Paola Andrea Campiño - Jose Gabriel Ruiz Bayer

BIBLIOTECA DE COMPONENTES

Universidad de los Andes
Facultad de Ingeniería

Universidad de los Andes | Vigilada Mineducación
Reconocimiento como Universidad: Decreto 1297 del 30 de mayo de 1964.
Reconocimiento personería jurídica: Resolución 28 del 23 de febrero de 19

PUBLISHED BY UNIVERSIDAD DE LOS ANDES - COLOMBIA

September 2018

Contenido

1. Diseño de componentes.

HENRY FORD

Si le hubiera preguntado a la gente lo que ellos querían, ellos habrían dicho caballos más rápidos.

1.1 Neurona Artificial

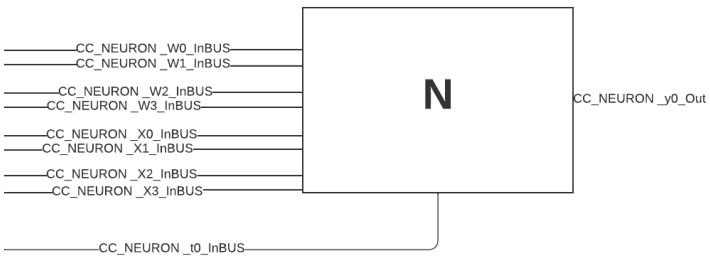
A continuación se presenta el componente combinacional **Neurona Artificial** mostrando su descripción con ecuación lógica.

1.1.1 Descripción del componente

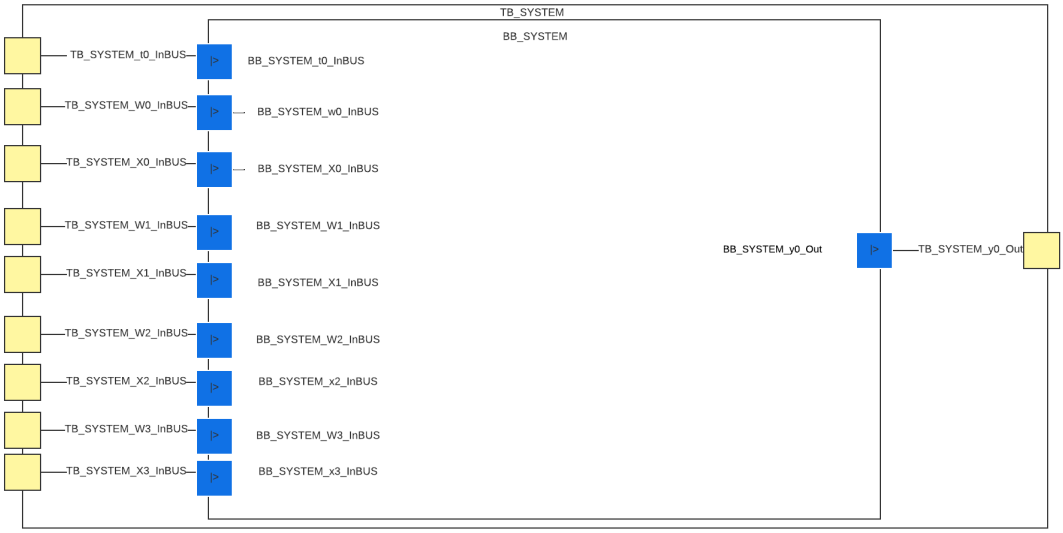
A. Descripción de componente:
Una neurona artificial es uno de los elementos más simples que conforman las redes neuronales. Una neurona artificial puede tener múltiples entradas y una sola salida. Cada entrada es un número de 8 bits, y se asocia con un peso que indica su importancia en la salida final. En el caso de esta neurona, cuenta con 9 entradas numéricas de 8 bits, 4 de las cuales son números y las otras cuatro son los pesos asociados a dichos números. Además, se tiene un número θ que se utiliza al final de la neurona para definir la salida. Cuando la suma de la multiplicación del peso asociado a un número con el número es menor a θ, la salida es 0; en cambio, cuando esta suma es mayor o igual a θ, la salida es 1. Es importante destacar que las neuronas artificiales son la base de las redes neuronales, y su complejidad aumenta a medida que se agregan más neuronas y conexiones entre ellas.
B. Especificaciones y restricciones:

- Entradas: 9 entradas de 8 bits
- $0 \leq \theta \leq 255$
- $\Sigma_{i=0}^{i=3}(peso_i * numero_i) \leq 255$
- *Salida: 1 salida de 8 bits con valor decimal de 1 o 0*

