

# Metodología de diseño de System-On-chip

## Proyecto del curso.

Proyecto-P3.

Alumno: Jaciel Hernandez Resendiz

### 1. Descripción del Problema

Este proyecto tiene como objetivo el diseño e implementación de un coprocesador de convolución utilizando el lenguaje de HDL System verilog, considerando los siguientes puntos:

- Aplicando la metodología top-down vista en clase.
- Usando una descripción estructural.
- Considerar los dos tipos de convolución FULL y SAME.

### 2. Diagrama de caja negra y definición de señales de entrada y salida. Breve explicación.

A continuación, se describen las entradas del diagrama de caja negra:

- dataX valor leído desde la memoria X.
- sizeX tamaño de la memoria X (registros).
- dataY valor leído desde la memoria Y.
- sizeZ tamaño de la memoria Y (registros).
- Datos almacenados en memoria Y.
- Start es la señal de inicio.
- Shape es la señal que controla el tipo de convolución SAME o FULL.

Salidas:

- Señal Done (One-shot) indica que la operación de convolución terminó.
- Señal de busy indica si el coprocesador de convolución está ocupado o no.
- MemX\_addr y memY\_addr son las direcciones de memoria para leer desde las memorias X y Y
- Resultado de la convolución dataZ.
- memZ\_addr es la dirección de memoria Z para guardar el resultado que contiene la salida dataZ.

A continuación, se muestra el diagrama de caja negra del coprocesador de convolución.

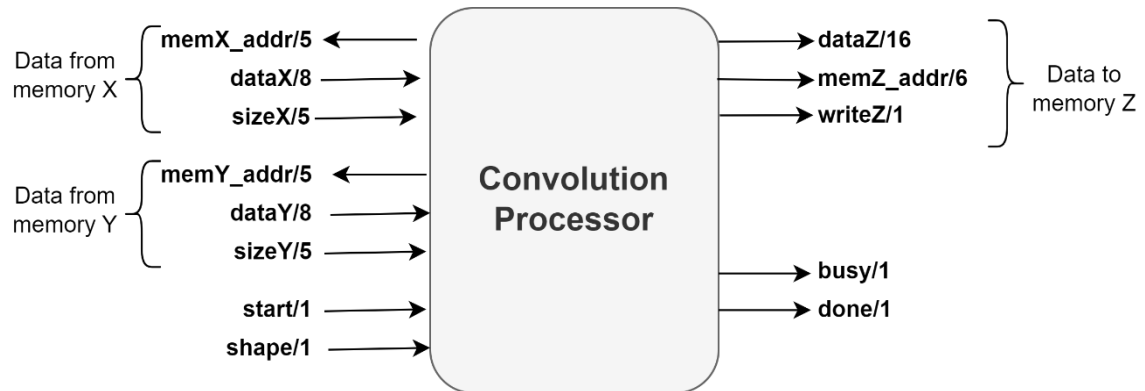


Figura 1 Diagrama de caja negra para el coprocesador de convolución

### 3. Pseudocódigo y breve explicación.

A continuación, se presenta el Pseudocódigo y su explicación de la idea para dar solución al problema planteado.

```

1. while start = 0 or busy 1
2. End while
3. dataZ_temp
4. busdatay=1
5. for k hasta (sizeX+sizeY-1)
6.     inicio = max(0, k-sizeY+1)
7.     fin     = min(k,sizeX)
8.     for j hasta range(inicio:fin)
9.         dataZ_temp[k] = dataZ_temp [k] + (dataX[j]* dataX[k-j])
10.    end for
11. end for
12. if shape =1
13.     dataZ = dataZ_temp[((sizeX-1)/2)-1:(( sizeX -1)/2)+ sizeX -1];
14.     done =1;
15. else:
16.     dataZ = dataZ_temp;
17.     done =1;
18. End if
19. If done =1
20.     busy =0
21.     goto 1
22. else
23.     go to 19
24. end if

```

Figura 2 Pseudocódigo de la implementación del coprocesador de convolución,

A continuación, se explica los pasos más importantes:

#Esperamos la señal de inicio por el maestro o esperamos si esta una operación en curso.

