Anexo I

Código VHDL - Implementación de una red SOM en FPGA

Mikel Díaz Rodríguez

Leioa, 19 de febrero de 2020

Índice

1. Mapas autoorganizados (SOM)

Los mapas autoorganizados, o también llamados Self-Organizing Maps (SOM), son un tipo de red neuronal artificial que se entrena utilizando técnicas de aprendizaje no supervisado para producir una representación discreta de baja dimensión del espacio de las muestras de entrada, llamado mapa. Los mapas autoorganizados difieren de otras redes neuronales artificiales, ya que aplican el aprendizaje competitivo en oposición al aprendizaje de corrección de errores, en el sentido que estos usan una función de vecindad para preservar las propiedades topológicas del espacio de entrada.

Un mapa autoorganizado está formado por componentes llamados nodos o neuronas. Cada neurona está asociada con un vector de pesos, de la misma dimensión de los vectores de entrada, y es una posición en el mapa (espacio de entrada). Normalmente las neuronas están colocadas en forma de una red bidimensional. El entrenamiento del SOM consiste en trasladar vectores de peso hacia los datos de entrada (reduciendo la distancia métrica) teniendo en cuenta la topología inducida por el espacio del mapa.

Una vez entrenado, se puede clasificar cualquier otro dato de entrada que se desee conocer a que grupo pertenece. Para ello, se busca la neurona que tenga la menor distancia a la muestra de entrada ya que la neurona ganadora pertenece a la misma agrupación que el dato de entrada. Para realizar este calculo se utiliza la implementación en hardware descrita en este anexo.

2. Arquitectura del SOM

La arquitectura diseñada está compuesta por 5 módulos: los registros de entrada, las neuronas, los comparadores, la ROM interna y el controlador. Con el fin de que la respuesta sea dada en el menor tiempo posible, la arquitectura ha sido diseñada para que los procesos se calculen en paralelo. Por este motivo para calcular cual es la neurona que tiene la menor distancia a la muestra de entrada se hace haciendo uso de los comparadores. Estos van comparando las salidas de las neuronas de dos en dos hasta obtener la neurona con la menor distancia. Además, mediante un leguaje de descripción de hardware VHDL (VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language)) se consigue escalar la arquitectura dependiendo del número de neuronas que se deseen implementar o del número de características que tenga la base de datos.

Dentro de la arquitectura de la FPGA, se implementa una ROM interna que guarda los pesos de las neuronas y las agrupaciones a las que pertenecen una vez se haya completado el entrenamiento del SOM y se hayan clasificado las neuronas. También, se utiliza una señal de Reset que borra todos los registros y una señal de Enable que habilita el controlador.

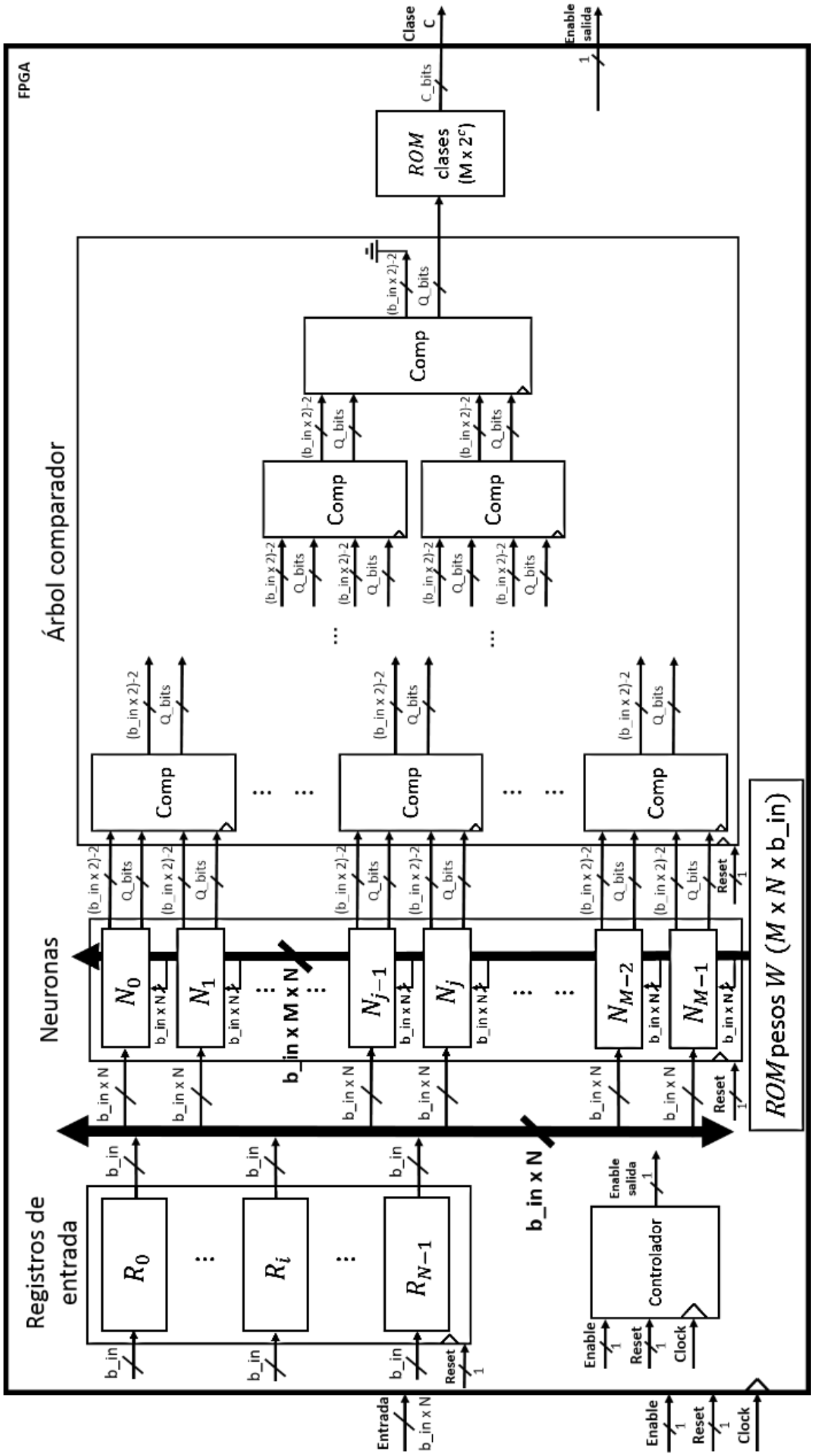

Figura 1: Arquitectura del circuito completo.

