Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

CE-4302 Arquitectura de Computadores II

Proyecto 2: Diseño e implementación de un arreglo sistólico para unidad de procesamiento neural (NPU)

Profesor:

Luis Alonso Barboza A.

Elaborado por:

Max Garro Mora

Naheem Johnson Solís

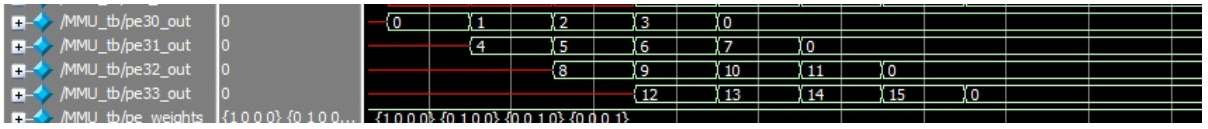
Daniel Montoya Rivera

Grupo 1

I Semestre 2025

- Prueba del funcionamiento del arreglo sistólico

  Esta prueba se hizo debido a que se quería probar las operaciones del arreglo sistólico incluyendo la matriz de pesos con la matriz de entrada, lo que se observa en la siguiente imagen son los PEs de salida.



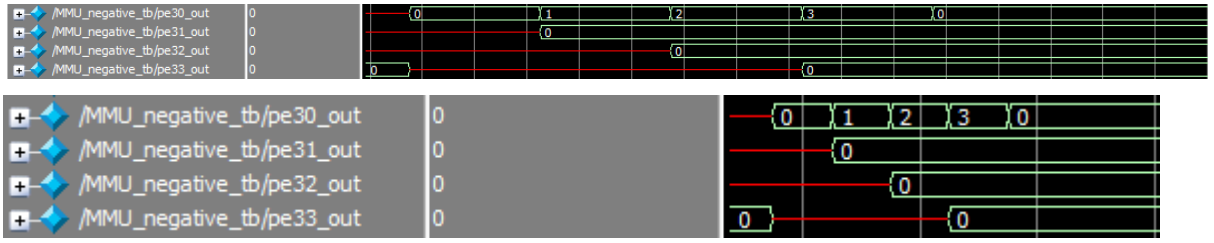
- Prueba del funcionamiento del arreglo sistólico con valores negativos

  Esta prueba se hizo debido a que se quería probar las operaciones del arreglo sistólico incluyendo la matriz de pesos con la matriz de entrada, lo que se observa en la siguiente imagen son los PEs de salida. Se usaron valores negativos para demostrar que el ReLU sí funciona y que cuando son menores a 0, dicho dato pasa a ser 0. Es por ello que se en esta imagen se muestra la matriz de pesos, pero con valores negativos:

```
// Carga por columnas (weight stationary)
@(posedge clk);
wt_arr = 64'hFFFF000000000000;  // Columna 0: [-1, 0, 0, 0]
print_weights();

@(posedge clk);
wt_arr = 64'h0000FFFF00000000;  // Columna 1: [0, -1, 0, 0]
print_weights();

@(posedge clk);
wt_arr = 64'h00000000FFFF0000;  // Columna 2: [0, 0, -1, 0]
print_weights();

@(posedge clk);
wt_arr = 64'h0000000000000001;  // Columna 3: [0, 0, 0, 1]
print_weights();
```

  Las siguientes imágenes corresponden a la matriz de salida, es decir, los PEs de salida y sus resultados.

```
# Pesos en los PEs (T=3250000):
#          Col0        Col1        Col2        Col3
# Fila     0:  0001        0000        0000        0000
# Fila     1:  0000        ffff        0000        0000
# Fila     2:  0000        0000        ffff        0000
# Fila     3:  0000        0000        0000        ffff
#
```

