Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Curso: Metodología de Diseño de SoCs
Nombre: Daniel Alejandro Lara López
No. de tarea: Tarea 1
Título: Diseño estructural de convolucionador a través de metodología Top - Down
Catedrático: Dr. Vidkar Anibal Delgado Gallardo

1.- Descripción del problema

Dentro de las aproximaciones de las metodologías de sistemas digitales, la metodología Top – Down permite realizar un diseño estructural basándonos en lógica de transferencia de registros o diseño a nivel de transferencia de registros (RTL). Esto nos lleva a plantear un diseño desde un nivel de abstracción de caja negra hacía abajo llegando hasta los registros y así describir el flujo de datos de un Core particular.

Para este proyecto se requiere diseñar un Core el cual nos permita realizar la convolución entre dos señales, una llamada X y la otra Y. Dichas entradas se requieren que sean de 8 bits y que sean procedentes de 2 memorias RAM instanciadas en el Testbench, estas serán cargadas a través de un algoritmo de generación de archivos a los cuales Quartus podrá acceder cada vez que se inicialice la cama de pruebas. A mismo, se requiere que este Core, cuente con dos entradas de Size, que manejan como entrada el tamaño de cada uno de los vectores de las señales y se requiere que sean de 5 bits de tamaño de palabra, también existen 2 entradas más, una llamada “Start” y otra llamada “Shape”, que deberán permitir tanto el inicio del algoritmo del Core y el modo en el que se requiere trabaje la convolución; el diseñador debe decidir que modo escoger al inicializar el testbench asignando un valor de 1 para full convolution o 0 para central convolution, o viceversa. De igual modo la bandera de Start indica el inicio y el fin del algoritmo.

Las salidas de este módulo deben ser las siguientes:

- dataZ/16 – Dicha salida es la salida de datos del Core, debe tener tamaño de 16 bits y es la encargada de almacenar en una memoria Z externa en cada una de sus direcciones el valor de la convolución.
- memZ_addr/6 – Es el puntero de dirección de la memoria de salida Z que almacena el valor de la convolución.
- writeZ/1 – es el Write enablo de la memoria de salida.
- Busy/1 – Es la bandera que se encuentra encendida mientras el algoritmo de convolución se desarrolla.
- Done/1 – Bandera de termino de algoritmo. Dicha salida es onehot, esto quiere decir que la bandera “done” se enciende un pulso de reloj en cuanto termina el algoritmo de convolución central o full convolution.
- memX_addr/5 – Salida de puntero de la memoria para la señal X.
- memY_addr/5 – Salida de puntero de la memoria para la señal Y.

De manera general, el algoritmo desempeñado de este core, realiza la convolución completa o central entre dos señales con una longitud máxima de 32 cada una, siempre tomando en cuenta a X como la señal mayor.

2.- Diagrama de caja negra y definición de señales de entrada y salida

A continuación, se muestra el diagrama de caja negra del convolucionador. En este podemos observar lo comentado en la sección 1 del reporte, se aprecia la arquitectura de mayor nivel llamada “convol.v”, dicha entidad de mayor nivel presenta dos entradas de datos de ocho bits, para la inserción de la señales X y Y, así como 2 enttradas de 5 bits para sus respectivos tamaños, las cuales son llamadas sizeX y sizeY, estas entradas permiten definir la cantidad de ciclos o iteraciones para las operaciones de multiplicación y acumulación en los estados para la operación de convolución. Del lado izquierdo, se aprecian las primeras dos salidas del Core, llamadas “memY_addr” y “memX_addr” que son los punteros de direccionamiento de las memorias X y Y, dichos punteros permiten el flujo de datos inicial del Core desde las memorias hasta sus salidas posterior a la operación de convolución.

Finalmente, la figura 1 muestra del lado izquierdo las salidas principales del convolucionador. La salida llamada “dataZ” es la salida de datos de 16 bits para cada uno de los elementos de la señal convolucionada. Mientras que la salida “memZ_addr” es el puntero de direccionamiento de la memoria Z para el resultado de la operación; continuando, writeZ (de 1 bit), es el habilitador de escritura de la memoria de salida que es habilitada por “convol.v” desde su FSM de control. Al momento de la realización del algoritmo de convolución, ya sea para central o completa, la bandera “busy” (salida) será habilitada con un 1 lógico mostrando que el Core esta ocupado en esa realización. Así mismo, “done” es la encargada de mostrar cada que la operación haya terminado, dado que se mantiene en 0 todo el tiempo de ejecución del algoritmo y en cuanto este termine, tanto para full como para central, la bandera se habilita con un 1 lógico por un ciclo, lo que quiere decir que esta es onehot.

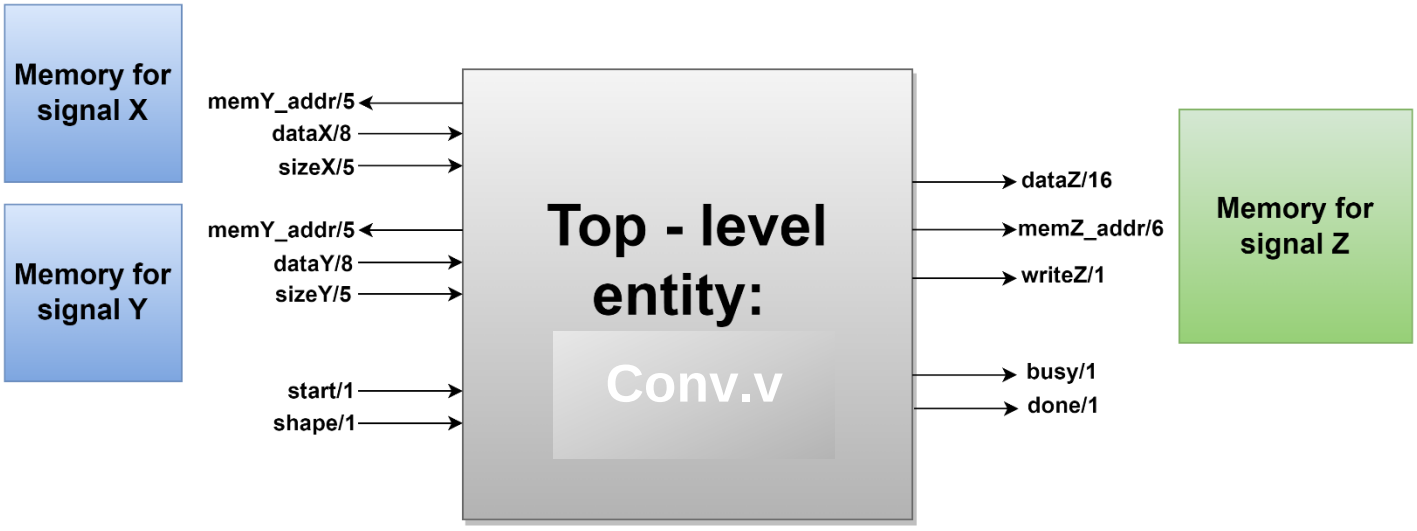


Figura 1.- Diagrama de caja negra de la entidad Top – level del convolucionador, con sus respectivas entradas y salidas.

Es importante mencionar que el diseño estructural a través de la metodología Top – Down hace posible la descripción del algoritmo de convolución tanto full, como central mediante la interconexión de “piezas” o “bloques” a nivel RTL y su control de flujo de datos mediante una maquina de estados que permite la activación de estos en función al algoritmo.