1. while start = 0 or busy 1
2. End while

```

3. dataZ_temp[0:16]
4. busdatay=1
#daIniciamos la operación de convolución entre dataX y dataY, donde el siguiente for k tomará los valores desde 0 hasta el total de
iteraciones de la convolución que es sizeX+sizeY-1
5. for k hasta (sizeX+sizeY-1)
    #Se calcula el valor de los índices que se usarán para seleccionar la información de dataX y dataY para las multiplicaciones
    y sumas entre dataX y dataY.
    #La variable inicio solo tomará el valor de 0 hasta que k sea mayor a sizeY, posterior a esto la variable inicio irá
    incrementando de 1 en 1 hasta que k llegue a sizeX+sizeY-1
6. inicio = max(0, k-sizeY+1)
    #Se calcula el valor de los índices hasta donde llegará las multiplicaciones y sumas entre dataX y dataY de acuerdo a la
    iteración actual, la variable fin tomará al inicio solo valores que va tomando la variable k, cuando k se mayor a sizeX, la
    variable fin tomará siempre el valor de sizeX-1
7. fin = min(k,sizeX)
8. for j hasta range(inicio:fin)
    #Se realiza la sumatoria para cada una de las iteraciones de la convolución, guardando la sumatoria en la
    posición dataZ[k], donde el índice j toma valores del rango inicio:fin. Mientras k-j toma valores de k (el número
    actual de la iteración de la convolución) - j que toma valores del rango inicio:fin.

    #Lo importante es el índice k-j, ya que con este ayuda a seleccionar los índices de dataY con base a la iteración
    de la convolución actual.
9. dataZ_temp[k] = dataZ_temp [k] + (dataX[j]* dataX[k-j])
10. end for
11. end for
    #Se valida si shape es igual 1 = SAME: La convolución SAME es la parte central de la convolución del mismo tamaño que sizeX.
    Para obtener la parte central de la convolución, se toma el siguiente rango de la convolución FULL dataZ_temp[(((sizeX-1)/2)-1:((
    sizeX -1)/2)+ sizeX -1];
12. if shape =1
13. dataZ = dataZ_temp[(((sizeX-1)/2)-1:(( sizeX -1)/2)+ sizeX -1];
14. done =1;
15. else:
    # De lo contrario se toma todo el resultado de la convolución (FULL).
16. dataZ = dataZ_temp;
17. done =1;
18. End if
    #Validamos si se ha terminado la convolución. Si es verdad esperamos que el maestro vuelva a enviar la señal de inicio.
19. If done =1
20. busy =0
21. goto 1
22. else
23. go to 19
24. end if

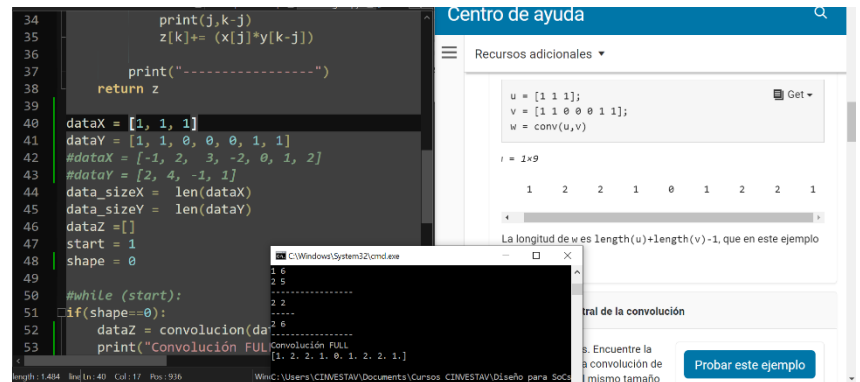
```

#### 4. Agregar como validaron el funcionamiento su pseudocódigo en Matlab, C, C++, Python etc.

Para la validación del pseudocódigo, este se implementó en el lenguaje de Python. A continuación, se muestra la implementación y la comprobación de su funcionamiento usando los ejemplos reportados en <https://la.mathworks.com/help/matlab/ref/conv.html>

