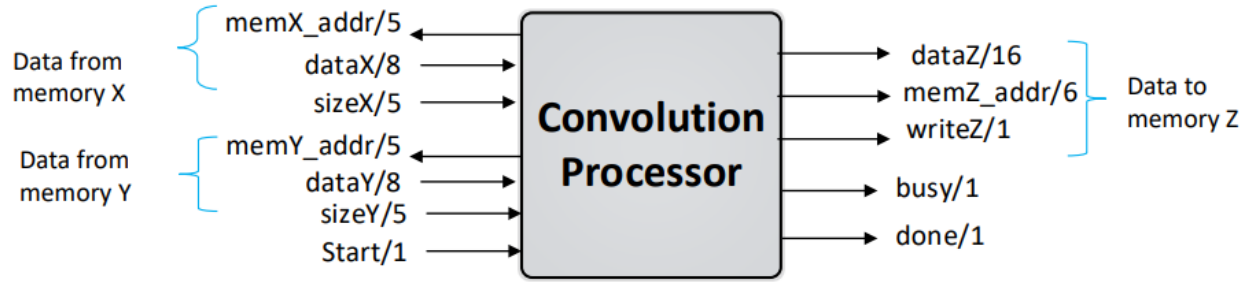
Convolution Processor

Alejandro Pardo Ordaz

Descripción del Problema

Implementar un procesador que realiza la convolución. El objetivo principal del coprocesador de convolución es realizar la convolución de dos señales de entrada, denotadas como dataX y dataY, y producir una salida dataZ que representa el resultado de la convolución. Además, el coprocesador está diseñado para trabajar con dos señales almacenadas en memorias ROM, memX y memY, las cuales contienen los datos de entrada dataX y dataY, respectivamente.

Diagrama de caja negra.



El diagrama de caja negra del coprocesador de convolución muestra las principales interfaces con el entorno externo.

Señales de Entrada:

- clk: Reloj del sistema para sincronizar las operaciones del coprocesador.
- rstn: Señal de reinicio asincrónico para restablecer el estado interno del coprocesador.
- dataX: Datos de entrada para la primera señal de convolución.
- sizeX: Tamaño de la primera señal de convolución.
- memX_addr: Dirección de memoria para acceder a los datos de dataX.
- dataY: Datos de entrada para la segunda señal de convolución.
- sizeY: Tamaño de la segunda señal de convolución.
- memY_addr: Dirección de memoria para acceder a los datos de dataY.
- start: Señal de inicio para iniciar la operación de convolución.

Señales de Salida:

- dataZ: Datos de salida que representan el resultado de la convolución.
- memZ_addr: Dirección de memoria para almacenar los datos de salida.
- writeZ: Señal que indica si se debe escribir en la memoria de salida.
- busy: Señal que indica si el coprocesador está ocupado realizando una operación.
- done: Señal que indica la finalización de la operación de convolución.

El coprocesador de convolución recibe dos señales de entrada (dataX y dataY) junto con sus respectivos tamaños y direcciones de memoria. Cuando se activa la señal de inicio (start), el coprocesador realiza la convolución de los datos de las dos memorias de entrada y genera una salida (dataZ) que representa el resultado de la convolución. Durante la operación, las señales (busy) y (done) indican el estado del coprocesador. Una vez que se completa la convolución, se activa la señal (done) se informa al sistema de que terminó la convolución y está lista para hacer otra convolución.

Pseudocódigo

```
1.  While start = 0
2.  End while
3.  addr_temp = 0
4.  busy = 1
5.  while addr_temp < sizeX + sizeY -1
6.      data_temp = 0
7.      j = 0
8.      while j < sizeY
9.          if (addr_temp - j >= 0 && addr_temp - j < sizeX)
10.             data_temp = data_temp + (dataX[addr_temp - j] * dataY[j])
11.             j++
12.         else
13.             data_temp = data_t + (0)
14.     End while
15.     dataZ = data_temp
16.     writeZ = 1
17.     writeZ = 0
18.     addr_temp ++
19. End white
20. busy = 0
21. done = 1
22. done = 0
```

23. if (init == 0)
24. goto 1
25. else
26. goto 22

Comienza esperando a que se active la señal de inicio (start). Luego, inicializa los contadores y las señales pertinentes antes de entrar en un bucle while para realizar la convolución. Dentro de este bucle while, se acumulan los resultados de la convolución en data_temp. Una vez completada la convolución, se escribe el resultado en la memoria de salida (dataZ) y se avanza al siguiente ciclo. Después de completar todas las operaciones, el coprocesador marca su estado como no ocupado (busy = 0) y termina la operación. Dependiendo del valor de init, el coprocesador puede reiniciar el proceso o finalizar su ejecución.

