// neuron.v

```
//
// Copyright 2011-2020 General Vision Holding LLC

/**************** Open Source License Notice ************************
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated
documentation files (the
"Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify,
merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the
Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
//*********************************************************************************************************//
// Neuron with individual prototype storage (Model CM1K -> OKI, QuarkSE Curie -> INTEL, NM500 manufactured by Nepes
Korea //
//*********************************************************************************************************//

module neuron(input clk, input reset_l, input ds, input read, input [3:0]register, input [15:0]data_in, input SR, input
KNN, input oktolearn_in, input dci, output reg dco, output reg id, output unclearn, output reg [15:0]data_out, output
reg ready);
//
//Write neurons commands (5'h0x))
parameter WNCR=5'h00;               // Write Context register
parameter WCOMP=5'h01;              // Write Component register
parameter WLCOMP=5'h02;             // Write Last Component register
parameter WINDEXCOMP=5'h03;         // Write Pattern memory index
parameter WCAT=5'h04;               // Write Category register
parameter WAIF=5'h05;               // Write Actual Influence Field update (SR mode only)
parameter WMINIF=5'h06;             // Write Minimum Influence Field setting
parameter WMAXIF=5'h07;             // Write Maximum Influence Field setting
parameter WTESTCOMP=5'h08;          // Write Test comp for global parallel testing of all neurons
parameter WTESTCAT=5'h09;           // Write Test category for global parallel testing of all neurons
parameter WGCR=5'h0B;               // Write Global Context Write
parameter RESETCHAIN=5'h0C;         // Write  to toint to the first physical neuron
parameter WNSR=5'h0D;
```

```
//Read commands (5'h1x)
parameter RNCR=5'h10;              // Read neuron context register (SR mode)
parameter RCOMP=5'h11;             // Read neuron component register (autoincrement in SR mode)
parameter RLCOMP=5'h12;            // Same as above
parameter RDIST=5'h13;             // Read the Distance of the fired neuron under focus
parameter RCAT=5'h14;              // Read the Category of the fired neuron under focus (and switch to the next, if
any)
parameter RAIF=5'h15;              // Read the Actual Influence Field (SR mode)
parameter RMINIF=5'h16;            // Read the Minimum Influence Field
parameter RMAXIF=5'h17;            // Read the Maximum Influence Field
parameter RNID=5'h1A;              // Read the Neuron Identifier
parameter RGCR=5'h1B;              // Read the Global Context Register
parameter RNCOUNT=5'h1F;           // Read the number of committed neurons
//
parameter defGCR=8'h01;            // Default Global Context
parameter defMAXIF=16'h4000;       // Default Maximum Influence Field Value
parameter defMINIF=16'h0002;       // Defaut Minimum Influence Field Value
//
reg oktolearn;
reg unc_l;
reg [7:0] ctxt;         // Bit[7]=Norm (0-L1, 1=LSUP), bit[6-0]=global context register
reg [15:0] aif;         // Actual Influence Field (neuron generalization factor)
reg [15:0] dist;        // Distance (L1 or LSup) between the broadcasted input pattern and the neuron pattern (Vector)
reg [15:0] cat;               // Category register)
reg [23:0] identifier;
reg fire;
reg nselect;            //true if NCR= GCR
reg exclude_loop;       //exclude firing neuron from dist_output list when its DIST is read
reg exclude_cat;        //exclude firing neuron from cat_output list when its CAT is read
reg [15:0]minif;
reg [7:0] ramindex;     //index of component in the neuron memory
wire [7:0] protodata;
reg [3:0] bit_ptr;
wire ramrw_;
wire [4:0]cmd_wire;  //used during CY_START1
reg [4:0] cmd;       //used after CY_START1
reg firing_id;
reg [7:0]comp;
//
// The neuron is a state machine with three states
reg [2:0]sm_state;
reg [2:0]sm_state_neg;
parameter CY_START1=0;          // First half of the START cycle
parameter CY_START2=1;          // Second half of the START cycle
parameter CY_LOOP=2;            // Loop cycle during the "Search & Sort operation"
```

```
parameter CY_LAST1=3;           //Last cycle of teh neuron
parameter CY_IDLE_NEG=4;        // Idel cycle
//
// Neuron individual status
wire [1:0]neuron_state;
parameter NS_IDLE=0;            // IDLE means no learned category and no daisy chain in active
parameter NS_RTL=1;             // RTL (Ready To Learn)  means no learned category but the neuron just before is
committed so next learning for me
parameter NS_COMMITTED=3;       // COMMITTED neuron has learned a category and if a pattern is broadcasted to all
neurons will possiblye react to it
//
assign unclearn= oktolearn & unc_l; // This signal is used for
                                          // a) enabling learn signal if no neuron recognize the entered pattern so RTL
can learn
                                        // b) During recognition signal uncertain if neurons with different catgory
react!
//
assign cmd_wire[4:0]={read,register[3:0]};
//
assign neuron_state[0]=dci;
assign neuron_state[1]=dco;
//
assign ramrw_= ds && ( (neuron_state[1:0]==NS_RTL && ~read && register[3:0]==4'h1) || (neuron_state[1:0]==NS_RTL &&
~read && register[3:0]==4'h2) || (~read && register[3:0]==4'h8)) ? 1'b0 :1'b1;

wire [7:0] abs;
assign abs[7:0]=(comp[7:0]>=protodata[7:0]) ? comp[7:0]-protodata[7:0]:protodata[7:0]-comp[7:0];
//
wire [15:0]min_aif;
wire [15:0]new_aif;
assign min_aif[15:0]= dco ? dist[15:0]:data_in[15:0];
wire degenerated = (min_aif[15:0] > minif[15:0]) ? 1'b0 : 1'b1;
assign new_aif[15:0]= degenerated ? minif[15:0] : min_aif[15:0];
//---------------------------------------
//Neuron RAM 256 bytes per neuron
//---------------------------------------
vector256 proto(.a(ramindex[7:0]),.d(data_in[7:0]),.clk(clk),.we(~ramrw_),.spo(protodata));
//---------------------------------------
// Positive edge of the clock
// Write the registers of the neuron
//---------------------------------------
reg [7:0]protoindex;
//
always @(posedge clk or negedge reset_l)
if(~reset_l)        // Reset operation initialize all value and register to default
```

```
    begin
    ctxt[7:0]<=defGCR[7:0];aif[15:0]<=defMAXIF[15:0];cat[15:0]<=16'h0000;
    minif[15:0]<=defMINIF[15:0];identifier[23:0]<=24'h000000;
    comp[7:0]<=8'h00;cmd[4:0]<=5'h00; protoindex[7:0]<=8'h00;
    exclude_loop<=0;exclude_cat<=0;fire<=0;
    sm_state[2:0]<=CY_START1;sm_state_neg[2:0]<=CY_IDLE_NEG;
    nselect<=1; oktolearn<=1;ready<=0;
    end
//
else begin
//
case(sm_state[2:0])
//
CY_START1:
if (~ds) begin sm_state_neg[2:0]<=CY_IDLE_NEG; ready<=1; end
//
else
//
begin
cmd[4:0]<=cmd_wire[4:0];
sm_state_neg[2:0]<=CY_START1;
ready<=0; oktolearn<=1;
//
if (cmd_wire==WINDEXCOMP) protoindex<=data_in[7:0];
//
if (SR)
    begin
    case (neuron_state[1:0])
    NS_RTL:
        case (cmd_wire)
        WNCR: ctxt[7:0]<=data_in[7:0];
        WMINIF: minif[15:0]<=data_in[15:0];
        WAIF: aif[15:0]<=data_in[15:0];
      WCAT: begin cat[15:0]<=data_in[15:0];if (data_in[15:0]!=0) identifier[23:0]<= identifier[23:0]+1;end
        endcase
     NS_IDLE:if(cmd_wire==WCAT && data_in[15:0]!=0)identifier[23:0]<= identifier[23:0]+1;
     endcase
    if (cmd_wire==WTESTCAT) cat[15:0]<=data_in[15:0];
    end
//
else // (~SR)
    begin
    case(neuron_state[1:0])
    NS_IDLE:
        case (cmd_wire)
```

```
        WGCR: begin ctxt[7:0]<=data_in[7:0];nselect<=1; end
        WMINIF: minif[15:0]<=data_in[15:0];
        WMAXIF: aif[15:0]<=data_in[15:0];
      WCAT: if (data_in[15:0]!=16'h0000) sm_state[2:0]<=CY_START2;
        endcase
    NS_RTL:
        case (cmd_wire)
        WGCR: begin ctxt[7:0]<=data_in[7:0];nselect<=1; end
        WMINIF: minif[15:0]<=data_in[15:0];
        WMAXIF: aif[15:0]<=data_in[15:0];
        WLCOMP:begin exclude_loop<=0;exclude_cat<=0;fire<=0;end
      WCAT:  begin cat[15:0]<={1'b0,data_in[14:0]};sm_state[2:0]<=CY_START2;end
        endcase
    NS_COMMITTED:
        begin
        if (nselect)
            case (cmd_wire)
            WMINIF: minif[15:0]<=data_in[15:0];
            WCOMP:comp[7:0]<=data_in[7:0];
            WLCOMP:begin comp[7:0]<=data_in[7:0];exclude_loop<=0;exclude_cat<=0;fire<=0;sm_state[2:0]<=CY_START2;end
            WCAT:
                begin
                if (fire)
                    begin
                    if (cat[14:0]==data_in[14:0]) oktolearn<=0;
                    else
                        begin
                        aif[15:0]<=new_aif[15:0]; cat[15]<=degenerated;
                        if (data_in[14:0]==0) oktolearn<=0;
                        end
                    end
                sm_state[2:0]<=CY_START2;
                end
            RDIST: if (fire) begin exclude_loop<=0;sm_state[2:0]<=CY_LOOP;end
            RCAT:  if (fire) sm_state[2:0]<=CY_START2;
            endcase
        if (cmd_wire==WGCR)
           if(data_in[15:0]==0) nselect<=1;
           else begin if (ctxt[7:0]!=data_in[7:0]) nselect<=0; else nselect<=1; end
        end
        endcase  //end of neuron state
    end //end of ~SR
end // End of START1
//
CY_START2:
```

```
begin
sm_state_neg[2:0]<=CY_START2;
case(neuron_state[1:0])
    NS_IDLE:
      if (cmd==WCAT)
            if (oktolearn_in) sm_state[2:0]<=CY_LAST1; //to increment identifier
            else sm_state[2:0]<=CY_START1;
    NS_RTL:
      if(cmd==WCAT)
            begin
            if (oktolearn_in) sm_state[2:0]<=CY_LOOP;
            else begin cat[15:0]<=16'h00;sm_state[2:0]<=CY_START1; end
            end
    NS_COMMITTED:
        case(cmd)
        WLCOMP:
            begin
            sm_state[2:0]<=CY_LAST1;
            if (~KNN)
               begin if (dist[15:0]>=aif[15:0]) fire<=0;else fire<=1; end
             else fire<=1;
             end
        WCAT:if (oktolearn_in) sm_state[2:0]<=CY_LOOP; else sm_state[2:0]<=CY_START1;
        RCAT:begin
            if (cat[14:0]!=data_in[14:0]) sm_state[2:0]<=CY_LOOP;else sm_state[2:0]<=CY_LAST1;
            end
         default: sm_state[2:0]<=CY_START1;
         endcase
endcase //case on neuron state
end // End Start2
//
CY_LOOP:
begin
    sm_state_neg[2:0]<=CY_LOOP;
    case(neuron_state[1:0])
    NS_RTL, NS_COMMITTED:
        begin
        if(data_out[bit_ptr[3:0]]>data_in[bit_ptr[3:0]])exclude_loop<=1;
        if (bit_ptr[3:0]==0) sm_state[2:0]<=CY_LAST1;
        end
    endcase
end
//
CY_LAST1:
begin
```

```
    case(neuron_state[1:0])
    NS_IDLE:
      if (cmd==WCAT)
            begin identifier[23:0]<= identifier[23:0]+1;sm_state_neg[2:0]<=CY_IDLE_NEG;end
    NS_RTL:
      if (cmd==WCAT)
            begin
            if (cat!=0)
                begin
                aif[15:0]<=new_aif[15:0]; cat[15]<=degenerated;
                identifier[23:0]<= identifier[23:0]+1;
                sm_state_neg[2:0]<=CY_IDLE_NEG;
                end
            sm_state_neg[2:0]<=CY_IDLE_NEG;
            end
    NS_COMMITTED:
        case(cmd)
        WLCOMP: sm_state_neg[2:0]<=CY_LAST1;
        RDIST: if (fire && ~exclude_loop && ~exclude_cat) sm_state_neg[2:0]<=CY_LAST1; else
sm_state_neg[2:0]<=CY_IDLE_NEG;
        RCAT:  if (fire && ~exclude_loop && ~exclude_cat) begin exclude_cat<=1;sm_state_neg[2:0]<=CY_LAST1;end else
sm_state_neg[2:0]<=CY_IDLE_NEG;
        endcase
    endcase //end of neuron state
sm_state[2:0]<=CY_START1;
end
//
endcase //end case on state machine status
//
end
//
//----------------------------------------
// Negative edge of the clock
// Update all the flags of the neuron
//----------------------------------------
always @(negedge clk or negedge reset_l)
if(~reset_l)
      begin
      id<=0;unc_l<=1;firing_id<=0;
    data_out[15:0]<=16'hFFFF;
    ramindex[7:0]<=8'h00;
    dist[15:0]<=16'h0000;
    bit_ptr[3:0]<=4'hF;
    dco<=0;
      end
```

```
else
//
begin
//
case(sm_state_neg[2:0])
//
CY_IDLE_NEG: if (~SR && cat[15:0]!=0) dco<=1;
//
CY_START1:
begin
    data_out[15:0]<=16'hFFFF;
    if (cmd==WNSR) begin ramindex[7:0]<=0; if (~SR) dist<=0; end
    if (cmd==WINDEXCOMP) ramindex[7:0]<=protoindex[7:0];
    if (cmd==RNCOUNT && neuron_state[1:0]==NS_RTL) data_out[15:0]<=identifier[15:0];
//
    if (SR)
    begin
    if (cmd==RESETCHAIN) begin ramindex[7:0]<=0; dco<=0; end
    if (cmd==WTESTCOMP) ramindex[7:0]<=ramindex[7:0]+1;
    if (neuron_state[1:0]==NS_RTL)
        case(cmd)
        RNCR: data_out[15:0]<={identifier[23:16],ctxt[7:0]};
        RCOMP: begin data_out[15:0]<={8'h00,protodata[7:0]};ramindex[7:0]<=ramindex[7:0]+1;end
        RAIF: data_out[15:0]<=aif[15:0];
        RMINIF: data_out[15:0]<=minif[15:0];
        RCAT: begin data_out[15:0]<=cat[15:0];ramindex[7:0]<=0; if (cat[15:0]!=0)dco<=1; end
        RDIST:data_out[15:0]<=dist[15:0];
        RNID: data_out[15:0]<=identifier[15:0];
        WCAT: begin ramindex[7:0]<=0; if (cat[15:0]!=0)dco<=1; end
        WCOMP: ramindex[7:0]<=ramindex[7:0]+1;
        endcase
    end
//
    else //if ~SR
    begin
    case(neuron_state[1:0])
    NS_RTL:
        case(cmd)
        RMINIF: data_out[15:0]<=minif[15:0];
        RMAXIF: data_out[15:0]<=aif[15:0];
        RGCR: data_out[15:0]<={identifier[23:16],ctxt[7:0]};
        WCOMP:  ramindex[7:0]<=ramindex[7:0]+1;
        WLCOMP: begin ramindex[7:0]<=0;id<=0; unc_l<=1; end
      WCAT:  unc_l<=1;
        endcase
```

```
    NS_COMMITTED:
        if (nselect)
            case(cmd)
            WCOMP,WLCOMP:
                begin
                firing_id<=0;id<=0; unc_l<=1;
                if (ramindex[7:0]==0) dist[15:0]<={8'h00,abs[7:0]};
                else
                    begin
                    if (ctxt[7]==0) dist[15:0]<=dist[15:0]+{8'h00,abs[7:0]};
                    else begin if (abs[7:0]>=dist[7:0]) dist[15:0]<={8'h00,abs[7:0]}; end
                    end
                if (cmd==WCOMP)ramindex[7:0]<=ramindex[7:0]+1;// else ramindex[7:0]<=0;
                end
            WCAT: unc_l<=1; //to work with wire-mux with oktolearn
            RDIST:if (fire && ~exclude_cat) data_out[15:0]<=dist[15:0];
            RNID: if (firing_id) begin data_out[15:0]<=identifier[15:0];firing_id<=0; end
            RNCR: if (firing_id) data_out[15:0]<={8'h00,identifier[23:16]};
            RCAT: if (fire && ~exclude_loop && ~exclude_cat) data_out[15:0]<={1'b0,cat[14:0]};
            endcase
//
        else begin id<=0; unc_l<=1; end
//
    endcase //end of the case on neuron_state
    bit_ptr[3:0]<=15;
    end //end of ~SR
end
//
CY_START2:
begin
    case(cmd)
    WLCOMP:
        begin
        if((neuron_state[1:0]==NS_COMMITTED) && fire) data_out[15:0]<={1'b0,cat[14:0]};
        ramindex[7:0]<=0;
        end
    WCAT:
        case(neuron_state[1:0])
        NS_RTL: data_out[15:0]<=aif[15:0];
        NS_COMMITTED: data_out[15:0]<=dist[15:0];
        endcase
    endcase
end
//
CY_LOOP:
```

```
begin
    case(cmd)
    WCAT:
        case(neuron_state[1:0])
        NS_RTL,NS_COMMITTED:if (exclude_loop) data_out[15:0]<=16'hFFFF;
        endcase
    RDIST, RCAT: if ((neuron_state[1:0]==NS_COMMITTED) && fire && exclude_loop) data_out[15:0]<=16'hFFFF;
    endcase
if (bit_ptr !=0) bit_ptr[3:0]<=bit_ptr[3:0]-1;
end
//
CY_LAST1:
    begin
    if (neuron_state[1:0]==NS_COMMITTED)
        case(cmd)
        WLCOMP:if(fire) begin if (cat[14:0]==data_in[14:0]) id<=1; else unc_l<=0;end
        RDIST: if (fire && ~exclude_loop && ~exclude_cat) data_out[15:0]<=dist[15:0];
        RCAT:  if (fire && ~exclude_loop && exclude_cat) begin data_out[15:0]<=cat[15:0];firing_id<=1; end
        endcase
    end
//
endcase //end of case on neuron_state
end
//
endmodule
```