2.1. Adder

Módulo incorporado en “neuron 4” encargado de sumar dos números en binario con flanco ascendente de reloj. Este módulo forma parte del árbol sumador.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Mikel Díaz
3   -- Create Date: 10/02/2020
4   -- Module Name: adder - Behavioral
5   -- Project Name: Parallel_SOM_drivers
6   -- Target Devices: Nexys4 DDR Rev. C
7   -- Description: Sumador con signo.
8   ----------------------------------------------------------------------------------
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use IEEE.NUMERIC_STD.ALL;
13  use IEEE.STD_LOGIC_SIGNED.ALL;
14
15  entity adder is
16      Generic (b_mult: INTEGER); -- número de bits después de multiplicar
17      Port ( D1 : in STD_LOGIC_VECTOR (b_mult-1 downto 0); -- Entrada 1
18             D2 : in STD_LOGIC_VECTOR (b_mult-1 downto 0); -- Entrada 2
19             Clock, Reset : in STD_LOGIC; -- Puerto de reloj y reset
20             Salida : out STD_LOGIC_VECTOR (b_mult-1 downto 0)); -- Salida
21  end adder;
22
23  architecture Behavioral of adder is
24
25  -- Declaración de señales
26  signal Salida_reg :STD_LOGIC_VECTOR (b_mult-1 downto 0) := (others =>'0'); -- Señal
    de salida de registro
27
28  begin
29
30  -- Suma D1 + D2 con flanco ascendente del Clock
31  process (Clock)
32  begin
33      if (Clock'event and Clock='1') then
34          if (Reset = '1') then
35              Salida_reg <= (others =>'0');
36          else
37              Salida_reg <= D1 + D2;
38          end if;
39      end if;
40  end process;
41
42  Salida <= Salida_reg;
43
44  end Behavioral;
```

2.2. Distance

Módulo incorporado en “neuron 4” encargado de calcular la diferencia entre una característica de la muestra de entrada y el peso asociado a dicha característica y lo eleva al cuadrado. El resultado se proporciona con flanco ascendente de reloj.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Mikel Díaz
3   -- Create Date: 10/02/2020
4   -- Module Name: distance - Behavioral
5   -- Project Name: Parallel_SOM
6   -- Target Devices: Nexys4 DDR Rev. C
7   -- Description: Cálculo de la distancia euclídea al cuadrado de dimensión uno.
8   ----------------------------------------------------------------------------------
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use IEEE.NUMERIC_STD.ALL;
13  use IEEE.STD_LOGIC_SIGNED.ALL;
14
15  entity distance is
16      Generic (b_in: INTEGER; -- número de bits de los datos de entrada
17               b_mult: INTEGER); -- número de bits después de multiplicar
18      Port ( X : in STD_LOGIC_VECTOR  (b_in-1 downto 0); -- Una característica de la
    muestra X
19             W : in STD_LOGIC_VECTOR  (b_in-1 downto 0); -- Una característica del
    peso W
20             Clock, Reset : in STD_LOGIC; -- Puerto de reloj y reset
21             d : out STD_LOGIC_VECTOR (b_mult-1 downto 0)); -- Salida
22  end distance;
23
24  architecture Behavioral of distance is
25
26  -- Declaración de señales
27  signal Resta, Resta_reg :STD_LOGIC_VECTOR (b_in-1 downto 0) := (others =>'0'); --
    Señal de conexión entre registros y señal de salida de registro
28  signal S_reg: STD_LOGIC_VECTOR ((b_in*2)-1 downto 0) := (others =>'0'); -- Señal de
    salida de registro
29
30  begin
31
32  -- Resta entre la muestra X y el peso W
33  process (Clock)
34  begin
35      if (Clock'event and Clock='1') then
36          if (Reset = '1') then
37              Resta_reg <= (others =>'0');
38          else
39              Resta_reg <= X - W;
40          end if;
41      end if;
42  end process;
43
44  Resta <= Resta_reg;
45
46  -- Cuadrado de la resta entre la muestra X y el peso W
47  process (Clock)
48  begin
49      if (Clock'event and Clock='1') then
50          if (Reset = '1') then
51              S_reg <= (others =>'0');
52          else
53              S_reg <= Resta * Resta;
54          end if;
```

```
55          end if;
56      end process;
57
58      -- Truncado de la señal de salida del multiplicador
59      d <= S_reg((b_in*2)-3 downto 2);
60
61      end Behavioral;
```

2.3 Neuron 4

Módulo encargado de calcular la distancia euclídea al cuadrado entre la muestra de entrada y el peso de una neurona. La arquitectura depende del número de características *N* que tengan las muestras de entrada y los pesos de las neuronas. En este caso, se ha diseñado para un número de características igual a 4.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Mikel Díaz
3   -- Create Date: 10/02/2020
4   -- Module Name: neuron_4 - Behavioral
5   -- Project Name: Parallel_SOM
6   -- Target Devices: Nexys4 DDR Rev. C
7   -- Description: Arquitectura de una neurona de 4 características.
8   ----------------------------------------------------------------------------------
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use ieee.numeric_std.all;
13
14  entity neuron_4 is
15      Generic (N: INTEGER;  -- número de características (atributos) de los datos de
    entrada (X)
16               b_in: INTEGER; -- número de bits de los datos de entrada
17               b_mult: INTEGER); -- número de bits después de multiplicar
18      Port ( X0_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 1 de la
    muestra X
19             X1_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 2 de la
    muestra X
20             X2_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 3 de la
    muestra X
21             X3_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 4 de la
    muestra X
22
23             W0_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 1 del
    peso W
24             W1_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 2 del
    peso W
25             W2_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 3 del
    peso W
26             W3_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);  -- Característica 4 del
    peso W
27
28             Clock, Reset : in STD_LOGIC; -- Puerto de reloj y reset
29
30             Salida : out STD_LOGIC_VECTOR (b_mult-1 downto 0)); -- Puerto de salida
31  end neuron_4;
32
33  architecture Behavioral of neuron_4 is
34
35  -- Declaración de señales
36
37  type tabla_X_n is array (0 to N-1) of std_logic_vector (b_in-1 downto 0);
38  signal mem_X_n: tabla_X_n; -- Guarda la muestra X
39
40  type tabla_W_n is array (0 to N-1) of std_logic_vector (b_in-1 downto 0);
41  signal mem_W_n: tabla_W_n;  -- Guarda el peso W
42
43  type tabla_S is array (0 to N-1) of std_logic_vector (b_mult-1 downto 0);
44  signal mem_S: tabla_S; -- Guarda la salida del componente "distance"
45
46  type tabla_suma is array (0 to 1) of std_logic_vector (b_mult-1 downto 0);
47  signal mem_suma: tabla_suma;  -- Guarda la salida del componente "adder"
48
49  signal dT: STD_LOGIC_VECTOR (b_mult-1 downto 0)  := (others =>'0');
```

