Appendix A: BAM Stability Convergence Proofs [8]

In this appendix, the proofs of the BAM convergence starting from a given initial state configuration to a local (or global) minimum will be presented.

1 Theorem 1

The (α^f, β^f) determined by the completion of the BAM set of the decoding cycles occurs as a local minimum of the energy function.

2 Proof 1

We have first to clarify the completion of the BAM by its stable configuration after further iterations of the decoding (processing) phase.

Assume first that the pair (α^f, β^f) is not a local minimum of the energy surface. If so, there exists an α', which differs from α^f only in the s^{th} component, i.e., $a'_s \neq a_s$,

$$\alpha' = [a_1, ..., a'_s, ..., a_n]$$

such that

$$-\alpha^f M \beta_f^T = E > E' = -\alpha' M \beta^T$$

. Therefore,

$$(\alpha' - \alpha_f) M \beta_f > 0$$

$$[0, ..., 0, (a'_s - a_s), 0, ..., 0] M \beta_f^T > 0$$

and

[8] M. H. Hassoun, Associative Neural Memories, Theory and Implementation, Oxford University Press, New York, 1993.

$$(a_s' - a_s)\sum_{j=1}^{p} m_{sj} b_j > 0$$

where

$$\beta_f = [b_1, ..., b_j, ..., b_p].$$

If $a_s = 1$ then $a_s' = 0$ and

$$a_s' - a_s < 0 \qquad \sum_{j=1}^{p} m_{sj} b_j < 0.$$

Therefore, on the next cycle, the value of a_s is:

$$a_s = \begin{cases} -1 & \textit{for bipolar} \\ 0 & \textit{for binary} \end{cases}$$

If a_s is –1 or 0, then

$$a_s' - a_s > 0 \qquad \sum_{j=1}^{p} m_{sj} b_j > 0.$$

Recall that the next value is determined by:

$$\Phi\left(\sum_{j=1}^{p} b_j m_{sj}\right).$$

Therefore, the next value of a_s is 1. This means that (α^f, β^f) will be changed. This contradicts the assumption that BAM set of decoding cycles has been completed.

3 Theorem 2

If a training pair (A_i, B_i) is a local minimum, then $\Phi(A_i M) = B_i$ and $\Phi(B_i M) = A_i$.

4 Proof 2: Define A_i and A_i' as:

$$A_i = [a_1, ..., a_s, ..., a_n]$$

$$A_i' = [a_1, ..., a_s', ..., a_n]$$

$$a_s \neq a_s'.$$

Then by the use of theorem 1 and the expression for the energy function and because (A_i, B_i) is a local minimum,

$$-A_i M B_i^T < -A_i' M B_i^T.$$

That is,

$$(a_s - a_s')\sum_{j=1}^{p} m_{sj} b_j > 0.$$

If $a_s = 1$, then $(a_s - a_s') > 0$ and

$$\sum_{j=1}^{p} m_{sj} b_j > 0$$

Hence, the next a_s is 1.

If a_s is -1 or 0, then $(a_s - a_s') < 0$ and

$$\sum_{j=1}^{p} m_{sj} b_j < 0.$$

Therefore, the next a_s will be either –1 or 0. This implies that no change occurs in a_s. A similar procedure can be used to show that b_s does not change.

5 The Decreasing Energy Function of the BAM

Let us assume (α, β) be the initial states and the next state to be (α, γ), where $\gamma = \Phi(\alpha M)$. Define E_1 and E_2 as:

$$E_1 = -\alpha M \beta^T = -\sum_i \sum_j \alpha_i m_{ij} \beta_j$$

$$E_2 = -\alpha M \gamma^T = -\sum_i \sum_j \alpha_i m_{ij} \gamma_j.$$

The change in energy is given by:

$$\Delta E = E_2 - E_1 = -\sum_i \sum_j \alpha_i m_{ij} (\gamma_j - \beta_j)$$
$$= -\sum_{j \in S} \Psi_j$$

with

$$S = \{k | \gamma_k \neq \beta_k \ .$$

If $\gamma_k = 1$, then

$$\sum_i \alpha_i m_{ik} > 0, \quad \gamma_k - \beta_k > 0 \Rightarrow \Psi_k > 0.$$

Whereas if $\gamma_k = -1$,

$$\sum_i \alpha_i m_{ik} < 0, \quad \gamma_k - \beta_k < 0 \Rightarrow \Psi_k > 0.$$

Therefore, $\Delta E < 0$ if $\beta \neq \gamma$. Because α and β are of finite dimension, ΔE will eventually become zero which implies that the next state will be the same as the current one.

Appendix B: VHDL X-Layer PLP-Neuron Description of BAM (24-32).

```
--VHDL Source Code for the 'X' layer Neuron (similar to the 'Y' Layer Neuron) of a B.A.M Network
-- composed by two layers of neurons: 24 neurons by 32 neurons.
--Parallel Implementation Version.
--Designed and tested by:
-- Mr. Saffih Faycal
--Biophysics Laboratory, Physics Department,
--University of Malaya, 50603 Kuala Lumpur, Malaysia.
--E-mail. Hosain@hotmail.com, W. W. Web homepage: http://members.tripod.com/~Saffih/faysal.html
--December 1998
-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity NRNXP24_32 is
  generic( N : integer := 32; B : integer := 5;
        S : integer := 5 ; T : integer := 9);
       --N: nb of incoming signals,
       --B: nb of BITS to store the number of memories
       --S: nb of BITS to store the ext|inh_sum
       --T>S: nb of BITS to store the tot_ext|inh
 port (
      ex_in   : in std_logic; --external value
      load    : in std_logic; --reset
      clk     : in std_logic; --clock beat
      w_up    : in std_logic; --wake_up signal
      tb_assv : in std_logic_vector(0 to N); --testbit_associated vector
      sgl_in  : in std_logic_vector(0 to N-1); --in-comming signal
      nb_mem  : in std_logic_vector(0 to B-1); --number of stored mem
      output  : out std_logic ; --neuron output
      send    : out std_logic); --getting the output
end ;


architecture ANRNXP24_32 of NRNXP24_32 is

 signal key   : std_logic_vector(1 downto 0);
 signal testlim : std_logic_vector(B-1 downto 0);
 signal testco  : std_logic_vector(B-1 downto 0);

 signal inhibi  : std_logic := '0';
 signal inh_sum : std_logic_vector(0 to S-1);
 signal sgl_h   : std_logic := '0';
 signal tot_inh : std_logic_vector(0 to T-1);

 signal ext     : std_logic_vector(0 to N-1);
 signal inh     : std_logic_vector(0 to N-1);

 signal excita  : std_logic := '0';
 signal ext_sum : std_logic_vector(0 to S-1);
 signal sgl_e   : std_logic := '0';
 signal tot_ext : std_logic_vector(0 to T-1);
```

```
begin

 process(tb_assv) --Calculation of Ext_Inh syn_str on the basis of the target bit and its ass_vector
  begin
   for i in 1 to N loop
     inh(i-1) <= tb_assv(0) xor tb_assv(i);
   end loop;
 end process;

 ext  <= not inh;

 key_cre:process(load,w_up)-- THE key UPDAYTING
 begin
   key <= load & w_up;
 end process;

 calcul:process(clk)--PRCESSING THE BEHAVIOUR OF THE NEURON
 variable counter   : std_logic_vector(0 to B-1) := (others => '0');
 variable pre_output : std_logic; -- previous output
 variable limit     : std_logic_vector(0 to B-1);
 variable temp      : std_logic;
 begin
 if(clk = '1') then  --wait until clk='1' and clk'event;
  case key is
  when "00" =>

    limit := nb_mem;
    testlim <= limit;
    send <= '0';  --IF ENTERING THE NB_MEM OR WORKING THE SEND
    counter := (others => '0');--SIGNAL SHOULD BE ATTENUATED -> '0'
    testco <= counter;

  when "10" =>

    pre_output := ex_in;
    output <= pre_output;
    send <= '0';
    counter := (others => '0');
    testco <= counter;

  when "01" =>

  temp := inh(0);
  for i in 1 to N-1 loop
     temp := temp or inh(i);
  end loop;

  if counter /= limit then
    if(temp /= '0') then
     counter := unsigned(counter) + 1;
     testco <= counter;

     excita <= not excita;
     inhibi <= not inhibi;
    end if;
  else
```

```
    if    tot_ext > tot_inh then
     pre_output := '1';
    elsif tot_ext < tot_inh then
      pre_output := '0';
    elsif tot_ext = tot_inh then
      pre_output := pre_output;
    end if;
      output <= pre_output;
      counter := (others => '0');
      testco <= counter;
      send <= '1';
  end if;

 when others =>
 end case ;
end if;
end process;

exito :process(excita)  --MULTIPLICATION, SUMMATION FOR EXCITATORY SYN_STRENGTH
variable ext_sumf  : unsigned(0 to S-1) ;
variable pext      : unsigned(0 to N-1);
begin
    pext := unsigned(sgl_in AND ext); --muliplying the input signal with --inh_syn_str
    ext_sumf := (ext_sumf'range => '0');

    for i in 0 to N-1 loop          --PARLL SUMMATION
    ext_sumf := ext_sumf + pext(i);
    end loop;

    ext_sum <= conv_std_logic_vector(ext_sumf,S);
    sgl_e <= not sgl_e;
end process;

acc_ext :process(sgl_e)
variable ext_acc   : unsigned(0 to T-1);
variable ext_ac1   : unsigned(0 to S-1); --is it necessary --<ext_ac1>=<ext_acc>
begin
    if (unsigned(testco) = 1) then
      ext_acc := (ext_acc'range => '0');
    end if;

    ext_ac1 := unsigned(ext_sum);
    ext_acc := ext_ac1 + ext_acc;
    tot_ext <= conv_std_logic_vector(ext_acc,T);
end process;

inhito :process(inhibi) --MULTIPLICATION, SUMMATION FOR INHIBITORY SYN_STRENGTH
variable inh_sumf  : unsigned(0 to S-1) ;
variable pinh      : unsigned(0 to N-1);
begin
    pinh := unsigned(sgl_in AND inh); --muliplying the input signal with --inh_syn_str
    inh_sumf := (inh_sumf'range => '0');

    for i in 0 to N-1 loop          --SUMMATION
    inh_sumf := inh_sumf + pinh(i);
    end loop;
```

```
    inh_sum <= conv_std_logic_vector(inh_sumf,S);
    sgl_h <= not sgl_h;
end process;

acc_inh :process(sgl_h)
variable inh_acc   : unsigned(0 to T-1);
variable inh_ac1   : unsigned(0 to S-1); --is it necessary --<ext_ac1>=<ext_acc>
begin
    if (unsigned(testco) = 1) then
      inh_acc := (others => '0');
    end if;

    inh_ac1 := unsigned(inh_sum);
    inh_acc := inh_ac1 + inh_acc;
    tot_inh <= conv_std_logic_vector(inh_acc,T);
end process;

end ANRNXP24_32;

configuration CNRNXP24_32 of NRNXP24_32 is
  for ANRNXP24_32
  end for;
end;
```