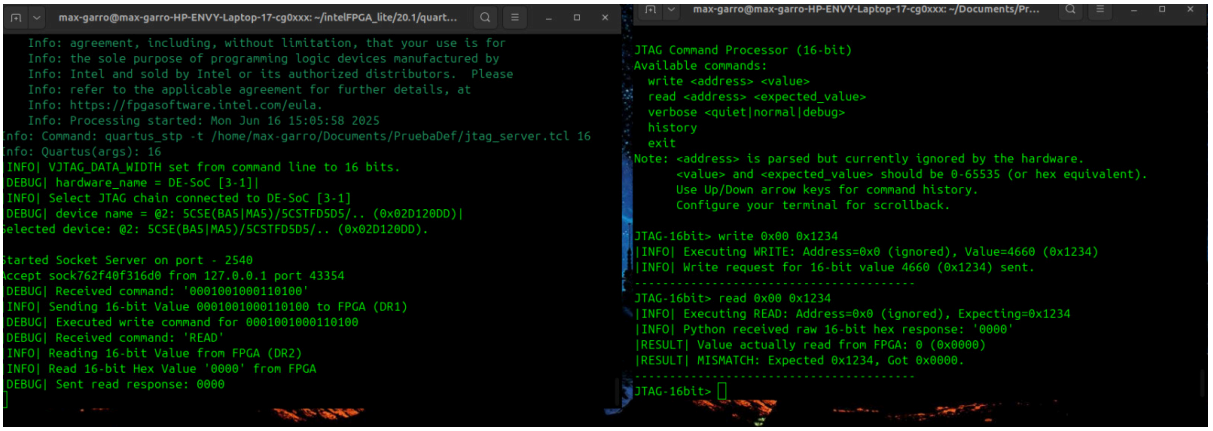


  En las imágenes anteriores se demuestra que se vuelven 0 cuando los datos son menores a 0.

- Prueba del JTAG

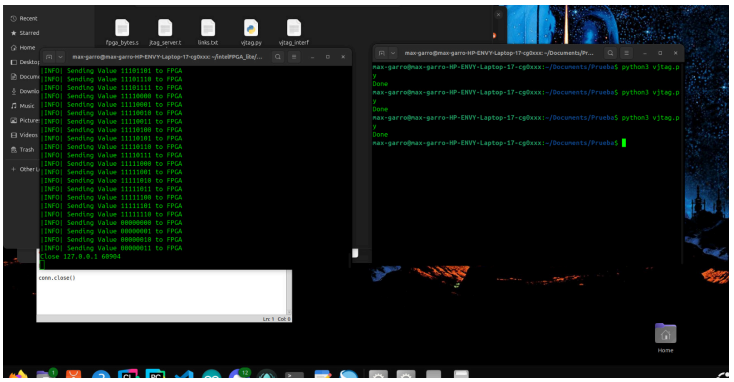
Ambas pruebas se hicieron en Linux debido a que era más sencillo modificar permisos de archivos y también acceder a archivos ejecutables dentro de la carpeta de instalación.

1 - Prueba 1 con escritura y lectura de datos en FPGA. (No se logra recuperar el valor escrito en la dirección asignada). Posible error en .tcl, debido a que no se logra leer nada y siempre retorna 0s.



Esta primera prueba se hizo para verificar la conexión entre la computadora (cliente) y la FPGA, por medio del archivo .tcl(Server).

2 - Prueba 2 con solo envío de 1024 datos desde el lado de la PC a la FPGA. (No se tiene certeza si se escribieron bien los valores).