3.- Pseudocódigo con una breve explicación

A continuación, se muestra el pseudocódigo que expresa el algoritmo a diseñar a través de la metodología Top – Down para la realización de las operaciones de convolución completa o central. La figura 2 expresa el Pseudocódigo del algoritmo.

```
%% Parameters
% Generating the signal vectors
X = 5;
Y = 10;
Start = 1;
Shape = 0; % 1 for "full" 0 for "same"

S_X = randi(10,1,X);
S_Y = randi(10,1,Y);

S_X_bin = dec2bin(S_X,8);
addressX = 'C:\Dani\Doctorado\convolucionador\simulation\modelsim\S_X.txt';
% filename1 = 'S_X.txt';
fileID1 = fopen(addressX, 'w');
for i = 1:numel(S_X_bin(:,1))
    fprintf(fileID1, '%s\n', S_X_bin(i,:));
end

addressY = 'C:\Dani\Doctorado\convolucionador\simulation\modelsim\S_Y.txt';
S_Y_bin = dec2bin(S_Y,8);
%%filename2 = 'S_Y.txt';
fileID2 = fopen(addressY, 'w');
for i = 1:numel(S_Y_bin(:,1))
    fprintf(fileID2, '%s\n', S_Y_bin(i,:));
end
```

```
% Signal Sizes and "Full" or "Same" Conv type
SizeX = numel(S_X);
SizeY = numel(S_Y);
Sz_full = (SizeX+SizeY)-1;
Sz_same = SizeX;

% Convolution using Matlab's function - Golden Model
Conv_Full = conv(S_X,S_Y,"full");
Conv_Same = conv(S_X,S_Y,"same");
```

```
%% Convolution Algorithm
S_Z = zeros(1,Sz_full);
S_Z_same = zeros(1,Sz_same);
```

```
while 1
    if Start == 1
        Z_Ind = 1;
        aux = 1;
        fprintf("Busy\n");
        for i = 1:SizeY
            for ii = 1:SizeX
                Mult = S_X(1,ii)*S_Y(1,i);
                S_Z(1,Z_Ind) = S_Z(1,Z_Ind) + Mult;
                Z_Ind = Z_Ind + 1;
                fprintf("Busy\n");
            end
            aux = aux + 1;
            Z_Ind = aux;
            fprintf("Busy\n");
        end

        if Shape == 0
            ptr = ceil((SizeY+1)/2);
            fprintf("Busy\n");
            for same = 1:Sz_same
                S_Z_same(1,same) = S_Z(1,ptr);
                ptr = ptr + 1;
                fprintf("Busy\n");
            end
            break;
        end
    else
        fprintf("Busy\n");
    end
    break;
end
fprintf("Done\n");
```

La primera parte del código, en primera instancia, expresa la longitud de los vectores de las señales X e Y, las cuales en este ejemplo son de tamaño 5 y 10 respectivamente, la siguiente línea muestra el valor de la variable “Shape” que expresa el tipo de operación que realizará el algoritmo, en este caso 1 representa que el algoritmo entra en modo full convolution. Continuando con las dos líneas antes del inicio del algoritmo en el ciclo While, se presentan 2 registros que almacenan los datos de las variables “S_X” y “S_Y”, dichos datos de los registros representan el tamaño real de los vectores de las señales X y Y.

Entrando a la sección principal del algoritmo, un ciclo while true permite el ingreso al algoritmo de convolución, mientras que el condicional if (Start == 1) valida el ingreso al algoritmo para la convolución completa. De este modo, al ingresar, se asigna a sz_full un vector con un tamaño sz_full, el cual es la suma de las longitudes de los vectores restándole el valor de 1, lo cual nos expresa el tamaño real del

vector resultante de la convolución, con cada valor inicializado en 0. Tomando en cuenta que este último vector se trata de una memoria interna.

Posterior a esto, se inicializan aux y Z_ind a uno, los cuales son los punteros de direccionamiento de los vectores tanto de y como del vector de salida de la convolución, también se expresa en la línea 16 la variable o registro “Busy”, el cual se inicializa a 1 expresando que el algoritmo de convolución inicio y el Core se encuentra en medio de la realización como tal.

Continuando, el algoritmo entra a un primer ciclo for dependiente de la longitud en el vector de Y, enseguida el algoritmo ingresa a un segundo ciclo for anidado con el primero que depende de la longitud del vector X. Esto se diseño así con el objetivo de que el algoritmo realice un barrido con cada valor de Y multiplicando cada valor de X, y posteriormente recorriéndose. Las líneas siguientes muestran la multiplicación y la acumulación. Como ya se mencionó cada dato de y multiplica a cada valor en el vector X uno a la vez mientras que la siguiente línea realiza la acumulación dentro del vector del resultado, la línea 22 aumenta el puntero de la memoria de salida (Z).

De esta forma el algoritmo sale del segundo for anidado, solo para incrementar el puntero de la memoria y e igualando el puntero de Z con el de aux, para que de este modo el resultado se vaya recorriendo; la convolución completa termina con una bandera de “Busy” representada con la línea de código `fprintf("Busy\n")`.

Un vez realizado el algoritmo de convolución completa, se toma una decisión en el pseudocódigo, la cual nos lleva a decidir entre realizar la operación de convolución central que comienza desde la línea 31, o bien a pasar a un estado de IDLE o Standing By, de decidir u optar por el valor de shape igual con cero en la condición, de elegir realizar la convolución central, el código a partir de 37 al itrerar toma el vector de la convolución completa y comienza desde un punto regido por la expresión en la línea 35, ceil((low_sz+1)/2), la cual nos permite comenzar a tomar valores del vector w_full a través del registro pointer dentro del for. Dicho ciclo se termina cuando se completan las iteraciones del número de elementos del vector de mayor tamaño, de esta forma es posible realizar la convolución central en cuanto se activa la entrada Shape.

3.1. Validación del algoritmo

En esta sección se presenta el algoritmo realizado en Matlab y sus respectivas simulaciones para comprobar su funcionamiento en las figuras 3, 4 y 5.

Primero se muestran los vectores asignados para X y Y.

```
S_X =

     7     1     9    10     7

S_Y =

  Columns 1 through 8

     8     8     4     7     2     8     1     3

  Columns 9 through 10

     1     1
```

El resultado de la convolución completa y central a través del modelo de oro con la función conv() de Matlab, es la siguiente:

```
Conv_Full =

    56    64   108   205   193   217   131   163   113   101    47    40    17     7

Conv_Same =

   217   131   163   113   101
```