```
50
51  -- Declaración de componentes
52
53  component distance
54      Generic (   b_in: INTEGER;
55                  b_mult: INTEGER);
56      Port ( X : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
57             W : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
58             Clock, Reset : in STD_LOGIC;
59             d : out STD_LOGIC_VECTOR (b_mult-1 downto 0));
60  end component;
61
62  component adder
63      Generic (b_mult: INTEGER);
64      Port ( D1 : in STD_LOGIC_VECTOR (b_mult-1 downto 0);
65             D2 : in STD_LOGIC_VECTOR (b_mult-1 downto 0);
66             Clock, Reset : in STD_LOGIC;
67             Salida : out STD_LOGIC_VECTOR (b_mult-1 downto 0));
68  end component;
69
70  begin
71
72  -- Guarda en memoria las entrada
73  mem_X_n(0) <= X0_n;
74  mem_X_n(1) <= X1_n;
75  mem_X_n(2) <= X2_n;
76  mem_X_n(3) <= X3_n;
77  mem_W_n(0) <= W0_n;
78  mem_W_n(1) <= W1_n;
79  mem_W_n(2) <= W2_n;
80  mem_W_n(3) <= W3_n;
81
82  -- Cálculo de la distancia euclidea al cuadrado
83
84  distancias:
85  for i in 0 to N-1 generate
86      dist: distance
87          GENERIC MAP (   b_in => b_in,
88                          b_mult => b_mult)
89          PORT MAP (mem_X_n(i), mem_W_n(i), Clock, Reset, mem_S(i));
90  end generate;
91
92  sumas:
93  for i in 0 to ((N/2)-1) generate
94      sum: adder
95          GENERIC MAP (b_mult => b_mult)
96          PORT MAP (mem_S(i*2), mem_S((i*2)+1), Clock, Reset, mem_suma(i));
97  end generate;
98
99  suma_T: adder
100     GENERIC MAP (b_mult => b_mult)
101     PORT MAP (mem_suma(0), mem_suma(1), Clock, Reset, dT);
102
103 Salida <= dT;
104
105 end Behavioral;
```

2.4. Comparer

El módulo tiene como entrada las distancias de dos neuronas a la muestra de entrada y el número de neuronas al que pertenecen, y devuelve la distancia más pequeña y el número de neurona al que pertenece dicha distancia. Este módulo forma parte del árbol comparador.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Mikel Díaz
3   -- Create Date: 10/02/2020
4   -- Module Name: comparer - Behavioral
5   -- Project Name: Parallel_SOM
6   -- Target Devices: Nexys4 DDR Rev. C
7   -- Description: Arquitectura de un comparador.
8   ----------------------------------------------------------------------------------
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use IEEE.NUMERIC_STD.ALL;
13
14  entity comparer is
15      Generic ( Q_bits: INTEGER; -- número de bits necesarios para enumerar todas las
    neuronas
16                b_mult: INTEGER); -- número de bits después de multiplicar
17      Port ( N1 : in STD_LOGIC_VECTOR (b_mult-1 downto 0); -- Entrada 1
18             N2 : in STD_LOGIC_VECTOR (b_mult-1 downto 0); -- Entrada 2
19             Q1 : in STD_LOGIC_VECTOR (Q_bits-1 downto 0); -- Neurona a la que
    pertenece la entrada 1
20             Q2 : in STD_LOGIC_VECTOR (Q_bits-1 downto 0); -- Neurona a la que
    pertenece la entrada 2
21             Clock, Reset : in STD_LOGIC; -- Puerto de reloj y reset
22             Salida_N : out STD_LOGIC_VECTOR (b_mult-1 downto 0); -- Puerto de salida
    (entrada)
23             Salida_Q : out STD_LOGIC_VECTOR (Q_bits-1 downto 0)); -- Puerto de salida
    (neurona)
24  end comparer;
25
26  architecture Behavioral of comparer is
27
28  -- Declaración de señales
29  signal Salida_N_reg : std_logic_vector(b_mult-1 downto 0):= (others =>'0'); -- Señal
    de salida de registro
30  signal Salida_Q_reg : std_logic_vector(Q_bits-1 downto 0):= (others =>'0'); -- Señal
    de salida de registro
31
32  begin
33
34  -- Comparación (salida <= la entrada más pequeña y la neurona a la que pertenece)
35  process (Clock)
36  begin
37      if (Clock'event and Clock='1') then
38          if (Reset = '1') then
39              Salida_N_reg <= (others =>'0');
40              Salida_Q_reg <= (others =>'0');
41          else
42              if (N1 > N2) then
43                  Salida_N_reg <= N2;
44                  Salida_Q_reg <= Q2;
45
46              else
47                  Salida_N_reg <= N1;
48                  Salida_Q_reg <= Q1;
49              end if;
50          end if;
51      end if;
```

```
52    end process;
53
54    Salida_N <= Salida_N_reg;
55    Salida_Q <= Salida_Q_reg;
56
57    end Behavioral;
```