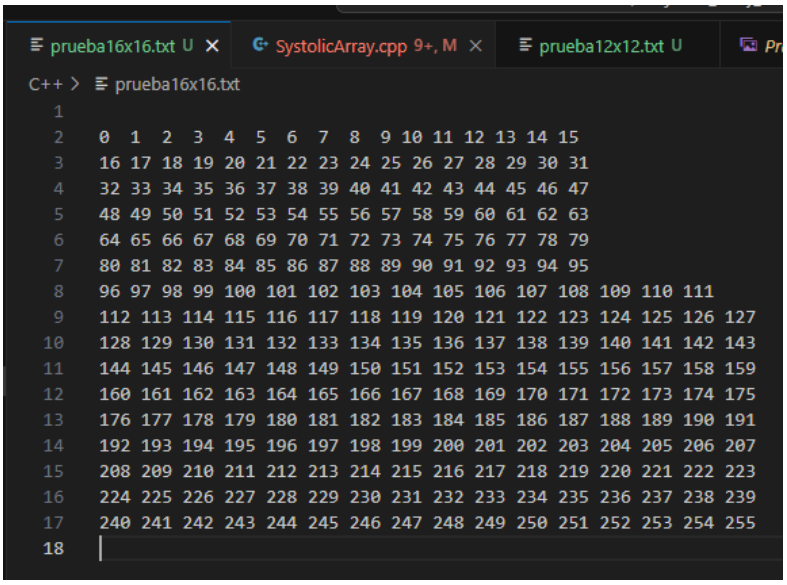
La prueba anterior consistió en enviar 1024 datos del cliente al server para verificar, que la conexión se estuviese haciendo correctamente.

Ambas pruebas han sido para revisar la conexión entre la computadora y la FPGA por medio de un .tcl, pero sucedió que al no haber una RAM conectada, los datos siempre retornaban 0s. Esto se debe a que se escribía en direcciones inexistentes y por ello los 0s.

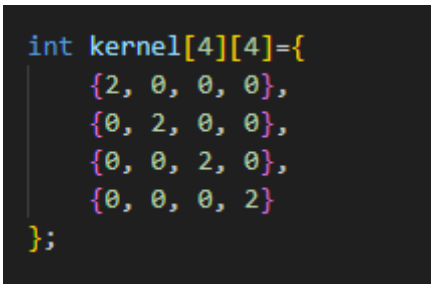
- Pruebas de alto nivel:

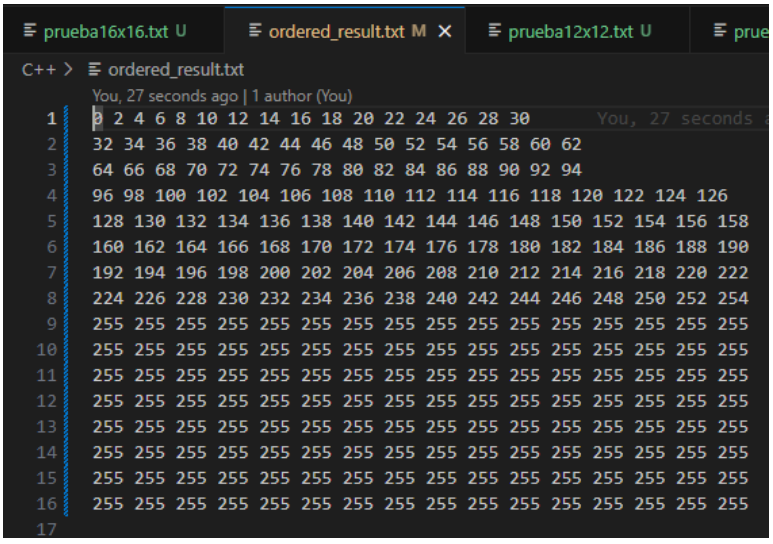
Ambas pruebas fueron ejecutadas para verificar el modelo inicial en alto nivel y probar así que todo estuviese funcionando, para luego hacer un programa con el mismo comportamiento. Su función era básicamente modelar el sistema de SystemVerilog.

