ÍNDEX ANNEXOS

Índex annexos....1
Capítol 1: Codis Matlab....2
1.1. Simulacions amb un únic oscil·lador....2
1.1.1. Funció de l'oscil·lador de van der Pol (VDP.m)....2
1.1.2. Funció d'integració numèrica mitjançant el mètode d'Euler (myeuler.m)....3
1.1.3. Simulacions d'un oscil·lador únic de van der Pol (Grafics_xarxa_unica.m)....4
1.2. Simulacions de la xarxa amb 6 oscil·ladors....7
1.2.1. Funció per l'oscil·lador de van der Pol acoblat (VDP.m)....7
1.2.2. Simulacions amb la xarxa de 6 oscil·ladors (simulador_6osc.m)....9
1.2.3. Simulació amb la xarxa de 6 oscil·ladors amb diferents canvis de matriu (transicions) (simulador_6osc_transicio.m)....11
Capítol 2: Codis VHDL....13
2.1. Oscil·lador únic....13
2.1.1. Disseny (VDP.vhd)....13
2.1.2. Testbench (simu.vhd)....16
2.2. Disseny final: Xarxa sencera amb enviament de dades per SPI....18
2.2.1. Component dels oscil·ladors (VDP.vhd)....18
2.2.2. Component SPI (spi_master.vhd)....22
2.2.3. Disseny principal (main.vhd)....27
2.2.4. Testbench (simu_main.vhd)....37
Capítol 3: Manual d'usuari....39
Capítol 4: Bibliografia....41

CAPÍTOL 1: CODIS MATLAB

A continuació s'exposen els codis que s'han fet servir per realitzar les simulacions presentades a la memòria.

1.1. Simulacions amb un únic oscil·lador

1.1.1. Funció de l'oscil·lador de van der Pol (VDP.m)

```
function [dydt] = VDP(t,y, alpha, p, omega)
%Oscil·lador de van der Pol únic
   dydt =  [y(2); alpha*(p^2-y(1)^2)*y(2)-omega^2*y(1)];
end
```

1.1.2. Funció d'integració numèrica mitjançant el mètode d'Euler (myeuler.m)

```
function [data] = myeuler( func, cini, time_steps )
%Integració numèrica d'un ODE de segon ordre mitjançant el mètode d'Euler
%  func -> funciò a integrar
%  cini -> vector de condicions inicials [y0 y0']
%  time_steps -> vector amb els intervals de temps a integrar (es pot fer
%  servir linspace())

   z(1) = cini(1); % x
   u(1) = cini(2); % x'
   for i=1:length(time_steps)
      if (i < length(time_steps-1))
         l = time_steps(i+1) - time_steps(i); %Delta T
         z(i+1) = z(i) + l * u(i); %x
         u(i+1) = u(i) + l*func(z(i),u(i)); %x'
      end;

   end;
   data = [z; u]; % [x ; x']
end
```

1.1.3. Simulacions d'un oscil·lador únic de van der Pol (Grafics_xarxa_unica.m)

```
%Resolució de l'oscil·lador de VDP mitjançant myeuler() i ode45()
%(integració numèrica)

%Paràmetres
alpha = 0.5;
p = 0.5;
omega = 0.5;
cini = [0.5 0.1]; %Condicions inicials
cini2 = [1 0.2];
cini3 = [0.7 0.8];
cini4 = [2 2];

 %Integració numèrica amb les funcions de matlab
[t2,y2] = ode45(@(t,y) VDP(t,y, alpha, p, omega),[0 100], cini);
[t3,y3] = ode45(@(t,y) VDP(t,y, alpha, p, omega),[0 100], cini2);
[t4,y4] = ode45(@(t,y) VDP(t,y, alpha, p, omega),[0 100], cini3);
[t5,y5] = ode45(@(t,y) VDP(t,y, alpha, p, omega),[0 100], cini4);

%Integració numèrica amb euler
%Com que el format requerit per ode45() és diferent al requerit per
%myeuler(), la funció VDP està declarada tant dins com fora de l'arxiu
%(VDP.m per ode45 i f per myeuler)
f =@(x,x2) alpha*(p^2-x^2)*x2-omega^2*x;
t = linspace(0,100,1000);
d = myeuler(f,cini,t);

%%Plots
%Evolució temporal euler
figure(1);
clf;
subplot(2,2,3)
plot(t,d);
```

```
legend('y','y'''');
xlabel('Temps');
ylabel('Amplitud');
%title('Evolució temporal mètode euler');

%Limit cycle euler
subplot(2,2,4);
plot(d(1,:),d(2,:));
xlabel('y');
ylabel('y'''');

%title('Limit cycle mètode euler');

%Evoluciò temporal ode45
subplot(2,2,1);
plot(t2,y2);
legend('y','y'''');
xlabel('Temps');
ylabel('Amplitud');
%title('Evolució temporal ode45');

%Limit cycle ode45 ( 4 simus)
subplot(2,2,2);
plot(y2(:,1),y2(:,2));
%title('Limit cycle ode45');
xlabel('y');
ylabel('y'''');
%Per mostrar més gràfics amb condicions inicials diferents cal
%descomentar aquesta secció
%hold on;
%plot(y3(:,1),y3(:,2));
%plot(y4(:,1),y4(:,2));
%plot(y5(:,1),y5(:,2));
legend('y(0)=0,5, y''(0)1=0.1','y(0)=1, y''(0)=0,2','y(0)=0,7, y''(0)=0,8','y(0)=2,
y''(0)=2');
cini2 = [1 0.2];
```

```
cini3 = [0.7 0.8];
cini4 = [2 2];


%%VDP canviant els paràmetres (ode45)
%Primer tram
alpha = 0.5;
p = 0.5;
omega = 0.5;
[t3,y3] = ode45(@(t,y) VDP(t,y, alpha, p, omega),[0 50], cini);

%Segon tram
alpha = 1;
p = 0.2;
omega = 1;
cini1 = [y3(end,1) y3(end,2)];
[t4,y4] = ode45(@(t,y) VDP(t,y, alpha, p, omega),[50 100], cini1);
ttot = [t3; t4];
ytot = [y3; y4];
figure(2);
clf;
plot(ttot,ytot);

text(10,1.25,'p=0.5, alpha=0.5, omega=0.5');
xlabel('Temps');
ylabel('Amplitud');
legend('y','y'''');
text(120, 1,'p=0.2, alpha=1, omega=1');
```

1.2. Simulacions de la xarxa amb 6 oscil·ladors

1.2.1. Funció per l'oscil·lador de van der Pol acoblat (VDP.m)

```
function [dydt] = VDP(t,y, alpha, p, omega, w, n)
% Oscil·ladors de van der Pol connectats en una xarxa amb n oscil·ladors.

% Paràmetres:
% alpha, p, omega: paràmetres comuns per a tots els oscil·ladors.
% n: nombre d'oscil·ladors de la xarxa.
% w: matriu nxn amb els pesos de les connexions internes.


    %preallocate (ho aconsella el propi matlab)
    contr = zeros(1,n);
    xa = zeros(1,n);
    x = zeros(1,n);

    %x = [x1 x2 ... xn] (per facilitar les coses)
    j = 1;
    for i=1:2:2*n
        x(j) = y(i);
        j = j + 1;
    end;

    %x = [y(1) y(3)]; %valor de x per a cada oscil·lador en un instant donat

    %Càlcul del vector de contribucions
    for i=1:n %per cada oscil·lador sumem les contribucions externes
        for j=1:n %oscil·ladors externs
            contr(j) = w(i,j) * x(j); %contribució de l'oscil·lador j sobre l'i
        end
        xa(i) = x(i) + sum(contr); %pes propi + pes contribuït extern
    end
```

```
    %Vectors dels estats interns
    % [y(2); f(xa(1),alpha,p,omega...); y(3); f(xa(2),alpha,p,omega...);
    % ... y(2n); f(xa(n),alpha,p,omega...)];
    j = 0;
    for i=1:2:2*n
        j = j + 1;
        d(i:i+1) = [y(i+1) alpha*(p^2-xa(j)^2)*y(i+1)-omega^2*xa(j)];
    end
    dydt = d'; %Matlab fa les coses en columnes, no en files
end
```

1.2.2. Simulacions amb la xarxa de 6 oscil·ladors (simulador_6osc.m)

```
%%Paràmetres
alpha = 1;
p = 2;
omega = 10;
n = 6;
w = [0 -0.1 0 -0.1 0 0; %Fast
    -0.1 0 -0.1 0 0 0;
    0 -0.1 0 0 0 -0.1;
    -0.1 0 0 0 -0.1 0;
    0 0 0 -0.1 0 -0.1;
    0 0 -0.1 0 -0.1 0];

% w = [0 -0.1 0 -0.1 0 0; %Medium
%     0 0 0 -0.1 -0.1 0;
%     -0.1 0 0 0 -0.1 0;
%     -0.1 0 0 0 -0.1 0;
%     -0.1 -0.1 0 0 0 0;
%     0 -0.1 0 -0.1 0 0];
%
% w = [0 -0.1 0 -0.1 0 0; %Slow
%     0 0 -0.1 0 -0.1 0;
%     -0.1 0 0 0 0 -0.1;
%     -0.1 0 0 0 -0.1 0;
%     0 -0.1 0 0 0 -0.1;
%     0 0 -0.1 -0.1 0 0];

n_simus = 1;
 bar = waitbar(i/n,'Simulant');

%%Simulacions
 for i=1:n_simus
    n2 = 0;
    waitbar(i/n_simus);
```

```
    cini = rand(1,n*2)*12-6;
    %cini = [1 1 1 1 1 1 1 1 1 1 1 1]
    for i= 1:2:size(cini,2)
        n2 = n2 + 1;
        fprintf('y%u(0)=%f,  y%u''(0)=%f\n',n2,cini(i),n2,cini(i+1));
    end
   [t,y] = ode45(@(t,y) VDP(t,y, alpha, p, omega,w,n),[0 10], cini);
   figure(1)
   clf
   plot(t,y(:,1),'',t,y(:,3),'',t,y(:,5),'',t,y(:,7),'',t,y(:,9),'',t,y(:,11),'');
   xlabel('Temps');
   ylabel('Amplitud');
   legend('1 - L1','2 - L2','3 - L3','4 - R1','5 - R2','6 - R3');
end

delete(bar);
```

1.2.3. Simulació amb la xarxa de 6 oscil·ladors amb diferents canvis de matriu (transicions) (simulador_6osc_transicio.m)

```
%%Paràmetres
alpha = 1;
p = 2;
omega = 5;
n = 6;
wf = [0 -0.1 0 -0.1 0 0; %Fast
    -0.1 0 -0.1 0 0 0;
    0 -0.1 0 0 0 -0.1;
    -0.1 0 0 0 -0.1 0;
    0 0 0 -0.1 0 -0.1;
    0 0 -0.1 0 -0.1 0];

wm = [0 -0.1 0 -0.1 0 0; %Medium
    0 0 0 -0.1 -0.1 0;
    -0.1 0 0 0 -0.1 0;
    -0.1 0 0 0 -0.1 0;
    -0.1 -0.1 0 0 0 0;
    0 -0.1 0 -0.1 0 0];

ws = [0 -0.1 0 -0.1 0 0; %Slow
    0 0 -0.1 0 -0.1 0;
    -0.1 0 0 0 0 -0.1;
    -0.1 0 0 0 -0.1 0;
    0 -0.1 0 0 0 -0.1;
    0 0 -0.1 -0.1 0 0];

n_simus = 1;

 bar = waitbar(i/n,'Simulant');

%%Simulacions
 for i=1:n_simus
    n2 = 0;
```

```
    waitbar(i/n_simus);
    cini = rand(1,n*2)*12-6;
    for i= 1:2:size(cini,2)
        n2 = n2 + 1;
        fprintf('y%u(0)=%f,  y%u''(0)=%f\n',n2,cini(i),n2,cini(i+1));
    end


   tspan = 10;
   %cini = [0.3; 0.1; -0.3; -0.1; 0.3; 0.1; 0.3; 0.1];
   [t,y] = ode45(@(t,y) VDP(t,y, alpha, p, omega,ws,n),[0 tspan], cini);


   cini2 = y(end,:);
   [t2,y2] = ode45(@(t,y) VDP(t,y, alpha, p, omega,wf,n),[tspan 2*tspan], cini2);


   cini3 = y2(end,:);
   [t3, y3] = ode45(@(t,y) VDP(t,y, alpha, p, omega,wm,n),[2*tspan 3*tspan], cini3);


   cini4 = y3(end,:);
   [t4, y4] = ode45(@(t,y) VDP(t,y, alpha, p, omega,wf,n),[3*tspan 4*tspan], cini4);


   tf = [t; t2; t3; t4];
   yf = [y; y2; y3; y4];


   figure(1)
   clf
   plot(tf,yf(:,1),'',tf,yf(:,3),'',tf,yf(:,5),'',tf,yf(:,7),'',tf,yf(:,9),'',tf,yf(:,11),'');
   xlabel('Temps');
   ylabel('Amplitud');
   legend('L1 - 1','L2 - 2','L3 - 3','R1 - 4','R2 - 5','R3 - 6');
end


delete(bar);
```

CAPÍTOL 2: CODIS VHDL

En aquesta secció s'exposen els codis del disseny que s'ha fet servir a la FPGA. És un disseny totalment paral·lel.