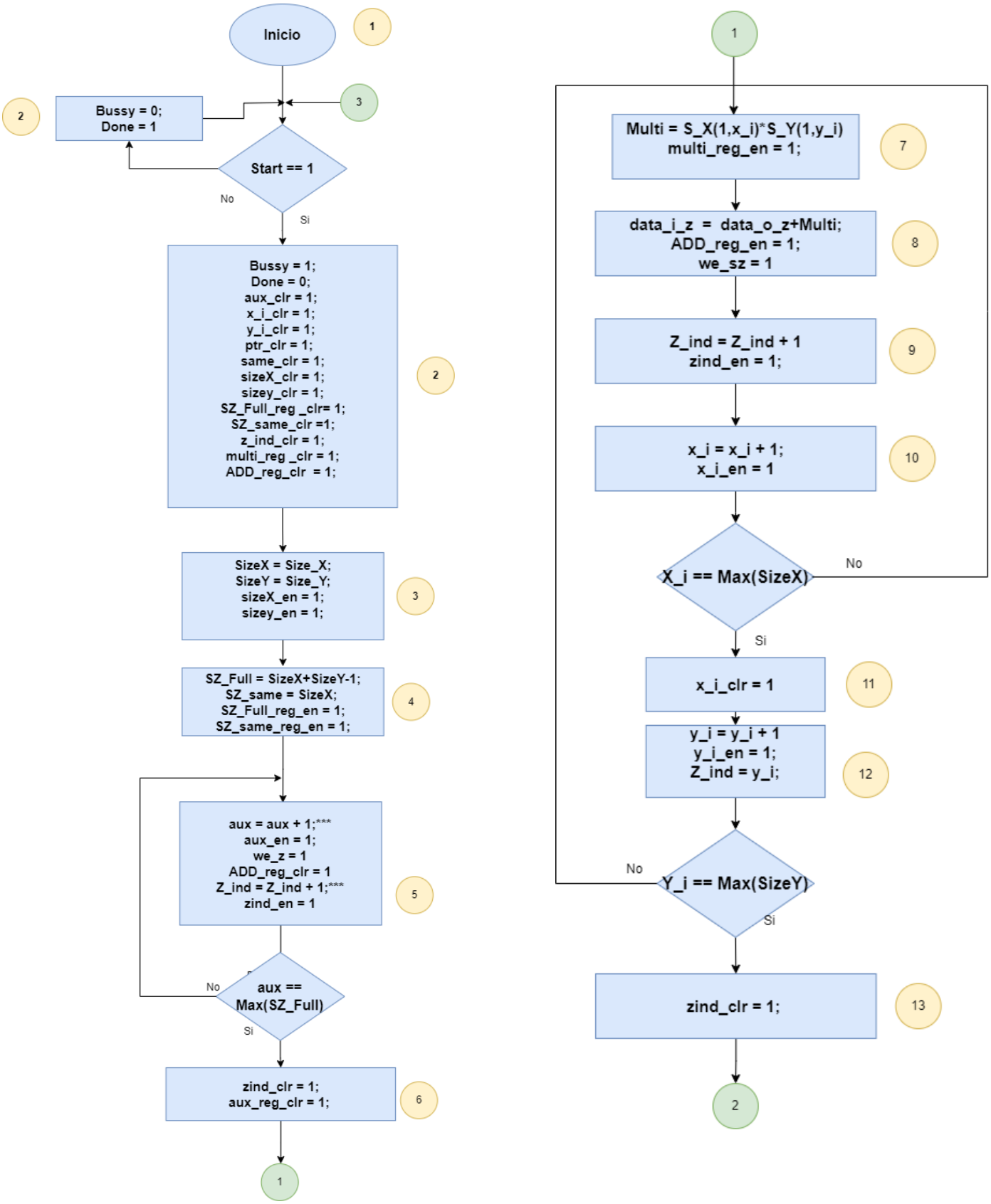
Y comparando este resultado con el del algoritmo realizado de la convolución tenemos:

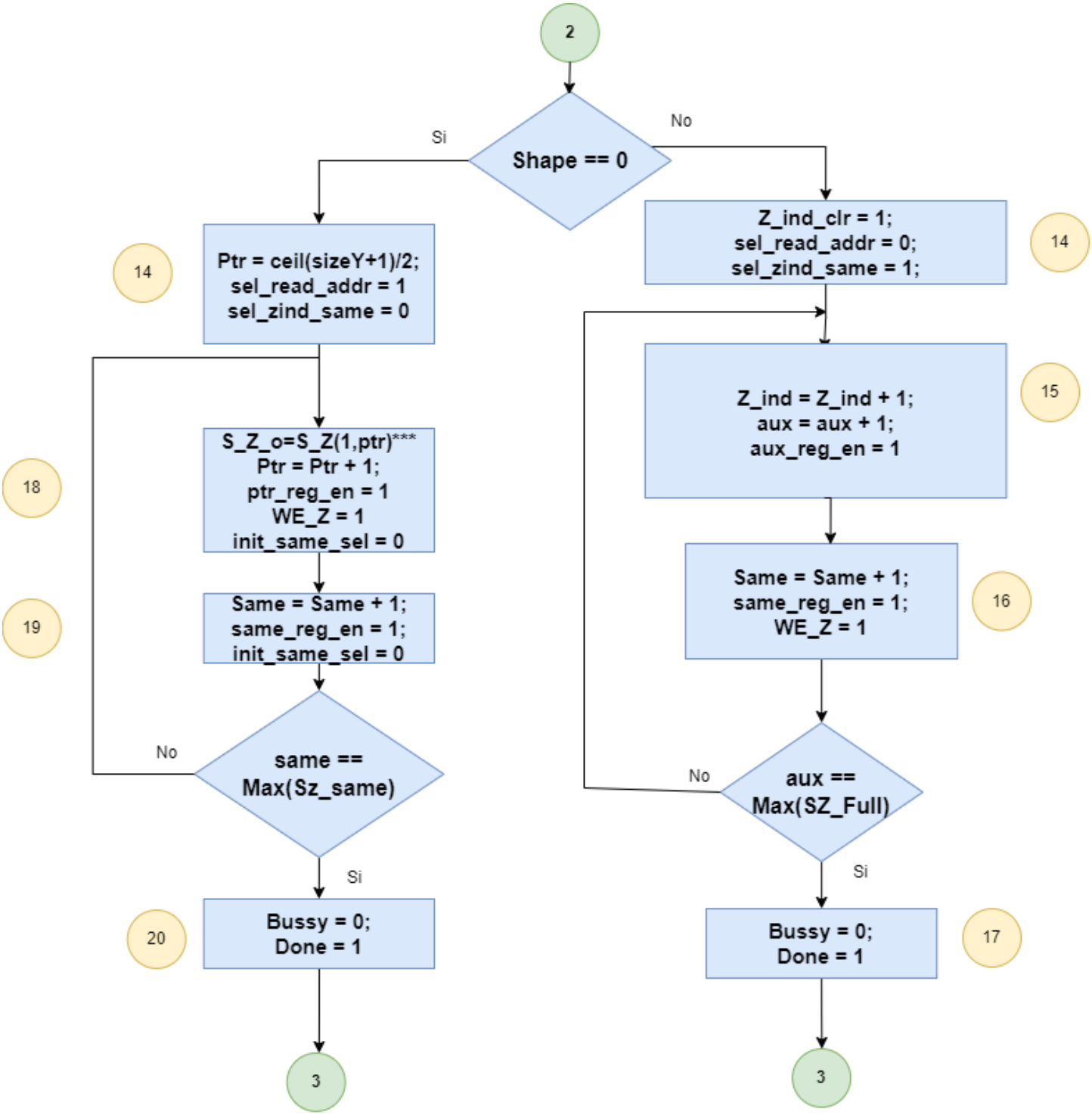
```
S_Z =

    56    64   108   205   193   217   131   163   113   101    47    40    17     7

S_Z_same =

   217   131   163   113   101
```

Podemos observar como el resultado para ambas convoluciones coincide, verificando el funcionamiento del mismo.

