clk190 : out  STD_LOGIC);
end component;

component fsm
    Port (clk : IN STD_LOGIC;
          sw : IN STD_LOGIC_VECTOR(7 downto 0);
          btnR, btnD, btnL, btnU, btnC : IN STD_LOGIC;
          led : OUT STD_LOGIC_VECTOR(1 downto 0));
end component;

begin
    cmp_clock : clkdiv port map (clk => clk, reset => '0', E190 => E190, clk190 => clockdiv);
    cmp_btnR : btn_pulse port map (inp => btnR, E => E190, clk => clockdiv, outp => outBtnR);
    cmp_btnD : btn_pulse port map (inp => btnD, E => E190, clk => clockdiv, outp => outBtnD);
    cmp_btnL : btn_pulse port map (inp => btnL, E => E190, clk => clockdiv, outp => outBtnL);
    cmp_btnU : btn_pulse port map (inp => btnU, E => E190, clk => clockdiv, outp => outBtnU);
    cmp_btnC : btn_pulse port map (inp => btnC, E => E190, clk => clockdiv, outp => outBtnC);
    cmp_fsm : fsm port map (clk => clockdiv, sw => sw, btnR => outBtnR, btnD => outBtnD, btnL =>
                                    outBtnL, btnU => outBtnU, btnC => outBtnC, led => led);
end Behavioral;
```

digicode_chenillard.vhd

```
entity digicode_chenillard is
    Port (clk : IN STD_LOGIC;
          sw : IN STD_LOGIC_VECTOR(7 downto 0);
          btnR, btnD, btnL, btnU, btnC : IN STD_LOGIC;
          led : OUT STD_LOGIC_VECTOR(7 downto 0));
end digicode_chenillard;

architecture Behavioral of digicode_chenillard is

signal led_s : STD_LOGIC_VECTOR(1 downto 0);
signal led_chenillard_s : STD_LOGIC_VECTOR(15 downto 0);
signal clock_4Hz : STD_LOGIC;
signal reset: std_logic := '1';
signal count: integer := 0;
```

```
component digicode
   Port (clk : IN STD_LOGIC;
     sw : IN STD_LOGIC_VECTOR(7 downto 0);
     btnR, btnD, btnL, btnU, btnC : IN STD_LOGIC;
     led : OUT STD_LOGIC_VECTOR(1 downto 0));
end component;

component chenillard
   Port ( btnC, btnU, btnL, btnR, btnD : in STD_LOGIC;
          sw : in STD_LOGIC_VECTOR (15 downto 0);
          led : out STD_LOGIC_VECTOR (15 downto 0);
          clk : in  STD_LOGIC);
end component;

component clk_div
   Port ( clk : in  STD_LOGIC;
          clk_4hz : inout STD_LOGIC := '0');
end component;

begin
   cmp_digicode : digicode port map (clk => clk, sw => sw, btnR => btnR, btnD => btnD, btnL =>
btnL, btnU => btnU, btnC => btnC, led => led_s);
   cmp_chenillard : chenillard port map (btnR => btnR, btnD => btnD, btnL => btnL, btnU => btnU,
btnC => btnC, sw => "0000000010000000", led => led_chenillard_s, clk => clk);
   cmp_clock : clk_div PORT MAP(clk => clk, clk_4hz => clock_4Hz);

   process(clock_4Hz, led_s, btnC)
   begin
   if (btnC = '1') then
      reset <= '1';
      count <= 0;
   end if;
   if(clock_4hz'event and clock_4hz = '1') then
      if(led_s = "10" and reset='1') then
         led <= led_chenillard_s(7 downto 0);
         count <=  count+1;
      elsif (led_s/="10" and reset='1') then
         led <= "000000"&led_s;
      end if;
      if (count = 8) then
         reset <= '0';
         count <= 0;
         led<= "00000000";
      end if;
   end if;
   end process;
end Behavioral;
```

fsm.vhd