2.1. Oscil·lador únic

2.1.1. Disseny (VDP.vhd)

```
--Entitat principal
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
entity VDP is
    Port ( y_out, y1_out : out sfixed(10 downto -10);
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           alpha,p,omega: in sfixed(0 downto -3));
end VDP;

architecture Behavioral of VDP is

    --Valors instantanis
    signal y, y1: sfixed(y_out'high downto y_out'low);
```

```
--Valors inicials
constant y_ini: sfixed(y'high downto y'low) := to_sfixed(0.09,y);
constant y1_ini: sfixed(y1'high downto y1'low) := to_sfixed(0.09,y1);



--p * p
constant p2_high: integer := sfixed_high(p, '*', p);
constant p2_low: integer := sfixed_low(p, '*', p);
signal p2: sfixed(p2_high downto p2_low);

--y * y
constant y2_high: integer := sfixed_high(y, '*', y);
constant y2_low: integer := sfixed_low(y, '*', y);
signal y2: sfixed(y2_high downto y2_low);

--p*p - y*y
constant p2y2_high: integer := sfixed_high(p2, '-', y2);
constant p2y2_low: integer := sfixed_low(p2, '-', y2);
signal p2y2: sfixed(p2y2_high downto p2y2_low); --p2 - y2

--alpha(p*p - y*y)
constant alphap2y2_high: integer := sfixed_high(alpha, '*', p2y2);
constant alphap2y2_low: integer := sfixed_low(alpha, '*', p2y2);
signal alphap2y2: sfixed(alphap2y2_high downto alphap2y2_low);

--alpha(p*p - y*y)y1
constant alphap2y2y1_high: integer := sfixed_high(alphap2y2, '*', y1);
constant alphap2y2y1_low: integer := sfixed_low(alphap2y2, '*', y1);
signal alphap2y2y1: sfixed(alphap2y2y1_high downto alphap2y2y1_low);

--w*w
constant w2_high: integer := sfixed_high(omega, '*', omega);
constant w2_low: integer := sfixed_low(omega, '*', omega);
signal w2: sfixed(w2_high downto w2_low);

--w*y
```

```
    constant w2y_high: integer := sfixed_high(w2, '*', y);
    constant w2y_low: integer := sfixed_low(w2, '*', y);
    signal w2y: sfixed(w2y_high downto w2y_low);

    --alpha(p*p - y*y)y1 - w*w*y (tot)
    constant tot_high: integer := sfixed_high(alphap2y2y1, '-', w2y);
    constant tot_low: integer := sfixed_low(alphap2y2y1, '-', w2y);
    signal tot: sfixed(tot_high downto tot_low);

    signal tot_petit: sfixed(16 downto -16);
begin
    y_out <= y;
    y1_out <= y1;
    p2 <= p * p;
    y2 <= y * y;
    p2y2 <= p2 - y2;
    alphap2y2 <= alpha * p2y2;
    alphap2y2y1 <= alphap2y2 * y1;
    w2 <= omega * omega;
    w2y <= w2 * y;
    tot <= alphap2y2y1 - w2y;
    tot_petit <= resize(tot,tot_petit);
    process(clk)
        begin
            if (clk = '1' and clk'event) then
                if (rst = '1') then
                    --Condicions inicials
                    y <= y_ini;
                    y1 <= y1_ini;
                else
                    y <= resize(y,y'high-1, y'low) + resize(y1,y'high-1, y'low);
                    y1 <= resize(y1+tot, y1'high, y1'low);
                end if;
            end if;
        end process;
end Behavioral;
```

2.1.2. Testbench (simu.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;

entity simu is
end simu;

architecture Behavioral of simu is

component VDP is
   Port ( y_out, y1_out: out sfixed(10 downto -10);
          clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          alpha,p,omega: in sfixed(0 downto -3));
end component;

signal clkk: std_logic := '0';
signal y, y1 : sfixed(10 downto -10);
signal rst : std_logic := '0';
signal alpha,p,omega: sfixed(0 downto -3);
constant CLK_period: time := 20 ns;

begin
   uut: VDP port map(
      alpha => alpha,
      p => p,
      omega => omega,
      clk => clkk,
      y_out => y,
      y1_out => y1,
      rst => rst
```

```
    );

    CLK_process: process
    begin
        clkk <= '0';
        wait for CLK_period/2;
        clkk <= '1';
        wait for CLK_period/2;
    end process;

    stim_process: process
    begin
        rst <= '1';
        alpha <= to_sfixed(0.5,alpha);
        omega <= to_sfixed(0.5,omega);
        p <= to_sfixed(0.5, p);
        wait for CLK_period * 3;
        rst <= '0';
        wait for 3 us;
        omega <= to_sfixed(0.25,omega);
        wait for 3 us;
        p <= to_sfixed(0.5,p);
        alpha <= to_sfixed(0.125,alpha);
        wait;

    end process;

end Behavioral;
```

2.2. Disseny final: Xarxa sencera amb enviament de dades per SPI

2.2.1. Component dels oscil·ladors (VDP.vhd)

```
--Entitat principal
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use ieee.std_logic_unsigned.all;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use ieee_proposed.fixed_float_types.all;
entity VDP is
  Generic(  param_high: integer;
            param_low: integer;
            internal_high: integer;
            internal_low: integer);

   Port ( clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          enable: in STD_LOGIC;
          alpha,p,omega: in sfixed(param_high downto param_low);
          cini: in sfixed(internal_high downto internal_low);
          y_ext: in sfixed(internal_high downto internal_low);
          y_out, y1_out: out sfixed(internal_high downto internal_low));
end VDP;


architecture Behavioral of VDP is

   --Valors instantanis
   signal y, y1: sfixed(internal_high downto internal_low); --y i y'
```