2.5. Input register

Arquitectura de un registro de entrada. Introduce la señal con el flanco ascendente del reloj.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Mikel Díaz
3   -- Create Date: 10/02/2020
4   -- Module Name: input_register - Behavioral
5   -- Project Name: Parallel_SOM
6   -- Target Devices: Nexys4 DDR Rev. C
7   -- Description: Arquitectura de un registro de entrada.
8   ----------------------------------------------------------------------------------
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12
13  entity input_register is
14      Generic (b_in: INTEGER); -- número de bits de los datos de entrada
15      Port (  X : in STD_LOGIC_VECTOR (b_in-1 downto 0); -- Puerto de entrada
16              Clock, Reset : in STD_LOGIC; -- Puerto de reloj y reset
17              X_act : out STD_LOGIC_VECTOR (b_in-1 downto 0)); -- Puerto de salida
18  end input_register;
19
20  architecture Behavioral of input_register is
21
22  -- Declaración de señales
23  signal X_act_reg: STD_LOGIC_VECTOR (b_in-1 downto 0) := (others =>'0');
24
25  begin
26
27  -- Adquiere la entrada X de manera síncrona
28  process (Clock, Reset)
29  begin
30      if (Reset = '1') then
31          X_act_reg <= (others =>'0');
32      elsif (Clock'event and Clock='1') then
33          X_act_reg <= X;
34      end if;
35  end process;
36
37  X_act <= X_act_reg;
38
39  end Behavioral;
```

2.6. Main iris

Arquitectura del circuito completo para la base de datos *Iris*.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Mikel Díaz
3   -- Create Date: 10/02/2020
4   -- Module Name: main_iris - architecture_main
5   -- Project Name: Parallel_SOM
6   -- Target Devices: Nexys4 DDR Rev. C
7   -- Description: Arquitectura del circuito completo.
8   ----------------------------------------------------------------------------------
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use IEEE.NUMERIC_STD.ALL;
13
14  entity main_iris is
15      Generic (N: INTEGER := 4;  -- número de características (atributos) de los datos
    de entrada (X)
16               M: INTEGER := 64; -- número total de neuronas
17               Q_bits: INTEGER := 6; -- número de bits necesarios para enumerar todas
    las neuronas
18               b_in: INTEGER := 8; -- número de bits de los datos de entrada
19               b_mult: INTEGER := 12; -- número de bits después de multiplicar
20               cycles: POSITIVE := 9); -- número de ciclos de reloj hasta obtener
    salida Clase_c válida
21                                       -- log2 (N) + log2 (M) + 1 = 2 + 6 + 1 = 9
22      Port (  X : in STD_LOGIC_VECTOR  ((b_in*N)-1 downto 0); -- muestra de entrada X
23              Clock, Reset, Enable : IN STD_LOGIC; -- Puerto de reloj y reset
24              Enable_salida : OUT STD_LOGIC; -- Salida válida
25              Clase_c : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)); -- Puerto de salida
26  end main_iris;
27
28   architecture architecture_main of main_iris is
29
30  type tabla_X_act is array (0 to N-1) of std_logic_vector (b_in-1 downto 0);
31  signal mem_X_act: tabla_X_act; -- Guarda todas las entradas X actuales del registro
32
33  type tabla_pesos_1 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0); --
    Valores de los pesos W para el atributo 1
34  signal mem_pesos_1: tabla_pesos_1 := ( "00101001", "00101011", "00101110",
    "00110010", "00110110", "00111010", "00111110", "00111110",
35                                         "00101001", "00101101", "00110000",
    "00110101", "00110111", "00111100", "00111101", "00111010",
36                                         "00101000", "00101011", "00101100",
    "00110010", "00110100", "00110110", "00110110", "00110111",
37                                         "00101000", "00100110", "00101101",
    "00110010", "00110010", "00110011", "00110011", "00110010",
38                                         "00100101", "00100111", "00101001",
    "00101110", "00110001", "00110001", "00110011", "00110010",
39                                         "00100011", "00101000", "00101100",
    "00101110", "00110000", "00110010", "00110100", "00110011",
40                                         "00100100", "00100110", "00101110",
    "00101100", "00101111", "00110000", "00110001", "00101110",
41                                         "00100101", "00101000", "00101010",
    "00100111", "00101101", "00101111", "00101101", "00101110" );
42  type tabla_pesos_2 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0); --
    Valores de los pesos W para el atributo 2
43  signal mem_pesos_2: tabla_pesos_2 := ( "00011110", "00011111", "00100000",
    "00011101", "00011000", "00011000", "00010101", "00011110",
44                                         "00011111", "00100010", "00011101",
    "00011000", "00010111", "00010111", "00011000", "00011101",
```

```
45                                          "00011100", "00011100", "00011010",
    "00010111", "00010110", "00010100", "00011000", "00011001",
46                                          "00011010", "00011011", "00010111",
    "00010010", "00010101", "00010110", "00010110", "00011011",
47                                          "00011101", "00011001", "00010111",
    "00010110", "00010110", "00010101", "00011000", "00011011",
48                                          "00011000", "00010111", "00010011",
    "00010101", "00011000", "00010101", "00011010", "00011010",
49                                          "00010010", "00010101", "00010101",
    "00010100", "00011000", "00011011", "00011001", "00010110",
50                                          "00010101", "00010011", "00010110",
    "00010100", "00011000", "00011010", "00010110", "00010110" );
51  type tabla_pesos_3 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0);  --
    Valores de los pesos W para el atributo 3
52  signal mem_pesos_3: tabla_pesos_3 := ( "00001111", "00001010", "00001010",
    "00011000", "00100110", "00101110", "00110111", "00110100",
53                                          "00001100", "00001100", "00010101",
    "00100011", "00101001", "00110001", "00110101", "00110001",
54                                          "00001011", "00001011", "00010011",
    "00100010", "00100101", "00101110", "00101011", "00101110",
55                                          "00001010", "00001111", "00011101",
    "00100011", "00101000", "00101010", "00101101", "00101101",
56                                          "00001000", "00001100", "00010011",
    "00100001", "00100110", "00101101", "00101100", "00101011",
57                                          "00001010", "00010010", "00011110",
    "00100000", "00100100", "00101000", "00101001", "00101010",
58                                          "00001010", "00010010", "00011100",
    "00100000", "00100010", "00100100", "00100111", "00101001",
59                                          "00010000", "00011001", "00011111",
    "00100100", "00100100", "00100110", "00100111", "00101001" );
60
61  type tabla_pesos_4 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0);  --
    Valores de los pesos W para el atributo 4
62  signal mem_pesos_4: tabla_pesos_4 := ( "00000011", "00000011", "00000010",
    "00000111", "00001100", "00001101", "00010010", "00010001",
63                                          "00000010", "00000010", "00000110",
    "00001011", "00001101", "00010001", "00010001", "00010100",
64                                          "00000010", "00000010", "00000100",
    "00001010", "00001100", "00001110", "00010001", "00010010",
65                                          "00000010", "00000010", "00001010",
    "00001010", "00001111", "00001111", "00010010", "00010011",
66                                          "00000010", "00000001", "00000100",
    "00001000", "00001010", "00001011", "00001110", "00010010",
67                                          "00000001", "00000100", "00001000",
    "00001010", "00001100", "00001100", "00010000", "00010010",
68                                          "00000010", "00000101", "00001000",
    "00001010", "00001100", "00001101", "00001111", "00010011",
69                                          "00000100", "00001000", "00001011",
    "00001110", "00001100", "00001110", "00010000", "00001111" );
70
71  type tabla_cluster is array (0 to M-1) of integer; -- Agrupación a la que pertenece
    cada neurona
72  signal mem_cluster: tabla_cluster := ( 1, 1, 1, 3, 3, 2, 2, 2,
73                                         1, 1, 1, 3, 2, 2, 2, 2,
74                                         1, 1, 1, 3, 3, 2, 2, 2,
75                                         1, 1, 3, 3, 2, 2, 2, 2,
76                                         1, 1, 1, 3, 3, 2, 2, 2,
77                                         1, 1, 3, 3, 3, 2, 2, 2,
78                                         1, 1, 3, 3, 3, 3, 3, 2,
79                                         1, 3, 3, 3, 3, 3, 3, 2 );
80
81  type tabla_salidas is array (0 to M-1) of std_logic_vector (b_mult-1 downto 0);
82  signal mem_salidas: tabla_salidas; -- Guarda las salidas de las neuronas
83
84  -- Salidas comparadores (por parejas y en paralelo)
85  ---- N ---> Guarda la distancia más pequeña
86
87  type tabla_N1 is array (0 to ((M/2)-1)) of std_logic_vector (b_mult-1 downto 0);
```

```
88    signal mem_N1: tabla_N1;
89
90    type tabla_N2 is array (0 to ((M/4)-1)) of std_logic_vector (b_mult-1 downto 0);
91    signal mem_N2: tabla_N2;
92
93    type tabla_N3 is array (0 to ((M/8)-1)) of std_logic_vector (b_mult-1 downto 0);
94    signal mem_N3: tabla_N3;
95
96    type tabla_N4 is array (0 to ((M/16)-1)) of std_logic_vector (b_mult-1 downto 0);
97    signal mem_N4: tabla_N4;
98
99    type tabla_N5 is array (0 to ((M/32)-1)) of std_logic_vector (b_mult-1 downto 0);
100   signal mem_N5: tabla_N5;
101
102   signal Salida_N : STD_LOGIC_VECTOR  (b_mult-1 downto 0) := (others =>'0'); --