Appendix C: VHDL PLP-Neuron-based BAM (24-32) Description.

```
--VHDL Source Code of the Top-Level Hierarchy (neurons assembling) of a B.A.M Network
-- B.A.M network is composed by two layers of neurons: 24 neurons by 32 neurons.
--Parallel Implementation Version.
--Designed and tested by:
-- Mr. Saffih Faycal
--Biophysics Laboratory, Physics Department,
--University of Malaya, 50603 Kuala Lumpur, Malaysia.
--E-mail. Hosain@hotmail.com, W. W. Web homepage: http://members.tripod.com/~Saffih/faysal.html
--December 1998
-----------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.typop24_32.all;

entity GBAMPD24_32 is

 generic( Tx : integer := 9; Ty : integer := 9; Sx : integer := 5; Sy : integer := 5);
 --Ny neurons in the 'Y' layer <-> Nx neurons in the 'X' layer.
 port (
    load      : in std_logic;
    clk       : in std_logic;
    nb_mem    : in std_logic_vector(0 to B-1);
    ex_iny    : in std_logic_vector(0 to Ny-1);
    ex_inx    : in std_logic_vector(0 to Nx-1);
    tb_x      : in std_logic_vector(0 to Nx-1);
    tb_y      : in std_logic_vector(0 to Ny-1);
    --WE TAKE Nx<Ny =>inpx_inh through the layer 'X';
    out_x     : out std_logic_vector(0 to Nx-1); --output values of the 'X' layer
    send_x    : out std_logic_vector(0 to Nx-1); --"send" signals of the 'X' layer
    out_y     : out std_logic_vector(0 to Ny-1); --output values of the 'Y' layer
    send_y    : out std_logic_vector(0 to Ny-1)); --"send" signals of the 'Y' layer
    -- tb_assvxy AND tb_assvyx should be SLICEABLE
end GBAMPD24_32;

architecture AGBAMPD24_32 of GBAMPD24_32 is
 component NRNXP24_32
 generic( N,B,S,T : in positive );--N: nb of incoming signals,
                  --B: nb of BITS to store the number of memories
                  --S: nb of BITS to store the ext|inh_sum
                  --T>S: nb of BITS to store the tot_ext|inh
port (
    ex_in  : in std_logic; --external value
    load   : in std_logic; --reset
    clk    : in std_logic; --clock beat
    w_up   : in std_logic; --wake_up signal
    tb_assv : in std_logic_vector(0 to N); -- testbit_associated vector
    sgl_in  : in std_logic_vector(0 to N-1); --in-comming signal
    nb_mem  : in std_logic_vector(0 to B-1); --number of stored mem
    output  : out std_logic ; --neuron output
    send    : out std_logic); --getting the output
end component;
```

```
 component  NRNYP32_24
 generic( N,B,S,T : in positive );--N: nb of incoming signals,
                  --B: nb of BITS to store the number of memories
                  --S: nb of BITS to store the ext|inh_sum
                  --T>S: nb of BITS to store the tot_ext|inh
port (
    ex_in   : in std_logic; --external value
    load    : in std_logic; --reset
    clk     : in std_logic; --clock beat
    w_up    : in std_logic; --wake_up signal
    tb_assv : in std_logic_vector(0 to N); -- testbit_associated vector
    sgl_in  : in std_logic_vector(0 to N-1); --in-comming signal
    nb_mem  : in std_logic_vector(0 to B-1); --number of stored mem
    output  : out std_logic ; --neuron output
    send    : out std_logic); --getting the output
end component;

 signal sx_wy      : std_logic_vector(0 to Nx-1); --sendS of 'X' to w_upS of 'Y'
 signal x_y        : std_logic_vector(0 to Nx-1); --outputS of 'X' to sgl_inS of 'Y'
 signal w_upy      : std_logic;
 signal sy_wx      : std_logic_vector(0 to Ny-1); --sendS of 'Y' to w_upS of 'X'
 signal y_x        : std_logic_vector(0 to Ny-1); --outputS of 'Y' to sgl_inS of 'X'
 signal w_upx      : std_logic;
 signal tb_assvx   : assbv_xy;
 signal tb_assvy   : assbv_yx;
 signal teststep   : std_logic;
 signal testws     : std_logic;
begin

 process(tb_x,tb_y)
 begin
  for i in 0 to (Nx - 1) loop
   tb_assvx(i)(0) <= tb_x(i);
   tb_assvx(i)(1 to Ny) <= tb_y;
  end loop;

  for i in 0 to (Ny - 1) loop
   tb_assvy(i)(0) <= tb_y(i);
   tb_assvy(i)(1 to Nx) <= tb_x;
  end loop;
 end process;

--COMPONENTS CONFIGURATION
g0 :for i in 0 to Nx-1 generate
  g1 : NRNXP24_32
     generic map (N => Ny,B => B,S => Sx,T => Tx)
     port map (ex_inx(i),load,clk,w_upx,tb_assvx(i),y_x,nb_mem,x_y(i),sx_wy(i));
 end generate;

 g2 :for i in 0 to Ny-1 generate
  g3 : NRNYP32_24
     generic map (N => Nx,B => B,S => Sy,T => Ty)
     port map (ex_iny(i),load,clk,w_upy,tb_assvy(i),x_y,nb_mem,y_x(i),sy_wx(i));
 end generate;
 send_x <= sx_wy ; send_y <= sy_wx ;
 out_x  <= x_y   ; out_y  <= y_x   ;
```

```
process(clk)
variable tmpx : std_logic;
variable tmpy : std_logic;
variable step : std_logic;
variable sw   : std_logic;
variable temp : std_logic;
variable ss : std_logic_vector(1 downto 0);
begin
ss := step & sw;

if (clk = '1') then
 if (load = '1') then
   step := '0';
   sw   := '0';
   tmpx := '0';
   tmpy := '0';
   --the keys should be "10" : storing initial value
   step := not step;    --step='1',sw='0'
 else --in nb_mem or processing calcul
  case ss is
  when "10" =>

    tmpx := '0';
    tmpy := '0';
    step := not step;   --step='0',sw='0'

  when "00" =>

       tmpx := not tmpy;
       tmpy := not tmpx;
       sw := '1';  --step='0',sw='1'

  when "01" =>

       if(sx_wy(0) = '1' or sy_wx(0) ='1') then
          step := '1'; sw := '1';
       end if;

  when "11" =>

    if (sx_wy(0) = '1') then
      -- SWTCHING BETWEEN THE WORKING PERIODS OF THE TWO LAYERS
       temp := tmpx;
       tmpx := tmpy;
       tmpy := temp;
       step := '0'; sw := '0';
    elsif (sy_wx(0) = '1') then
       temp := tmpx;
       tmpx := tmpy;
       tmpy := temp;
       step := '0'; sw := '0';
    end if;

  when others =>
  end case;
```

```
    end if;
  end if;

  testws <= sw;
  teststep <= step;
  w_upx <= tmpx;
  w_upy <= tmpy;
 end process;

end AGBAMPD24_32;

configuration CGBAMPD24_32 of GBAMPD24_32 is
 for AGBAMPD24_32
  for g0
   for g1 : NRNXP24_32
   use entity work.NRNXP24_32(ANRNXP24_32);
   end for;
  end for;
  for g2
   for g3 : NRNYP32_24
   use entity work.NRNYP32_24(ANRNYP32_24);
   end for;
  end for;
 end for;
end CGBAMPD24_32;
```