1.1.2 Símbolo



1.1.3 Diagrama caja negra



1.1.4 Funcionalidad

Ecuación característica

$$1 \equiv CC\_SYSTEM\_y0\_Out <= \Sigma_{i=0}^{i=3}(CC\_wi\_In * CC\_NEURON\_xi\_In) \geq CC\_NEURON\_t0\_In$$

or

$$0 \equiv CC\_SYSTEM\_y0\_Out <= \Sigma_{i=0}^{i=3}(CC\_wi\_In * CC\_NEURON\_xi\_In) < CC\_NEURON\_t0\_In$$

Macro algoritmo

Algorithm 1: Neurona artificial

Data: $CC\_NEURON\_W0_{I}nBUS$
,$CC\_NEURON\_W1\_InBUS$
,$CC\_NEURON\_W2\_InBUS$
,$CC\_NEURON\_W3\_InBUS$
,$CC\_NEURON\_X0\_InBUS$
,$CC\_NEURON\_X1\_InBUS$
,$CC\_NEURON\_X2\_InBUS$
,$CC\_NEURON\_X3\_InBUS, CC\_NEURON\_t0\_InBUS$)
Result: $CC\_NEURON\_y0\_Out$ (0, 1)

$$1 \equiv CC\_SYSTEM\_y0\_Out <= \Sigma_{i=0}^{i=3}(CC\_wi\_In * CC\_NEURON\_xi\_In) \geq CC\_NEURON\_t0\_In$$

or

$$0 \equiv CC\_SYSTEM\_y0\_Out <= \Sigma_{i=0}^{i=3}(CC\_wi\_In * CC\_NEURON\_xi\_In) < CC\_NEURON\_t0\_In$$

1.1.5 HDL: Caja negra

```
module BB_SYSTEM (
////////////// OUTPUTS //////////
BB_SYSTEM_y0_Out,
////////////// INPUTS //////////
BB_SYSTEM_t0_InBUS,
BB_SYSTEM_x0_InBUS,
BB_SYSTEM_w0_InBUS,
BB_SYSTEM_x1_InBUS,
BB_SYSTEM_w1_InBUS,
BB_SYSTEM_x2_InBUS,
BB_SYSTEM_w2_InBUS,
BB_SYSTEM_x3_InBUS,
BB_SYSTEM_w3_InBUS
);
//=======================================================
//  PARAMETER declarations
//=======================================================
parameter NUMBER_DATAWIDTH = 8;// tamaño  en terminos de bits
//=======================================================
//  PORT declarations
//=======================================================
output BB_SYSTEM_y0_Out;//salida
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_t0_InBUS;//theta
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_x0_InBUS; // numero 0
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_w0_InBUS;// peso numero 0
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_x1_InBUS;// numero 1
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_w1_InBUS;// peso numero 1
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_x2_InBUS;// numero 2
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_w2_InBUS;// peso numero 2
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_x3_InBUS;// numero 3
input [NUMBER_DATAWIDTH-1:0] BB_SYSTEM_w3_InBUS;// peso numero 3
//=======================================================
//  REG/WIRE declarations
```

```
//=====================================================

//=====================================================
//  Structural coding
//=====================================================
CC_NEURON #(.NUMBER_DATAWIDTH(NUMBER_DATAWIDTH)) CC_NEURON_u0 (
// port map - connection between master ports and signals/registers
.CC_NEURON_y0_Out(BB_SYSTEM_y0_Out),
.CC_NEURON_t0_InBUS(BB_SYSTEM_t0_InBUS),
.CC_NEURON_x0_InBUS(BB_SYSTEM_x0_InBUS),
.CC_NEURON_w0_InBUS(BB_SYSTEM_w0_InBUS),
.CC_NEURON_x1_InBUS(BB_SYSTEM_x1_InBUS),
.CC_NEURON_w1_InBUS(BB_SYSTEM_w1_InBUS),
.CC_NEURON_x2_InBUS(BB_SYSTEM_x2_InBUS),
.CC_NEURON_w2_InBUS(BB_SYSTEM_w2_InBUS),
.CC_NEURON_x3_InBUS(BB_SYSTEM_x3_InBUS),
.CC_NEURON_w3_InBUS(BB_SYSTEM_w3_InBUS)
);
endmodule
```

1.1.6 Definición vectores de prueba

En este caso como los exiten muchos casos posibles, se ha seleccionado los siguientes casos para verificar la validez del programa:

#Prueba	t0	x0	w0	x1	w1	x2	w2	x3	w3	y0_out
0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0
2	40	1	1	2	2	3	3	4	4	0
3	128	0	0	0	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0	0	1
5	2	1	1	2	2	3	3	4	4	1
6	255	1	1	2	2	3	3	4	4	0
7	1	1	0	100	2	100	0	55	1	1

Se consideró 6 casos para realizar las pruebas

- El primero para garantizar que cuando todos los valores son iguales el resultado debe ser igual a 1, ya que t0 no es mayor a la sumatoria del los nuermos multiplicados por el peso.
- El segundo para garantizar que cuando t0 es mayor el resutado es igual a 0.
- El tercero para probar que t0 cuando t0 es mayor el resultado es igual a 0 solo que con nuemeros más variados.
- El quinto para confirmar que cuando dos numeros son iguales el resutlado es igual a 0
- El sexto ciendo caso contrario a la prueba 2 que debería ser igual a 1.
- Las pruebas 7ma y 8va tienen como fin comprobar que en efecto los numeros más grandes los reconoce la neurona que son con 255.

1.1.7 HDL: Vectores de prueba

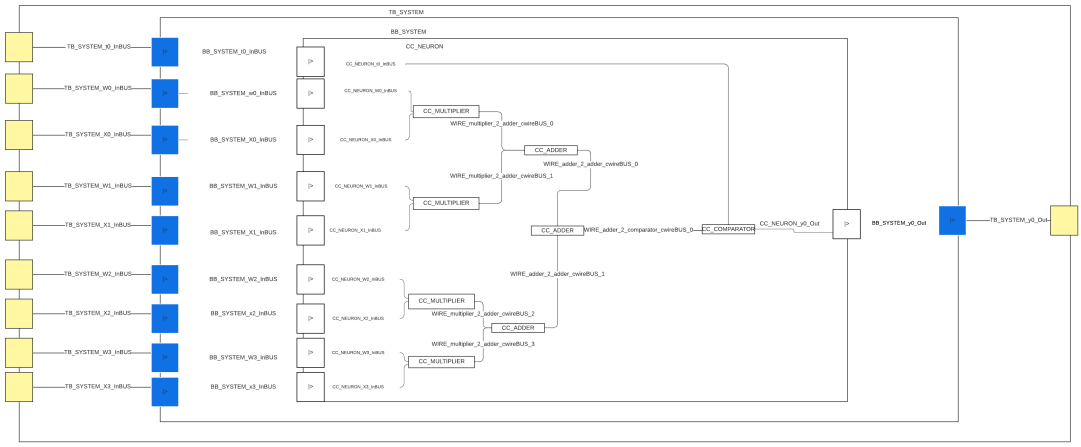
```
    module TB_SYSTEM();
// constants
// ========================================
// Parameter ( may differ for physical synthesis)
// ========================================
// general purpose registers
```

```
reg eachvec;
parameter NUMBER_DATAWIDTH = 8;
integer ii=0;

// test vector input registers
// ========================================
// INTERNAL WIRE AND REG DECLARATIONS
// ========================================
// wires (OUTPUTS)
wire TB_SYSTEM_y0_Out;
// reg (INPUTS)
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_t0_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_x0_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_w0_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_x1_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_w1_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_x2_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_w2_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_x3_InBUS;
reg [NUMBER_DATAWIDTH-1:0] TB_SYSTEM_w3_InBUS;

// assign statements (if any)
BB_SYSTEM BB_SYSTEM_u0 (
// port map - connection between master ports and signals/registers
.BB_SYSTEM_y0_Out(TB_SYSTEM_y0_Out),
.BB_SYSTEM_t0_InBUS(TB_SYSTEM_t0_InBUS),
.BB_SYSTEM_x0_InBUS(TB_SYSTEM_x0_InBUS),
.BB_SYSTEM_w0_InBUS(TB_SYSTEM_w0_InBUS),
.BB_SYSTEM_x1_InBUS(TB_SYSTEM_x1_InBUS),
.BB_SYSTEM_w1_InBUS(TB_SYSTEM_w1_InBUS),
.BB_SYSTEM_x2_InBUS(TB_SYSTEM_x2_InBUS),
.BB_SYSTEM_w2_InBUS(TB_SYSTEM_w2_InBUS),
.BB_SYSTEM_x3_InBUS(TB_SYSTEM_x3_InBUS),
.BB_SYSTEM_w3_InBUS(TB_SYSTEM_w3_InBUS)
);
initial
begin
// code that executes only once
// insert code here --> begin

// --> end
$display("Running testbench");
end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)

begin
// code executes for every event on sensitivity list
// insert code here --> begin
#10000 $monitor( "%1d:", ii, "Tiempo: %2d." , $time, "TB_SYSTEM_x0_InBUS = %b",
 TB_SYSTEM_x0_InBUS, "  |  TB_SYSTEM_w0_InBUS= %b", TB_SYSTEM_w0_InBUS, "  |  TB_SYSTEM_y0_Out =
 TB_SYSTEM_y0_Out);


#0  TB_SYSTEM_t0_InBUS <= 0; TB_SYSTEM_x0_InBUS <= 0; TB_SYSTEM_w0_InBUS <= 0; TB_SYSTEM_x1_InBUS
```

```
 TB_SYSTEM_x2_InBUS <= 0; TB_SYSTEM_w2_InBUS <= 0;
 TB_SYSTEM_x3_InBUS <= 0; TB_SYSTEM_w3_InBUS <= 0; // CHECK 1

#20000  TB_SYSTEM_t0_InBUS <= 1; TB_SYSTEM_x0_InBUS <= 0; TB_SYSTEM_w0_InBUS <= 0;
 TB_SYSTEM_x1_InBUS <= 0; TB_SYSTEM_w1_InBUS <= 0;
 TB_SYSTEM_x2_InBUS <= 0; TB_SYSTEM_w2_InBUS <= 0;
 TB_SYSTEM_x3_InBUS <= 0; TB_SYSTEM_w3_InBUS <= 0; // CHECK 0

#20000 TB_SYSTEM_t0_InBUS <= 40; TB_SYSTEM_x0_InBUS <= 1; TB_SYSTEM_w0_InBUS <= 1;
 TB_SYSTEM_x1_InBUS <= 2; TB_SYSTEM_w1_InBUS <= 2;
 TB_SYSTEM_x2_InBUS <= 3; TB_SYSTEM_w2_InBUS <= 3;
 TB_SYSTEM_x3_InBUS <= 4; TB_SYSTEM_w3_InBUS <= 4; // CHECK 0

#20000 TB_SYSTEM_t0_InBUS <= 128; TB_SYSTEM_x0_InBUS <= 0; TB_SYSTEM_w0_InBUS <= 0;
 TB_SYSTEM_x1_InBUS <= 0; TB_SYSTEM_w1_InBUS <= 0;
 TB_SYSTEM_x2_InBUS <= 0; TB_SYSTEM_w2_InBUS <= 0;
 TB_SYSTEM_x3_InBUS <= 0; TB_SYSTEM_w3_InBUS <= 0; // CHECK 0

#20000 TB_SYSTEM_t0_InBUS <= 1; TB_SYSTEM_x0_InBUS <= 1; TB_SYSTEM_w0_InBUS <= 1;
 TB_SYSTEM_x1_InBUS <= 0; TB_SYSTEM_w1_InBUS <= 0;
 TB_SYSTEM_x2_InBUS <= 0; TB_SYSTEM_w2_InBUS <= 0;
 TB_SYSTEM_x3_InBUS <= 0; TB_SYSTEM_w3_InBUS <= 0; // CHECK 1

#20000 TB_SYSTEM_t0_InBUS <= 2; TB_SYSTEM_x0_InBUS <= 1; TB_SYSTEM_w0_InBUS <= 1;
 TB_SYSTEM_x1_InBUS <= 2; TB_SYSTEM_w1_InBUS <= 2;
 TB_SYSTEM_x2_InBUS <= 3; TB_SYSTEM_w2_InBUS <= 3;
 TB_SYSTEM_x3_InBUS <= 4; TB_SYSTEM_w3_InBUS <= 4; // CHECK 1

#20000 TB_SYSTEM_t0_InBUS <= 255; TB_SYSTEM_x0_InBUS <= 1; TB_SYSTEM_w0_InBUS <= 1;
 TB_SYSTEM_x1_InBUS <= 2; TB_SYSTEM_w1_InBUS <= 2;
 TB_SYSTEM_x2_InBUS <= 3; TB_SYSTEM_w2_InBUS <= 3;
 TB_SYSTEM_x3_InBUS <= 4; TB_SYSTEM_w3_InBUS <= 4; // CHECK 1

#20000 TB_SYSTEM_t0_InBUS <= 1; TB_SYSTEM_x0_InBUS <= 1; TB_SYSTEM_w0_InBUS <= 0;
 TB_SYSTEM_x1_InBUS <= 100; TB_SYSTEM_w1_InBUS <= 2;
 TB_SYSTEM_x2_InBUS <= 100; TB_SYSTEM_w2_InBUS <= 0;
 TB_SYSTEM_x3_InBUS <= 55; TB_SYSTEM_w3_InBUS <= 1; // CHECK 1


@eachvec;
$finish;
// --> end
end
endmodule
```

1.1.8 Diagrama caja blanca

En este caso el diagrama corresponde solamente a la neurona.