      neurona con la distancia más pequeña (Salida <= distancia)
103
104   ---- Q ---> Guarda el valor de la neuron (se enumeran las neuronas mediante mem_Q)
105
106   type tabla_Q is array (0 to M-1) of std_logic_vector (Q_bits-1 downto 0); --
      Enumeración de las neuronas
107   signal mem_Q: tabla_Q := ( "000000", "000001", "000010", "000011", "000100",
      "000101", "000110", "000111",
108                              "001000", "001001", "001010", "001011", "001100",
      "001101", "001110", "001111",
109                              "010000", "010001", "010010", "010011", "010100",
      "010101", "010110", "010111",
110                              "011000", "011001", "011010", "011011", "011100",
      "011101", "011110", "011111",
111                              "100000", "100001", "100010", "100011", "100100",
      "100101", "100110", "100111",
112                              "101000", "101001", "101010", "101011", "101100",
      "101101", "101110", "101111",
113                              "110000", "110001", "110010", "110011", "110100",
      "110101", "110110", "110111",
114                              "111000", "111001", "111010", "111011", "111100",
      "111101", "111110", "111111" );
115
116   type tabla_Q1 is array (0 to ((M/2)-1)) of std_logic_vector (Q_bits-1 downto 0);
117   signal mem_Q1: tabla_Q1;
118
119   type tabla_Q2 is array (0 to ((M/4)-1)) of std_logic_vector (Q_bits-1 downto 0);
120   signal mem_Q2: tabla_Q2;
121
122   type tabla_Q3 is array (0 to ((M/8)-1)) of std_logic_vector (Q_bits-1 downto 0);
123   signal mem_Q3: tabla_Q3;
124
125   type tabla_Q4 is array (0 to ((M/16)-1)) of std_logic_vector (Q_bits-1 downto 0);
126   signal mem_Q4: tabla_Q4;
127
128   type tabla_Q5 is array (0 to ((M/32)-1)) of std_logic_vector (Q_bits-1 downto 0);
129   signal mem_Q5: tabla_Q5;
130
131   signal Salida_Q : STD_LOGIC_VECTOR  (Q_bits-1 downto 0) := (others =>'0'); --
      neurona con la distancia más pequeña (Salida <= número de neuron)
132
133   signal cnt : unsigned(cycles-1 DOWNTO 0); -- contador
134
135   -- Declaración de componentes
136
137   component neuron_4 -- Distancia entre la muestra de entrada y el peso de la neurona
138       Generic (N: INTEGER;
139                b_in: INTEGER;
140                b_mult: INTEGER);
141       Port ( X0_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
142              X1_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
143              X2_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
144              X3_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
```

```
145
146             W0_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
147             W1_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
148             W2_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
149             W3_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
150
151             Clock, Reset : in STD_LOGIC;
152
153             Salida : out STD_LOGIC_VECTOR (b_mult-1 downto 0) := (others =>'0'));
154     end component;
155
156     component comparer -- Comparación de distancias entre dos neuronas (Resultado <=
        distancia más pequeña)
157         Generic ( Q_bits: INTEGER;
158                   b_mult: INTEGER);
159         Port ( N1 : in STD_LOGIC_VECTOR (b_mult-1 downto 0);
160                N2 : in STD_LOGIC_VECTOR (b_mult-1 downto 0);
161                Q1 : in STD_LOGIC_VECTOR (Q_bits-1 downto 0);
162                Q2 : in STD_LOGIC_VECTOR (Q_bits-1 downto 0);
163                Clock, Reset : in STD_LOGIC;
164                Salida_N : out STD_LOGIC_VECTOR (b_mult-1 downto 0);
165                Salida_Q : out STD_LOGIC_VECTOR (Q_bits-1 downto 0));
166     end component;
167
168     component input_register -- Registro de las muestras de entrada X
169         Generic ( b_in: INTEGER);
170         Port (  X : in STD_LOGIC_VECTOR (b_in-1 downto 0);
171                 Clock, Reset : in STD_LOGIC;
172                 X_act : out STD_LOGIC_VECTOR (b_in-1 downto 0));
173     end component;
174
175     begin
176
177     -- Declaración de los registros de entrada
178     registros_entradas:
179     for i in 0 to N-1 generate
180         reg_X: input_register
181             Generic map (   b_in => b_in)
182             Port map (    X      => X((b_in*(i+1))-1 downto i*b_in),
183                           Clock => Clock,
184                           Reset => Reset,
185                           X_act => mem_X_act(i));
186     end generate;
187
188     -- Cálculo de las distancia entre las neuronas y la muestra de entrada
189     dist_pesos:
190     for i in 0 to M-1 generate
191         calc_neurona : neuron_4
192                 Generic map (   N => N,
193                                 b_in => b_in,
194                                 b_mult => b_mult)
195                 Port map (  X0_n => mem_X_act(0),
196                             X1_n => mem_X_act(1),
197                             X2_n => mem_X_act(2),
198                             X3_n => mem_X_act(3),
199                             W0_n => mem_pesos_1(i),
200                             W1_n => mem_pesos_2(i),
201                             W2_n => mem_pesos_3(i),
202                             W3_n => mem_pesos_4(i),
203                             Clock => Clock,
204                             Reset => Reset,
205                             Salida => mem_salidas(i));
206
207     end generate;
208
209     -- Comparación de las distancias (por parejas y en paralelo)
210     comp1_T:
211     for i in 0 to ((M/2)-1) generate
```

```
212         comp1: comparer
213             Generic map (   Q_bits => Q_bits,
214                             b_mult => b_mult)
215             Port map (  N1 => mem_salidas(i*2),
216                         N2 => mem_salidas((i*2)+1),
217                         Q1 => mem_Q (i*2),
218                         Q2 => mem_Q ((i*2)+1),
219                         Clock => Clock,
220                         Reset => Reset,
221                         Salida_N => mem_N1 (i),
222                         Salida_Q => mem_Q1 (i));
223     end generate;
224
225     comp2_T:
226     for i in 0 to ((M/4)-1) generate
227         comp2: comparer
228             Generic map (   Q_bits => Q_bits,
229                             b_mult => b_mult)
230             Port map (  N1 => mem_N1(i*2),
231                         N2 => mem_N1((i*2)+1),
232                         Q1 => mem_Q1 (i*2),
233                         Q2 => mem_Q1 ((i*2)+1),
234                         Clock => Clock,
235                         Reset => Reset,
236                         Salida_N => mem_N2 (i),
237                         Salida_Q => mem_Q2 (i));
238     end generate;
239
240     comp3_T:
241     for i in 0 to ((M/8)-1) generate
242         comp3: comparer
243             Generic map (   Q_bits => Q_bits,
244                             b_mult => b_mult)
245             Port map (  N1 => mem_N2(i*2),
246                         N2 => mem_N2((i*2)+1),
247                         Q1 => mem_Q2 (i*2),
248                         Q2 => mem_Q2 ((i*2)+1),
249                         Clock => Clock,
250                         Reset => Reset,
251                         Salida_N => mem_N3 (i),
252                         Salida_Q => mem_Q3 (i));
253     end generate;
254
255     comp4_T:
256     for i in 0 to ((M/16)-1) generate
257         comp3: comparer
258             Generic map (   Q_bits => Q_bits,
259                             b_mult => b_mult)
260             Port map (  N1 => mem_N3(i*2),
261                         N2 => mem_N3((i*2)+1),
262                         Q1 => mem_Q3 (i*2),
263                         Q2 => mem_Q3 ((i*2)+1),
264                         Clock => Clock,
265                         Reset => Reset,
266                         Salida_N => mem_N4 (i),
267                         Salida_Q => mem_Q4 (i));
268     end generate;
269
270     comp5_T:
271     for i in 0 to ((M/32)-1) generate
272         comp3: comparer
273             Generic map (   Q_bits => Q_bits,
274                             b_mult => b_mult)
275             Port map (  N1 => mem_N4(i*2),
276                         N2 => mem_N4((i*2)+1),
277                         Q1 => mem_Q4 (i*2),
278                         Q2 => mem_Q4 ((i*2)+1),
279                         Clock => Clock,
```

```
280                     Reset => Reset,
281                     Salida_N => mem_N5 (i),
282                     Salida_Q => mem_Q5 (i));
283 end generate;
284
285 comp6: comparer
286     Generic map (   Q_bits => Q_bits,
287                     b_mult => b_mult)
288     Port map (  N1 => mem_N5(0),
289                 N2 => mem_N5(1),
290                 Q1 => mem_Q5 (0),
291                 Q2 => mem_Q5 (1),
292                 Clock => Clock,
293                 Reset => Reset,
294                 Salida_N => Salida_N,
295                 Salida_Q => Salida_Q );
296
297 -- Una vez obtenida la neurona ganadora (neurona con la distancia más pequeña a la
    muestra de entrada),
298 -- se da como resultado, en binario, la agrupación a la que pertenece
299
300 Clase_c <= std_logic_vector( to_unsigned (mem_cluster
    (to_INTEGER(unsigned(Salida_Q))), Clase_c'length) );
301
302 -- Para conocer cuando la salida de la arquitectura es válida, se implementa un
    contador que da como resultado
303 -- el flanco de reloj para el cual la salida Clase_c es válida.
304
305 process (Clock, Reset)
306  begin
307      if reset = '1' then
308          cnt <= (others => '0');
309          Enable_salida <= '0';
310      elsif (Clock'event and Clock='1') then
311          if Enable='1' then
312              if cnt <= cycles then
313                  cnt <= cnt + 1;
314              else
315                  Enable_salida <= '1';
316              end if;
317          end if;
318      end if;
319 end process;
320
321 end architecture_main;
```

