Avance Convolution Processor

Profesor: Vidkar Anibal Delgado Gallardo

Alumno: Efrén Armando Limón Bultrago

30/04/2024

1. DESCRIPCIÓN DEL PROBLEMA:

Desarrollar una unidad de proceso de convolución de dos señales, realizar bloques básicos para su implementación, proponer propuestas para el algoritmo y utilizar la metodología de top-down e instanciación de módulos, seguir las buenas prácticas de programación, utilizaremos lo siguiente como referencia.

Inputs: Data stores in memory Y. Size of the signal stores in this memory. Start signal.

Outputs: Done signal (One-shot Type). Busy signal- Store result in memory Z.

Note: signal X(n) will be stored internally in a ROM memory.

FORMULA Y PARAMETROS A UTILIZAR:

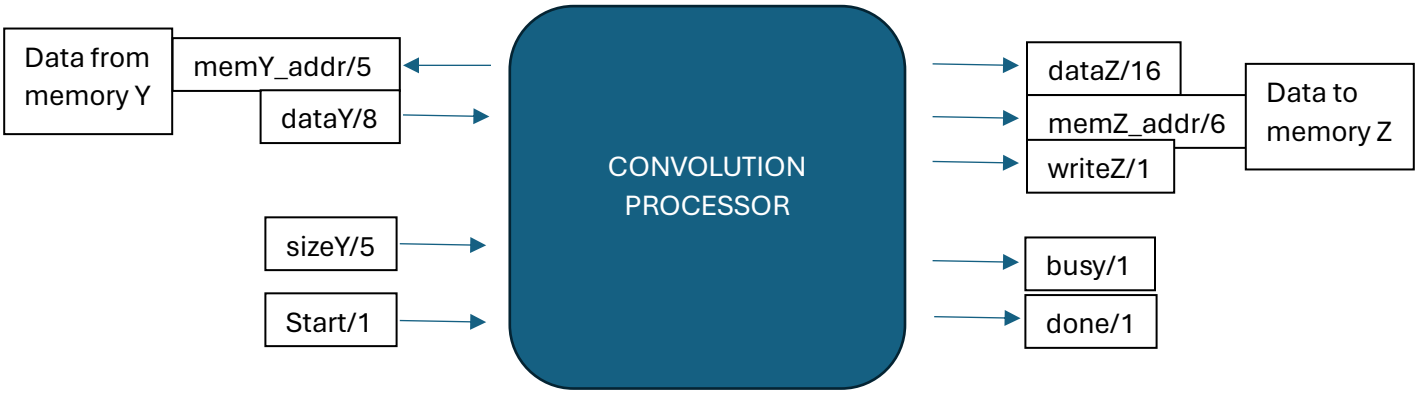
$$\boldsymbol{Z(n) = x * y}$$

$$\boldsymbol{Z(n) = \sum_{k=-\infty}^{\infty} x[k] * y[n-k]}$$

$$\boldsymbol{x[n] = \{2, 1, 0\} = \{X_0, X_1, X_2\}}$$

$$\boldsymbol{y[n] = \{3, 5, 6, 8, 4\} = \{X_0, X_1, X_2, X_3\}}$$

2. DEFINICIÓN DE I/O Y DISEÑO DE CAJA NEGRA:



Inputs:

Start (1bit)

dataY (8 bits)

sizeY (5 bits)

Outpus:

Busy (1 bit)

Done (1 bit)

Write (1 bit)

dataZ (16 bits)

memZ (6 bits)

Relacion entre I/O:

Si *start = 0*, *busy* y *done* también serán 0.

Si *start = 1*, comenzara la convolución y *busy* = 1, esto hasta que termine la convolución, cuando termine el proceso, *busy = 0* y *done = 1*.

Si *start = 1* y *done = 1*, *start = 0*, *done = 0* finalizara la convolución.

3. SEUDO-CÓDIGO Y ALGORITMO:

1.Verificación de inicio del sistema:

2.Verifica si start es igual a 1 y si el sistema no está ocupado (busy == 0).

3.Si se cumple, se procede a iniciar el sistema; de lo contrario, se sale de la función.

4.Inicialización de variables:

5.Se inicializa busy en 1 para indicar que el sistema está ocupado.