Validación del Pseudocódigo

```
3  // Tamaño de los arreglos de datos X, Y y Z
4  #define SIZE_X 3
5  #define SIZE_Y 3
6  #define SIZE_Z (SIZE_X + SIZE_Y - 1)
7
8  // Función para realizar la convolución entre dos arreglos
9  void convolution(int X[], int Y[], int Z[]) {
10     // Iterar sobre todos los puntos de convolución
11     for (int i = 0; i < SIZE_Z; i++) {
12         Z[i] = 0; // Inicializar el valor en Z[i]
13
14         // Realizar la operación de convolución para el punto i
15         for (int j = 0; j < SIZE_Y; j++) {
16             // Verificar si el índice es válido en X
17             int idx = i - j;
18             if (idx >= 0 && idx < SIZE_X) {
19                 Z[i] += X[idx] * Y[j];
20             }
21         }
22     }
23 }
```

DataX = {1, 2, 1, 4, 3, 7}

DataY = {1, 3, 5, 6, 7}

Resultado con C

```
Resultado de la convolución:
1 5 12 23 39 56 67 81 63 49
```

Resultado con Matlab

```
Resultado de la convolución:
     1     5    12    23    39    56    67    81    63    49
```

Diagrama ASM

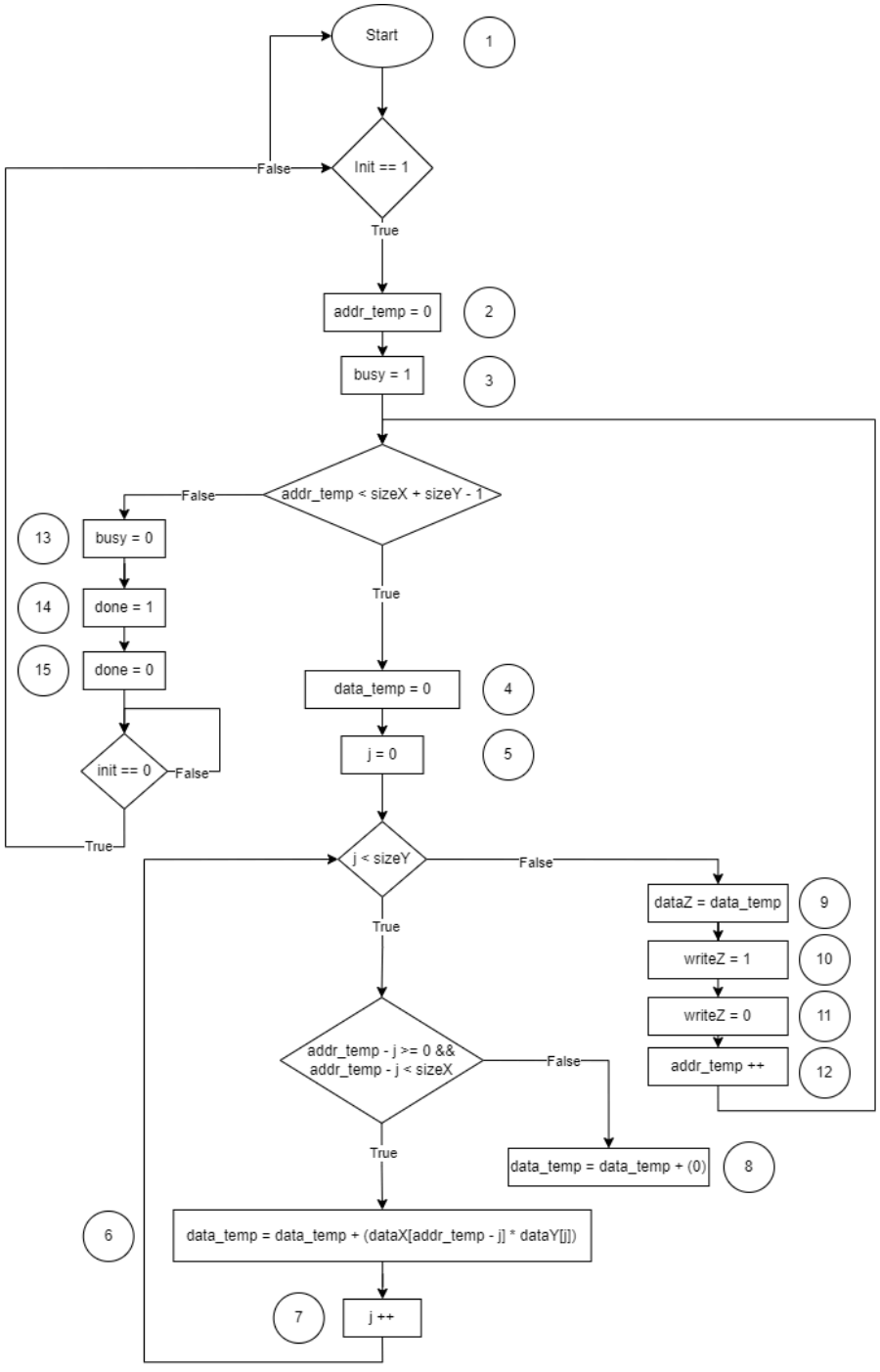
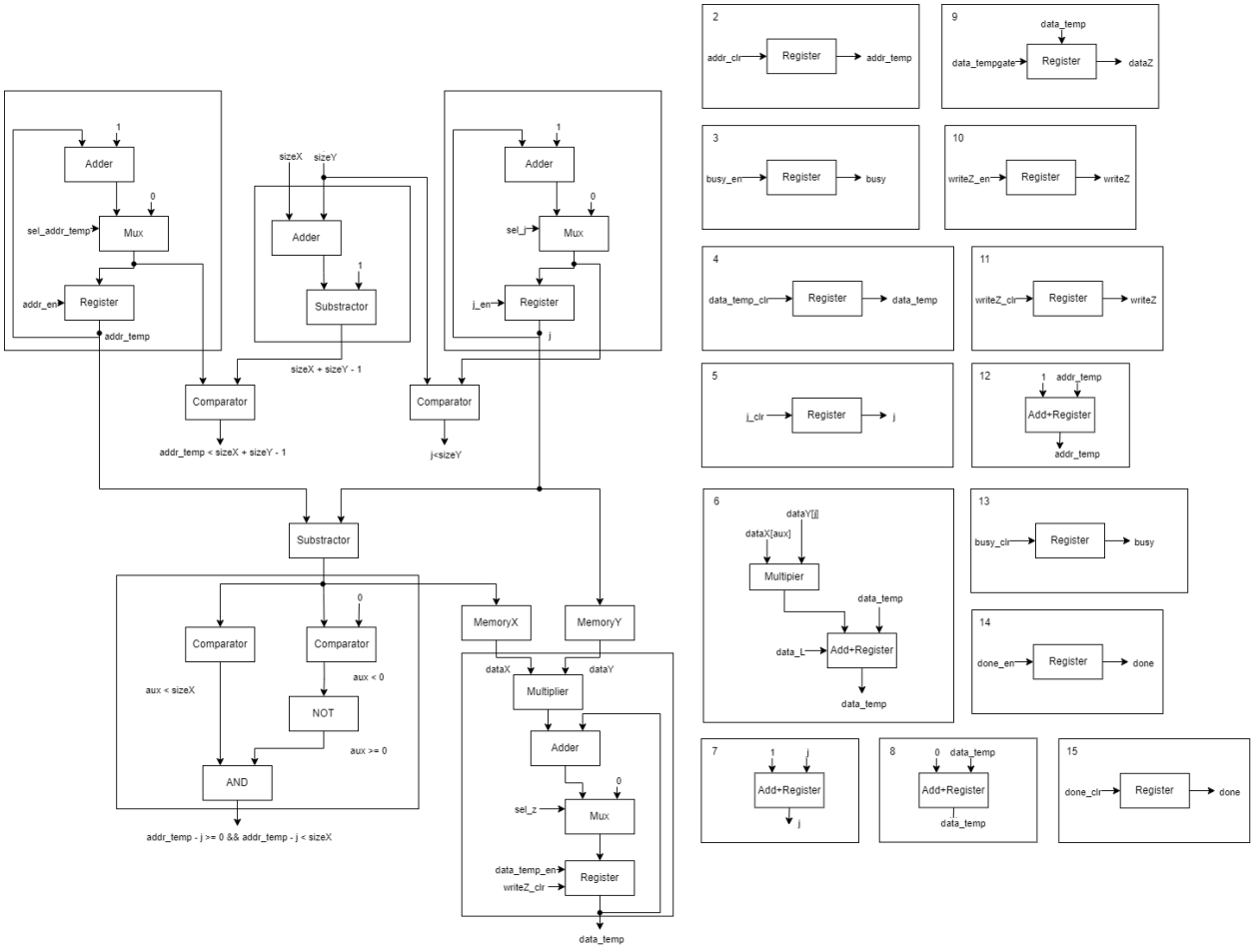