2.7. Main drivers

Arquitectura del circuito completo para la clasificación de los conductores según el consumo de combustible.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Mikel Díaz
3   -- Create Date: 10/02/2020
4   -- Module Name: main_drivers - architecture_main
5   -- Project Name: Parallel_SOM
6   -- Target Devices: Nexys4 DDR Rev. C
7   -- Description: Arquitectura del circuito completo.
8   ----------------------------------------------------------------------------------
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use IEEE.NUMERIC_STD.ALL;
13
14  entity main_drivers is
15      Generic (N: INTEGER := 4;  -- número de características (atributos) de los datos
    de entrada (X)
16               M: INTEGER := 36; -- número total de neuronas
17               Q_bits: INTEGER := 6; -- número de bits necesarios para enumerar todas
    las neuronas
18               b_in: INTEGER := 9; -- número de bits de los datos de entrada
19               b_mult: INTEGER := 14; -- número de bits después de multiplicar
20               cycles: POSITIVE := 9); -- número de ciclos de reloj hasta obtener
    salida Clase_c válida
21      Port (  X : in STD_LOGIC_VECTOR  ((b_in*N)-1 downto 0); -- muestra de entrada X
22              Clock, Reset, Enable : IN STD_LOGIC; -- Puerto de reloj y reset
23              Enable_salida : OUT STD_LOGIC; -- Salida válida
24              Clase_c : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)); -- Puerto de salida
25  end main_drivers;
26
27   architecture architecture_main of main_drivers is
28
29  type tabla_X_act is array (0 to N-1) of std_logic_vector (b_in-1 downto 0);
30  signal mem_X_act: tabla_X_act; -- Guarda todas las entradas X actuales del registro
31
32  type tabla_pesos_1 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0); --
    Valores de los pesos W para el atributo 1
33  signal mem_pesos_1: tabla_pesos_1 := ( "010101011", "010100110", "010100000",
    "010011100", "010010111", "010001000",
34                                         "010100100", "010100010", "010011110",
    "010011010", "010010001", "010000000",
35                                         "010011010", "010011111", "010100000",
    "010011100", "010010001", "010000101",
36                                         "010010111", "010011001", "010011010",
    "010000110", "010000001", "001110010",
37                                         "010010010", "010010101", "010001110",
    "010001001", "001111110", "001111011",
38                                         "010001111", "010010001", "010001001",
    "010000100", "010000100", "010000100" );
39  type tabla_pesos_2 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0); --
    Valores de los pesos W para el atributo 2
40  signal mem_pesos_2: tabla_pesos_2 := ( "001000110", "001000110", "001001001",
    "001001000", "001000111", "001001000",
41                                         "001000010", "001000100", "001000011",
    "001000001", "001001001", "001001010",
42                                         "000110111", "000111111", "000111100",
    "000111100", "000111111", "001000100",
43                                         "000110100", "000111000", "000111011",
    "000111110", "000111111", "001000001",
44                                         "000110100", "000110001", "000110101",
    "000111001", "000111111", "000111111",
```