6.Se reinicia la señal done a 0 para indicar que la convolución aún no ha terminado.

7.Bucle externo para recorrer memoriaY:

8.Se itera sobre cada elemento de memoriaY, que contiene direcciones memY_addr, datos dataY, y tamaños sizeY.

9.Inicialización del resultado de la convolución:

10.Se inicializa resultado en 0 para acumular el resultado de la convolución para cada conjunto de datos de memoriaY.

11.Bucle interno para recorrer memoriaX:

12.Se itera sobre cada elemento de memoriaX, que contiene direcciones memX_addr, datos dataX, y tamaños sizeX.

13.Cálculo de la convolución:

14.Se calcula el índice contador para acceder a los elementos de memoriaX y realizar la multiplicación y suma para la convolución.

15.Se acumula el resultado en resultado.

16.Verificación de finalización de la convolución para el conjunto actual:

17.Se verifica si contador es menor que sizeY + sizeX - 1, es decir, si se ha terminado la convolución para el conjunto actual de datos.

18.Si se ha terminado, se sale del bucle interno y se procede al siguiente conjunto de datos en memoriaY.

19.Escritura del resultado en memoriaZ:

20.Se usa la dirección de memoria memY_addr como dirección de memoria en memoriaZ (memZ_addr = memY_addr).

21.Se guarda el resultado de la convolución resultado en dataZ.

22.Se activa la señal writeZ para indicar que se debe escribir en memoriaZ.

23.Indicación de finalización de la convolución y liberación del sistema:

24.Se activa la señal done para indicar que la convolución ha terminado.

25.Se establece busy en 0 para indicar que el sistema ya no está ocupado.

IMPLEMENTACIÓN EN LENGUAJE C:

```
// Efrén Armando Limón Bultrago.
// Version:    1.7
// Metodología de Diseño de (SOC)System-On-Chip
// Librerias
#include <stdio.h>
// Constantes
#define SIZE_X 3
#define SIZE_Y 5
#define SIZE_Z (SIZE_X + SIZE_Y - 1)
// Estructura para representar una señal en memoria
struct Signal {
    int data;
    int size;
};
// Funciónes
// Función para realizar la convolución
void convolution(struct Signal memoriaY[],
                 struct Signal memoriaX[],
                 struct Signal memoriaZ[],
                 int *start,
                 int *busy,
                 int *done);
int main() {
    // Definición de señales y memorias
    struct Signal memoriaY[SIZE_Y] = {{3}, {5}, {6}, {8}, {4}}; // 5 valores
    struct Signal memoriaX[SIZE_X] = {{2}, {1}, {0}};   // 3 valores
    struct Signal memoriaZ[SIZE_Z] = {0}; // Inicializar memoriaZ con ceros
    // Señales de entrada
    int start = 1; // Indicar inicio del sistema
    int busy  = 0;  // Indicar sistema no ocupado
```

```
    int done  = 0;  // Indicar convolución no terminada
    // Mandar a llamar la función convolution
    convolution(memoriaY, memoriaX, memoriaZ, &start, &busy, &done);
    // Señales de salida
    int busy_out = busy; // Señal de salida busy
    int done_out = done; // Señal de salida done
    // Imprimir resultados
    printf("Resultado de la convolución:\n");
    for (int i = 0; i < SIZE_Z; i++) {
        printf("%d : ", memoriaZ[i].data);
    }
    printf("\n");
    return 0;
}
// Función para realizar la convolución
void convolution(struct Signal memoriaY[],
                 struct Signal memoriaX[],
                 struct Signal memoriaZ[],
                 int *start,
                 int *busy,
                 int *done) {
    int i, j, countX, actZ;
    // Verificación de inicio del sistema
    if (*start == 1 && *busy == 0) {
        *busy = 1; // El sistema está ocupado
        *done = 0; // Reiniciar la señal de done
        // Bucle externo para recorrer el arreglo resultante Z
        for (i = 0; i < SIZE_Z; i++) {
            actZ = 0; // Inicializar el acumulador para la posición i de Z
            // Bucle interno para recorrer la señal Y
            for (j = 0; j < SIZE_Y; j++) {
```

```
            countX = i - j; // Calcular el índice en X para la multiplicación
            // Verificar si el índice calculado está dentro del rango de X
            if (countX >= 0 && countX < SIZE_X) {
                // Multiplicar elementos de X y Y, y acumular el resultado
                actZ += memoriaX[countX].data * memoriaY[j].data;
            }
        }
        // Almacenar el resultado de la convolución en memoriaZ
        memoriaZ[i].data = actZ;
        memoriaZ[i].size = 1; // Tamaño fijo para la convolución
    }
    *done = 1; // Indicar que la convolución ha terminado
    *busy = 0; // El sistema ya no está ocupado
  }
}
```