1- Para valores positivos:

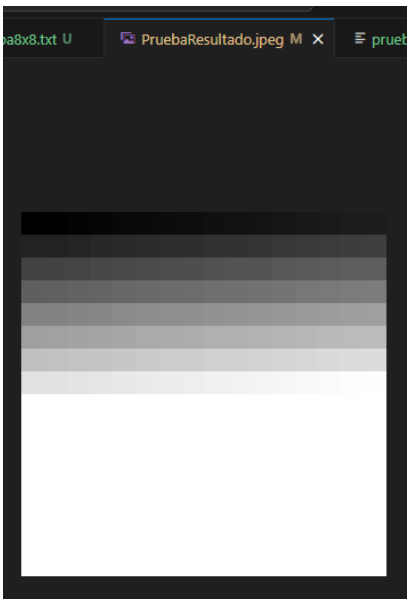


En esta caso se usa una matriz de pesos de 2 para duplicar los valores de entrada.





Las imágenes anteriores corresponden a los datos de entrada, que van a ser procesados por medio del filtro que se hizo. Además, se aprecia que dichos valores de salida corresponden al doble de los valores de entrada, tal cual cómo se esperaba que se comportara.



La imagen anterior corresponde a los datos de salida de la matriz de prueba.

Las imágenes anteriores corresponden al resultado final después de todo el procesamiento de estas mismas.

2- Para valores negativos:

La siguiente imagen corresponde a la matriz de pesos negativos, la cual provocará que el resultado sea distinto al anterior con valores positivos.

```
int kernel[4][4]={
    {-1, 0, 0, 0},
    {0, 1, 0, 0},
    {0, 0, 1, 0},
    {0, 0, 0, 1}
};
```

prueba16x16.txt U | ordered_result.txt M | prueba12x12.txt U

C++ > prueba16x16.txt

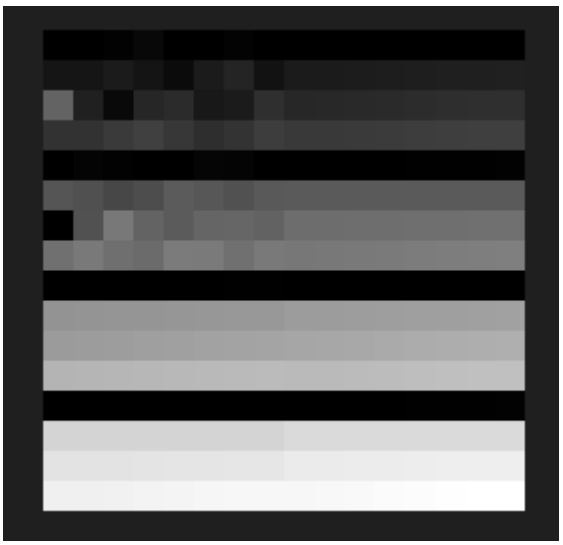
```
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
```

prueba16x16.txt U | ordered_result.txt M | SystolicArray.cpp 9+, M

C++ > ordered_result.txt

You, 2 minutes ago | 1 author (You)

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
96 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
0 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
```



En las imágenes anteriores se pudo comprobar que los resultados fueron distintos y que el comportamiento esperado sí se cumple.

- Prueba de precargado de la RAM con un .mif (Ram loader testbench):

En esta prueba se usó una ram con un .mif precargado con valores random, para así comprobar que todos estos se están procesando dentro del código.

Matriz inicial:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	1	2	3	4	5	6	7	8	.....
8	9	10	11	12	13	14	15	16	.....
16	17	18	19	20	21	22	23	24	.....
24	25	26	27	28	29	30	31	32	.....
32	33	34	35	36	37	38	39	40	!"#$%&'(
40	41	42	43	44	45	46	47	48	)*+,-./0
48	49	50	51	52	53	54	55	56	12345678
56	57	58	59	60	61	62	63	64	9:;<=>?@

El arreglo de salida:

```
# === TEST: Carga secuencial desde RAM ===
# T=65000 [STATE]:   LOAD, Word=0, Base= 0, Addr= 0
# T=75000 [STATE]:   LOAD, Word=0, Base= 0, Addr= 0
# T=85000 [STATE]:   LOAD, Word=0, Base= 0, Addr= 0
# T=95000 [STATE]:   LOAD, Word=0, Base= 0, Addr= 0
# T=105000 [STATE]:   LOAD, Word=1, Base= 0, Addr= 1
# T=115000 [STATE]:   LOAD, Word=1, Base= 0, Addr= 1
# T=125000 [STATE]:   LOAD, Word=1, Base= 0, Addr= 1
# T=135000 [STATE]:   LOAD, Word=2, Base= 0, Addr= 2
# T=145000 [STATE]:   LOAD, Word=2, Base= 0, Addr= 2
# T=155000 [STATE]:   LOAD, Word=2, Base= 0, Addr= 2
# T=165000 [STATE]:   LOAD, Word=3, Base= 0, Addr= 3
# T=175000 [STATE]:   LOAD, Word=3, Base= 0, Addr= 3
# T=185000 [STATE]:   LOAD, Word=3, Base= 0, Addr= 3
# T=185000 [BLOCK]: Valid=1, Done=0, Out=0004000300020001
#   PE0: 0001, PE1: 0002, PE2: 0003, PE3: 0004
# T=195000 [STATE]:   LOAD, Word=0, Base= 4, Addr= 4
# T=205000 [STATE]:   LOAD, Word=0, Base= 4, Addr= 4
# T=215000 [STATE]:   LOAD, Word=0, Base= 4, Addr= 4
# T=225000 [STATE]:   LOAD, Word=1, Base= 4, Addr= 5
# T=235000 [STATE]:   LOAD, Word=1, Base= 4, Addr= 5
# T=245000 [STATE]:   LOAD, Word=1, Base= 4, Addr= 5
# T=255000 [STATE]:   LOAD, Word=2, Base= 4, Addr= 6
# T=265000 [STATE]:   LOAD, Word=2, Base= 4, Addr= 6
# T=275000 [STATE]:   LOAD, Word=2, Base= 4, Addr= 6
# T=285000 [STATE]:   LOAD, Word=3, Base= 4, Addr= 7
# T=295000 [STATE]:   LOAD, Word=3, Base= 4, Addr= 7
# T=305000 [STATE]:   LOAD, Word=3, Base= 4, Addr= 7
# T=305000 [BLOCK]: Valid=1, Done=0, Out=0008000700060005
#   PE0: 0005, PE1: 0006, PE2: 0007, PE3: 0008
# T=315000 [STATE]:   LOAD, Word=0, Base= 8, Addr= 8
# T=325000 [STATE]:   LOAD, Word=0, Base= 8, Addr= 8
# T=335000 [STATE]:   LOAD, Word=0, Base= 8, Addr= 8
# T=345000 [STATE]:   LOAD, Word=1, Base= 8, Addr= 9
# T=355000 [STATE]:   LOAD, Word=1, Base= 8, Addr= 9
# T=365000 [STATE]:   LOAD, Word=1, Base= 8, Addr= 9
# T=375000 [STATE]:   LOAD, Word=2, Base= 8, Addr=10
# T=385000 [STATE]:   LOAD, Word=2, Base= 8, Addr=10
# T=395000 [STATE]:   LOAD, Word=2, Base= 8, Addr=10
# T=405000 [STATE]:   LOAD, Word=3, Base= 8, Addr=11
# T=415000 [STATE]:   LOAD, Word=3, Base= 8, Addr=11
# T=425000 [STATE]:   LOAD, Word=3, Base= 8, Addr=11
# T=425000 [BLOCK]: Valid=1, Done=0, Out=000c000b000a0009
```

...

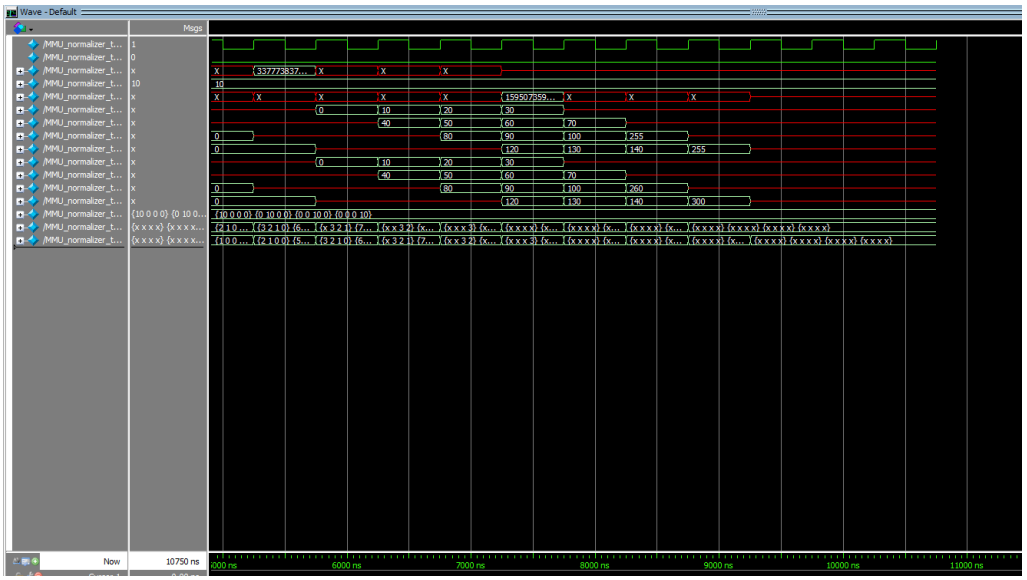
```
#   PE0: 0039, PE1: 003a, PE2: 003b, PE3: 003c
# T=1875000 [STATE]:   LOAD, Word=0, Base=60, Addr=60
# T=1885000 [STATE]:   LOAD, Word=0, Base=60, Addr=60
# T=1895000 [STATE]:   LOAD, Word=0, Base=60, Addr=60
# T=1905000 [STATE]:   LOAD, Word=1, Base=60, Addr=61
# T=1915000 [STATE]:   LOAD, Word=1, Base=60, Addr=61
# T=1925000 [STATE]:   LOAD, Word=1, Base=60, Addr=61
# T=1935000 [STATE]:   LOAD, Word=2, Base=60, Addr=62
# T=1945000 [STATE]:   LOAD, Word=2, Base=60, Addr=62
# T=1955000 [STATE]:   LOAD, Word=2, Base=60, Addr=62
# T=1965000 [STATE]:   LOAD, Word=3, Base=60, Addr=63
# T=1975000 [STATE]:   LOAD, Word=3, Base=60, Addr=63
# T=1985000 [STATE]:   LOAD, Word=3, Base=60, Addr=63
# T=1985000 [BLOCK]: Valid=1, Done=0, Out=0040003f003e003d
#   PE0: 003d, PE1: 003e, PE2: 003f, PE3: 0040
# T=1995000 [STATE]: FINISH, Word=0, Base= 0, Addr= 0
# T=2005000 [STATE]:   DONE, Word=0, Base= 0, Addr= 0
#
# === RESULTADOS FINALES ===
# Bloques completos cargados: 