```
45                                          "000110111", "000110101", "000110011",
    "000111101", "000111101", "000111101" );
46    type tabla_pesos_3 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0);  --
    Valores de los pesos W para el atributo 3
47    signal mem_pesos_3: tabla_pesos_3 := ( "001001010", "001001000", "001001001",
    "001001001", "001001010", "001001110",
48                                          "001000100", "001000011", "001000111",
    "001001100", "001001000", "001001111",
49                                          "000111001", "001000001", "001000011",
    "001000100", "001001101", "001010001",
50                                          "000110100", "000111010", "000111110",
    "001001001", "001001110", "001011110",
51                                          "000101101", "000101110", "000110110",
    "000111000", "001000101", "001001100",
52                                          "000101101", "000110000", "000111001",
    "000111001", "000111001", "000111001" );
53
54    type tabla_pesos_4 is array (0 to M-1) of std_logic_vector (b_in-1 downto 0);  --
    Valores de los pesos W para el atributo 4
55    signal mem_pesos_4: tabla_pesos_4 := ( "000001000", "000001000", "000000110",
    "000000110", "000000111", "000000110",
56                                          "000001000", "000001000", "000000111",
    "000000111", "000000110", "000000110",
57                                          "000000110", "000000110", "000000110",
    "000000111", "000001011", "000000101",
58                                          "000000101", "000000101", "000000101",
    "000001111", "000010001", "000010110",
59                                          "000000011", "000000100", "000000100",
    "000001010", "000010010", "000010011",
60                                          "000000010", "000000011", "000000101",
    "000010000", "000010000", "000010000" );
61
62    type tabla_cluster is array (0 to M-1) of integer; -- Agrupación a la que pertenece
    cada neurona
63    signal mem_cluster: tabla_cluster := ( 1, 1, 1, 1, 1, 2,
64                                           1, 1, 1, 1, 2, 2,
65                                           3, 1, 1, 1, 2, 2,
66                                           3, 3, 1, 2, 2, 2,
67                                           3, 3, 3, 3, 2, 2,
68                                           3, 3, 3, 2, 2, 2 );
69
70    type tabla_salidas is array (0 to M-1) of std_logic_vector (b_mult-1 downto 0);
71    signal mem_salidas: tabla_salidas; -- Guarda las salidas de las neuronas
72
73    -- Salidas comparadores (por parejas y en paralelo)
74    ---- N ---> Guarda la distancia más pequeña
75
76    type tabla_N1 is array (0 to 17) of std_logic_vector (b_mult-1 downto 0);
77    signal mem_N1: tabla_N1;
78
79    type tabla_N2 is array (0 to 8) of std_logic_vector (b_mult-1 downto 0);
80    signal mem_N2: tabla_N2;
81
82    type tabla_N3 is array (0 to 4) of std_logic_vector (b_mult-1 downto 0);
83    signal mem_N3: tabla_N3;
84
85    type tabla_N4 is array (0 to 1) of std_logic_vector (b_mult-1 downto 0);
86    signal mem_N4: tabla_N4;
87
88    signal mem_N5 : STD_LOGIC_VECTOR  (b_mult-1 downto 0) := (others =>'0');
89
90    signal Salida_N : STD_LOGIC_VECTOR  (b_mult-1 downto 0) := (others =>'0'); --
    neurona con la distancia más pequeña (Salida <= distancia)
91
92    ---- Q ---> Guarda el valor de la neuron (se enumeran las neuronas mediante mem_Q)
93
94    type tabla_Q is array (0 to M-1) of std_logic_vector (Q_bits-1 downto 0); --
    Enumeración de las neuronas
```

```
 95    signal mem_Q: tabla_Q := ( "000000", "000001", "000010", "000011", "000100",
       "000101",
 96                               "000110", "000111", "001000", "001001", "001010",
       "001011",
 97                               "001100", "001101", "001110", "001111", "010000",
       "010001",
 98                               "010010", "010011", "010100", "010101", "010110",
       "010111",
 99                               "011000", "011001", "011010", "011011", "011100",
       "011101",
100                               "011110", "011111", "100000", "100001", "100010",
       "100011" );
101
102   type tabla_Q1 is array (0 to 17) of std_logic_vector (Q_bits-1 downto 0);
103   signal mem_Q1: tabla_Q1;
104
105   type tabla_Q2 is array (0 to 8) of std_logic_vector (Q_bits-1 downto 0);
106   signal mem_Q2: tabla_Q2;
107
108   type tabla_Q3 is array (0 to 4) of std_logic_vector (Q_bits-1 downto 0);
109   signal mem_Q3: tabla_Q3;
110
111   type tabla_Q4 is array (0 to 1) of std_logic_vector (Q_bits-1 downto 0);
112   signal mem_Q4: tabla_Q4;
113
114   signal mem_Q5 : STD_LOGIC_VECTOR  (Q_bits-1 downto 0) := (others =>'0');
115
116   signal Salida_Q : STD_LOGIC_VECTOR  (Q_bits-1 downto 0) := (others =>'0'); --
      neurona con la distancia más pequeña (Salida <= número de neuron)
117
118   signal cnt : unsigned(cycles-1 DOWNTO 0); -- contador
119
120   -- Declaración de componentes
121
122   component neuron_4 -- Distancia entre la muestra de entrada y el peso de la neurona
123       Generic (N: INTEGER;
124                b_in: INTEGER;
125                b_mult: INTEGER);
126       Port ( X0_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
127              X1_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
128              X2_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
129              X3_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
130
131              W0_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
132              W1_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
133              W2_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
134              W3_n : in STD_LOGIC_VECTOR  (b_in-1 downto 0);
135
136              Clock, Reset : in STD_LOGIC;
137
138              Salida : out STD_LOGIC_VECTOR (b_mult-1 downto 0) := (others =>'0'));
139   end component;
140
141   component comparer -- Comparación de distancias entre dos neuronas (Resultado <=
      distancia más pequeña)
142       Generic ( Q_bits: INTEGER;
143                 b_mult: INTEGER);
144       Port ( N1 : in STD_LOGIC_VECTOR (b_mult-1 downto 0);
145              N2 : in STD_LOGIC_VECTOR (b_mult-1 downto 0);
146              Q1 : in STD_LOGIC_VECTOR (Q_bits-1 downto 0);
147              Q2 : in STD_LOGIC_VECTOR (Q_bits-1 downto 0);
148              Clock, Reset : in STD_LOGIC;
149              Salida_N : out STD_LOGIC_VECTOR (b_mult-1 downto 0);
150              Salida_Q : out STD_LOGIC_VECTOR (Q_bits-1 downto 0));
151   end component;
152
153   component input_register -- Registro de las muestras de entrada X
154       Generic ( b_in: INTEGER);
```