Appendix D: VHDL Systolic-like BAM (24-32) Expansion Description.

```
--VHDL Source Code of the Systolic-like architecture with its application for the expandability of the
-- B.A.M network. Expansion factor is equal to four for interconnection four B.A.M units
-- B.A.M units are composed by two layers of neurons: 24 neurons by 32 neurons.
--Parallel Implementation Version.
--Designed and tested by:
-- Mr. Saffih Faycal
--Biophysics Laboratory, Physics Department,
--University of Malaya, 50603 Kuala Lumpur, Malaysia.
--E-mail. Hosain@hotmail.com, W. W. Web homepage: http://members.tripod.com/~Saffih/faysal.html
--December 1998
-----------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

package system_compo is
 constant Nx : integer := 24;--:NUMBER OF NEURONS IN THE 'X' LAYER
 constant Ny : integer := 32;--:NUMBER OF NEURONS IN THE 'Y' LAYER
 constant B  : integer := 5 ;--:NUMBER OF BITS TO STORE THE NUM_MEM
 constant AS4 : integer := 4 ;-- nb of associations of the system
 constant ser_cap : integer := 24;
 constant sh_cap  : integer := 32;
 constant Nx4     : integer := 96;
 constant Ny4     : integer := 128;
 type assbv_xy is array (0 to Nx-1) of std_logic_vector(0 to Ny);
 type assbv_yx is array (0 to Ny-1) of std_logic_vector(0 to Nx);
 type tb_xy4 is array (0 to AS4-1) of std_logic_vector(0 to (Nx4 + Ny4 - 1));
 type sh_memo is array (0 to 3) of std_logic_vector(0 to sh_cap - 1);--for'Y'(32)
 type ser_memo is array (0 to 3) of std_logic_vector(0 to ser_cap - 1);--for'X'(24)
 type a_intercon is array (0 to 3) of std_logic_vector(0 to (Nx4 + Ny4 - 1));--shifter of the assoc.'s.
 type m_intercon is array (0 to 3) of std_logic_vector(0 to sh_cap - 1);--shift_memory .
 type conf4x is array (0 to 3) of std_logic_vector(0 to Nx-1);
 type conf4y is array (0 to 3) of std_logic_vector(0 to Ny-1);
end;
-----------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity PAR_SHIFT is
 generic (N          : integer );
 port (
      SD, LOAD, RSTn, CLK : in std_logic;--SD -> staied at the current memo
      LOADVec       : in std_logic_vector(0 to N-1);
      IN_PUT        : in std_logic_vector(0 to N-1);
      OUT_PUT       : out std_logic_vector(0 to N-1));
end;

architecture A_SHIFT of PAR_SHIFT is
 begin

process (RSTn, CLK)
  variable temp : std_logic_vector (0 to N-1);
begin
```

```
   if   (RSTn ='0') then
     OUT_PUT <= (others => '0');--load
   elsif (CLK = '1' and CLK'event) then
    if (SD = '0') then
     if (LOAD = '0') then
      OUT_PUT <= LOADVec;
     else
      OUT_PUT <= IN_PUT;
     end if;
    else
      null;
    end if;
   end if;
 end process;
end ;

configuration C_SHIFT of PAR_SHIFT is
   for A_SHIFT
   end for;
end ;
-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.system_compo.all;

entity synap_memo4 is
port (
       rst : in std_logic;
       LOAD : in std_logic;
       clk : in std_logic;
       stay : in std_logic;
       sh_input : in sh_memo;
       ser_input : in ser_memo;
       sh_output : out std_logic_vector(0 to sh_cap-1);
       ser_output : out ser_memo);
end;

architecture a_synap_memo of synap_memo4 is

 component PAR_SHIFT32
  generic (N              : in integer := 32);
  port (
        SD, LOAD, RSTn, CLK : in std_logic;--rstn=0->load, rstn=1->shift
        LOADVec             : in std_logic_vector(0 to N-1);
        IN_PUT              : in std_logic_vector(0 to N-1);
        OUT_PUT             : out std_logic_vector(0 to N-1));
end component;

component PAR_SHIFT24
 generic (N              : in integer := 24);
 port (
       SD, LOAD, RSTn, CLK : in std_logic;--rstn=0->load, rstn=1->shift
       LOADVec             : in std_logic_vector(0 to N-1);
       IN_PUT              : in std_logic_vector(0 to N-1);
       OUT_PUT             : out std_logic_vector(0 to N-1));
end component;
```

```
 signal zero_ser   : std_logic_vector(0 to ser_cap-1);
 signal postblok   : m_intercon;
 begin

zero_ser <= (others => '0');

 paralleling: for i in 0 to 3 generate
  par1: PAR_SHIFT24 --these units are only for || loading No-shifting=>zero_ser
      generic map (N => ser_cap)
      port map (stay, stay, rst, clk, ser_input(i), zero_ser, ser_output(i));
 end generate;--2nd loc for LOAD before: look at stay and LOAD in this unit

 shifting: for i in 0 to 3 generate
  sh1: if (i = 0) generate
   blok0: PAR_SHIFT32
     generic map (N => sh_cap)
     port map (stay, LOAD, rst, clk, sh_input(i), postblok(3), postblok(i));
  end generate;

  sh2: if (i > 0) and (i < 3) generate
   blok1: PAR_SHIFT32
     generic map (N => sh_cap)
     port map (stay, LOAD, rst, clk, sh_input(i), postblok(i-1), postblok(i));
  end generate;
  sh3: if (i = 3) generate
   blok3: PAR_SHIFT32
     generic map (N => sh_cap)
     port map (stay, LOAD, rst, clk, sh_input(i), postblok(i-1), postblok(i));
  end generate;
 end generate;

sh_output <= postblok(3);
end a_synap_memo;

configuration c_synap_memo of synap_memo4 is
 for a_synap_memo
  for paralleling
    for all : PAR_SHIFT24
    use entity work.PAR_SHIFT(A_SHIFT);
    end for;
  end for;

  for shifting
   for sh1
    for all : PAR_SHIFT32
    use entity work.PAR_SHIFT(A_SHIFT);
    end for;
   end for;
   for sh2
    for all : PAR_SHIFT32
     use entity work.PAR_SHIFT(A_SHIFT);
    end for;
   end for;
   for sh3
    for all : PAR_SHIFT32
     use entity work.PAR_SHIFT(A_SHIFT);
```

```
    end for;
  end for;
 end for;
end for;

end c_synap_memo;
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.system_compo.all;

entity int_4bamplp24_32 is
 port (
    start  : in std_logic;--starting processing/loading
    clk    : in std_logic;--beating clocks
    rst    : in std_logic;--reset signal of the internal memories
    nb_mem : in std_logic_vector(0 to B-1);--fro. outside->inside
    ex_inx4 : in conf4x;--fro. out -> c_loc_mem
    ex_iny4 : in conf4y;--fro. out -> c_loc_mem
    tb_x4  : in conf4x;--fro. out -> s_loc_mem
    tb_y4  : in conf4y;--fro. out -> s_loc_mem
    out_x4 : out ser_memo;-- fro. x_layers chips -> out
    out_y4 : out sh_memo);-- fro. y_layers chips -> out
 end;

architecture a_integer4s of int_4bamplp24_32 is

 component GBAMPD24_32
  generic( Tx : integer := 9; Ty : integer := 9; Sx : integer := 5; Sy : integer := 5);
  --Ny neurons in the 'Y' layer <-> Nx neurons in the 'X' layer.

    port (
          load      : in  std_logic;--resp switch bet layers
          clk       : in  std_logic;
          nb_mem    : in  std_logic_vector(0 to B-1);
          ex_iny    : in  std_logic_vector(0 to Ny-1);
          ex_inx    : in  std_logic_vector(0 to Nx-1);
          tb_x      : in  std_logic_vector(0 to Nx-1);
          tb_y      : in  std_logic_vector(0 to Ny-1);
          -- Nx<Ny =>inpx_inh through the layer 'X';
          out_x     : out std_logic_vector(0 to Nx-1); --output values  of the 'X' layer
          send_x    : out std_logic_vector(0 to Nx-1); --"send" signals of the 'X' layer
          out_y     : out std_logic_vector(0 to Ny-1); --output values  of the 'Y' layer
          send_y    : out std_logic_vector(0 to Ny-1); --"send" signals of the 'Y' layer
          key_x     : out std_logic_vector(0 to 1); --state of the layer "X"
          key_y     : out std_logic_vector(0 to 1)); -- state of the layer "Y"
end component;

component  synap_memo4
    port (
          rst       : in std_logic;
          LOAD      : in std_logic;
          clk       : in std_logic;
          stay      : in std_logic;