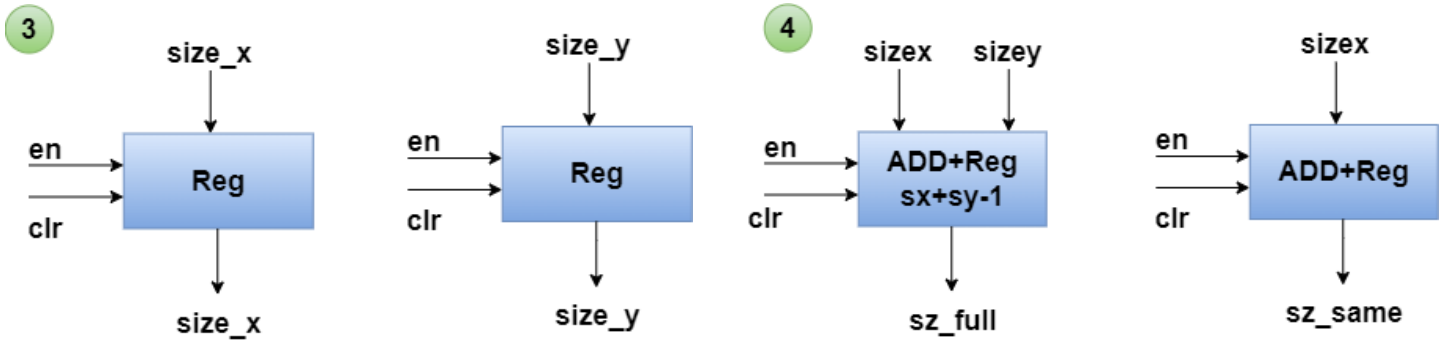
4.- Diagrama ASM

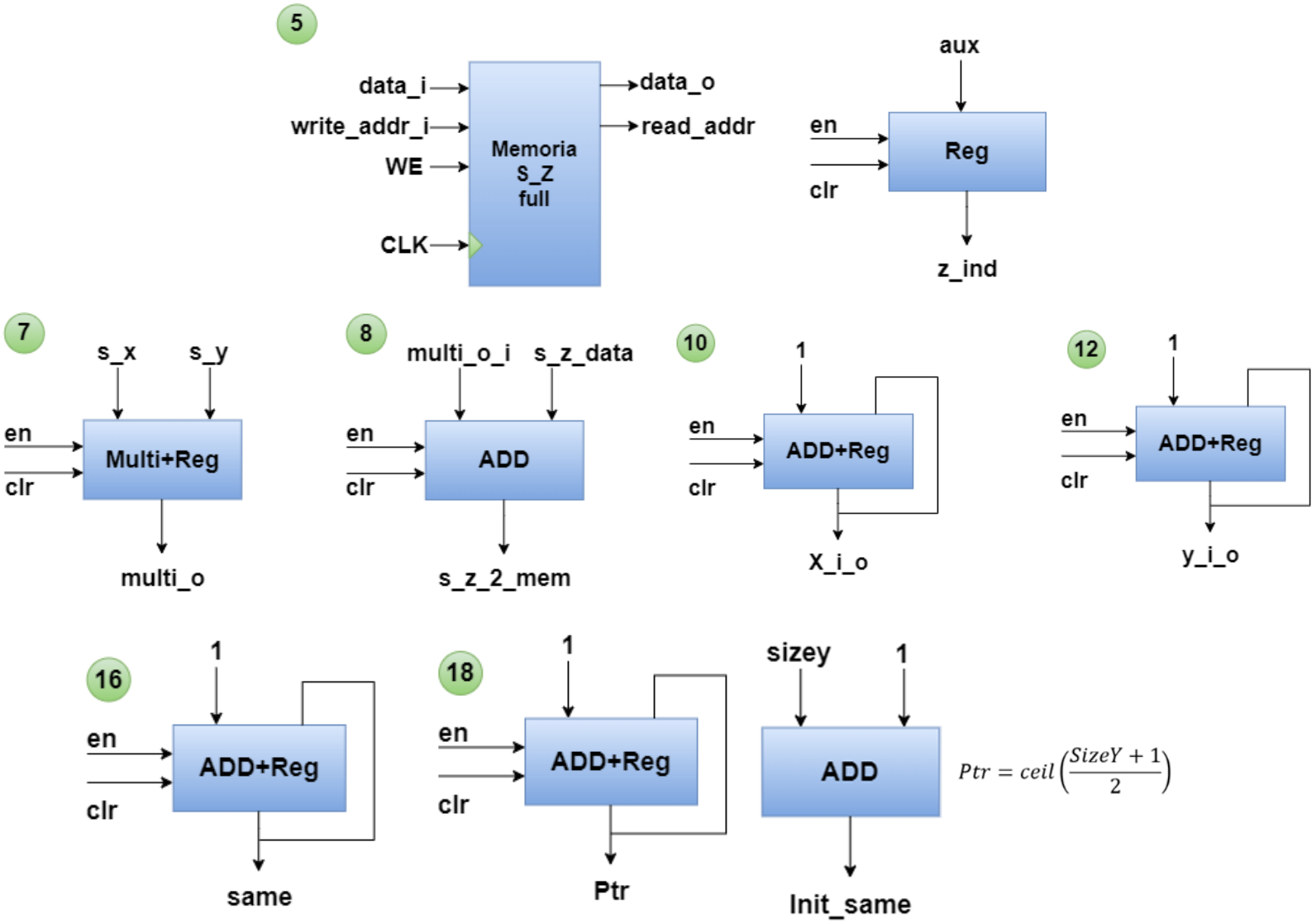
Se muestra el diagrama ASM en la siguiente figura:





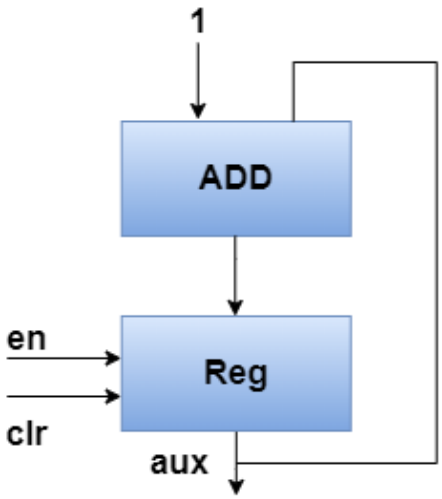
A continuación, se presentan los bloques traducidos a Hardware propuestos a partir del diagrama ASM del algoritmo para la entidad del convolucionador se muestra en lo posterior.