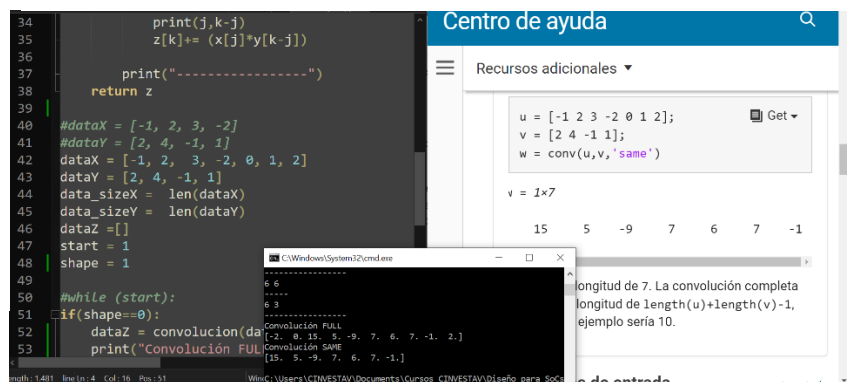
```
1 import numpy as np
2 import math
3 def convolucion(n,y,shape):
4     n = len(y) #Obtenemos el tamaño del dataX
5     m = len(y) #Obtenemos el tamaño del dataY
6     z = []
7     z = np.zeros(n+m-1) #Generamos dataZ del tamaño del dataX+dataY-1
8     inicio = 0
9     #h tomara los valores desde 0 hasta el total de iteraciones de la convolucion,
10    # que es n+m-1
11    for k in range(n+m-1):
12        if(shape ==1 and ((math.floor((n-1)/2)-1) or (k > (math.floor((n-1)/2)-1)+1))):
13            continue
14        #Se calcula el valor de los índices que se usaran para seleccionar la información de
15        #dataX y dataY para las
16        #Multiplicaciones y sumas entre dataX y dataY
17        #La variable inicio tomara solo tomara el valor de 0 hasta que h sea mayor a m, posterior
18        #a esto la
19        #La variable inicio irá incrementando de 1 en 1 hasta que h llegue a n+m-1
20        inicio = max(0, k-m+1)
21        #Se calcula el valor de los hasta donde llegará las mutiplicaciones y sumas entre dataX y
22        #dataY
23        #La variable fin tomara al inicio solo valores que va tomando la variable h, cuando h sea
24        #mayor a n,
25        #La variable fin tomara siempre el valor de n-1
26        fin = min(k, n-1)
27        print("Inicio: ",inicio)
28        print("Fin: ",fin)
29        print("Rango: ")
30        print(list(range(inicio,fin+1)))
31        for j in range(inicio,fin+1):
32            print("Indice: ", j, " Indice: ",k-j)
33            #Se realiza la sumatoria para cada una de las iteraciones de la convolucion
34            #Guardando la sumatoria en las posición dataZ[k], donde el índice j toma valores
35            #del rango inicio:fin
36            #Mientras h-j toma valores de h(el número actual de la iteración de la convolucion)
37            # - j que toma valores del rango inicio:fin
38            #Lo importante es el índice h-j, ya que con este te ayuda a seleccionar los índices
39            #de dataY con base al la iteración de
40            # la convolucion actual.
41            z[k] += (y[j])*(y[k-j])
42        return z
43 #dataX = [1, 1, 0, 0, 0, 1, 1]
44 #dataY = [1, 1, 1]
45 dataX = [-1, 2, 3, -2, 0, 1, 2]
46 dataY = [2, 4, -1, 1]
47 data_sizeX = len(dataX)
48 data_sizeY = len(dataY)
49 dataZ = []
50 start = 1 # Estado de inicio
51 shape = 1 # Tipo de convolución
52 dataZ = convolucion(dataX,dataY,shape)
53 print("Convolución")
54 print(dataZ)
```

convolución usando el lenguaje Python.



The screenshot shows a code editor with Python code for full convolution. The code defines a function `convolucion` that takes `n`, `y`, and `shape` as inputs. It calculates the output `z` by iterating over the input arrays and performing element-wise multiplication and summation. The code is run in a terminal window, and the output is displayed. To the right, a MATLAB help window for the `conv` function is visible, showing the syntax `w = conv(u,v)` and the result of the convolution for the example inputs `u = [1 1 1]` and `v = [1 1 0 0 1 1]`.

Figura 3 Ejemplo de los resultados de la implementación de convolución FULL comparado con resultados de Matlab.



The screenshot shows a code editor with Python code for same convolution. The code defines a function `convolucion` that takes `n`, `y`, and `shape` as inputs. It calculates the output `z` by iterating over the input arrays and performing element-wise multiplication and summation. The code is run in a terminal window, and the output is displayed. To the right, a MATLAB help window for the `conv` function is visible, showing the syntax `w = conv(u,v,'same')` and the result of the convolution for the example inputs `u = [-1 2 3 -2 0 1 2]` and `v = [2 4 -1 1]`.

Figura 5 Ejemplo dos de los resultados de la implementación de convolución SAME comparado con resultados de Matlab.

## 5. Diagrama ASM.

En la figura 6 se presenta el diagrama ASM de la solución propuesta, se identificaron 21 posibles estados.

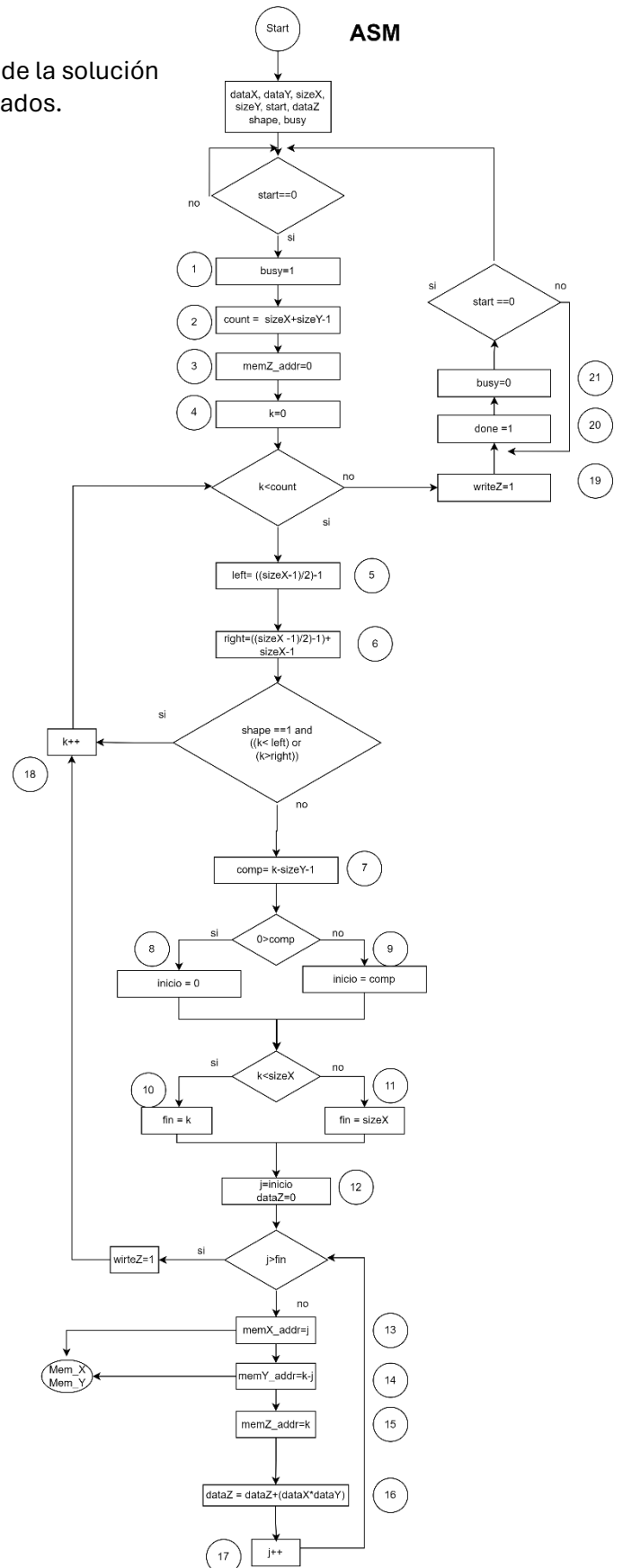


Figura 6 Diagrama ASM de lógica para la implementación del coprocesador de convolución.

## 6. Diagrama del datapath y máquina de estados. (Explicar si combinaron estados, que bloques óptimos decidieron agregar, etc.)

En la figura 7 se muestra el diagrama datapath de acuerdo con los pasos identificados en el diagrama ASM (figura 6). En el diagrama datapath se identificaron 21 posibles bloques considerando por separado cada tarea identificada en el diagrama ASM. Entre los bloques identificados se encuentran: sumas, restas, división y comparación y registros.