VALIDACION DEL ALGORITMO EN:

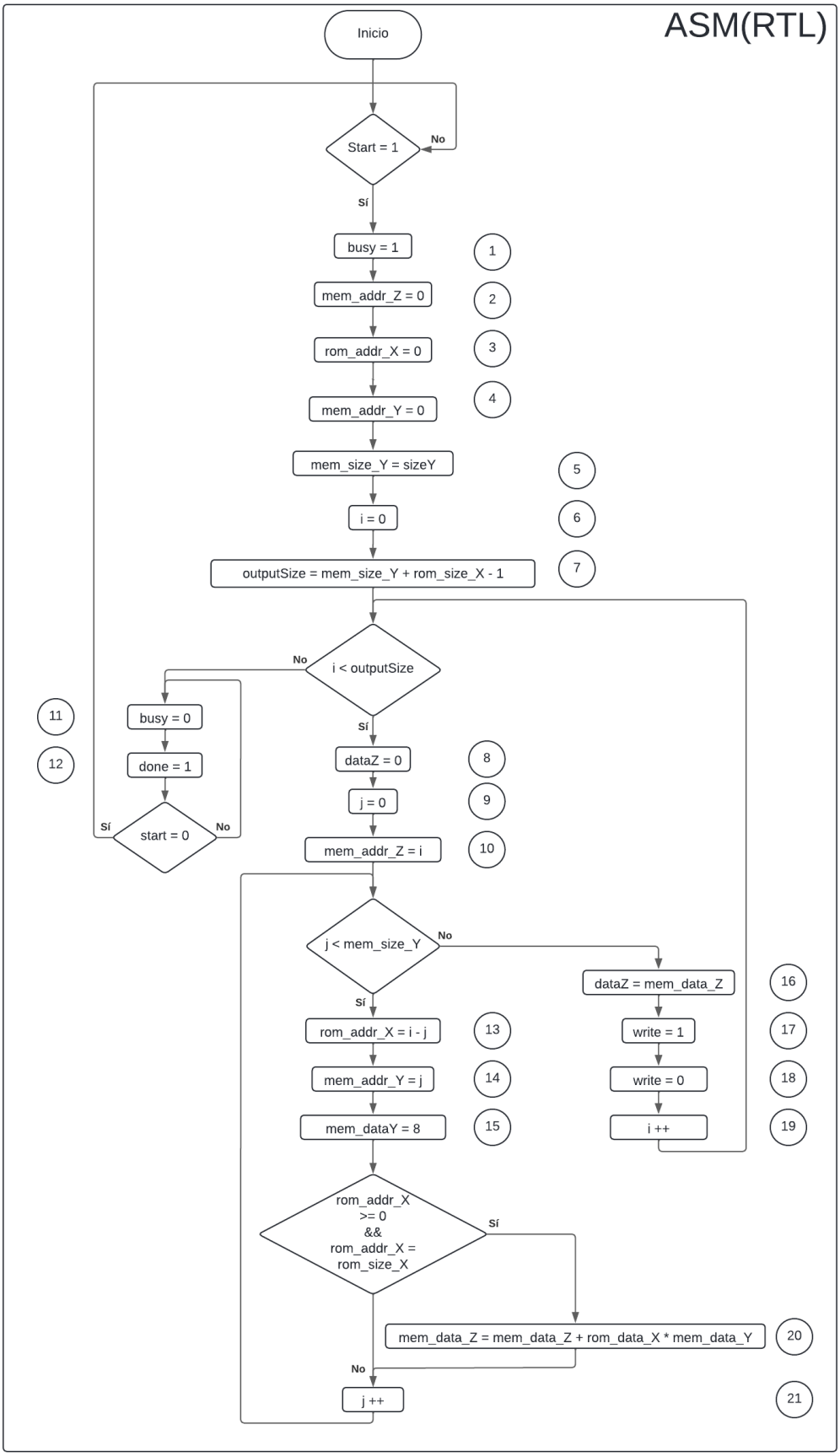
MATLAB:

```
>> x = [2,1,0]

x =

     2     1     0

>> y = [3,5,6,8,4]

y =

     3     5     6     8     4

>> z = conv(x,y)

z =

     6    13    17    22    16     4     0
```

