```
------------------------------------------------------------------------------------------------------------------------------------------------
-- nano_pkg.vhd
------------------------------------------------------------------------------------------------------------------------------------------------
-- 2008_0425 nanoproc
-- auteur : Vincent Fristot
-- Ecole Phelma Grenoble INP
--
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;

package NANO_PKG is
    constant len_data_bus : integer :=16;
    constant len_addr_bus : integer :=8;
    type REGISTER_FILE is array (natural range<>) of std_logic_vector(len_data_bus-1 downto 0
);

    constant alu_add : std_logic_vector(1 downto 0) := "00"; -- liste des codes alu
    constant alu_sub : std_logic_vector(1 downto 0) := "01";
    constant alu_and : std_logic_vector(1 downto 0) := "10";



end NANO_PKG;
------------------------------------------------------------------------------------------------------------------------------------------------
-- System.vhd
------------------------------------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use ieee.numeric_std.all;
library lib_nanoproc;
use lib_nanoproc.all;
USE lib_nanoproc.nano_pkg.all;

-- Systeme nanoprocesseur
-- Ecole PHELMA - Grenoble INP
-- Auteur : Vincent FRISTOT
-- date : Juillet 2008
--
entity SYSTEM is
port
(
    RESET, CLOCK_50 : in std_logic;
    KEY          : in std_logic_vector(1 downto 0); -- deux boutons poussoirs
    Filter_In    : in std_logic_vector(7 downto 0);
    Filter_Out   : out std_logic_vector(7 downto 0);
    ADC_Eocb     : in std_logic;       -- interface conv CAN
    ADC_Convstb : out std_logic;
    ADC_Rdb,DAC_wrb : out std_logic;           -- resultat seuillage
    HEX0,HEX1,HEX2,HEX3 : out std_logic_vector(6 downto 0) -- afficheurs 7 seg
);
end SYSTEM;

architecture a of SYSTEM is

component SYSTEM_PROC
```

```
PORT (  Resetn, Clock   : in std_logic;
    io_inp      : in std_logic_vector(31 downto 0);
    io_out      : out std_logic_vector(31 downto 0));
end component;

component ROM_DIGIT
port (  digit : in std_logic_vector(3 downto 0) ;
    output  : out std_logic_vector(6 downto 0) ) ;
end component;

type STATE is (S0,S1,S2);
signal Current_State, Next_State    : STATE ;
signal Reg_d, Reg_q               : std_logic_vector(7 downto 0);
signal Eocb_sync,adc_rd_csb          : std_logic;
signal syst_in,syst_out           : std_logic_vector(31 downto 0);

begin
U1:system_proc port map(
        Resetn => RESET,
        Clock  => CLOCK_50,
        io_inp => syst_in,
        io_out => syst_out);

U2:rom_digit port map(
        digit => syst_out(19 downto 16),
        output => HEX0);

U3:rom_digit port map(
        digit => syst_out(23 downto 20),
        output => HEX1);


    HEX3    <=(others=>'1');      -- devalide l'afficheur
    HEX2    <=(others=>'1');      -- devalide l'afficheur
    syst_in <= KEY & "0000000000000000000000"& reg_q; -- sortie CAN verrouillee
    Filter_Out <= syst_out(0)&"0000000"; -- 00 ou 80 selon bit 0 port I/O 00FE

    ADC_Rdb  <= ADC_rd_csb;

-- Machine Etat controle du CAN

    P_STATE : process(CLOCK_50) -- M‰emorisation des variables d'etats
    begin
      if (CLOCK_50 = '1' and CLOCK_50'EVENT ) then
    -- Initialisation € l'etat 0 sous contrŸle de "reset"
        if (RESET='0') then
        Current_State <= S0 ;
        reg_q <= "00000000";
        Eocb_sync <= '0';
        else
        Current_State <= Next_State ;
        reg_q <= reg_d;
        Eocb_sync <= ADC_Eocb;
        end if ;
      end if;
    end process P_STATE;
    P_FSM:process(Current_State, Eocb_sync, reg_q, Filter_in) -- ajout de Filter_in dans la liste de sensiiblité
    begin
    reg_d <= reg_q;         -- maintien registre
    DAC_wrb <='1' ;         -- pas d'ecriture DAC

    case Current_State is
         when S0 =>
            ADC_Convstb <= '1';         -- start conversion ADC
            ADC_rd_csb <= '1'; -- shu
            Next_State <= S1;
         when S1 =>
            ADC_Convstb <= '0';                                   -- lancement  conversion
            if (Eocb_sync='0') then
                ADC_rd_csb <= '0'; -- shu
                reg_d <= Filter_in;         -- lecture
                Next_State <= S2;
            else
                ADC_rd_csb <= '1'; -- shu
                Next_State <= S1;
            end if;

          when S2 =>
            ADC_rd_csb <= '1'; -- shu
                if (Eocb_sync='0') then        -- attente que Eocb repasse à "1"
             Next_State <= S2;
            else
             Next_State <= S0;
            end if;
            ADC_Convstb <= '1';
            DAC_wrb <='0' ;             -- Ecriture DAC

     end case;
     end process P_FSM;

end a;
```