16
# SeÃ±al done: 1
# ERROR en primer bloque
# TEST PASADO: Todos los datos se cargaron correctamente
# ** Note: $finish    : C:/Users/maxga/OneDrive/Documentos/PARQ2/Systolic_Array_BrightnessFilter/SystemVerilogBrightness/RAM_Loader_tb.sv(87)
#    Time: 2005 ns  Iteration: 2  Instance: /RAM_Loader_tb
# 1
```

En este caso también se comprobó que se logró cargar los datos desde la ram y que cumplió con su comportamiento esperado.

Este módulo es el RAMloader, el cual consiste en una máquina de estados que cada cierto tiempo emplea chunks de 4x4. Esos chunks son los que se ven en las imágenes anteriores.

- Prueba del Normalizer:

Se hicieron pruebas con los mismos datos precargados en la ram, para luego usar el módulo del normalizador para así comprobar el comportamiento esperado.

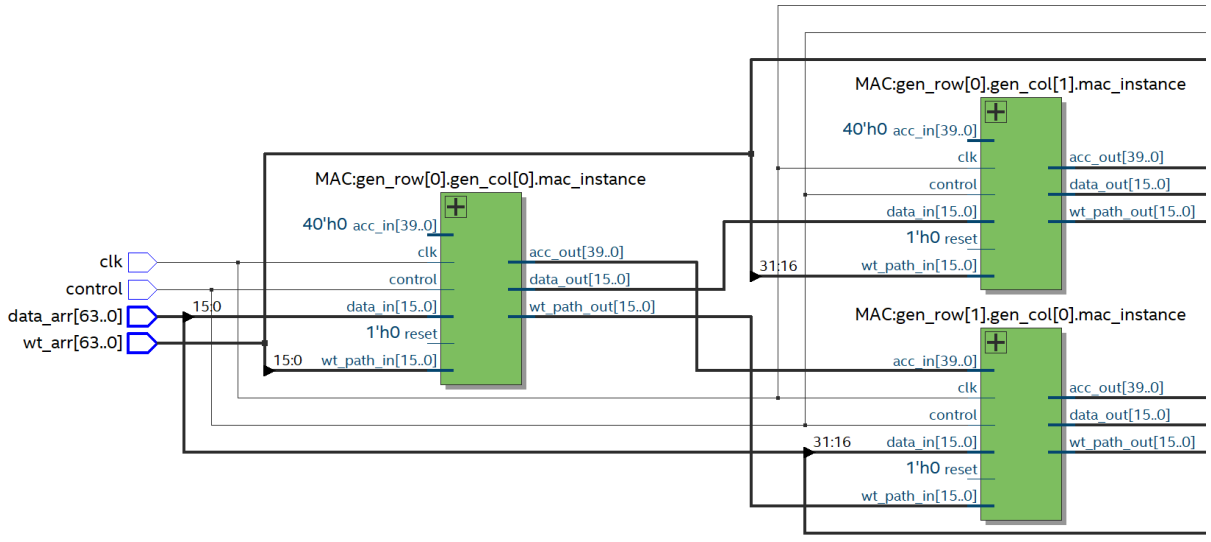


El normalizador (“Normalizer”) se encarga de comprobar si el dato es mayor a 255, entonces que sea seteado en 255 y si es menor a 0, que nunca va a pasar, se usa la función ReLU para establecerlos en 0.

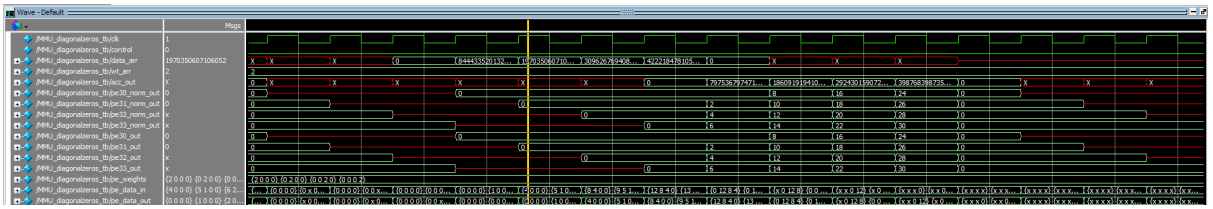
- Test de diagonal zeros:

Dicho test fue para comprobar que se están distribuyendo los datos de la matriz de entrada cómo se espera.

Esta estructura muestra cómo se distribuyen los datos que usa cada MAC: data_array como datos de entrada y otro usa los datos del weight array como datos de entrada y luego estos se procesan para lograr conseguir la matriz de salida.



En la siguiente imagen se muestra cómo los datos terminan de ser procesados después de todo:



Dicha prueba fue exitosa debido a que se observa cómo los datos procesados están en las salidas.