En el diagrama se puede ver los componentes necesarios para completar una neurona, la multiplicación entre numero y peso, La sumatoria resultado de estas multiplicaciones para así llegar finalizar con una comparación con con t0 y evidenciar una respuesta (1 o 0). El diagrama muestra en detalle los cables con respectivos nombres para llegar a este componente.

1.1.9 HDL: Caja blanca

```
module CC_NEURON (
////////////// OUTPUTS //////////
CC_NEURON_y0_Out,
////////////// INPUTS //////////
CC_NEURON_t0_InBUS,
CC_NEURON_x0_InBUS,
CC_NEURON_w0_InBUS,
CC_NEURON_x1_InBUS,
CC_NEURON_w1_InBUS,
CC_NEURON_x2_InBUS,
CC_NEURON_w2_InBUS,
CC_NEURON_x3_InBUS,
CC_NEURON_w3_InBUS
);
//=========================================================
//  PARAMETER declarations
//=========================================================
parameter NUMBER_DATAWIDTH = 8;
//=========================================================
//  PORT declarations
//=========================================================
output  CC_NEURON_y0_Out;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_t0_InBUS;//coeficiente o
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_x0_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_w0_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_x1_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_w1_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_x2_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_w2_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_x3_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_NEURON_w3_InBUS;

//=========================================================
```

```
//  REG/WIRE declarations
//=====================================================
wire [NUMBER_DATAWIDTH-1:0] WIRE_multiplier_2_adder_cwireBUS_0;
wire [NUMBER_DATAWIDTH-1:0] WIRE_multiplier_2_adder_cwireBUS_1;
wire [NUMBER_DATAWIDTH-1:0] WIRE_multiplier_2_adder_cwireBUS_2;
wire [NUMBER_DATAWIDTH-1:0] WIRE_multiplier_2_adder_cwireBUS_3;

wire [NUMBER_DATAWIDTH-1:0] WIRE_adder_2_adder_cwireBUS_0;
wire [NUMBER_DATAWIDTH-1:0] WIRE_adder_2_adder_cwireBUS_1;

wire [NUMBER_DATAWIDTH-1:0] WIRE_adder_2_comparator_cwireBUS_0;

//=====================================================
//  Structural coding
//=====================================================
//STRUCTURAL
CC_MULTIPLIER CC_MULTIPLIER_u0 (// instanciación multiplier x0 y w0
// port map - connection between master ports and signals/registers
.CC_MULTIPLIER_product_Out(WIRE_multiplier_2_adder_cwireBUS_0),
.CC_MULTIPLIER_m0_InBUS(CC_NEURON_x0_InBUS),
.CC_MULTIPLIER_m1_InBUS(CC_NEURON_w0_InBUS)
);

CC_MULTIPLIER CC_MULTIPLIER_u1 (// instanciación multiplier x1 y w1
// port map - connection between master ports and signals/registers
.CC_MULTIPLIER_product_Out(WIRE_multiplier_2_adder_cwireBUS_1),
.CC_MULTIPLIER_m0_InBUS(CC_NEURON_x1_InBUS),
.CC_MULTIPLIER_m1_InBUS(CC_NEURON_w1_InBUS)

);

CC_MULTIPLIER CC_MULTIPLIER_u2 (// instanciación multiplier x2 y w2
// port map - connection between master ports and signals/registers
.CC_MULTIPLIER_product_Out(WIRE_multiplier_2_adder_cwireBUS_2),
.CC_MULTIPLIER_m0_InBUS(CC_NEURON_x2_InBUS),
.CC_MULTIPLIER_m1_InBUS(CC_NEURON_w2_InBUS)


);

CC_MULTIPLIER CC_MULTIPLIER_u3 (//// instanciación multiplier  x3 y w3
// port map - connection between master ports and signals/registers
.CC_MULTIPLIER_product_Out(WIRE_multiplier_2_adder_cwireBUS_3),
.CC_MULTIPLIER_m0_InBUS(CC_NEURON_x3_InBUS),
.CC_MULTIPLIER_m1_InBUS(CC_NEURON_w3_InBUS)

);

CC_ADDER CC_ADDER_u0 (// sumatoria x0*w0 + x1*w1
// port map - connection between master ports and signals/registers
.CC_ADDER_sum_Out(WIRE_adder_2_adder_cwireBUS_0),
.CC_ADDER_a0_InBUS(WIRE_multiplier_2_adder_cwireBUS_0),
.CC_ADDER_a1_InBUS(WIRE_multiplier_2_adder_cwireBUS_1)
);

CC_ADDER CC_ADDER_u1 (// sumatoria x2*w2 + x3*w3
```

```
// port map - connection between master ports and signals/registers
.CC_ADDER_sum_Out(WIRE_adder_2_adder_cwireBUS_1),
.CC_ADDER_a0_InBUS(WIRE_multiplier_2_adder_cwireBUS_2),
.CC_ADDER_a1_InBUS(WIRE_multiplier_2_adder_cwireBUS_3)

);

CC_ADDER CC_ADDER_u2 (// sumatoria x0*w0 + x1*w1 + x2*w2 + x3*w3
// port map - connection between master ports and signals/registers
.CC_ADDER_sum_Out(WIRE_adder_2_comparator_cwireBUS_0),
.CC_ADDER_a0_InBUS(WIRE_adder_2_adder_cwireBUS_1),
.CC_ADDER_a1_InBUS(WIRE_adder_2_adder_cwireBUS_0)


);

CC_COMPARATOR CC_COMPARATOR_u2 (
// (x0*w0 + x1*w1 + x2*w2 + x3*w3)> t0 => 0 ||
//  (x0*w0 + x1*w1 + x2*w2 + x3*w3)<= t0 => 1

// port map - connection between master ports and signals/registers
.CC_COMPARATOR_result_Out (CC_NEURON_y0_Out),
.CC_COMPARATOR_c0_InBUS(CC_NEURON_t0_InBUS),
.CC_COMPARATOR_c1_InBUS(WIRE_adder_2_comparator_cwireBUS_0)


);


endmodule
```