```
155         Port (  X : in STD_LOGIC_VECTOR (b_in-1 downto 0);
156                 Clock, Reset : in STD_LOGIC;
157                 X_act : out STD_LOGIC_VECTOR (b_in-1 downto 0));
158     end component;
159 
160     begin
161 
162     -- Declaración de los registros de entrada
163     registros_entradas:
164     for i in 0 to N-1 generate
165         reg_X: input_register
166             Generic map (   b_in => b_in)
167             Port map (    X      => X((b_in*(i+1))-1 downto i*b_in),
168                           Clock => Clock,
169                           Reset => Reset,
170                           X_act => mem_X_act(i));
171     end generate;
172 
173     -- Cálculo de las distancia entre las neuronas y la muestra de entrada
174     dist_pesos:
175     for i in 0 to M-1 generate
176         calc_neurona : neuron_4
177                 Generic map (   N => N,
178                                 b_in => b_in,
179                                 b_mult => b_mult)
180                 Port map (  X0_n => mem_X_act(0),
181                             X1_n => mem_X_act(1),
182                             X2_n => mem_X_act(2),
183                             X3_n => mem_X_act(3),
184                             W0_n => mem_pesos_1(i),
185                             W1_n => mem_pesos_2(i),
186                             W2_n => mem_pesos_3(i),
187                             W3_n => mem_pesos_4(i),
188                             Clock => Clock,
189                             Reset => Reset,
190                             Salida => mem_salidas(i));
191 
192     end generate;
193 
194     -- Comparación de las distancias (por parejas y en paralelo)
195     comp1_T:
196     for i in 0 to 17 generate
197         comp1: comparer
198             Generic map (   Q_bits => Q_bits,
199                             b_mult => b_mult)
200             Port map (  N1 => mem_salidas(i*2),
201                         N2 => mem_salidas((i*2)+1),
202                         Q1 => mem_Q (i*2),
203                         Q2 => mem_Q ((i*2)+1),
204                         Clock => Clock,
205                         Reset => Reset,
206                         Salida_N => mem_N1 (i),
207                         Salida_Q => mem_Q1 (i));
208     end generate;
209 
210     comp2_T:
211     for i in 0 to 8 generate
212         comp2: comparer
213             Generic map (   Q_bits => Q_bits,
214                             b_mult => b_mult)
215             Port map (  N1 => mem_N1(i*2),
216                         N2 => mem_N1((i*2)+1),
217                         Q1 => mem_Q1 (i*2),
218                         Q2 => mem_Q1 ((i*2)+1),
219                         Clock => Clock,
220                         Reset => Reset,
221                         Salida_N => mem_N2 (i),
222                         Salida_Q => mem_Q2 (i));
```

```
223   end generate;
224
225   comp3_T:
226   for i in 0 to 3 generate
227       comp3: comparer
228           Generic map (   Q_bits => Q_bits,
229                           b_mult => b_mult)
230           Port map (  N1 => mem_N2(i*2),
231                       N2 => mem_N2((i*2)+1),
232                       Q1 => mem_Q2 (i*2),
233                       Q2 => mem_Q2 ((i*2)+1),
234                       Clock => Clock,
235                       Reset => Reset,
236                       Salida_N => mem_N3 (i),
237                       Salida_Q => mem_Q3 (i));
238   end generate;
239
240   comp4_T:
241   for i in 0 to 1 generate
242       comp3: comparer
243           Generic map (   Q_bits => Q_bits,
244                           b_mult => b_mult)
245           Port map (  N1 => mem_N3(i*2),
246                       N2 => mem_N3((i*2)+1),
247                       Q1 => mem_Q3 (i*2),
248                       Q2 => mem_Q3 ((i*2)+1),
249                       Clock => Clock,
250                       Reset => Reset,
251                       Salida_N => mem_N4 (i),
252                       Salida_Q => mem_Q4 (i));
253   end generate;
254
255   comp5: comparer
256       Generic map (   Q_bits => Q_bits,
257                       b_mult => b_mult)
258       Port map (  N1 => mem_N4(0),
259                   N2 => mem_N4(1),
260                   Q1 => mem_Q4 (0),
261                   Q2 => mem_Q4 (1),
262                   Clock => Clock,
263                   Reset => Reset,
264                   Salida_N => mem_N5,
265                   Salida_Q => mem_Q5 );
266
267   comp6: comparer
268       Generic map (   Q_bits => Q_bits,
269                       b_mult => b_mult)
270       Port map (  N1 => mem_N5,
271                   N2 => mem_N2(8),
272                   Q1 => mem_Q5,
273                   Q2 => mem_Q2 (8),
274                   Clock => Clock,
275                   Reset => Reset,
276                   Salida_N => Salida_N,
277                   Salida_Q => Salida_Q );
278
279   -- Una vez obtenida la neurona ganadora (neurona con la distancia más pequeña a la
      muestra de entrada),
280   -- se da como resultado, en binario, la agrupación a la que pertenece
281
282   Clase_c <= std_logic_vector( to_unsigned (mem_cluster
      (to_INTEGER(unsigned(Salida_Q))), Clase_c'length) );
283
284   -- Para conocer cuando la salida de la arquitectura es válida, se implementa un
      contador que da como resultado
285   -- el flanco de reloj para el cual la salida Clase_c es válida.
286
287   process (Clock, Reset)
```

```
288     begin
289         if reset = '1' then
290             cnt <= (others => '0');
291             Enable_salida <= '0';
292         elsif (Clock'event and Clock='1') then
293             if Enable='1' then
294                 if cnt <= cycles then
295                     cnt <= cnt + 1;
296                 else
297                     Enable_salida <= '1';
298                 end if;
299             end if;
300         end if;
301     end process;
302 
303     end architecture_main;
```