          sh_input  : in sh_memo;
```

```
        ser_input  : in ser_memo;
        sh_output  : out std_logic_vector(0 to sh_cap-1);
        ser_output : out ser_memo);
end component;

component PAR_SHIFT
  generic (N          : in  integer);
  port (
      SD, LOAD, RSTn, CLK : in std_logic;--rstn=0->load, rstn=1->shift
      LOADVec             : in std_logic_vector(0 to N-1);
      IN_PUT              : in std_logic_vector(0 to N-1);
      OUT_PUT             : out std_logic_vector(0 to N-1));
end component;

signal confx_mem : conf4x;--stroring the network config
signal confy_mem : conf4y;--stroring the network config
--local memory of the whole configuration
signal key_x1   : std_logic_vector(0 to 1);
signal key_y1   : std_logic_vector(0 to 1);
--KEY OF THE 1ST BAM_UNIT
signal sendx1   : std_logic_vector(0 to Nx-1);
signal sendy1   : std_logic_vector(0 to Ny-1);
--SEND OF THE 1ST BAM_UNIT
signal LOADVec_asso : tb_xy4;--ass_memo STORE.
--local memory of the all the "learning" ass_memories which could
--be "well suited to" change dynamicaly during learning processing
signal sh_input  : sh_memo;--for 'X'
signal sh_output : std_logic_vector(0 to sh_cap-1);--for 'X'
signal LOAD_syn  : std_logic;
signal LOAD_asso : std_logic;
signal load_units : std_logic;
signal ser_input : ser_memo;--for 'Y'
signal ser_output : ser_memo;--for 'Y'
signal postblok  : a_intercon;
signal stay_syn  : std_logic;--'1' for freezing the syn_memo
signal freeze_asso: std_logic;--'1' for freezing the
signal RSTn_asso : std_logic;
signal rst_syn   : std_logic;
signal int_clk   : std_logic;--internal clock
signal nb_membam : std_logic_vector(0 to B-1);
begin
--int_clk of the controle unit should be in advance of the BAM's clk

process(clk)
 begin
    int_clk <= not clk;-- the internal clock
 end process;

 DIST_MEMO: synap_memo4
       port map (rst_syn, LOAD_syn, int_clk, stay_syn, sh_input, ser_input, sh_output, ser_output);

ser_input(0) <= postblok(3)(0 to Nx-1);
ser_input(1) <= postblok(3)(Nx to Nx + Nx -1);
ser_input(2) <= postblok(3)(Nx+Nx to Nx+Nx+Nx - 1);
ser_input(3) <= postblok(3)(Nx+Nx+Nx to Nx+Nx+Nx+Nx - 1);
```

```
sh_input(0)  <= postblok(3)(Nx+Nx+Nx+Nx to Nx+Nx+Nx+Nx+Ny - 1);
sh_input(1)  <= postblok(3)(Nx+Nx+Nx+Nx+Ny to Nx+Nx+Nx+Nx+Ny+Ny - 1);
sh_input(2)  <= postblok(3)(Nx+Nx+Nx+Nx+Ny+Ny to Nx+Nx+Nx+Nx+Ny+Ny+Ny - 1);
sh_input(3)  <= postblok(3)(Nx+Nx+Nx+Nx+Ny+Ny+Ny to Nx+Nx+Nx+Nx+Ny+Ny+Ny+Ny - 1);
--relate between the assoc_shifter and the synap_memo entries

UNITIS : for i in 0 to 3 generate-
  FIRSTU: if (i = 0) generate
    bam_unit1 : GBAMPD24_32
        port map (load_units, clk, nb_membam, confy_mem(i), confx_mem(i), ser_output(i),
        sh_output, out_x4(i), sendx1, out_y4(i), sendy1, key_x1, key_y1);
    end generate;
  OTHERUs: if (i > 0) generate
    bam_others: GBAMPD24_32
        port map (load_units, clk, nb_membam, confy_mem(i), confx_mem(i), ser_output(i),
        sh_output, out_x4(i), open, out_y4(i), open, open, open);
  end generate;
end generate;

shifting: for i in 0 to 3 generate----------------------for the LOADVec_asso--last: int_clk
  sh1: if (i = 0) generate
    memo0 : PAR_SHIFT
      generic map (N => Nx4 + Ny4)
      port map (freeze_asso, LOAD_asso, RSTn_asso, clk, LOADVec_asso(i), postblok(3),
                postblok(i));
  end generate;
  sh2: if (i > 0) and (i < 3) generate
    memo1_2 : PAR_SHIFT
      generic map (N => Nx4 + Ny4)
      port map (freeze_asso, LOAD_asso, RSTn_asso, clk, LOADVec_asso(i), postblok(i-1),
                postblok(i));
  end generate;
  sh3: if (i = 3) generate
    memo3 : PAR_SHIFT
      generic map (N => Nx4 + Ny4)
      port map (freeze_asso, LOAD_asso, RSTn_asso, clk, LOADVec_asso(i), postblok(i-1),
                postblok(i));
  end generate;
  --the first Nx4-bits are of tb_x4, others for tb_y4
end generate;

 PRO : process(int_clk, rst)
variable memo         : integer;
variable nbr_memo     : integer;
variable step         : integer;
variable nb_assos     : integer;
variable tmp_nbmembam : integer;
variable tmp_frzmemo  : std_logic;
variable tmp_staysyn  : std_logic;
variable tmp_loadunits : std_logic;
variable tmp_rstsyn   : std_logic;
variable tmp_RSTnasso : std_logic;
variable tmp_LOADasso : std_logic;
variable tmp_LOADsyn  : std_logic;
variable go_on        : std_logic;
```

```
begin

  if   (rst ='0') then--clear internal memo
   for i in 0 to 3 loop
    confx_mem(i)   <= (others => '0');
    confy_mem(i)   <= (others => '0');
    LOADVec_asso(i) <= (others => '0');
   end loop;

    rst_syn   <= '0'; --clear the internal of the synap_memo
    RSTn_asso <= '0';--clear the internal of the LOADVec_assoo
    stay_syn  <= '0';--(1) --not freeze the temp memo (syn_memo)/crtl be '1' for next 'load'
    freeze_asso  <= '0'; --not freeze the LOADVec_assoo (1)
    LOAD_asso <= '0';--after this step must go loading
    LOAD_syn  <= '0';--after this step must go loading
    memo := 0;
    nbr_memo := 0;
    step := 0;
    nb_assos := 0;
    go_on := '1';

  elsif (int_clk = '1' and int_clk'event) then--int_clk avoiding mis_synchronization
   if (start = '0') then ----------START IN LOADING PHASE

    if (memo = 0) then--load the initial configuration
      confx_mem <= ex_inx4;
      confy_mem <= ex_iny4;  --1 time and 4 ever<the ex- must be pres now!>
    end if;

    if ((memo < AS4) and (RSTn_asso = '0')) then --the chips number
      LOADVec_asso(memo)(0 to Nx4 - 1) <= tb_x4(3) & tb_x4(2) & tb_x4(1) & tb_x4(0);
      --load the ass_memories
      LOADVec_asso(memo)(Nx4 to (Nx4 + Ny4 - 1)) <= tb_y4(0) & tb_y4(1) & tb_y4(2) &    tb_y4(3);
      --the static storing in a periodical way (with the 'clk'
      memo := memo + 1;--fule the asso_memory 'input'
    else --finish loading memo to tmp_memo => start || loadin asso+syn
      rst_syn   <= '1';--let loading the int of the synap_memo (1st asso)
      RSTn_asso <= '1';--S/let loading the int of the LOADVec_assoo in ||
    end if;

    nbr_memo := conv_integer(unsigned(nb_mem));
    tmp_nbmembam := nbr_memo * AS4;
    nb_membam <= conv_std_logic_vector(tmp_nbmembam, B); --get the nbr of assoc.'s
    load_units <= '1';--synth/load of the bam's: nb_mem

  elsif (start = '1') then

    load_units <= '0'; --synth/starting the processing in the BAM
    LOAD_asso <= '1';--!! || shifting or freezing for ever

  if ((sendy1(0)='1' and key_x1="01") or (sendx1(0)='1' and key_y1="01")
      or (key_x1="00" and key_y1="00" and sendy1(0) = '0' and sendx1(0)='0')
      or go_on = '0') then --1st for swaping, 2nd for starting & last for continuation
    if  (step < nbr_memo-1) then
      stay_syn <= '0';
      freeze_asso  <= '1';--freeze the LOADVec_assoo
```

```
        LOAD_syn <= '1';--go shifting the synap
            --shifting for the "sh32" only the load of "ser" = stay_syn
        step := step + 1;
        go_on := '0';--added
       else
        freeze_asso <= '0';--defreeze the asso & shift in ||
        LOAD_syn  <= '0';--!! || default function loading else shifting .
        step := 0;
        nb_assos := nb_assos + 1;
        if (nb_assos > AS4-1) then
         go_on := '1';--added
         nb_assos := 0;
        end if;
       end if;----------------------------Responsible of 4-period_sending

    else
       freeze_asso <= '1';--freeze the assoc
       stay_syn <= '1';--freeze the synap
       step := 0;
    end if; ------------ controlling the sending
   end if;
  end if;

 end process;
end;
```