```
--Valors inicials
constant y_ini: sfixed(y'high downto y'low) := to_sfixed(0.09,y);
constant y1_ini: sfixed(y1'high downto y1'low) := to_sfixed(0.09,y1);

--y + y_ext
constant y3_high: integer := sfixed_high(y,'+',y_ext);
constant y3_low: integer := sfixed_low(y,'+',y_ext);
signal y3: sfixed(y3_high downto y3_low);

--p * p
constant p2_high: integer := sfixed_high(p, '*', p);
constant p2_low: integer := sfixed_low(p, '*', p);
signal p2: sfixed(p2_high downto p2_low);

--y * y
constant y2_high: integer := sfixed_high(y3, '*', y3);
constant y2_low: integer := sfixed_low(y3, '*', y3);
signal y2: sfixed(y2_high downto y2_low);

--p*p - y*y
constant p2y2_high: integer := sfixed_high(p2, '-', y2);
constant p2y2_low: integer := sfixed_low(p2, '-', y2);
signal p2y2: sfixed(p2y2_high downto p2y2_low); --p2 - y2

--alpha(p*p - y*y)
constant alphap2y2_high: integer := sfixed_high(alpha, '*', p2y2);
constant alphap2y2_low: integer := sfixed_low(alpha, '*', p2y2);
signal alphap2y2: sfixed(alphap2y2_high downto alphap2y2_low);

--alpha(p*p - y*y)y1
constant alphap2y2y1_high: integer := sfixed_high(alphap2y2, '*', y1);
constant alphap2y2y1_low: integer := sfixed_low(alphap2y2, '*', y1);
signal alphap2y2y1: sfixed(alphap2y2y1_high downto alphap2y2y1_low);

--w*w
constant w2_high: integer := sfixed_high(omega, '*', omega);
```

```
    constant w2_low: integer := sfixed_low(omega, '*', omega);
    signal w2: sfixed(w2_high downto w2_low);

    --w*y
    constant w2y_high: integer := sfixed_high(w2, '*', y3);
    constant w2y_low: integer := sfixed_low(w2, '*', y3);
    signal w2y: sfixed(w2y_high downto w2y_low);

    --alpha(p*p - y*y)y1 - w*w*y (tot)
    constant tot_high: integer := sfixed_high(alphap2y2y1, '-', w2y);
    constant tot_low: integer := sfixed_low(alphap2y2y1, '-', w2y);
    signal tot: sfixed(tot_high downto tot_low);
begin

    y3 <= y + y_ext;-- + y_ext; --Valor intern + contribuciÃ³ externa

    p2 <= p * p;
    y2 <= y3 * y3;
    p2y2 <= p2 - y2;
    alphap2y2 <= alpha * p2y2;
    alphap2y2y1 <= alphap2y2 * y1;
    w2 <= omega * omega;
    w2y <= w2 * y3;
    tot <= alphap2y2y1 - w2y;
       y_out <= y; --Sortida de l'oscilÂ·lador
       y1_out <= y1;

    process(clk) --ActualitzaciÃ³ de valors i reset
    begin
       if (clk = '1' and clk'event) then
          if (rst = '1') then --Condicions inicials
             y <= cini;
             y1 <= y1_ini;
          else
             if (enable = '1') then --Assignar nou valors oscilÂ·lador
                            y <= resize(y + (y1 sra 0), y, fixed_saturate, fixed_round);
```

```
                    y1 <= resize(y1+(tot sra 0), y1, fixed_saturate, fixed_round);
            end if;
        end if;
    end if;
  end process;
end Behavioral;
```

2.2.2. Component SPI (spi_master.vhd)

Font: [1]

```
---------------------------------------------------------------------------------
--
--  FileName:         spi_master.vhd
--  Dependencies:     none
--  Design Software:  Quartus II Version 9.0 Build 132 SJ Full Version
--
--  HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--  WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--  PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--  BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--  DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--  PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--  BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--  ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--  Version History
--  Version 1.0 7/23/2010 Scott Larson
--    Initial Public Release
--  Version 1.1 4/11/2013 Scott Larson
--    Corrected ModelSim simulation error (explicitly reset clk_toggles signal)
--
---------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY spi_master IS
  GENERIC(
    slaves  : INTEGER := 4;  --number of spi slaves
    d_width : INTEGER := 2); --data bus width
```

```
  PORT(
    clock   : IN     STD_LOGIC;                             --system clock
    reset_n : IN     STD_LOGIC;                             --asynchronous reset
    enable  : IN     STD_LOGIC;                             --initiate transaction
    cpol    : IN     STD_LOGIC;                             --spi clock polarity
    cpha    : IN     STD_LOGIC;                             --spi clock phase
    cont    : IN     STD_LOGIC;                             --continuous mode command
    clk_div : IN     INTEGER;                               --system clock cycles per 1/2 period of sclk
    addr    : IN     INTEGER;                               --address of slave
    tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
    miso    : IN     STD_LOGIC;                             --master in, slave out
    sclk    : BUFFER STD_LOGIC;                             --spi clock
    ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
    mosi    : OUT    STD_LOGIC;                             --master out, slave in
    busy    : OUT    STD_LOGIC;                             --busy / data ready signal
    rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
END spi_master;

ARCHITECTURE logic OF spi_master IS
  TYPE machine IS(ready, execute);                          --state machine data type
  SIGNAL state       : machine;                             --current state
  SIGNAL slave       : INTEGER;                             --slave selected for current transaction
  SIGNAL clk_ratio   : INTEGER;                             --current clk_div
  SIGNAL count       : INTEGER;                             --counter to trigger sclk from system clock
  SIGNAL clk_toggles : INTEGER RANGE 0 TO d_width*2 + 1;    --count spi clock toggles
  SIGNAL assert_data : STD_LOGIC;                           --'1' is tx sclk toggle, '0' is rx sclk toggle
  SIGNAL continue    : STD_LOGIC;                           --flag to continue transaction
  SIGNAL rx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --receive data buffer
  SIGNAL tx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --transmit data buffer
  SIGNAL last_bit_rx : INTEGER RANGE 0 TO d_width*2;        --last rx data bit location
BEGIN
  PROCESS(clock, reset_n)
  BEGIN