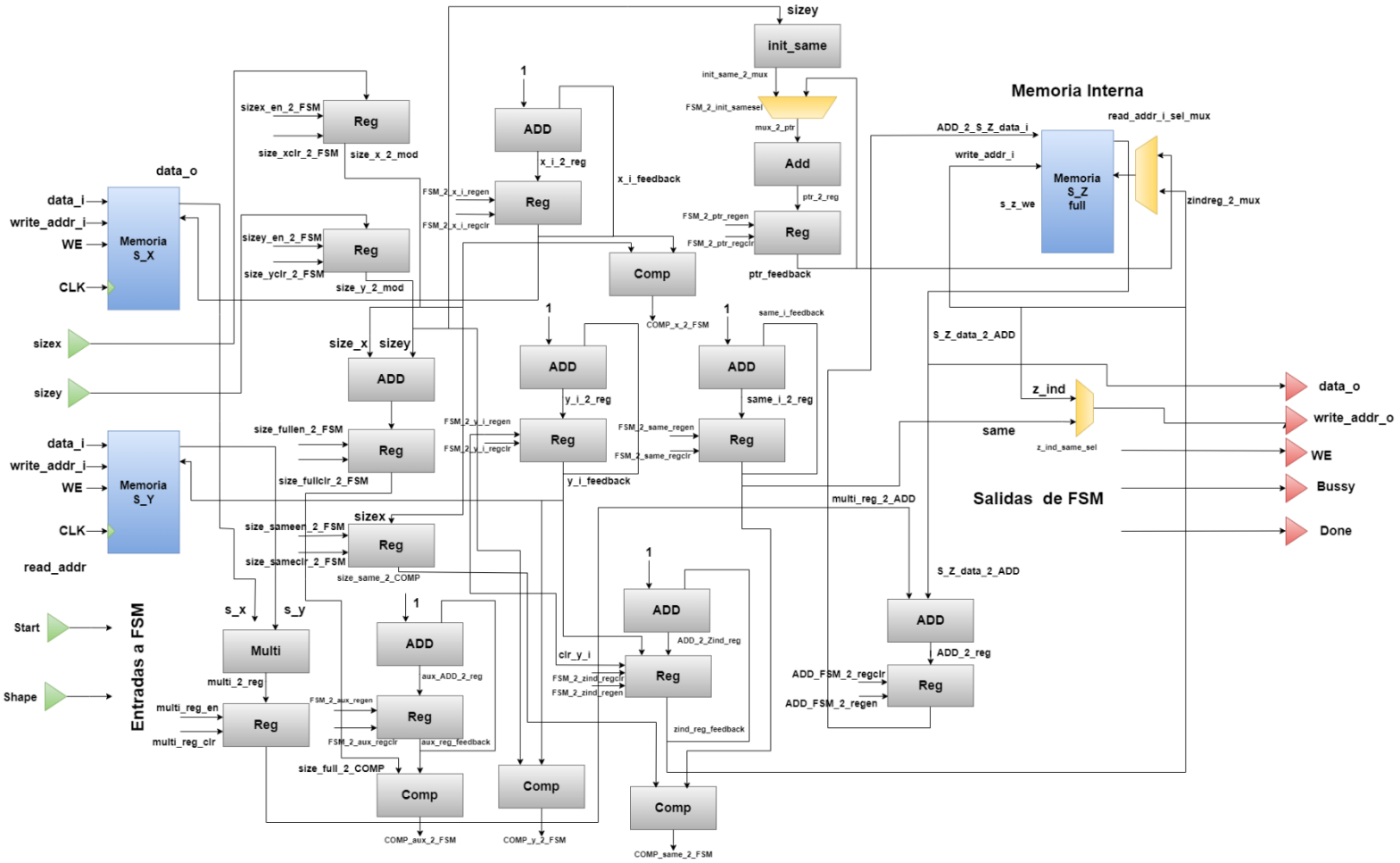
Debemos recordar que esta metodología establece el diseño de cualquier algoritmo a través de la unión de bloques simples RTL, por lo que es importante mencionar que cada bloque de sumador es la unión de un bloque combinacional de suma con un registro, y la unidad mas simple a usarse es tal. A continuación, se presenta una figura con el datapath que es la interconexión de estos bloques.

Se muestra la estructura de todos los sumadores con su respectivo registro:

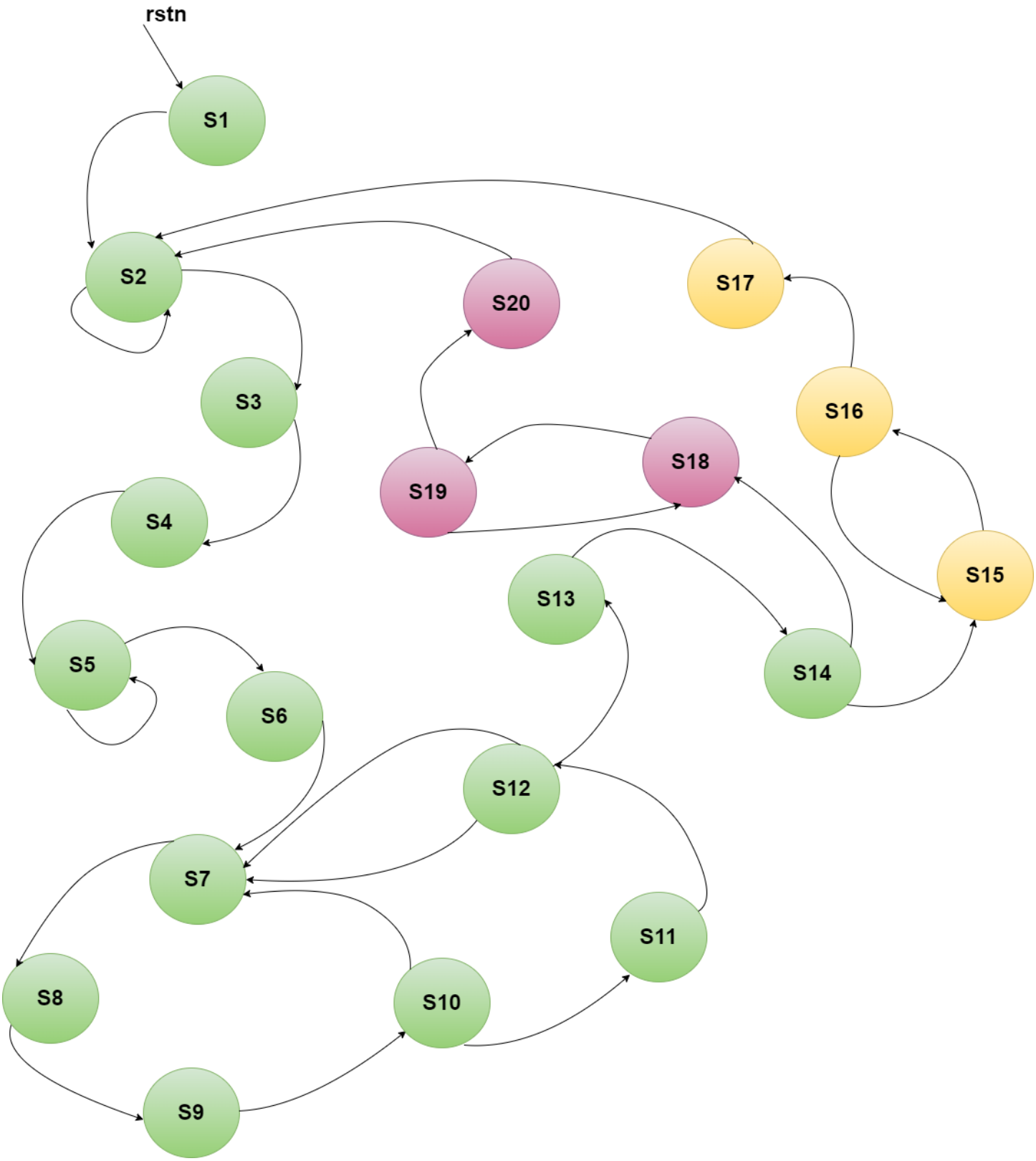


5.- Diagrama del Datapath y máquina de estados

Se presenta el diagrama del Datapath:



Se presenta el diagrama de estados de la máquina de estados:



Se presenta una lista con las salidas utilizadas en cada estado de la FSM Moore:

S1	busy <= 1'b0; done <= 1'b0;

S2	busy <= 1'b1;
	done <= 1'b0;
	aux_clr <= 1'b1;
	x_ind_clr <= 1'b1;
	y_ind_clr <= 1'b1;
	same_ind_clr <= 1'b1;
	ptr_clr <= 1'b1;
	sizex_clr <= 1'b1;
	sizey_clr <= 1'b1;
	size_full_clr <= 1'b1;
	size_same_clr <= 1'b1;
	zind_clr <= 1'b1;
	multi_reg_clr <= 1'b1;
	ADD_reg_clr <= 1'b1;
	ptr_en <= 1'b1;

S3	sizex_en <= 1'b1; sizey_en <= 1'b1;

S4	size_full_en <= 1'b1;
	size_same_en <= 1'b1;
	ptr_en <= 1'b1;

S5	aux_en <= 1'b0;
	we_s_z <= 1'b0;
	ADD_reg_clr <= 1'b0;

S6	aux_clr <= 1'b1; zind_clr <= 1'b1;
S7	multi_reg_en <= 1'b1;
S8	ADD_reg_en <= 1'b1;
S9	zind_en <= 1'b1; we_s_z <= 1'b1;
S10	x_ind_clr <= 1'b1; y_ind_en <= 1'b1;
S11	aux_2_zind <= 1'b1;
S12	Estado Dummy
S13	zind_clr <= 1'b1;
S14	read_addr_sel <= 1'b0; z_ind_same_reg_sel <= 1'b0;

S15	zind_en <= 1'b1;
	aux_en <= 1'b1;
	read_addr_sel <= 1'b0;

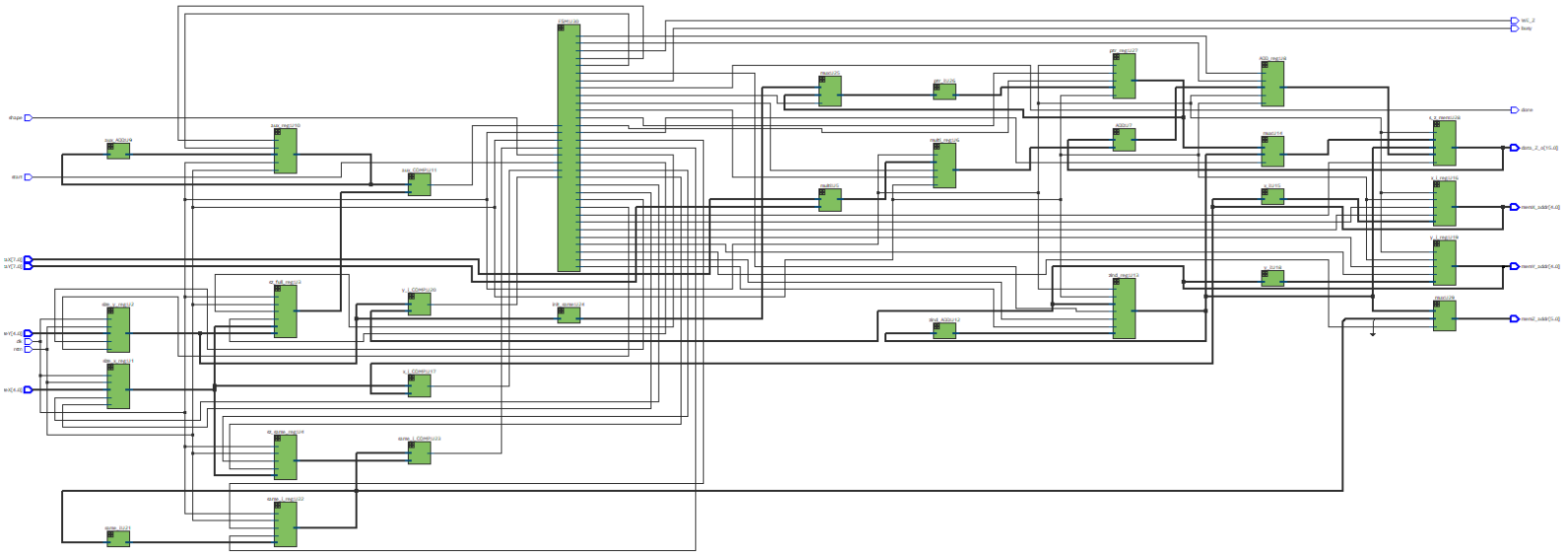
S16	WE_Z <= 1'b1; same_ind_en <= 1'b1;
S17	busy <= 1'b0; done <= 1'b1;
S18	ptr_en <= 1'b1 init_same_sel <= 1'b0;
S19	WE_Z <= 1'b1; init_same_sel <= 1'b0;

S20	same_ind_en <= 1'b1;
	busy <= 1'b0;
	done <= 1'b1;

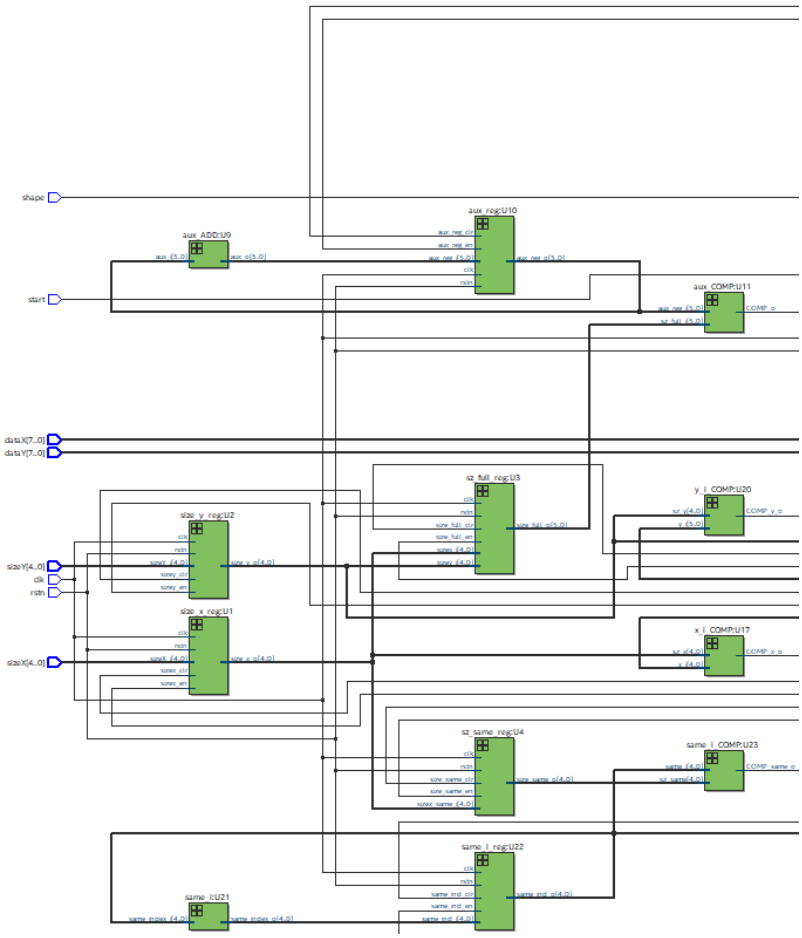
7.- Resultados de simulación y síntesis en FPGA.

a. *Esquemático Top Level generado por la herramienta*

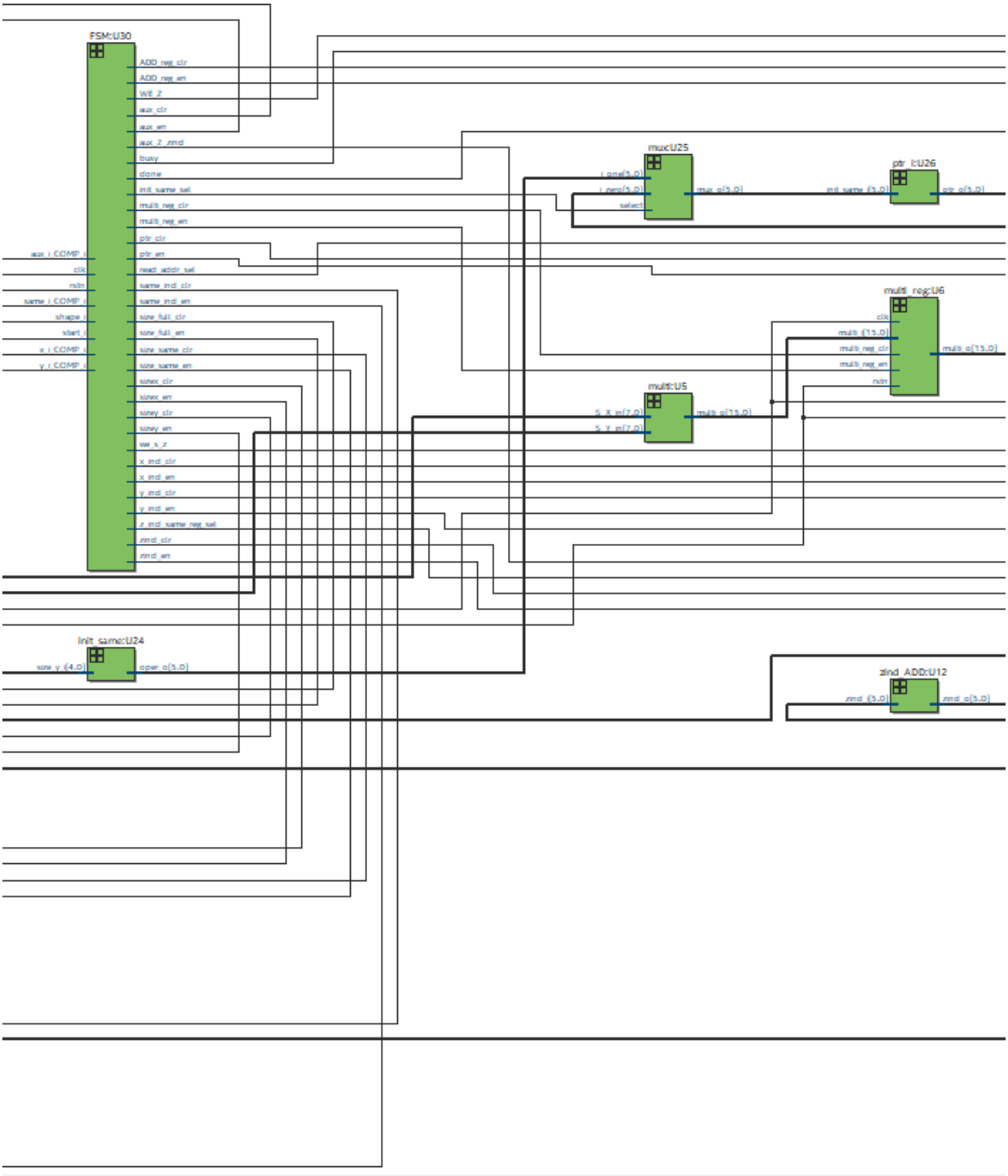
Se muestra el diagrama esquemático:



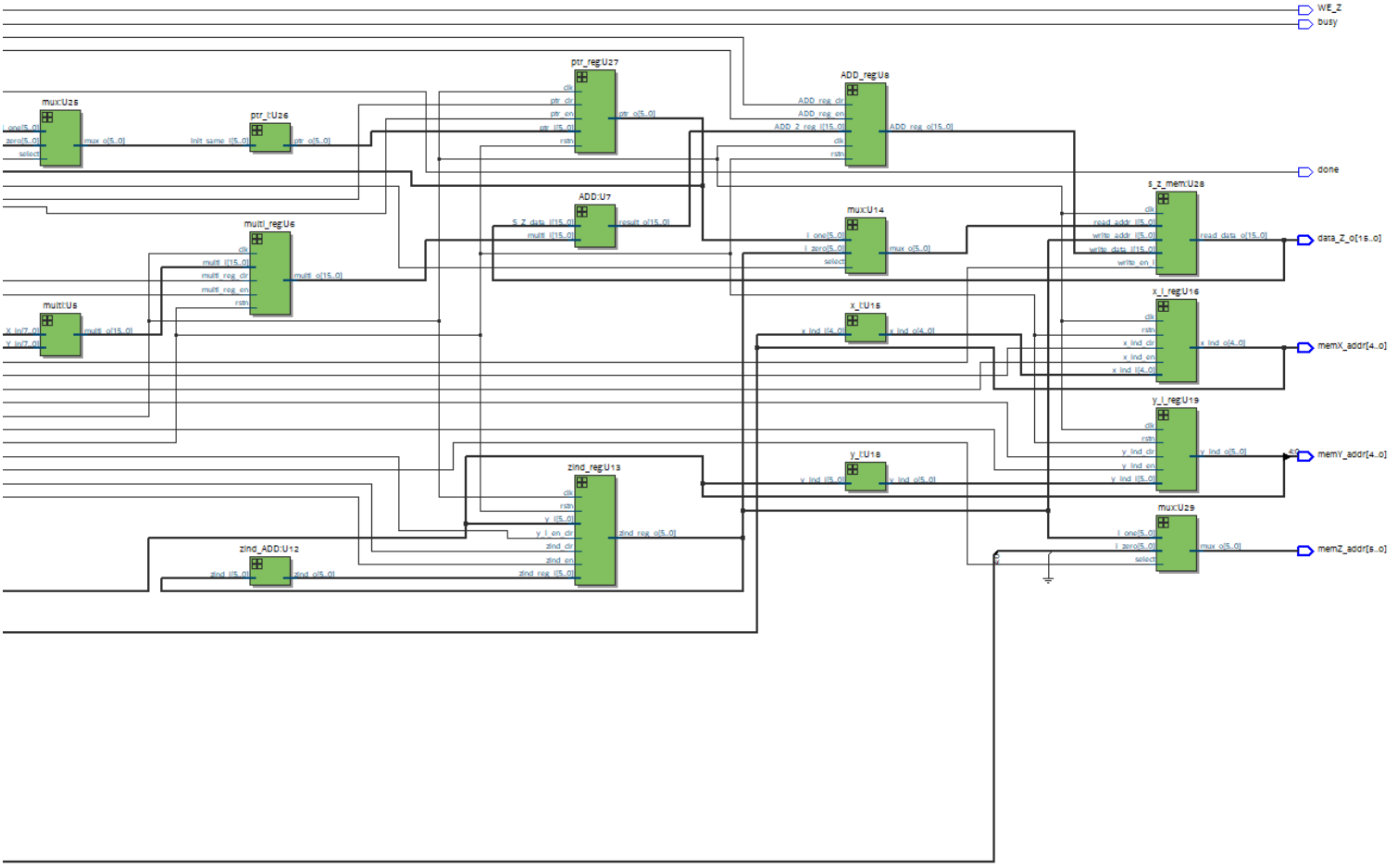
Primera parte:



Parte media del RTL:



Parte final del RTL:

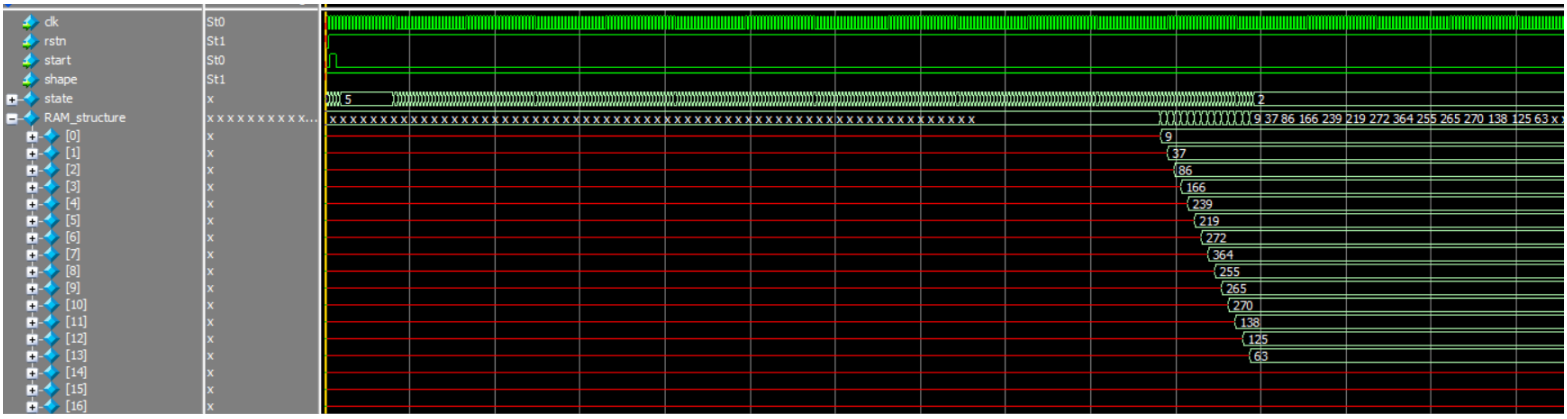


b. Programa que genere archivos para poder ser cargados en las memorias.

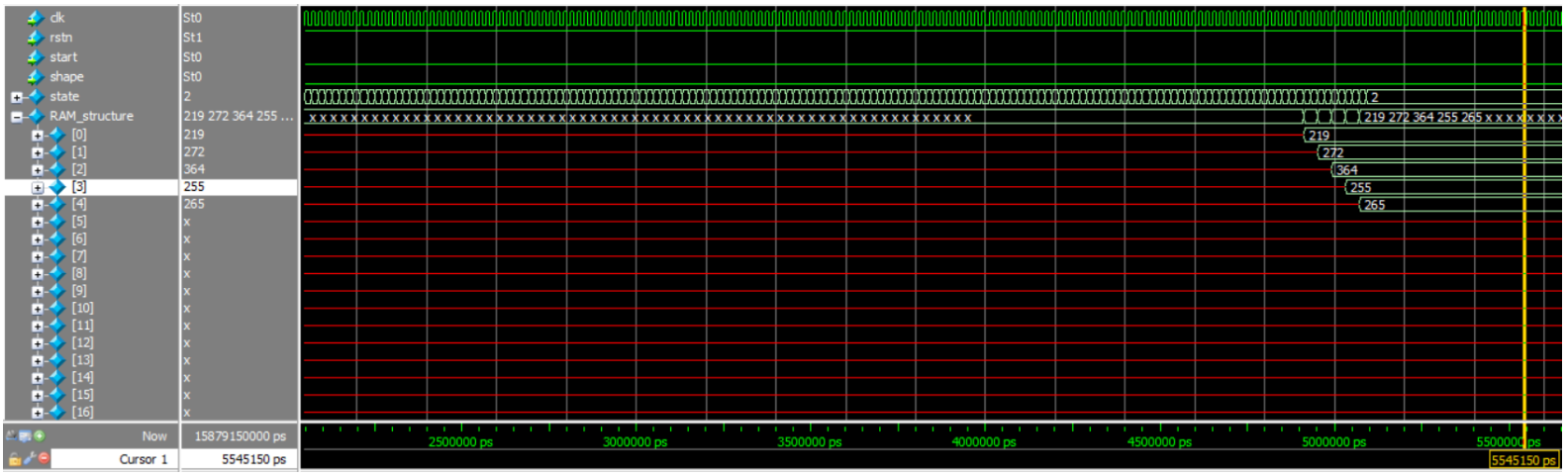
Dicho programa se ve en la figura siguiente:

```
%% Parameters
% Generating the signal vectors
X = 5;
Y = 10;
Start = 1;
Shape = 0; % 1 for "full" 0 for "same"

S_X = randi(10,1,X);
S_Y = randi(10,1,Y);

S_X_bin = dec2bin(S_X,8);
addressX = 'C:\Dani\Doctorado\convolucionador\simulation\modelsim\S_X.txt';
% filename1 = 'S_X.txt';
fileID1 = fopen(addressX, 'w');
for i = 1:numel(S_X_bin(:,1))
    fprintf(fileID1, '%s\n', S_X_bin(i,:));
end

addressY = 'C:\Dani\Doctorado\convolucionador\simulation\modelsim\S_Y.txt';
S_Y_bin = dec2bin(S_Y,8);
%%filename2 = 'S_Y.txt';
fileID2 = fopen(addressY, 'w');
for i = 1:numel(S_Y_bin(:,1))
    fprintf(fileID2, '%s\n', S_Y_bin(i,:));
end
```

c. Waveform de la simulación de dos señales de X tamaño 5 y Y tamaño 10.



Convolución completa



Convolución central

Siendo a simple vista imperceptible, sin embargo después de realizar algunos cálculos, se puede decir que el número de ciclos de reloj que tarda el módulo para poder realizar la convolución completa es de 271 ciclos. Para la convolución central toma solo 262 ciclos de reloj.

d. Área

El área total ocupada por este diseño se expresa a través de su número total de registros utilizados por Quartus al inferir el hardware y hacer uso de los recursos del chip FPGA cyclone IV E EP4CE115F29C7, en este caso podemos observar en la figura el reporte de compilación el cual dice que para nuestro diseño se hicieron uso de 76 registros, los cuales pueden representar desde registros hasta Look Up Tables (LUTs), de igual forma expresa un total de 187 elementos lógicos y al rederos de 1024 bits de memoria utilizados y haciendo uso de 1 solo multiplicador embebido de elementos de 9 bits.

Flow Summary

<<Filter>>

Flow Status	Successful - Sun May 05 13:50:27 2024
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	conv_top
Top-level Entity Name	conv_top
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	206 / 49,760 (< 1 %)
Total registers	125
Total pins	65 / 360 (18 %)
Total virtual pins	0
Total memory bits	1,024 / 1,677,312 (< 1 %)
Embedded Multiplier 9-bit elements	1 / 288 (< 1 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

e. Frecuencia máxima

Para saber el valor de la frecuencia máxima de operación es necesario tomarla en el peor caso que es cuando el Core opera a una temperatura de alrededor de 85 °C, de este modo pudimos observar en el menú llamado “slow 1200 mV 85C Model Fmax Summary” que el core presenta una frecuencia máxima de operación a 85 °C de 200.16 MHz, lo cual puede observarse en la figura.

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	186.57 MHz	186.57 MHz	clk	

f. Número de ciclos de reloj para hacer convolución a partir de que “start” es puesto en nivel alto hasta que la señal de “Done” es puesta en nivel alto. Para dos señales de entrada guardadas en memoria (aquí se deben cargar los archivos generados en el punto b) X e Y, una de 5 muestras y la otra de 10 muestras.

Siendo a simple vista imperceptible, sin embargo después de realizar algunos cálculos, se puede decir que el número de ciclos de reloj que tarda el módulo para poder realizar la convolución completa es de 271 ciclos. Para la convolución central toma solo 262 ciclos de reloj.