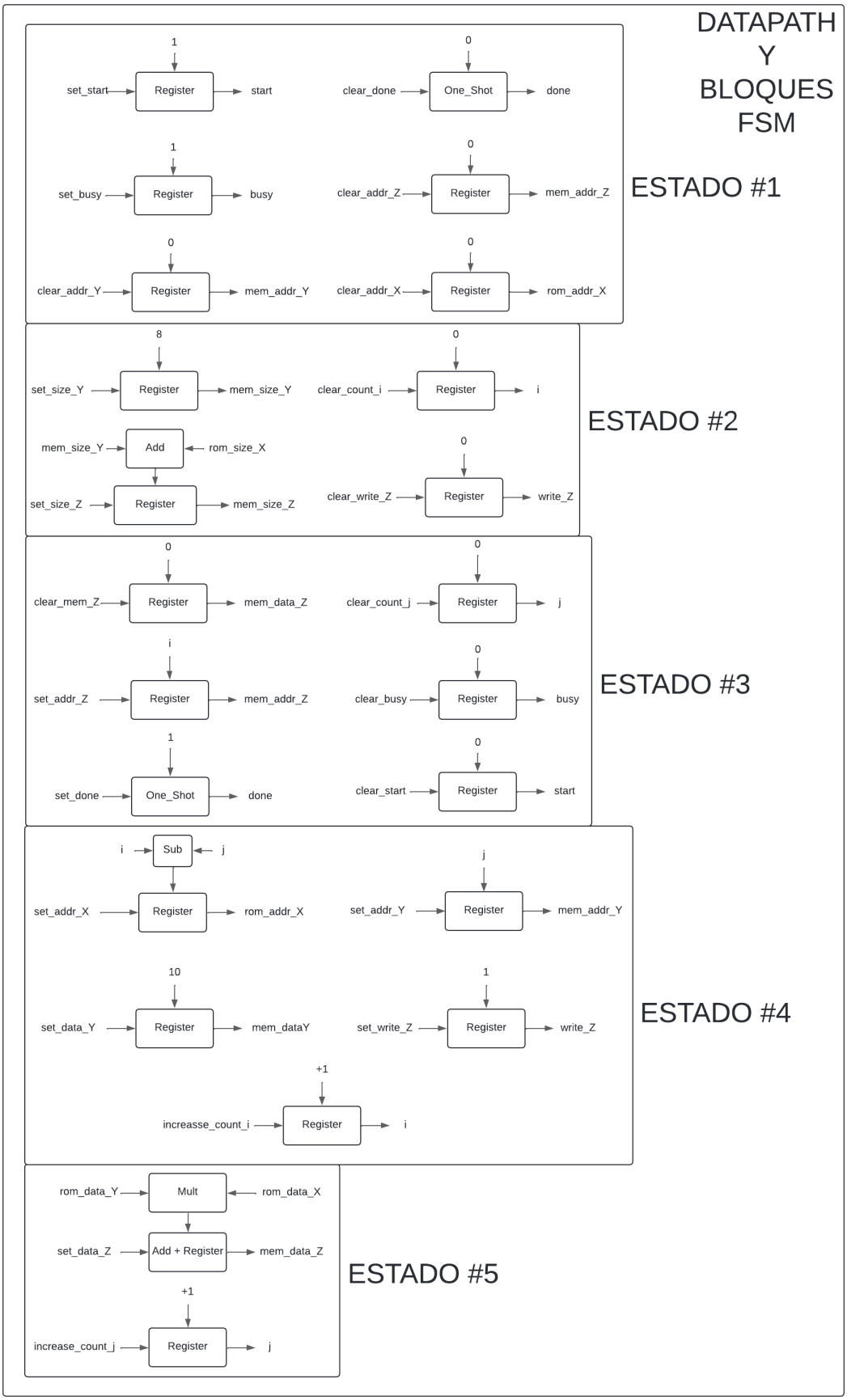
C:

```
Resultado de la convoluci%n:
6 : 13 : 17 : 22 : 16 : 4 : 0 :
```