```
entity fsm is
  Port (clk : IN STD_LOGIC;
        sw : IN STD_LOGIC_VECTOR(7 downto 0);
        btnR, btnD, btnL, btnU, btnC : IN STD_LOGIC;
        led : OUT STD_LOGIC_VECTOR(1 downto 0));
end fsm;

architecture Behavioral of fsm is
type state_type is (s1, s2, s3, s4, sOpen);
signal state, next_state : state_type;
signal sig_i : std_logic_vector(1 downto 0);
signal btn : STD_LOGIC_VECTOR(1 downto 0);
signal btnP : STD_LOGIC;

component btn_to_number
    Port ( btnR : in STD_LOGIC;
           btnD : in STD_LOGIC;
           btnL : in STD_LOGIC;
           btnU : in STD_LOGIC;
           num : out STD_LOGIC_VECTOR (1 downto 0);
           isP : OUT STD_LOGIC);
end component;

begin

SYNC_PROC : process (clk, btnC)
begin
if rising_edge(clk) then
    if (btnC = '1') then
        state <= s1;
        led <= "00";
    else
        state <= next_state;
        led <= sig_i;
    end if;
end if;
end process;

OUTPUT_DECODE : process (state)
    begin
    case (state) is
        when sOpen =>
            sig_i <= "10";
        when s1 =>
            sig_i <= "00";
        when others =>
            sig_i <= "01";
    end case;
end process;
```

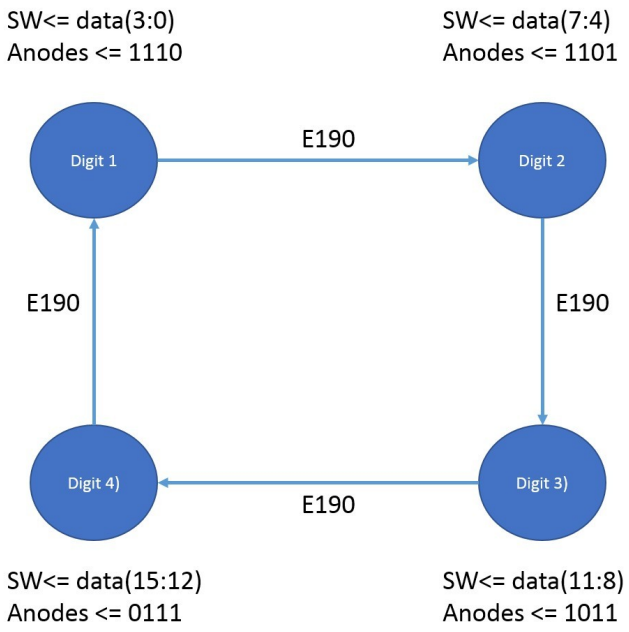
```
cmp : btn_to_number port map (btnR => btnR, btnD => btnD, btnL => btnL, btnU => btnU, num =>
btn, isP=>btnP);

NEXT_STATE_DECODE : process (state, sw, btnP, btn)
begin
   next_state <= state;
   if (btnP='1') then
      case (state) is
         when s1 =>
            if (sw(1 downto 0)=btn) then
               next_state <= s2;
            else
               next_state <= s1;
            end if;
         when s2 =>
            if (sw(3 downto 2)=btn) then
               next_state <= s3;
            else
               next_state <= s1;
            end if;
         when s3 =>
            if (sw(5 downto 4)=btn) then
               next_state <= s4;
            else
               next_state <= s1;
            end if;
         when s4 =>
            if (sw(7 downto 6)=btn) then
               next_state <= sOpen;
            else
               next_state <= s1;
            end if;
         when others =>
            next_state <= state;
      end case;
   end if;
end process;

end Behavioral;
```

afficheur_16bits.vhd

```
entity afficher_16bits is
  Port (clk : IN STD_LOGIC;
        sw : IN STD_LOGIC_VECTOR (15 downto 0);
        seg : OUT STD_LOGIC_VECTOR (6 downto 0);
        an : OUT STD_LOGIC_VECTOR (3 downto 0));
end afficher_16bits;

architecture Behavioral of afficher_16bits is

signal clockdiv : STD_LOGIC;
signal an_s : STD_LOGIC_VECTOR (3 downto 0);
signal sw_7seg : STD_LOGIC_VECTOR (3 downto 0);

component x7seg
    Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
           sevenseg : out STD_LOGIC_VECTOR (6 downto 0));
end component;

component clk190
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           E190, clk190 : out  STD_LOGIC);
end component;

component fsm_seg
  Port (clk : IN STD_LOGIC;
        an : OUT STD_LOGIC_VECTOR (3 downto 0));
end component;

begin
    cmp_clock : clk190 port map (clk => clk, reset => '0', E190 => open, clk190 => clockdiv);
    cmp_7seg : x7seg port map (sw => sw_7seg, sevenseg => seg);
    cmp_fsm : fsm_seg port map (clk => clockdiv, an => an_s);

    with an_s select
        sw_7seg <= sw(3 downto 0) when "1110",
                   sw(7 downto 4) when "1101",
                   sw(11 downto 8) when "1011",
                   sw(15 downto 12) when others;
    an <= an_s;
end Behavioral;
```

fsm_seg.vhd