    IF(reset_n = '0') THEN      --reset system
      busy <= '1';              --set busy signal
```

```
    ss_n <= (OTHERS => '1');    --deassert all slave select lines
    mosi <= 'Z';                --set master out to high impedance
    rx_data <= (OTHERS => '0'); --clear receive data port
    state <= ready;             --go to ready state when reset is exited

  ELSIF(clock'EVENT AND clock = '1') THEN
    CASE state IS               --state machine

      WHEN ready =>
        busy <= '0';            --clock out not busy signal
        ss_n <= (OTHERS => '1'); --set all slave select outputs high
        mosi <= 'Z';            --set mosi output high impedance
        continue <= '0';        --clear continue flag

        --user input to initiate transaction
        IF(enable = '1') THEN
          busy <= '1';          --set busy signal
          IF(addr < slaves) THEN   --check for valid slave address
            slave <= addr;        --clock in current slave selection if valid
          ELSE
            slave <= 0;           --set to first slave if not valid
          END IF;
          IF(clk_div = 0) THEN     --check for valid spi speed
            clk_ratio <= 1;       --set to maximum speed if zero
            count <= 1;           --initiate system-to-spi clock counter
          ELSE
            clk_ratio <= clk_div;  --set to input selection if valid
            count <= clk_div;      --initiate system-to-spi clock counter
          END IF;
          sclk <= cpol;           --set spi clock polarity
          assert_data <= NOT cpha; --set spi clock phase
          tx_buffer <= tx_data;    --clock in data for transmit into buffer
          clk_toggles <= 0;        --initiate clock toggle counter
          last_bit_rx <= d_width*2 + conv_integer(cpha) - 1; --set last rx data bit
          state <= execute;        --proceed to execute state
        ELSE
```

```
    state <= ready;         --remain in ready state
  END IF;

WHEN execute =>
  busy <= '1';       --set busy signal
  ss_n(slave) <= '0'; --set proper slave select output

  --system clock to sclk ratio is met
  IF(count = clk_ratio) THEN
    count <= 1;                 --reset system-to-spi clock counter
    assert_data <= NOT assert_data; --switch transmit/receive indicator
    IF(clk_toggles = d_width*2 + 1) THEN
      clk_toggles <= 0;             --reset spi clock toggles counter
    ELSE
      clk_toggles <= clk_toggles + 1; --increment spi clock toggles counter
    END IF;

    --spi clock toggle needed
    IF(clk_toggles <= d_width*2 AND ss_n(slave) = '0') THEN
      sclk <= NOT sclk; --toggle spi clock
    END IF;

    --receive spi clock toggle
    IF(assert_data = '0' AND clk_toggles < last_bit_rx + 1 AND ss_n(slave) = '0') THEN
      rx_buffer <= rx_buffer(d_width-2 DOWNTO 0) & miso; --shift in received bit
    END IF;

    --transmit spi clock toggle
    IF(assert_data = '1' AND clk_toggles < last_bit_rx) THEN
      mosi <= tx_buffer(d_width-1);             --clock out data bit
      tx_buffer <= tx_buffer(d_width-2 DOWNTO 0) & '0'; --shift data transmit buffer
    END IF;

    --last data receive, but continue
    IF(clk_toggles = last_bit_rx AND cont = '1') THEN
      tx_buffer <= tx_data;             --reload transmit buffer
```

```
        clk_toggles <= last_bit_rx - d_width*2 + 1; --reset spi clock toggle counter
        continue <= '1';                    --set continue flag
      END IF;

      --normal end of transaction, but continue
      IF(continue = '1') THEN
        continue <= '0';     --clear continue flag
        busy <= '0';         --clock out signal that first receive data is ready
        rx_data <= rx_buffer; --clock out received data to output port
      END IF;

      --end of transaction
      IF((clk_toggles = d_width*2 + 1) AND cont = '0') THEN
        busy <= '0';           --clock out not busy signal
        ss_n <= (OTHERS => '1'); --set all slave selects high
        mosi <= 'Z';           --set mosi output high impedance
        rx_data <= rx_buffer;    --clock out received data to output port
        state <= ready;        --return to ready state
      ELSE                     --not end of transaction
        state <= execute;      --remain in execute state
      END IF;

    ELSE      --system clock to sclk ratio not met
      count <= count + 1; --increment counter
      state <= execute;   --remain in execute state
    END IF;

  END CASE;
 END IF;
END PROCESS;
END logic;
```

2.2.3. Disseny principal (main.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use ieee_proposed.fixed_float_types.all;
use IEEE.NUMERIC_STD.ALL;



entity main is
    Port( clk_in: in STD_LOGIC;
    enable: in STD_LOGIC;
            weight_select: in STD_LOGIC_vector(1 downto 0) := "00";
            rst: in STD_LOGIC;
            miso: in STD_LOGIC;

            spi_clk: buffer STD_LOGIC;
            slave_select: buffer STD_LOGIC_VECTOR(2 downto 0);

            mosi: out STD_LOGIC;
            Led: out STD_LOGIC_VECTOR(7 downto 0);
            fast_osc: in STD_LOGIC
            );
end main;

architecture Behavioral of main is

component spi_master IS
  GENERIC(
    slaves  : INTEGER := 3;  --number of spi slaves
    d_width : INTEGER := 8); --data bus width
  PORT(
    clock   : IN     STD_LOGIC;                     --system clock
```

```
    reset_n : IN     STD_LOGIC;                     --asynchronous reset
    enable  : IN     STD_LOGIC;                     --initiate transaction
    cpol    : IN     STD_LOGIC;                     --spi clock polarity
    cpha    : IN     STD_LOGIC;                     --spi clock phase
    cont    : IN     STD_LOGIC;                     --continuous mode command
    clk_div : IN     INTEGER;                       --system clock cycles per 1/2 period of sclk
    addr    : IN     INTEGER;                       --address of slave
    tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
    miso    : IN     STD_LOGIC;                     --master in, slave out
    sclk    : BUFFER STD_LOGIC;                       --spi clock
    ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
    mosi    : OUT    STD_LOGIC;                     --master out, slave in
    busy    : OUT    STD_LOGIC;                     --busy / data ready signal
    rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
END component;


constant internal_high: integer := 2;
constant internal_low: integer := -7;
constant param_high: integer := 0;
constant param_low: integer := -5;
component VDP is
  Generic(  param_high: integer := param_high;
        param_low: integer := param_low;
        internal_high: integer := internal_high;
        internal_low: integer := internal_low);

 Port ( clk : in STD_LOGIC;
       rst : in STD_LOGIC;
       enable: in STD_LOGIC;
       alpha,p,omega: in sfixed(param_high downto param_low);
       cini: in sfixed(internal_high downto internal_low);
       y_ext: in sfixed(internal_high downto internal_low);
       y_out,y1_out: out sfixed(internal_high downto internal_low));
end component;
```

```
signal data_out, data_in: STD_LOGIC_VECTOR(7 downto 0);
signal servo_select: STD_LOGIC_VECTOR(1 downto 0);

signal busy: STD_LOGIC; --Indica quan el SPI estÃ  enviant
signal send_signal, send_out: STD_LOGIC; --Senyals d'enable del SPI


constant count_max: integer := 64; --64 sim --4375000
constant count2_max: integer := 500; --500 sim --125000
signal count: integer range 0 to count_max:= 0 ; --Cicles d'espera entre cada actualitzaciÃ³ de tots els servos
signal count2: integer range 0 to count2_max:= 0; --Cicles d'espera entre l'actualitzaciÃ³ d'un servo i el segÃŒent

signal sel_osc: integer range -1 to 12 := -1;

constant count_osc_max_normal: integer := 2; --2 sim --4375000 --337500
constant count_osc_max_fast: integer := 1;
signal count_osc_max: integer range 0 to count_osc_max_normal := count_osc_max_normal;
signal count_osc: integer range 0 to count_osc_max_normal:= 0;
signal enable_osc: STD_LOGIC := '0';

signal ss: STD_LOGIC_vector(2 downto 0); --slave select
signal addr: integer range -1 to 2; --nmero del slave a qui enviem (el modul spi ho converteix en el ss que toqui)

----------
--Valors de cada oscilÂ·lador (per fer les connexions desprÃ©s)
type y_array is array (0 to 5) of sfixed(internal_high downto internal_low);
signal y: y_array;
signal y1: y_array;

--Matriu de les connexions
constant w: sfixed(0 downto -3) := to_sfixed(-0.125,0,-3); --Valor de les connexions. --Hauria de ser 0.1 perÃ² aixÃ² requeriria mÃ©s bits
constant z: sfixed(0 downto -3) := to_sfixed(0.0,0,-3);
```

```
type weight_array is array (0 to 5) of sfixed(w'high downto w'low); --'0' quan la connexiÃ³ val 0, '1' quan la connexiÃ³ Ã©s different de 0 (els valors s'assignen mÃ©s endavant)

type weight_matrix is array (0 to 5) of weight_array;

constant fast: weight_matrix :=

                ((z,w,z,w,z,z),

                (w,z,w,z,z,z),

                (z,w,z,z,z,w),

                (w,z,z,z,w,z),

                (z,z,z,w,z,w),

                (z,z,w,z,w,z));


constant medium: weight_matrix :=

        ((z,w,z,w,z,z),

        (z,z,z,w,w,z),

        (w,z,z,z,w,z),

        (w,z,z,z,w,z),

        (w,w,z,z,z,z),

        (z,w,z,w,z,z));


constant slow: weight_matrix :=

                ((z,w,z,w,z,z),

                (z,z,w,z,w,z),

                (w,z,z,z,z,w),

                (w,z,z,z,w,z),

                (z,w,z,z,z,w),

                (z,z,w,w,z,z));




signal weights: weight_matrix;-- := fast; --Matriu que faig servir



--Sumes de les contribucions externes dels oscilÂ·ladors

type contrib_array is array(0 to 5) of sfixed(internal_high downto internal_low);

signal contrib: contrib_array;
```

```
--Valors inicials dels oscilÂ·ladors. Han de ser diferents ja que si no s'acoblen els oscilÂ·ladors entre ells.
type cini_array is array(0 to 5) of sfixed(internal_high downto internal_low);
signal cini: cini_array;

--Valors dels oscilÂ·ladors adaptats al format necessari pel servo
type servo_pos_array is array(0 to 11) of unsigned(5 downto 0);
signal servo_pos: servo_pos_array;
type servo_pos_tmp_array is array(0 to 11) of ufixed(2 downto -3);
signal servo_pos_tmp: servo_pos_tmp_array;
signal send_pos: unsigned(5 downto 0); --Senyal que enviem pel SPI (vindrÃ  del servo_pos(I) que toqui)

begin
    SPI_M: spi_master port map(
        clock => clk_in,
        reset_n => not rst,
        enable => send_signal,
        cpol => '0',--
        cpha => '0',---comprovar?
        cont => '0',
        clk_div => 20,--200
        addr => addr,
        tx_data => data_out,
        miso => miso,
        sclk => spi_clk,
        ss_n => ss,
        mosi => mosi,
        busy => busy,
        rx_data => data_in);

    --Condicions inicials dels oscilÂ·ladors
    asd: for I in 0 to 5 generate
        cini(I) <= to_sfixed(-1.0+0.21*real(I),cini(I));
    end generate asd;
```

```
--Contribucions en funciÃ³ dels valors
c: for I in 0 to 5 generate
    contrib(I) <= resize(weights(I)(0) * y(0) +
                  weights(I)(1) * y(1) +
                  weights(I)(2) * y(2) +
                  weights(I)(3) * y(3) +
                  weights(I)(4) * y(4) +
                  weights(I)(5) * y(5),
                        contrib(I), fixed_saturate, fixed_round); --amb fixed_truncate no acaba de funcionar, perÃ² estalvia bastants recursos
  end generate c;


  --Connexions dels oscilÂ·ladors
    OSCILADORS: for I in 0 to 5 generate
      OSC: VDP port map(
      clk => clk_in,
      enable => enable_osc,
      rst => rst,
      alpha => to_sfixed(0.1,param_high,param_low),
      p => to_sfixed(0.1,param_high,param_low),
      omega => to_sfixed(0.2,param_high,param_low),
      cini => cini(I),
      y_ext => contrib(I), --to_sfixed(0,internal_high,internal_low)
      y_out => y(I),
      y1_out => y1(I));
  end generate OSCILADORS;



  --AdaotaciÃ³ dels senyals dels oscilÂ·ladors al format numÃ©ric per l'arduino
  --Els servos de la dreta van al revÃ©s
  ADAPT_ESQUERRA_H: for I in 0 to 2 generate
    servo_pos(I) <= unsigned(resize(y(I),1,-4)) + "100000";
    --servo_pos_tmp(I) <= to_ufixed(resize(y(I) + 1.3,servo_pos_tmp(I)));
        --servo_pos(I) <= unsigned(servo_pos_tmp(I)); --Passem a vector de 6 bits sense signe --VALOR MAXIM/MINIM DE LA ONA?
  end generate ADAPT_ESQUERRA_H;
```

```
    ADAPT_DRETA_H: for I in 3 to 5 generate
            servo_pos(I) <= unsigned(resize(-y(I),1,-4)) + "100000";
    end generate ADAPT_DRETA_H;

    SVL: for I in 6 to 8 generate
        servo_pos(I) <= "111111" when y1(I-6)(internal_high) = '0' else "100000"; --0 -> down, 1 -> up
    end generate SVL;

    SVR: for I in 9 to 11 generate
        servo_pos(I) <= "000000" when y1(I-6)(internal_high) = '0' else "100000"; -- 1 -> down, 0 -> up
    end generate SVR;

      count_osc_max <= count_osc_max_normal when fast_osc = '0' else count_osc_max_fast; --Mode normal o fast forward
        PRESCALER_OSC: process(clk_in) --Prescaler per generar els senyals d'enable dels oscilÂ·ladors
    begin
      if (rising_edge(clk_in)) then
        if (rst = '1') then
          count_osc <= 0;
          enable_osc <= '0';
        elsif (enable = '1') then
          count_osc <= count_osc + 1;
          if (count_osc = count_osc_max) then --700000
            enable_osc <= '1';
            count_osc <= 0;
            count_osc <= 0;
          else
            enable_osc <= '0';
          end if;
        end if;
      end if;
    end process;
```

```
ORDRE_SPI: process (clk_in) --prescaler + mÃ quina d'estats per decidir quin oscilÂ·lador enviar pel spi
    begin
        if (rising_edge(clk_in)) then
            if (rst = '1') then
                count <= 0;
                count2 <= 0;
                sel_osc <= -1;
                send_out <= '0';
            elsif (enable = '1') then
                if (count < count_max) then--0700000
                    count <= count + 1;
                else
                    if (count = count_max) then--0700000 --Una tirada d'actualitzacions cada X temps
                        --count2 <= count2 + 1;
                        --if (busy = '0') then --ComenÃ§em a comptar quan ja no estÃ  ocupat
                count2 <= count2 + 1;
            --end if;

            if (count2 = 0) then
                    if (sel_osc > -1 and sel_osc < 12) then --NomÃ©s enviem si hi ha servo selÂ·leccionat
                send_out <= '1';
              end if;
            else
              if (count2 = count2_max) then--20000 busy = '0'--actualitzem un servo cada X + n*t temps
                count2 <= 0;
                sel_osc <= sel_osc + 1;
              else
                send_out <= '0';
              end if;

              if (sel_osc = 12) then --Tornem a esperar
                sel_osc <= -1;
                count <= 0;
```

```
                end if;
                        end if;
                    end if;

                end if;
            end if;
        end if;
    end process;
    send_signal <= send_out and not busy;


        --Multiplexor per enviar les posicions de cada servo --(ara nomÃ©s fem servir el servo horitzontal)
    with sel_osc select servo_select <= --seleccIÃ³ de servo
      "11" when 0, --L1H
         "10" when 6, --L1V
      "01" when 3, --R1H
         "00" when 9, --R1V

      "11" when 1, --L2H
         "10" when 7, --L2V
      "01" when 4, --R2H
         "00" when 10,--R2V

      "11" when 2, --L3H
         "10" when 8, --L3V
      "01" when 5, --R3H
         "00" when 11,--R3V
      "UU" when others;

    with sel_osc select send_pos <= --seleccIÃ³ de posiciÃ³
      servo_pos(0) when 0,
      servo_pos(1) when 1,
      servo_pos(2) when 2,
      servo_pos(3) when 3,
      servo_pos(4) when 4,
```

```
    servo_pos(5) when 5,
    servo_pos(6) when 6,
    servo_pos(7) when 7,
    servo_pos(8) when 8,
    servo_pos(9) when 9,
    servo_pos(10) when 10,
    servo_pos(11) when 11,
    "UUUUUU" when others;

  with sel_osc select addr <= --selecciÃ³ de slave (segment del robot)
    0 when 0,
    1 when 1,
    2 when 2,
    0 when 3,
    1 when 4,
    2 when 5,
    0 when 6,
    1 when 7,
    2 when 8,
    0 when 9,
    1 when 10,
    2 when 11,
    -1 when others;

  weights <= slow when weight_select = "00" else
          medium when weight_select = "01" else
          fast;

  slave_select <= ss;

  data_out <= std_logic_vector(send_pos) & servo_select;

  Led <= std_logic_vector(servo_pos(0)) & weight_select;
end Behavioral;
```