Appendix E: VHDL Description of the ECT Injection in BAM (24-32)

```
--VHDL Source Code of the Encoding-Decoding technique applied in B.A.M network
--B.A.M Network is composed by two layers of neurons: 24 neurons by 32 neurons.
--Parallel Implementation Version.
--Designed and tested using the Synopsis Tools based on Unix platform by:
-- Mr. Saffih Faycal
--Biophysics Laboratory, Physics Department,
--University of Malaya, 50603 Kuala Lumpur, Malaysia.
--E-mail. Hosain@hotmail.com, W. W. Web homepage: http://members.tripod.com/~Saffih/faysal.html
--December 1998
-----------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
package typos24_32 is

   constant Nx : integer := 24;--:NUMBER OF NEURONS IN THE 'X' LAYER
   constant Ny : integer := 32;--:NUMBER OF NEURONS IN THE 'Y' LAYER
   type assbv_xy is array (0 to Nx-1) of std_logic_vector(0 to Ny);
   type assbv_yx is array (0 to Ny-1) of std_logic_vector(0 to Nx);
end;                    -- ^                  -- ^
                        -- |                  -- |
                        -- |                  -- Transfered tb_assv'data
                        -- This is nb of N's in the target layer
-----------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity SHIFT51 is --Similarly SHIFT56, SHIFTN2 and SHIFTN3 are (and must be) defined
                  --through the parameter N to be set equal to 56, 2, 3 successively.
 generic (N      : in  integer :=51);
 port (
    RSTn, SI, CLK   : in std_logic;
    PRESET_CLRV : in std_logic_vector(0 to N-1);
    OUT_PUT         : out std_logic_vector(0 to N-1);
    SO              : out std_logic);
end;

architecture A_SHIFT51 of SHIFT51 is
 begin

 process (RSTn, CLK)
 variable temp : std_logic_vector (0 to N-1);
 begin
   if  (RSTn ='0') then
     temp := PRESET_CLRV;
     OUT_PUT <= temp;
   elsif (CLK = '1' and CLK'event) then
     temp := SI & temp(0 to N-2); -- signal & variable : posssible
     OUT_PUT <= temp;
     SO <= temp(N-1);
   end if;
  end process; -- if we use 1 variable into 2 process => declare it twice > its ported value will be unknown
end ;
```

```
configuration C_SHIFTN of SHIFT51 is
 for A_SHIFT51
 end for;
end ;
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity coder is
port(
     clk    : in std_logic;--if rst='1': shifting + counting
     rst_sh : in std_logic;--if rst='0': store data + clear shiftes + clear counter
     rst_cnt : in std_logic;
     si_w   : in std_logic;
     input_w : in std_logic_vector(0 to 50);
     out_cod : out std_logic_vector(0 to 55);
     steps  : out std_logic_vector(0 to 5));--the counter output
end;

architecture A_coder of coder is

 component SHIFT51
  generic (N      : in  integer := 51);
  port (
       RSTn, CLK, SI  : in  std_logic;
       PRESET_CLRV : in std_logic_vector(0 to N-1);
       OUT_PUT        : out std_logic_vector(0 to N-1);
       SO             : out std_logic);
 end component;

 component SHIFT56
  generic (N      : in  integer := 56);
  port (
       RSTn, CLK, SI : in  std_logic;
       PRESET_CLRV  : in std_logic_vector(0 to N-1);
       OUT_PUT      : out std_logic_vector(0 to N-1);
       SO           : out std_logic);
 end component;

 component SHIFTN2
  generic (N      : in  integer := 2);
  port (
       RSTn, CLK, SI : in  std_logic;
       PRESET_CLRV  : in std_logic_vector(0 to N-1);
       OUT_PUT      : out std_logic_vector(0 to N-1);
       SO           : out std_logic);
 end component;

 component SHIFTN3
  generic (N      : in  integer := 3);
  port (
       RSTn, CLK, SI : in  std_logic;
       PRESET_CLRV  : in std_logic_vector(0 to N-1);
       OUT_PUT      : out std_logic_vector(0 to N-1);
       SO           : out std_logic);
 end component;
```

```
component Scounter6
  generic( N : in integer := 6);
  port(
       RST   : in  std_logic;
       CP    : in  std_logic;
      count : BUFFER std_logic_vector(0 to N-1)); --!-!
end component;

signal count          : std_logic_vector(0 to 6-1);
signal so_w           : std_logic;
signal in_word        : std_logic_vector(0 to 50);
signal si2,so2        : std_logic;
signal inp2, out_shift2 : std_logic_vector(0 to 1);
signal si3,so3        : std_logic;
signal inp3, out_shift3 : std_logic_vector(0 to 2);
signal in_cod         : std_logic_vector(0 to 55);
signal si_cod         : std_logic;
signal tsteps         : std_logic_vector(0 to 5);

for in_data : SHIFT51 use entity work.SHIFT51(A_SHIFT51);
for shift2  : SHIFTN2 use entity work.SHIFTN2(A_SHIFTN2);
for shift3  : SHIFTN3 use entity work.SHIFTN3(A_SHIFTN3);
for sh_code : SHIFT56 use entity work.SHIFT56(A_SHIFT56);
for cnt_gate : Scounter6 use entity work.Scounter6(A_Scounter6);
begin

 in_data : SHIFT51 -- input data shift register
      generic map (N => 51)
      port map(RSTn => rst_sh, CLK => clk, SI => si_w,
      PRESET_CLRV => input_w,OUT_PUT => in_word, SO => so_w);
 shift2  : SHIFTN2  --hidden shift register 2 bits
      generic map(N => 2)
      port map(RSTn => rst_sh, CLK => clk, SI => si2,
      PRESET_CLRV => inp2, OUT_PUT => out_shift2, SO => so2);
 shift3  : SHIFTN3  --hidden shift register 3 bits
      generic map (N => 3)
      port map(RSTn => rst_sh, CLK => clk, SI => si3,
      PRESET_CLRV => inp3, OUT_PUT => out_shift3, SO => so3);
 sh_code  : SHIFT56 -- output coded data shift register
      generic map(N => 56)
      port map(RSTn => rst_sh, CLK => clk, SI => si_cod,
      PRESET_CLRV => in_cod, OUT_PUT => out_cod, SO => open);
 cnt_gate :  Scounter6
      generic map(N => 6)
      port map(RST => rst_cnt, CP => clk, count => tsteps);

     steps <= tsteps;
     inp2  <= (others => '0');
     inp3  <= (others => '0');
     in_cod <= (others => '0');
    si3 <= so2 xor si2;

process(clk,so_w)
variable tmp : std_logic;
begin
 tmp := so_w;
```

```
   if (tsteps >= "110011") then -- =? 51 bits <-> shifting the data
      si2   <= '0';
      si_cod <= so3;
   else
      si2   <= tmp xor so3; --si2 , so_w , so3
      si_cod <= tmp;        --si_cod , so_w
   end if;
  end process;

  end ;

  configuration C_coder of coder is
   for a_coder
   end for;
  end;
-------------------------------------------------------------------------------
  library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use work.typos24_32.all;

  entity GBAMPD24_32 is

   generic( Tx : integer := 9; Ty : integer := 9; Sx : integer := 5; Sy : integer := 5);
   port (
      load      : in  std_logic;
      clk       : in  std_logic;
      ex_iny    : in  std_logic_vector(0 to Ny-1);
      ex_inx    : in  std_logic_vector(0 to Nx-1);
      tb_x      : in  std_logic_vector(0 to Nx-1);--tb's asso to "X"
      tb_y      : in  std_logic_vector(0 to Ny-1);--tb's asso to "Y"
      out_x     : out std_logic_vector(0 to Nx-1);
      out_y     : out std_logic_vector(0 to Ny-1);
      pstored   : out std_logic;
      nstored   : out std_logic);
  end;

  architecture AGBAMPD24_32 of GBAMPD24_32 is

   component  NRNXP24_32
    generic( N,S,T : in positive );
   port (
      ex_in   : in std_logic; --external value
      load    : in std_logic; --reset
      clk     : in std_logic; --clock beat
      w_up    : in std_logic; --wake_up signal
      tb_assv : in std_logic_vector(0 to N); --target_bit ass_vector
      sgl_in  : in std_logic_vector(0 to N-1); --in-comming signal
      output  : out std_logic ; --neuron output
      send    : out std_logic); --getting the output
  end component;

  component  NRNYP32_24
   generic( N,S,T : in positive );
  port (
      ex_in     : in std_logic; --external value
```

```
    load      : in std_logic; --reset
    clk       : in std_logic; --clock beat
    w_up      : in std_logic; --wake_up signal
    tb_assv : in std_logic_vector(0 to N); --target_bit ass_vector
    sgl_in    : in std_logic_vector(0 to N-1); --in-comming signal
    output    : out std_logic ; --neuron output
    send      : out std_logic); --getting the output
end component;

component coder
 port(
   clk        : in std_logic;
   rst_sh     : in std_logic;
   rst_cnt    : in std_logic;
   si_w       : in std_logic := '0';
   input_w  : in std_logic_vector(0 to 50);
   out_cod  : out std_logic_vector(0 to 55);
   steps      : out std_logic_vector(0 to 5));
end component;

signal loadl        : std_logic; --load signal shared between the two(2) layers
signal sx_wy        : std_logic_vector(0 to Nx-1); --sendS of 'X' to w_upS of 'Y'
signal x_y          : std_logic_vector(0 to Nx-1); --outputS of 'X' to sgl_inS of 'Y'
signal w_upy        : std_logic;
signal sy_wx        : std_logic_vector(0 to Ny-1); --sendS of 'Y' to w_upS of 'X'
signal y_x          : std_logic_vector(0 to Ny-1); --outputS of 'Y' to sgl_inS of 'X'
signal w_upx        : std_logic;
signal testws       : std_logic;
signal teststep     : std_logic;
signal tb_assvx     : assbv_xy;
signal tb_assvy     : assbv_yx;
signal rst_cod      : std_logic;
signal sto_dat      : std_logic_vector(0 to 50);
signal bisto_dat    : std_logic_vector(0 to 50);
signal cod_dat      : std_logic_vector(0 to 55);
signal bits         : std_logic_vector(0 to 5);
signal start_co     : std_logic;
signal stage        : std_logic;
signal nsto_dat     : std_logic_vector(0 to 50);
signal ncod_dat     : std_logic_vector(0 to 55);
signal inpv         : std_logic;
signal changer      : std_logic;
begin

process(tb_x,tb_y)
begin
 for i in 0 to (Nx - 1) loop
   tb_assvx(i)(0) <= tb_x(i);
   tb_assvx(i)(1 to Ny) <= tb_y;
 end loop;

 for i in 0 to (Ny - 1) loop
   tb_assvy(i)(0) <= tb_y(i);
   tb_assvy(i)(1 to Nx) <= tb_x;
 end loop;
end process;
```

```
 V0 :for i in 0 to Nx-1 generate
  V1 : NRNXP24_32
     generic map (N => Ny,S => Sx,T => Tx)
     port map (ex_inx(i),loadl,clk,w_upx,tb_assvx(i),y_x,x_y(i),sx_wy(i));
  end generate;

 V2 :for i in 0 to Ny-1 generate
  V3 : NRNYP32_24
     generic map (N => Nx,S => Sy,T => Ty)
     port map (ex_iny(i),loadl,clk,w_upy,tb_assvy(i),x_y,y_x(i),sy_wx(i));
  end generate;

 cod_cmp: coder port map (clk => clk, rst_sh => rst_cod, rst_cnt => rst_cod, si_w => open,
                  input_w => bisto_dat, out_cod => cod_dat ,steps => bits);

 out_x  <= x_y  ; out_y  <= y_x  ;

 process(ex_inx,ex_iny)
 variable t_inp : std_logic;
 begin
  t_inp := ex_inx(0);
  for i in 1 to Nx-1 loop
   t_inp := t_inp or ex_inx(i);
  end loop;

  for i in 0 to Ny-1 loop
   t_inp := t_inp or ex_iny(i);
  end loop;

  if(t_inp = '1') then
    inpv <= '1';
  end if;
 end process;

 process(clk)
 variable tmpx    : std_logic;
 variable tmpy    : std_logic;
 variable step    : std_logic;
 variable sw      : std_logic;
 variable temp    : std_logic;
 variable ss      : std_logic_vector(1 downto 0);
 variable memo_st : std_logic_vector(0 to Nx-1) := (others => '0');
 variable simila  : unsigned(0 to Nx-1);
 variable tmp     : std_logic;
 begin

 ss := step & sw;

 if (clk = '1') then
   if (load = '1') then
    step := '0';
    sw  := '0';
    tmpx := '1';
    tmpy := '1'; --ENTER THE ASSOCIATED VECTORS(LEARNING)
    loadl <= '1';
```

```
 if(inpv = '1') then --ENTER THE INITIAL CONFIGURATION
   tmpx := '0';
   tmpy := '0';
 end if;
 --the keys should be "11" : storing the associted (synaptic strength)
 step := '1';    --step='1',sw='0'

else --in nb_mem or processing calcul
 loadl <= '0';
 case ss is
 when "10" =>

   tmpx := '0';
   tmpy := '0';
   step := not step;   --step='0',sw='0'

 when "00" =>

      tmpx := not tmpy;
      tmpy := not tmpx;
      sw := '1';  --step='0',sw='1'

 when "01" =>

      if(sx_wy(0) = '1' or sy_wx(0) ='1')
        then   step := '1'; sw := '1';
      end if;

 when "11" =>

   if (sx_wy(0) = '1') then
     -- SWTCHING BETWEEN THE WORKING PERIODS OF THE TWO LAYERS
     --STARTING BY THE "X' LAYER;
     simila := unsigned(memo_st) - unsigned(x_y);
     tmp := '0';

     for i in 0 to Nx-1 loop
       tmp := tmp or simila(i);
     end loop;

       if(tmp = '0') then --TESTING THE STABILITY OF THE SIGNAL "X"
         start_co <= '1';-- starting to coding of the output.
         sto_dat(0 to Ny-6) <= y_x(5 to Ny-1);
         sto_dat(Ny-5 to 50) <= x_y(0 to Nx-1);
         nsto_dat(0 to Nx-1) <= not x_y;
         nsto_dat(Nx to 50) <= not y_x(5 to Ny-1);
       end if;

       temp := tmpx;
       tmpx := tmpy;
       tmpy := temp;
       memo_st := x_y;
       step := '0'; sw := '0';

  elsif (sy_wx(0) = '1') then
       temp := tmpx;
```

```
        tmpx := tmpy;
        tmpy := temp;
        step := '0'; sw := '0';
    end if;
  when others =>
  end case;
 end if;
end if;

testws <= sw;
teststep <= step;
w_upx <= tmpx;
w_upy <= tmpy;
end process;

MUX:process(clk, changer)
begin
 if  (changer = '0') then
  bisto_dat <= sto_dat;
 elsif (changer = '1') then
  bisto_dat <= nsto_dat; -- the coder input
 end if;
end process;

process(clk) --THE CODER PROCESS: STARTING BY THE STATIONARY STATE OF THE BAM
variable sto : std_logic;
begin
 if(clk = '1') then
  if(start_co = '1') then
   case stage is
   when '0' =>

    rst_cod <= '0';--FOR LOADING
    if(sto /= '1') then
     stage <= not stage;
    end if;

   when '1' =>

    rst_cod <= '1';--STARTING CODING
    if(bits = "111000") then --COMPARING THE 2 CODES
     changer <= not changer;
     if  (unsigned(cod_dat(0 to 4)) - unsigned(y_x(0 to 4)) = 0) then
      pstored <= '1';
      sto := '1';
      stage <= not stage;
     else
      pstored <= '0';
      if  (changer = '1' and (unsigned(cod_dat(0 to 4)) - unsigned(y_x(0 to 4)) = 0)) then
       nstored <= '1';
      elsif (changer = '1') then
       nstored <= '0';
      end if;
     end if;
   end if;
```

```
    when others =>
    end case;
    else
      rst_cod <= '0'; --update the coder (sh_registers and counters)
      stage <= '0';
      changer <= '0';
    end if;
  end if;
end process;
end AGBAMPD24_32;

configuration CGBAMPD24_32 of GBAMPD24_32 is
 for AGBAMPD24_32
  for V0
   for V1 : NRNXP24_32
   use entity work.NRNXP24_32(ANRNXP24_32);
   end for;
  end for;
  for V2
   for V3 : NRNYP32_24
   use entity work.NRNYP32_24(ANRNYP32_24);
   end for;
  end for;
  for cod_cmp : coder
   use entity work.coder(A_coder);
  end for;
 end for;
end CGBAMPD24_32;
```