1.1.10 HDL: bloques

Para los bloques ADDER u0, u1 y u2 sus codigos corresponden al siguente:

```
module CC_ADDER (
////////////// OUTPUTS //////////
CC_ADDER_sum_Out,
////////////// INPUTS //////////
CC_ADDER_a0_InBUS,
CC_ADDER_a1_InBUS
);
//=======================================================
//  PARAMETER declarations
//=======================================================
parameter NUMBER_DATAWIDTH = 8;
//=======================================================
//  PORT declarations
//=======================================================
output  [NUMBER_DATAWIDTH-1:0] CC_ADDER_sum_Out;
input  [NUMBER_DATAWIDTH-1:0] CC_ADDER_a0_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_ADDER_a1_InBUS;
//=======================================================
//  REG/WIRE declarations
//=======================================================

//=======================================================
```

```
//  Structural coding
//=======================================================


assign CC_ADDER_sum_Out= CC_ADDER_a0_InBUS + CC_ADDER_a1_InBUS;

endmodule
```

Para el bloque COMPARATOR su codigo corresponde al siguente:

```
module CC_COMPARATOR (
/////////////// OUTPUTS //////////
CC_COMPARATOR_result_Out,
/////////////// INPUTS //////////
CC_COMPARATOR_c0_InBUS,
CC_COMPARATOR_c1_InBUS
);
//=======================================================
//  PARAMETER declarations
//=======================================================
parameter NUMBER_DATAWIDTH = 8;
//=======================================================
//  PORT declarations
//=======================================================
output reg CC_COMPARATOR_result_Out;
input  [NUMBER_DATAWIDTH-1:0] CC_COMPARATOR_c0_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_COMPARATOR_c1_InBUS;
//=======================================================
//  REG/WIRE declarations
//=======================================================

//=======================================================
//  Structural coding
//=======================================================
always @(*) begin

if (CC_COMPARATOR_c0_InBUS>CC_COMPARATOR_c1_InBUS) begin
CC_COMPARATOR_result_Out = 0;
end else begin
CC_COMPARATOR_result_Out = 1;
end
end


endmodule
```