2.2.4. Testbench (simu_main.vhd)

```
-- TestBench Template
 LIBRARY ieee;
 USE ieee.std_logic_1164.ALL;
 USE ieee.numeric_std.ALL;

 ENTITY testbench IS
 END testbench;

 ARCHITECTURE behavior OF testbench IS

 -- Component Declaration
        COMPONENT main is
                Port( clk_in: in STD_LOGIC;
                                enable: in STD_LOGIC;
                               weight_select: in STD_LOGIC_VECTOR := "11";
                                rst: in STD_LOGIC;
                                miso: in STD_LOGIC;
                                spi_clk: buffer STD_LOGIC;
                        slave_select: buffer STD_LOGIC_vector(2 downto 0);

                                mosi: out STD_LOGIC;
                                fast_osc: in STD_LOGIC);

        END COMPONENT;
signal clkk: std_logic := '0';
constant CLK_period: time := 20 ns;
signal rst: std_logic := '0';


 BEGIN
  CLK_process: process
  begin
     clkk <= '0';
     wait for CLK_period/2;
```

```
        clkk <= '1';
        wait for CLK_period/2;
    end process;

 -- Component Instantiation
        uut: main PORT MAP(
                clk_in => clkk,
                enable => '1',
                rst => rst,
                miso => '0',

                spi_clk => open,
                slave_select => open,

                mosi => open,
                fast_osc => '0');


 --  Test Bench Statements
     tb : PROCESS
     BEGIN
rst <= '1';
        wait for CLK_period * 5 ; -- wait until global set/reset completes
rst <= '0';
        -- Add user defined stimulus here

        wait; -- will wait forever
     END PROCESS tb;
 --  End Test Bench

 END;
```

CAPÍTOL 3: MANUAL D'USUARI

En aquest apartat es descriu el control del sistema de locomoció per al robot hexàpode mitjançant xarxes neuronals.