```
entity fsm_seg is
  Port (clk : IN STD_LOGIC;
        an : OUT STD_LOGIC_VECTOR (3 downto 0));
end fsm_seg;

architecture Behavioral of fsm_seg is

type state_type is (st1, st2, st3, st4);
signal anodes_s : STD_LOGIC_VECTOR (3 downto 0);
signal state, next_state : state_type;

begin

SYNC_PROC : process (clk)
begin
    if rising_edge(clk) then
        state <= next_state;
        an <= anodes_s;
    end if;
end process;

OUTPUT_DECODE: process (state)
begin
    if (state = st1) then
        anodes_s <= "1110";
    elsif (state = st2) then
        anodes_s <= "1101";
    elsif (state = st3) then
        anodes_s <= "1011";
    else
        anodes_s <= "0111";
    end if;
end process;

NEXT_STATE_DECODE: process (state)
begin
    next_state <= state;
    case (state) is
        when st1 =>
            next_state <= st2;
        when st2 =>
            next_state <= st3;
        when st3 =>
            next_state <= st4;
        when st4 =>
            next_state <= st1;
        when others =>
            next_state <= st1;
      end case;
   end process;

end Behavioral;
```



add.hmd (additionneur mots de 16 bits non signé, résultats sur 17 bits => 16 bits de poids faibles, et en allumant la LED0 en cas de retenue)

```
0c00 0000 0014 ffff
201f a402 8804 201f
a402 8804 c820 8002
2010 c82b a003 03f5
1400 ffff ffff ffff
1000 0000 0004 1c00
ffff ffff ffff ffff
1c00 ffff ffff ffff
ffff ffff ffff ffff
```

random.vhd

```
entity random is
   generic ( width : integer :=  32 );
   port (
      clk , reset : in std_logic;
      enable : in std_logic;
      random_num : out std_logic_vector (width-1 downto 0)   --output vector
   );
end random;

architecture Behavioral of random is

   signal r : std_logic_vector(31 downto 0) := x"80000000";

begin
process(clk)
   begin
           if ( clk'event and clk = '1' ) then
              if (reset = '1') then
                 r <= x"80000000";
              elsif(enable = '1') then
                 r(30 downto 0) <= r(31 downto 1);
                 r(31) <= (((r(0) xor r(2)) xor r(3)) xor r(4));
              end if;
         random_num <= r;
           end if;
         end process;
end Behavioral;
```

IP_rdm.vhd

```
entity IP_rdm is
   GENERIC (Mycode : std_logic_vector(10 downto 0) := "00000010000"
);
   port (
      clk , reset : in std_logic;
      IPcode : in std_logic_vector (10 downto 0);
      Tout : out std_logic_vector (31 downto 0);
      Nout : out std_logic_vector (31 downto 0)
   );
end IP_rdm;

architecture Behavioral of IP_Rdm is

   component random
      generic ( width : integer :=  32 );
      port (
         clk , reset : in std_logic;
         enable : in std_logic;
         random_num : out std_logic_vector (width-1 downto 0)   --output vector
      );
   end component;

signal enable_s : STD_LOGIC;
signal random_num_s : std_logic_vector (31 downto 0);

begin
         cmp : random port map (clk => clk, reset => reset, enable => enable_s, random_num =>
                                    random_num_s);
         enable_s <=  '1' when ipcode(10 downto 0)= Mycode else '0';
         -- Tout <= random_num_s when ipcode(10 downto 0)= Mycode else (others =>'Z');
         Tout <= "00000000000000000000" & random_num_s(11 downto 0) when ipcode(10 downto
                  0)= Mycode else (others =>'Z');
         Nout <= "00000000000000000000" & random_num_s(23 downto 12) when ipcode(10 downto
                  0)= Mycode else (others =>'Z');
end Behavioral;
```

IP_square2.vhd