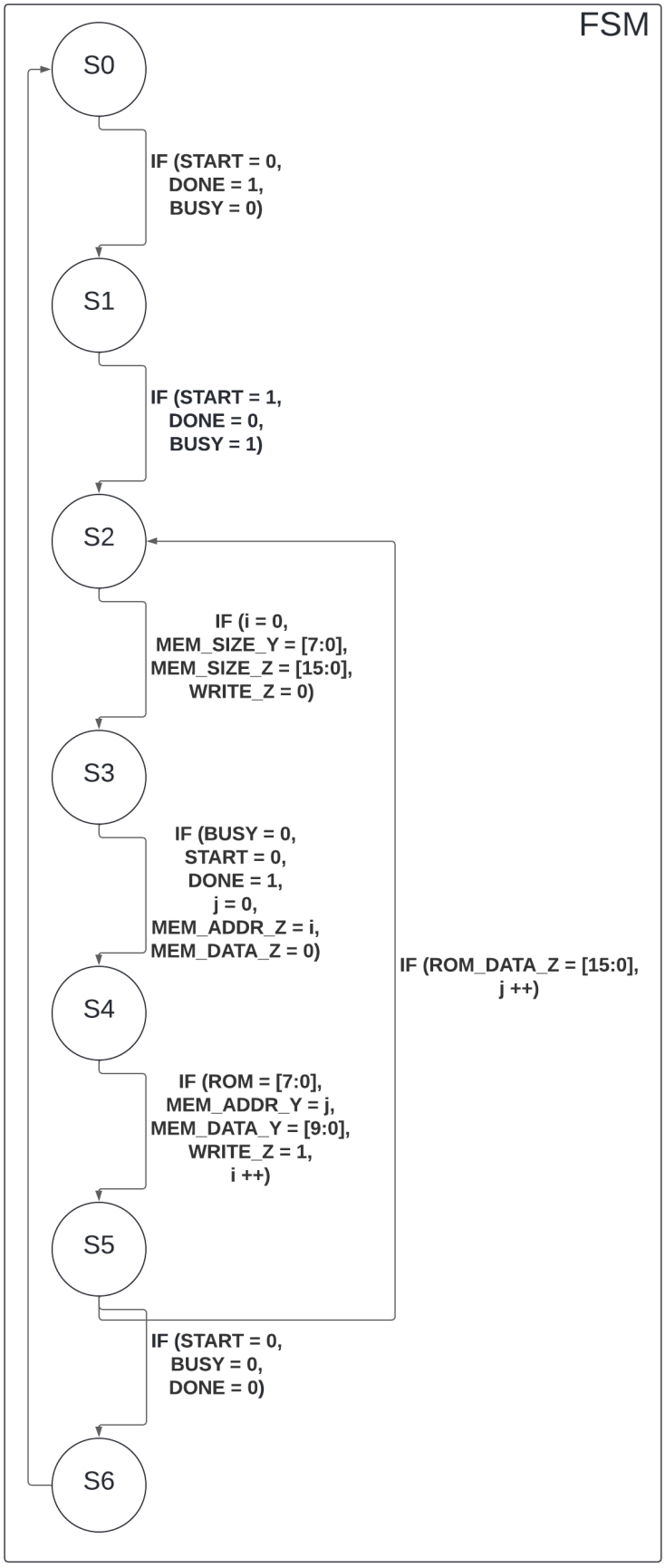
4. DESCRIPCIÓN DEL ALGORITMO EN ASM (RTL):



5. DATAPATH A BLOQUES FSM:



6. DIAGRAMA FSM:



7. DATAPATH OPTIMIZADO: