

// **nm96_top.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.

*/

//

// 3 clusters of 32 neurons

// option to instantiate more clusters if they fit into the FPGA

`timescale 1 ns / 100 ps // timescale for following modules

// See NM500 documentation for operation

module nm_top (input G_CLK, input G_RESET_l, input CS_l, input DS, input RW_l, input [3:0]REG, inout [15:0]DATA, inout ID_l, inout UNC_l, input DCI, output DCO, output RDY);

//

wire reset_l;

wire [15:0]data_in;

wire [15:0]data_out;

//

// Ready lines tree

// Wires declarations

wire ready_n1; wire ready_n2; wire ready_n3;

// Build global AND

wire ready_n=ready_n1 & ready_n2 & ready_n3;

//

// ID lines tree

// Wires declarations

wire id1; wire id2; wire id3;

// Build global OR

wire id= id1 | id2 | id3;

//

```

// UNC lines tree
// Wire declarations
wire unlearn1; wire unlearn2; wire unlearn3;
// Build global AND
wire unlearn_w= unlearn1 & unlearn2 & unlearn3;
//
wire oktolearn;
reg [15:0]NSR;    //network status register
wire ds_in=DS;
wire read_in=RW_1;
wire [3:0]reg_in=REG[3:0];
wire ready_out;
wire unlearn_in;
wire id_1_in;
// -----
// Generation of clocks and standby signals
// -----
reg standby_r;
always @(posedge G_CLK or negedge reset_1)
if (~reset_1) standby_r<=0; else standby_r<=CS_1;
//
wire G_clk_w= ~standby_r & G_CLK;
// -----
// Upon external G_RESET_1, generation of a clean reset_1 with a duration of 256 G_clk
// -----
reg [15:0]reset_counter;

always @(posedge G_CLK or negedge G_RESET_1)
    if(~G_RESET_1)reset_counter[15:0]<=16'h00FF;
    //if(~G_RESET_1)reset_counter[15:0]<=16'h000F; // For simu make it shorter
    else if (reset_counter[15:0]!=16'h0000) reset_counter[15:0]<=reset_counter[15:0]-16'h0001;
//
assign reset_1=(reset_counter[15:0]==16'h0000)? 1'b1: 1'b0;
// -----
// Access to Status Registers
// -----
// Update the control bits of the Network Status and Recognition Status registers
parameter NETWORK_STATUS=5'hD;
//
always @(posedge G_clk_w or negedge reset_1)
begin
if(~reset_1)
    begin NSR[15:0]<=16'h0000; end
else
    begin
        NSR[3]<=~id_1_in;NSR[2]<=~unlearn_in;
        if ((reg_in[3:0]==NETWORK_STATUS) & ds_in & ~read_in) NSR[15:0]<=data_in[15:0];
    end
end

```

```

end
end
//
wire [3:0]reg_in_pad= REG[3:0];
assign ready_out= (reset_l & ready_n);
assign RDY=ready_out ? 1'b1 :1'b0;
//-----
// Assign data pad (muxed with category in stand alone)
//-----
wire [15:0]data_in_pad;
assign data_in[15:0]= data_in_pad[15:0];
// Data out from cluster dotted AND bus
// Wires declaration
wire [15:0]data_out_n1; wire [15:0]data_out_n2;wire [15:0]data_out_n3;
//Build global AND bus
wire [15:0]data_out_n= data_out_n1[15:0] & data_out_n2[15:0] & data_out_n3[15:0]
& (read_in & (reg_in[3:0]==NETWORK_STATUS) ? NSR[15:0]: 16'hFFFF) | (RW_l ? 16'h0000 : 16'hFFFF) ; // test
//
// PADS for Xilinx implementation
IOBUF DATApad00(.I(1'b0),.O(data_in_pad[0]),.T(data_out_n[0]),.IO(DATA[0]));
IOBUF DATApad01(.I(1'b0),.O(data_in_pad[1]),.T(data_out_n[1]),.IO(DATA[1]));
IOBUF DATApad02(.I(1'b0),.O(data_in_pad[2]),.T(data_out_n[2]),.IO(DATA[2]));
IOBUF DATApad03(.I(1'b0),.O(data_in_pad[3]),.T(data_out_n[3]),.IO(DATA[3]));
IOBUF DATApad04(.I(1'b0),.O(data_in_pad[4]),.T(data_out_n[4]),.IO(DATA[4]));
IOBUF DATApad05(.I(1'b0),.O(data_in_pad[5]),.T(data_out_n[5]),.IO(DATA[5]));
IOBUF DATApad06(.I(1'b0),.O(data_in_pad[6]),.T(data_out_n[6]),.IO(DATA[6]));
IOBUF DATApad07(.I(1'b0),.O(data_in_pad[7]),.T(data_out_n[7]),.IO(DATA[7]));
IOBUF DATApad08(.I(1'b0),.O(data_in_pad[8]),.T(data_out_n[8]),.IO(DATA[8]));
IOBUF DATApad09(.I(1'b0),.O(data_in_pad[9]),.T(data_out_n[9]),.IO(DATA[9]));
IOBUF DATApad10(.I(1'b0),.O(data_in_pad[10]),.T(data_out_n[10]),.IO(DATA[10]));
IOBUF DATApad11(.I(1'b0),.O(data_in_pad[11]),.T(data_out_n[11]),.IO(DATA[11]));
IOBUF DATApad12(.I(1'b0),.O(data_in_pad[12]),.T(data_out_n[12]),.IO(DATA[12]));
IOBUF DATApad13(.I(1'b0),.O(data_in_pad[13]),.T(data_out_n[13]),.IO(DATA[13]));
IOBUF DATApad14(.I(1'b0),.O(data_in_pad[14]),.T(data_out_n[14]),.IO(DATA[14]));
IOBUF DATApad15(.I(1'b0),.O(data_in_pad[15]),.T(data_out_n[15]),.IO(DATA[15]));
//
IOBUF ID_pad(.I(~(id & unclean_in)),.O(id_l_in),.T(~(id & unclean_in) ),.IO(ID_l));
IOBUF UNC_lpad(.I(unclearn_w),.O(unclearn_in),.T( unclean_w),.IO(UNC_l));
// END of PADS for Xilinx
assign oktolearn= unclean_in;           // Propagate the okay for the   RTL (Ready to Learn)
// -----
// 6 clusters of 16 neurons each
// Chip DCI should be connected to first cluster "dci" and DCO to the last cluster "dco"
// Master chip DCI must have a pullup resistor
// -----
// Daisy chain wires declarations
wire dc1, dc2 ;

```

```

parameter FORGET=4'hF;
//
wire ds_n= DS;
wire reset_n_1= (reg_in[3:0]==FORGET) & ~read_in ? ~ds_in : reset_1;
//
neuroncluster cluster1(.clk(G_clk_w), .reset_1(reset_n_1), .ds(ds_n), .read(read_in),
    .register(reg_in[3:0]), .data_in(data_in[15:0]), .SR(NSR[4]), .KNN(NSR[5]),
    .oktolearn_in(oktolearn), .dci(DCI), .dco(dc1), .id(id1), .unclearn(unclearn1), .data_out(data_out_n1[15:0]),
    .ready(ready_n1));
//
neuroncluster cluster2(.clk(G_clk_w), .reset_1(reset_n_1), .ds(ds_n), .read(read_in),
    .register(reg_in[3:0]), .data_in(data_in[15:0]), .SR(NSR[4]), .KNN(NSR[5]),
    .oktolearn_in(oktolearn), .dci(dc1), .dco(dc2), .id(id2), .unclearn(unclearn2), .data_out(data_out_n2[15:0]),
    .ready(ready_n2));
//
neuroncluster cluster3(.clk(G_clk_w), .reset_1(reset_n_1), .ds(ds_n), .read(read_in),
    .register(reg_in[3:0]), .data_in(data_in[15:0]), .SR(NSR[4]), .KNN(NSR[5]),
    .oktolearn_in(oktolearn), .dci(dc2), .dco(DCO), .id(id3), .unclearn(unclearn3), .data_out(data_out_n3[15:0]),
    .ready(ready_n3));
//

```