Para los bloques MULTIPLIER u0, u1 y u2 sus codigos corresponden al siguente:

```
module CC_MULTIPLIER (
/////////////// OUTPUTS //////////
CC_MULTIPLIER_product_Out,
/////////////// INPUTS //////////
CC_MULTIPLIER_m0_InBUS,
CC_MULTIPLIER_m1_InBUS
);
```

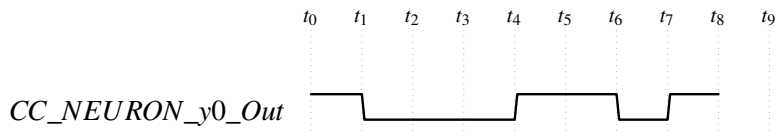
```
//=====================================================
//  PARAMETER declarations
//=====================================================
parameter NUMBER_DATAWIDTH = 8;
//=====================================================
//  PORT declarations
//=====================================================
output  [NUMBER_DATAWIDTH-1:0] CC_MULTIPLIER_product_Out;
input  [NUMBER_DATAWIDTH-1:0] CC_MULTIPLIER_m0_InBUS;
input  [NUMBER_DATAWIDTH-1:0] CC_MULTIPLIER_m1_InBUS;
//=====================================================
//  REG/WIRE declarations
//=====================================================

//=====================================================
//  Structural coding
//=====================================================

assign CC_MULTIPLIER_product_Out = CC_MULTIPLIER_m0_InBUS * CC_MULTIPLIER_m1_InBUS;


endmodule
```

1.1.11 Simulación temporal

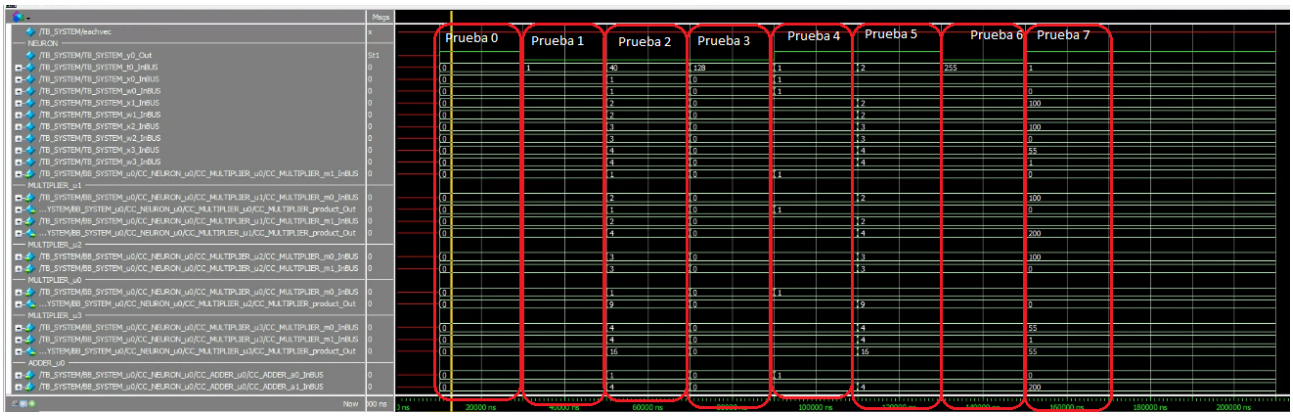
A continuación se presenta un diagrama de lo que se espera obtener del sistema. Cabe resaltar que el orden de las pruebas es igual como se muestra en la tabla con los vectores de prueba como se ve a continuaicón:

#Prueba	t0	x0	w0	x1	w1	x2	w2	x3	w3	y0_out
0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0
2	40	1	1	2	2	3	3	4	4	0
3	128	0	0	0	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0	0	1
5	2	1	1	2	2	3	3	4	4	1
6	255	1	1	2	2	3	3	4	4	0
7	1	1	0	100	2	100	0	55	1	1

Por lo que resultados de la simulación debería verse de la siguiente manera:

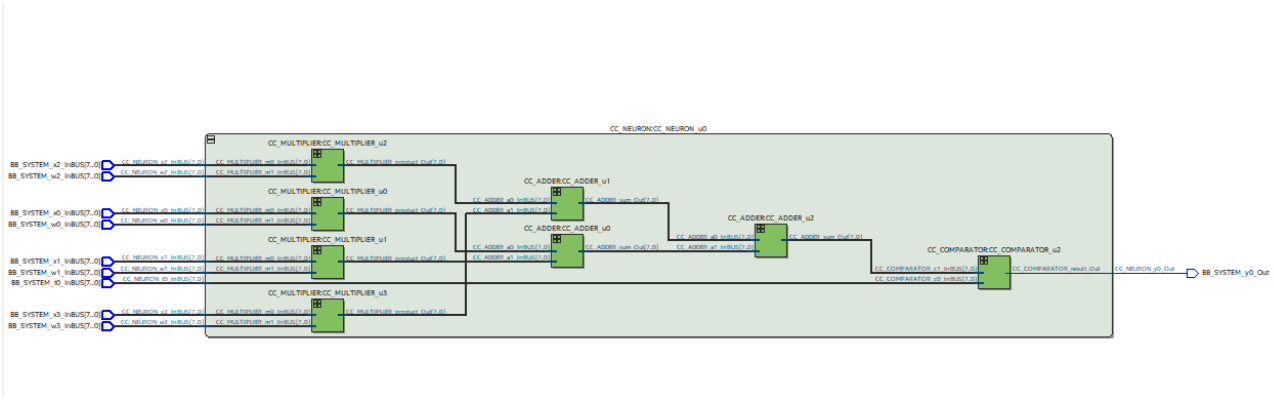


El siguiente diagrama de tiempos obtenido en la herramienta Quartus de Altera. Se puede ver los resultados de la simulación:



En base a los resultados anteiormente obtenidos se puede decir que los procedimientos efectuados son los esperados por lo tanto el funcionamiento de la neurona es correcta obteniendo 0 y 1 en los casos previstos y calculados de manera teorica.

1.1.12 Diagramas QUARTUS



Vista general del componente CC_Neuron

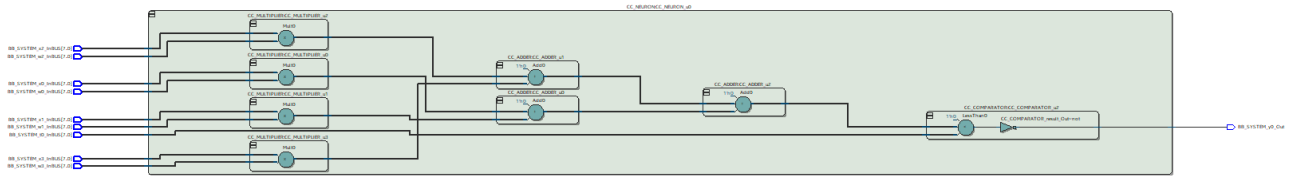
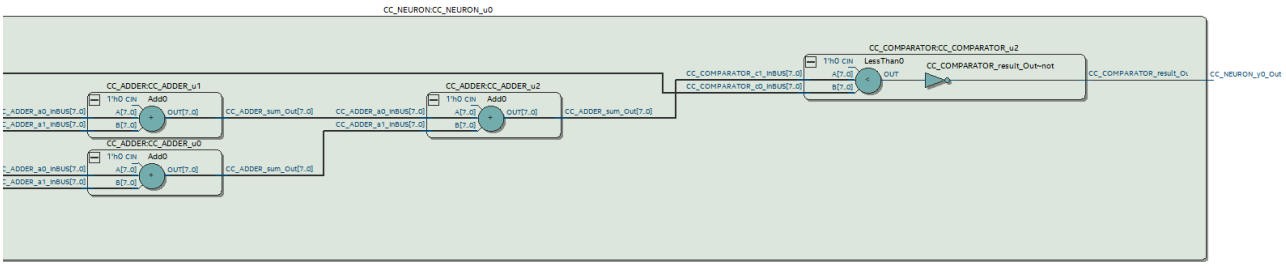


Imagen detallando de la caja blanca de CC_Neuron con cada uno de sus componentes



Vista cercana de los componentes adder y comparer de CC_Neuron

1.1.13 Resultados y lecciones aprendidas

Después de realizar la práctica utilizando el programa Quartus y la manipulacion de diversos codigos en Verilog para el desarrollo de una neurona artificial, se puede concluir que se ha logrado comprender los elementos básicos de las redes neuronales y su funcionamiento. La neurona artificial es un elemento simple pero importante en la construcción de redes neuronales, ya que es la base para la creación de capas y redes más complejas. La práctica permitió entender cómo funciona una neurona artificial y cómo los valores y parametros de entrada influyen en la salida final de la neurona. Se logró implementar una neurona artificial en Quartus que cumple con los criterios establecidos. La práctica también permitió adquirir conocimientos prácticos y aplicados sobre el lenguaje Verilog, lo cual puede ser de gran utilidad en futuros proyectos que involucren el desarrollo de hardware digital.