El mòdul de control és la placa *Nexys 2*, i consta de diferents interruptors i botons. Es designen els elements fent servir la nomenclatura de la placa, és a dir, *BTN[0-3]* i *SW[0-7]*. La funció assignada a cadascun d'ells és la següent:

- BTN0: Botó de reset. Assigna els valors inicials al sistema. Un cop es deixa de prémer comença a oscil·lar la xarxa.
- BTN1: Botó d'avenç ràpid. Provoca que els càlculs del sistema s'efectuïn molt més ràpid. Mentre es prem aquest botó no s'envia cap dada pel SPI, per evitar que el robot realitzi moviments bruscos. Útil per no haver d'esperar a que el robot faci tots els moviments durant el període transitori. Es recomana fer-lo servir, prement-lo durant uns instants, després del reset i abans d'engegar el robot.
- BTN[2-3]: No s'utilitzen.

En quant als interruptors,

- SW0: Activa i desactiva els càlculs de la xarxa i l'enviament de dades pel SPI. Per tant, quan es desactiva, el robot es manté en la posició anterior.
- SW[1-2]: Selecció de matriu per la xarxa. Les matrius disponibles són les següents:
    - «00» → Matriu lenta.
    - «01» → Matriu mitjana.
    - «10» o «11» → Matriu ràpida.

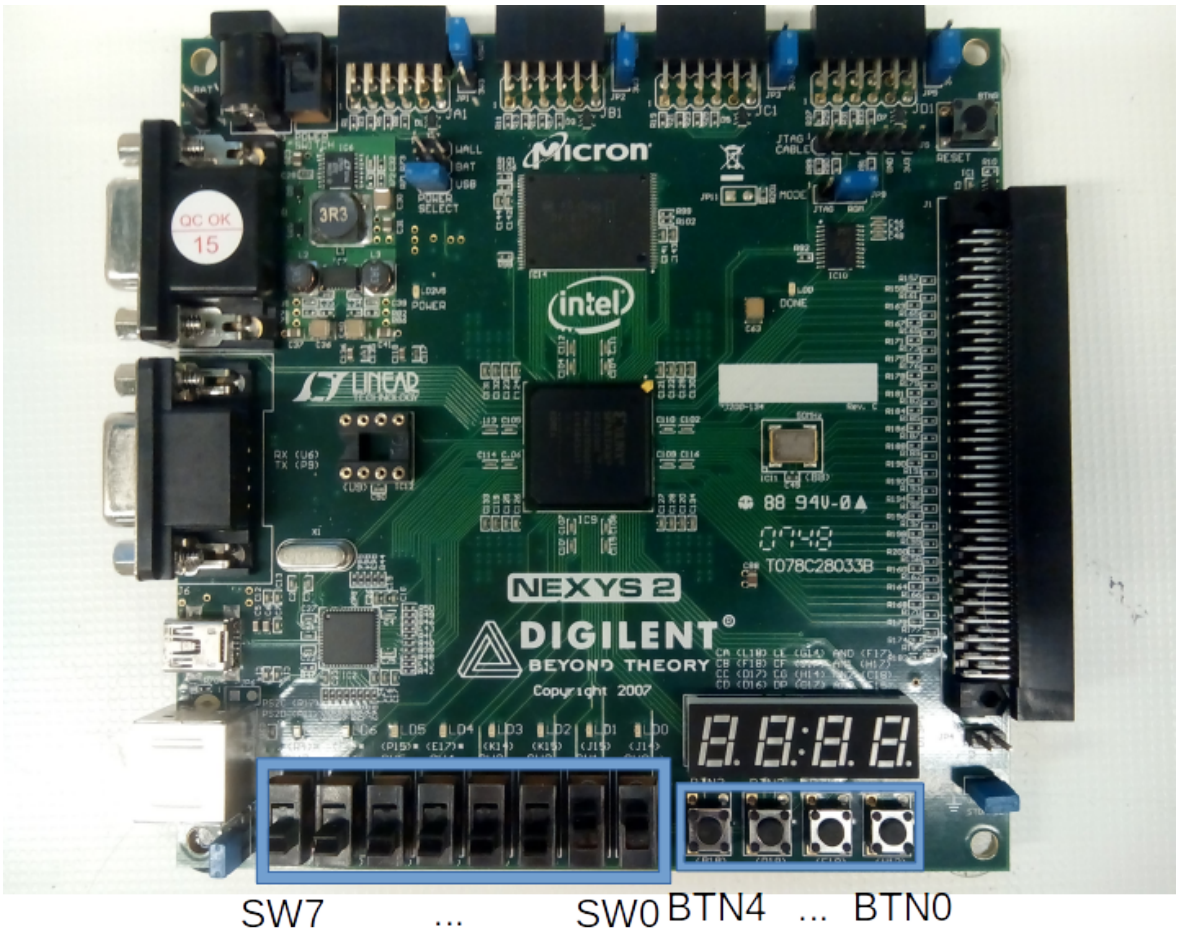

Figura 1: *Disposició dels botons i interruptors a la Nexys 2.*

La placa també disposa de 8 LEDs (LD[0-7]). Els LEDs LD[7-2] mostren el valor en binari d'un dels oscil·ladors, i poden ser útils per comprovar si la xarxa ha començat a oscil·lar.

Cal notar que si, s'engega el robot abans que la xarxa hagi arribat al règim permanent, el robot possiblement es desestabilitzarà. És per això que és recomanable fer servir el botó d'avenç ràpid ABANS d'engegar el robot.

Un altre aspecte a tenir en compte és que canvis en els codis dels *Arduinos* presents als mòduls poden implicar que el sistema deixi de funcionar correctament, ja sigui per un canvi en el protocol d'enviament de dades o en la interpretació dels valors de posició enviats.

CAPÍTOL 4: BIBLIOGRAFIA

Scott Larson, 2013. Serial Peripheral Interface (SPI) Master (VHDL). https://eewiki.net/pages/viewpage.action?pageId=4096096 (últim accés el Maig, 2016). [1]