Diagrama del datapath y máquina de estados.


1
Adder
0
sel_addr_temp
Mux
addr_en
Register
addr_temp
sizeX
sizeY
Adder
1
Substractor
sizeX + sizeY - 1
1
Adder
0
sel_j
Mux
j_en
Register
j
Comparator
addr_temp < sizeX + sizeY - 1
Comparator
j<sizeY
Substractor
Comparator
0
Comparator
aux < sizeX
aux < 0
NOT
aux >= 0
AND
addr_temp - j >= 0 && addr_temp - j < sizeX
MemoryX
MemoryY
dataX
dataY
Multiplier
Adder
0
sel_z
Mux
data_temp_en
writeZ_clr
Register
data_temp
2
addr_clr
Register
addr_temp
9
data_temp
data_tempgate
Register
dataZ
3
busy_en
Register
busy
10
writeZ_en
Register
writeZ
4
data_temp_clr
Register
data_temp
11
writeZ_clr
Register
writeZ
5
j_clr
Register
j
12
1
addr_temp
Add+Register
addr_temp
6
dataY[j]
dataX[aux]
Multipier
data_temp
data_L
Add+Register
data_temp
13
busy_clr
Register
busy
14
done_en
Register
done
7
1
j
Add+Register
j
8
0
data_temp
Add+Register
data_temp
15
done_clr
Register
done