```
entity IP_square2 is
   GENERIC (Mycode : std_logic_vector(10 downto 0):="00000000000" );
   Port ( Tin : in  STD_LOGIC_VECTOR (31 downto 0);
          Nin : in  STD_LOGIC_VECTOR (31 downto 0);
          IPcode : in  STD_LOGIC_vector(10 downto 0) ;
          Tout : out  STD_LOGIC_VECTOR (31 downto 0));
end IP_square2;

architecture Behavioral of IP_square2 is
component multiply
        port (
        a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        p: out std_logic_vector(31 downto 0));
end component;

signal mul1, mul2 : STD_LOGIC_VECTOR (31 downto 0);

begin
        my_mul16_1 : multiply port map (a => Tin(15 downto 0), b => Tin (15 downto 0), p => mul1);
        my_mul16_2 : multiply port map (a => Nin(15 downto 0), b => Nin (15 downto 0), p => mul2);
        Tout <= (mul1+mul2) when Mycode = IPcode else (others =>'Z');
end Behavioral;
```

IP_cmpFFE001.vhd

```
entity IP_cmpFFE001 is
   GENERIC (Mycode : std_logic_vector(10 downto 0) := "00000000000");
   port ( Tin : in std_logic_vector (31 downto 0);
          IPcode : in std_logic_vector (10 downto 0);
          Tout : out std_logic_vector(31 downto 0));
end IP_cmpFFE001;

architecture Behavioral of IP_cmpFFE001 is

signal s : std_logic_vector(31 downto 0);

begin
   process(Tin)
      begin
         if(Tin <= x"00FFE001") then
            s <= x"00000001";
         else
            s <= x"00000000";
         end if;
    end process;

    Tout <= s when IPcode = Mycode  else (others =>'Z');
end Behavioral;
```

IP_detecte_fin.vhd

```
entity IP_detecte_fin is
   GENERIC (Mycode : std_logic_vector(10 downto 0) := "00000000000");
   port (
         Tin : in std_logic_vector (31 downto 0);
        IPcode : in std_logic_vector (10 downto 0);
        Tout : out std_logic_vector(31 downto 0)
    );
end IP_detecte_fin;

architecture Behavioral of IP_detecte_fin is

signal s : std_logic_vector(31 downto 0);

begin
   process(Tin)
      begin
         if(Tin <= 0) then
            s <= x"FFFFFFFF";
         else
            s <= x"00000000";
         end if;
    end process;
    Tout <= s when IPcode = Mycode  else (others =>'Z');
end Behavioral;
```

IP_generateFFFF.vhd

```
entity IP_generate_FFFF is
   GENERIC (Mycode : std_logic_vector(10 downto 0) := "00000000000");
   port (
         IPcode : in std_logic_vector (10 downto 0);
         Tout : out std_logic_vector(31 downto 0)
   );
end IP_generate_FFFF;

architecture Behavioral of IP_generate_FFFF is
begin
    Tout <= x"00010001" when ipcode= Mycode else (others =>'Z');
end Behavioral;
```

add32.fsh

lit 2 nbr de 16 bits, les place sur la pile et affiche le résultat sur l'afficheur 7seg pour les 16 bits de poids faibles et le 17ieme bit sur les led

```
slave
start

master
:lire_switch
  $1f
  btn
  switch
;
: main
  lire_switch
  lire_switch
  +
  7segdup
  $10
  ->
  led
;

start
  main
  $1f
  btn
endprogram
```

rdm.fsh

affiche les 16 bits de poids faibles du 50ième nombre aléatoire de la série en utilisant IP_rdm

```
:IP rdm $8810;
:IP rdm2 $8010;

slave
start

master
: main
  $30
  for
    rdm2
  next
  rdm
  7seg
;

start
  main
  $1f
  btn
endprogram
```

pi.fsh

```
:IP rdm $8810;

slave
start

master
: main
  ticraz
  0
  $10000
  for
    rdm
    dup
    $fff
    and
    dup
    mul16
    swap
    12
    ->
    $fff
    and
    dup
    mul16
    add
    $ffe001
    <=
    if
      inc
    endif
  next
  tic
  swap
  7seg
  $1f
  btn
  dup
  16
  ->
  7seg
  $1f
  btn
  7seg
  $1f
  btn
;

start
  main
endprogram
```

pi_f.fsh

```
:IP rdm $9010;
:IP square2 $C814;
:IP cmpFFE001 $AB00;
:IP detectefin $AB01;
:IP generateFFFF $8B02;

slave
start

master
: main
  ticraz
  0
  generateFFFF
  begin
    swap
    rdm
    square2
    cmpFFE001
    add
    swap
    dec
    dup
    detectefin
  until
  pop
  tic
  swap
  7seg
  $1f
  btn
  dup
  16
  ->
  7seg
  $1f
  btn
  7seg
  $1f
  btn
;

start
  main
endprogram
```