Appendix F: VHDL Description of the Hopfield net Bus-based Architecture.

```
--VHDL Source Code of the Bus-Based architecture with the VHDL description of the Neuron Unit and
-- the Arbiter Unit.
-- Parallel Implementation Version.
--Designed and tested using the Synopsis Tools based on Unix platform by:
-- Mr. Saffih Faycal
--Biophysics Laboratory, Physics Department,
--University of Malaya, 50603 Kuala Lumpur, Malaysia.
--E-mail. Hosain@hotmail.com, W. W. Web homepage: http://members.tripod.com/~Saffih/faysal.html
--December 1998
-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
package pac_wanrn is

  constant N : integer := 8;constant S : integer := 4;
  type st_syn is array (0 to N-1) of std_logic_vector(0 to S-1);--:Synaptic Strength
  type parall_memory is array (0 to N-1) of st_syn; --parall_memory1;--in the TOPWAN

end;
-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.pac_wanrn.all;

entity WANRN8 is
 generic( N : integer := 8 ;S : integer := 4 ;--S(0)=sgn_bit
       A : integer := 3 ;H : integer := 12);
      -- N: nb of incoming signals: for the initial state,
      -- S: nb of BITS to store the digits of the weight
      -- H>S: nb of BITS to store the accumuation of the weights
      -- A : nb of bits for in/out address
      -- H : nb of bits for the local field (broadcasting)
port (
    clk     : in std_logic;--clock beat: "CHIP" : OUTPAD
    start   : in std_logic;--storing ID or receiving nbus_grant
    init_out : in std_logic;--init configuration
    sgl_in   : in std_logic_vector(0 to N-1);--"CHIP" : OUTPAD-- the input signal for the initial loc field
    key      : in std_logic_vector(0 to 1); --load & wup : CTRL UNIT
    nbus_grant: in std_logic;--coming form the control unit
    upd_nrn  : inout std_logic_vector(0 to A); --: to the BUS
    --updated neuron , 1st bit:value(wanto '1' or '0')& others=address
    lc_adfd  : out std_logic_vector(0 to H);--: CTRL UNIT
    --sent field, 1st bit to indicate clip_appl:'1' if 0->1; '0' if 1->0 and 'Z' otherwise
    synaps   : in st_syn;--"CHIP" : OUTPAD
    output   : out std_logic ); --neuron output "CHIP" : OUTPAD
end ;

architecture AWANRN8 of WANRN8 is
 signal weight  : st_syn;
begin

LOAD_SS:process(synaps)
 begin
```

```
  if (key(0) = '1') then --When start(load='1')
      weight <= synaps;--(connect)(0 to S-1)
  else
    for i in 0 to N-1 loop
     weight(i) <= (others => '0');
    end loop;
  end if;
end process;----------SYN_STR

calcul:process(clk)--PRCESSING THE BEHAVIOUR OF THE NEURON
variable pre_output   : std_logic; -- previous output
variable test         : std_logic;
variable accumulator : unsigned(0 to H-1);
variable tmp_accum   : unsigned(0 to H-1);
variable multip       : std_logic_vector(0 to S-1);
variable loc_field    : unsigned(0 to H-1);--local field for initial
variable wgt_addr     : unsigned(0 to H-1);
variable int_addr     : integer;
variable ID           : std_logic_vector(0 to A-1);
variable take_addr    : std_logic;
-- and perpetual state (initially take the value of the accumulator);
begin
 if(clk = '1' and clk'event) then
  case key is
  when "10" => --initialization of the configuration

    accumulator := (others => '0');
    tmp_accum := (others => '0');

    if (start = '0') then--init sys_configuration
      pre_output := init_out;
    end if;

    if (nbus_grant = '1') then
      if   (start = '0') then
        ID := upd_nrn(0 to A-1);
      elsif (start = '1') then
        pre_output := not pre_output;
      end if;
    else upd_nrn <= (others => 'Z');
    end if;

  when "00" => --initialization of the loc_fld

    accumulator := (others => '0');
    tmp_accum := (others => '0');

  for i in 0 to N-1 loop --MULTn, SUMn and ACCUMULATION FOR SYN_STRENGTH
    if   (sgl_in(i) = '1') then
      multip := weight(i);
    elsif (sgl_in(i) = '0') then
      multip := (others => '0');
    end if;
    tmp_accum(0) := multip(0); --MSB
    if   (multip(0) = '1') then
      for i in 1 to H-S loop
        tmp_accum(i) := '1';
      end loop;
    elsif (multip(0) = '0') then
```

```
      for i in 1 to H-S loop
       tmp_accum(i) := '0';
      end loop;
    end if;

    tmp_accum(H-S+1 to H-1) := unsigned(multip(1 to S-1));
    test := '0';

    for i in 1 to H-1 loop
      test := test or tmp_accum(i);
    end loop;

    if (test = '0') then next;
    end if;

    accumulator := tmp_accum + accumulator ;

  end loop;

    loc_field  := accumulator;
    take_addr := '0';

  when "01" =>--updating 1 neuron in the second clock cycle take the addresse given in the first clock cycle

  if   (nbus_grant = '0') then --read the updated neuron address
    take_addr := not take_addr;

  if (take_addr = '0') then
    int_addr := conv_integer(unsigned(upd_nrn(1 to A)));
    if   (weight(int_addr)(0) = '1') then
      for i in 0 to H-S loop
        wgt_addr(i) := '1';
      end loop;
    elsif (weight(int_addr)(0) = '0') then
      for i in 0 to H-S loop
        wgt_addr(i) := '0';
      end loop;
    end if;
    wgt_addr(H-S+1 to H-1) := unsigned(weight(int_addr)(1 to S-1));
    if   (upd_nrn(0) = '0') then --<=> change from 1 to 0 => (-)
      loc_field := loc_field - wgt_addr;
    elsif (upd_nrn(0) = '1') then --<=> change from 0 to 1 => (+)
      loc_field := loc_field + wgt_addr;
    end if;
    --we use here a bipolar units IN ORDER to simulate ISING MODEL of spins so (0)->(-1); (1)->(+1)
  end if;--take_addr

  elsif (nbus_grant = '1') then --(old)bus_standsend your local ID
    upd_nrn <= loc_field(0) & ID; --send the bus your ID' nbr and requirement
  end if;

  when others =>

end case ;

    if   ((loc_field(0) = '0') and (pre_output = '0')) then
      lc_adfd(0) <= '1' ; lc_adfd(1 to H) <= conv_std_logic_vector(loc_field, H);
      --send the loc_field with the 1st bit '1'--WANT TO BE '1'
    elsif ((loc_field(0) = '0') and (pre_output = '1')) then
```

```
    lc_adfd(0 to H) <= ("ZZZZZZZZZZZZZ");

   elsif ((loc_field(0) = '1') and (pre_output = '0')) then
    lc_adfd(0 to H) <= ("ZZZZZZZZZZZZZ");

   elsif ((loc_field(0) = '1') and (pre_output = '1')) then
    lc_adfd(0) <= '0' ; lc_adfd(1 to H) <= conv_std_logic_vector(loc_field, H);
    --send the loc_field with the 1st bit '0'.--WANT TO BE '0'
   end if;
   --this section is needed when key="10" and "01"

   output <= pre_output;

 end if;
end process;

end AWANRN8;

configuration CWANRN8 of WANRN8 is
 for AWANRN8
 end for;
end;
-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

package pac_ctl is

  constant N : integer := 8; constant A : integer := 3;
  constant H : integer := 12;
  type field_in is array (0 to N-1) of std_logic_vector (0 to H);
end;
-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.pac_ctl.all;--pac_ctl.

entity CTRL_WANRN8 is
 port (
      clk : in  std_logic;--Top control: "CHIP": OUTPAD
      rst : in  std_logic;--to shifter dist b_grant: "CHIP": OUTPAD
      start : in  std_logic; --Top control: "CHIP" : OUTPAD
      config : in  std_logic_vector(0 to N-1);--configuring : OUTPAD
      state : in  std_logic_vector(0 to N-1);--configuration of the NRN_sys
      recei_fld : in  field_in;  --1st bit:value,2nd bit:local?:TO WANRN8"bus_request"
      bus_grant : out std_logic_vector(0 to N-1);--addr_out. send the address : TO WANRN8
      ini_conf : out std_logic_vector(0 to N-1);--initial configuration vector
      load_wup : out std_logic_vector(0 to 1); --control load & w_up:TO WANRN
      crl_count : out std_logic);--control the counter
      --(1)_when we want to initialise the net : "load & ctrl" (to "10")
      --(2)_for the first time we must have :"load & ctrl"="00"(init loc_fld)
      --(3)_when we want to update one neuron  : "load & ctrl" (to "01")
      --and send its address after (load & ctrl = "00")to the other neurons
      --for a new evaluation
end ;
```

```
architecture ACTRL_WANRN8 of CTRL_WANRN8 is

 component SHIFTN
 generic (N        : in  integer);
 port (RSTn, SI, CLK: in std_logic;
     PRESET_CLRV  : in std_logic_vector(0 to N-1);
     OUT_PUT      : out std_logic_vector(0 to N-1);
     SO           : out std_logic);
 end component;

 signal si       : std_logic;
 signal init_grant  : std_logic_vector(0 to N-1);
 signal tmp_bgrant  : std_logic_vector(0 to N-1);
 for shifting : SHIFTN use entity work.SHIFTN(A_SHIFTN);
 begin

 si  <= '0';
 shifting : SHIFTN generic map (N => N)
           port map (rst, si, clk, init_grant, tmp_bgrant, open);

CAL: process(clk, start, rst)
 variable tmp_ldcl : std_logic_vector(0 to 1);
 variable going   : std_logic;
 --next variables are for choosing the NRN
 variable step      : unsigned(0 to A);
 variable temp_adfld : std_logic_vector(0 to H);  --the hold adresse
 variable posdiff   : unsigned(0 to H-1);       --positive difference
 variable negdiff   : unsigned(0 to H-1);       --negative difference
 variable ld_padd   : std_logic_vector(0 to A-1);--store the pos_addr
 variable ld_nadd   : std_logic_vector(0 to A-1);--store the neg_addr
 variable address   : std_logic_vector(0 to A+1);--pos/neg + val + locl?
 variable maxpos_fld : std_logic_vector(0 to H-1);--positive field
 variable maxneg_fld : std_logic_vector(0 to H-1);--negative field.
 variable dif_pn_fld : unsigned(0 to H-1);--negative field.
 variable p_one_ti  : std_logic;
 variable n_one_ti  : std_logic;
 variable temp_conf  : std_logic_vector(0 to N-1);
 variable fst_timaddr  : std_logic;
 variable tmp_busgrant : std_logic_vector(0 to N-1);
 variable change_state : std_logic;--from "01" to "10"
 variable addr_conf   : std_logic;
begin

if   (start = '0') then ----------INITIALIZATION
    --send the ini_conf to the NRNs--update the NRN's key to "10": initilization
     ini_conf <= config;
     tmp_ldcl := "10";
     going := '0';
     fst_timaddr := '0';
     change_state := '0';

   if  (rst = '0') then--assign each neuron with an ID using a counter;
     crl_count <= '0';
     init_grant(0) <= '1';
     init_grant(1 to N-1) <= (others => '0');
   elsif (clk = '0' and clk'event) then
```

```
    if (conv_integer(unsigned(tmp_bgrant)) /= N) then
     crl_count <= '1';--start counting
     tmp_busgrant := tmp_bgrant;
     --through a mux:let shifter choose bus master
    else
     crl_count <= '0';
        --this should indicate to the counter that it must
        --disconnects and updates its output to the (others => '0');
        --:look to the Scounter_bus.vhd
     tmp_busgrant := (others => '0');--no neuron is bus_master
    end if;
   end if;

elsif (clk = '0' and clk'event) then
 if   (going = '0') then--caluculation of the loc_field
      tmp_ldcl := "00";
      going := not going;
      addr_conf := '0';--step for NRN'choosing;
     -- crl_count <= '1';--stop the count--no role
elsif (going = '1') then--the cliping between:confe

 if (addr_conf = '0') then-- sending addresse phase
     step := (others => '0');
     p_one_ti := '0';
     n_one_ti := '0';
     tmp_busgrant := (others => '0');

   while (step <= N-1) loop --for step in 0 to N-1 loop
     temp_adfld := recei_fld(conv_integer(step))(0 to H);

    if (temp_adfld(0) = '1') then
    --NRN want to flip -> '1':recei positive val
     if   (p_one_ti = '0') then --initial step
      maxpos_fld := temp_adfld(1 to H);
      ld_padd := conv_std_logic_vector(step, A);
      --STR the NRN class (addr)
      p_one_ti := not p_one_ti;
     elsif (p_one_ti = '1') then --next step
      posdiff := unsigned(temp_adfld(1 to H)) - unsigned(maxpos_fld);
      if (posdiff(0) = '0') then  -- MSBit (positive)
       maxpos_fld := temp_adfld(1 to H);
       ld_padd := conv_std_logic_vector(step, A);
       --STR the NRN class (addr)
      end if;
     end if;
    elsif (temp_adfld(0) = '0') then
    --NRN want to flip -> '0': recei negative val
     if   (n_one_ti = '0') then --initial step
      maxneg_fld := temp_adfld(1 to H);
      ld_nadd := conv_std_logic_vector(step, A);
      --STR the NRN class (addr)
      n_one_ti := not n_one_ti;
     elsif (n_one_ti = '1') then
      negdiff := unsigned(temp_adfld(1 to H)) - unsigned(maxneg_fld);
      if (negdiff(0) = '1') then  -- MSBit (negative)
       maxneg_fld := temp_adfld(1 to H);
```

```
          ld_nadd := conv_std_logic_vector(step, A);
          --STR the NRN class (addr)
        end if;
      end if;
    end if;
    step := step + 1;
  end loop;

  --choose the best candidate for the updating between pos and neg fld
   dif_pn_fld := unsigned(maxneg_fld) + unsigned(maxpos_fld);

  if   (p_one_ti = '1' and n_one_ti = '1') then
  --both initialization
    if   (dif_pn_fld(0) = '0') then
      address(2 to A+1) := ld_padd;
    elsif (dif_pn_fld(0) = '1') then
      address(2 to A+1) := ld_nadd;
    end if;
    --addess(1) = '0' : normal porcessing (no initialisation)
  elsif (p_one_ti = '0' and n_one_ti = '1') then
     address(2 to A+1) := ld_nadd;
  elsif  (p_one_ti = '1' and n_one_ti = '0') then
    address(2 to A+1) := ld_padd;
  end if;
     --temp_conf contains the new configuration
     --address contains the choosen NRN
 end if; --elsif (ADDRE_CONF = '1') then

 end if;-- of the if (going = '0') then

 if   (addr_conf = '0') then
 --send the new address::the first executed loop
   addr_conf := not addr_conf;
   tmp_ldcl := "01";--change the state of the neuron
   tmp_busgrant(conv_integer(unsigned(address))) := '1';--protection!!!
 elsif (addr_conf = '1') then
  --send the new configuration::the second executed loop
  --two clock cycles to enter the loop
   change_state := not change_state;

 if (change_state = '0') then
   addr_conf := not addr_conf;
   tmp_ldcl := not tmp_ldcl;
   --change the nrn state:if granted=>change output
 end if;
end if;

   load_wup <= tmp_ldcl;
   bus_grant <= tmp_busgrant;
end if;--of :if (start = '0') then

end process;
end ACTRL_WANRN8;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
use work.pac_nrnx24_32.all;
use work.pac_ctl.all;
use work.all;

entity TOPWAN is
      generic( N : integer := 8; S : integer := 4);
  port
     ( rst   : in std_logic;--for the control unit
       clk   : in std_logic;--to the neuron
       start : in std_logic;--the control unit
      config : in std_logic_vector(0 to N-1);--to the control unit
     memo_syn : in parall_memory; --parall_memory2;
     out_net  : out std_logic_vector(0 to N-1));
 end ;

 architecture A_TOPWAN of TOPWAN is
 component WANRN8
  generic( N : integer := 8 ;S : integer := 4 ;--S(0)=sgn_bit
       A : integer := 3 ;H : integer := 12);
     -- N: nb of incoming signals: for the initial state,
     -- S: nb of BITS to store the digits of the weight
     -- H>S: nb of BITS to store the accumuation of the weights
     -- A : nb of bits for in/out address
     -- H : nb of bits for the local field (broadcasting)
 port (
     clk     : in std_logic;--clock beat: "CHIP" : OUTPAD
     start   : in std_logic;--storing ID or receiving bus_grant
     init_out : in std_logic;--init configuration
     sgl_in  : in std_logic_vector(0 to N-1);--"CHIP" : OUTPAD
     -- the input signal for the initial loc field
     key     : in std_logic_vector(0 to 1); --load & wup : CTRL UNIT
     nbus_grant: in std_logic;--coming form the control unit
     upd_nrn  : inout std_logic_vector(0 to A); --: to the BUS
     --updated neuron , 1st bit:value(wanto '1' or '0', others=address
     lc_adfd  : out std_logic_vector(0 to H);--: CTRL UNIT
     --sent field, 1st bit to indicate clip_appl:'1' if 0->1; '0' if 1->0, 'Z' otherwise
     synaps   : in st_syn;--"CHIP" : OUTPAD
     output   : out std_logic ); --neuron output "CHIP" : OUTPAD
 end component;

 component CTRL_WANRN8
  port (
      clk : in std_logic;
      rst : in std_logic;--
      start : in std_logic; --Top control: "CHIP" : OUTPAD
      config : in std_logic_vector(0 to N-1);--configuring : OUTPAD
      state : in std_logic_vector(0 to N-1);--configuration of the NRN_sys
      recei_fld : in field_in;  --1st bit:value,2nd bit:local?:TO WANRN8"bus_request"
      bus_grant : out std_logic_vector(0 to N-1);--addr_out. send the address : TO WANRN8
      ini_conf : out std_logic_vector(0 to N-1);--initial configuration vector
     load_wup : out std_logic_vector(0 to 1); --control load & w_up:TO WANRN
     crl_count : out std_logic);--control the counter
     --(1)_when we want to initialise the net : "load & ctrl" (to "10")
     --(2)_for the first time we must have :"load & ctrl"="00"(init loc_fld)
     --(3)_when we want to update one neuron  : "load & ctrl" (to "01")
     --and send its address after (load & ctrl = "00")to the other neurons
```

```
    --for a new evaluation
end component ;

component Scounter
  generic( N : in integer);
  port(
     RST   : in  std_logic;--we have an entry pin : jk always in '1'
     CP    : in  std_logic;
     count : buffer std_logic_vector(0 to N-1));
end component;

signal intersgl  : std_logic_vector(0 to N-1);
signal load_wup  : std_logic_vector(0 to 1);
signal recei_fld : field_in;
signal state     : std_logic_vector(0 to N-1);
signal bus_grant : std_logic_vector(0 to N-1);
signal ini_conf : std_logic_vector(0 to N-1);
signal upd_nrn  : resolved std_ulogic_vector(0 to A) bus;
signal crl_count : std_logic;
begin

out_net <= state;
NRNT  : for i in 0 to N-1 generate
  NRN1 : WANRN8
        generic map (N => 8, S => 4, A => 3, H => 12)
        port map (clk, start, ini_conf(i), state, load_wup, bus_grant(i),
        upd_nrn, recei_fld(i), memo_syn(i), state(i)); --state<->sgl_in
end generate;

CONTROL: CTRL_WANRN8 port map (clk, rst, start, config, state, recei_fld, bus_grant, ini_conf,
load_wup, crl_count);

ADDRESSING: Scounter
        generic map (N => 4) -- =>0 to A <> upd_nrn
        port map (crl_count, clk, upd_nrn);
end;

configuration C_TOPWAN of TOPWAN is
for A_TOPWAN
 for NRNT
   for NRN1 : WANRN8
     use entity work.WANRN8(AWANRN8);
   end for;
 end for;
 for CONTROL : CTRL_WANRN8
     use entity work.CTRL_WANRN8(ACTRL_WANRN8);
 end for;
 for ADDRESSING : Scounter
     use entity work.Scounter(A_Scounter);
 end for;
end for;
end;
```