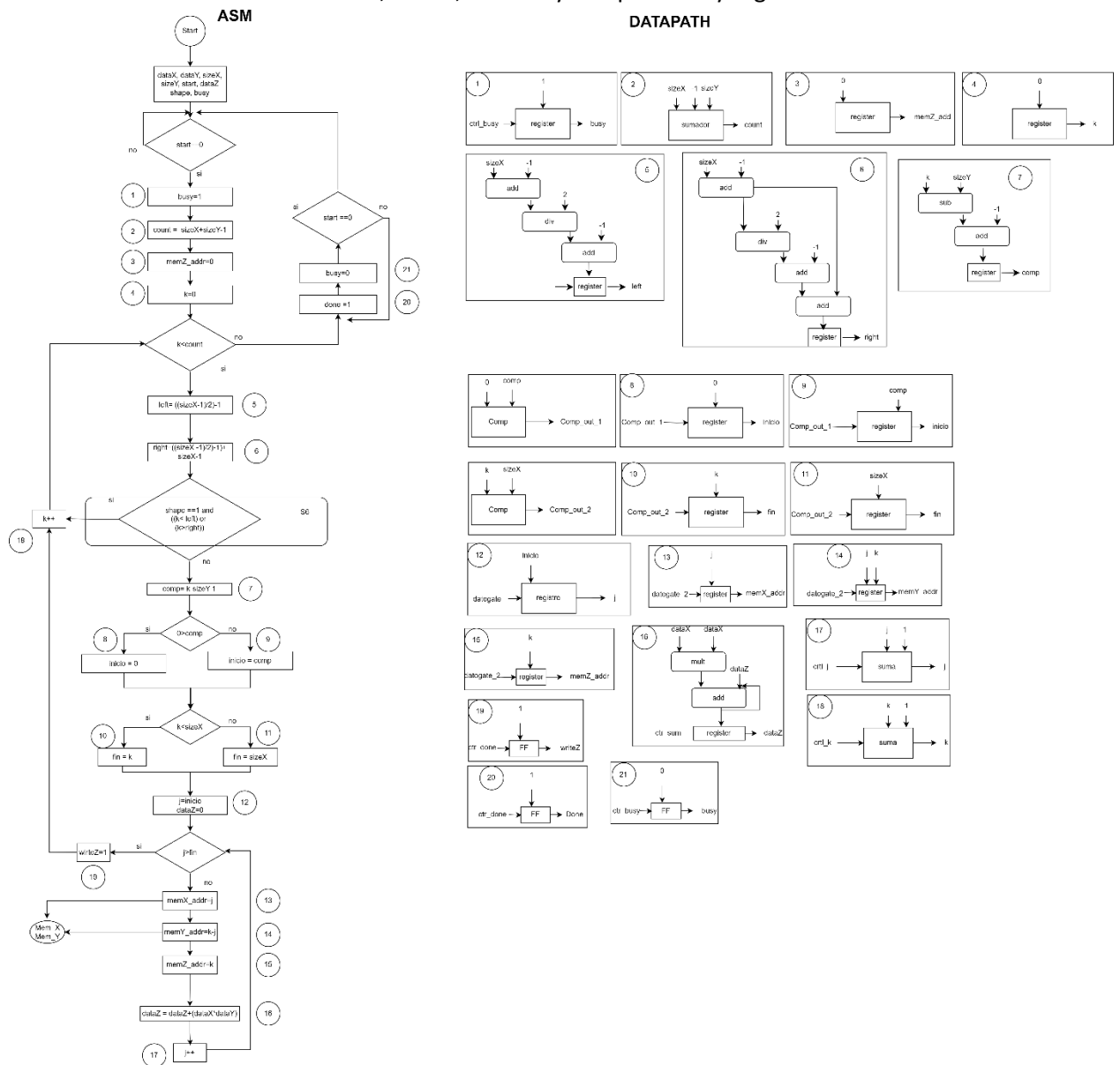


Figura 7 Diagrama datapath sin optimizar bloques.

A continuación, en la figura 8 se muestran los bloques optimizados, la optimización partió en agrupar bloques que tiene las mismas entradas, pero diferentes salidas, para esto nos apoyamos de bloques mux.

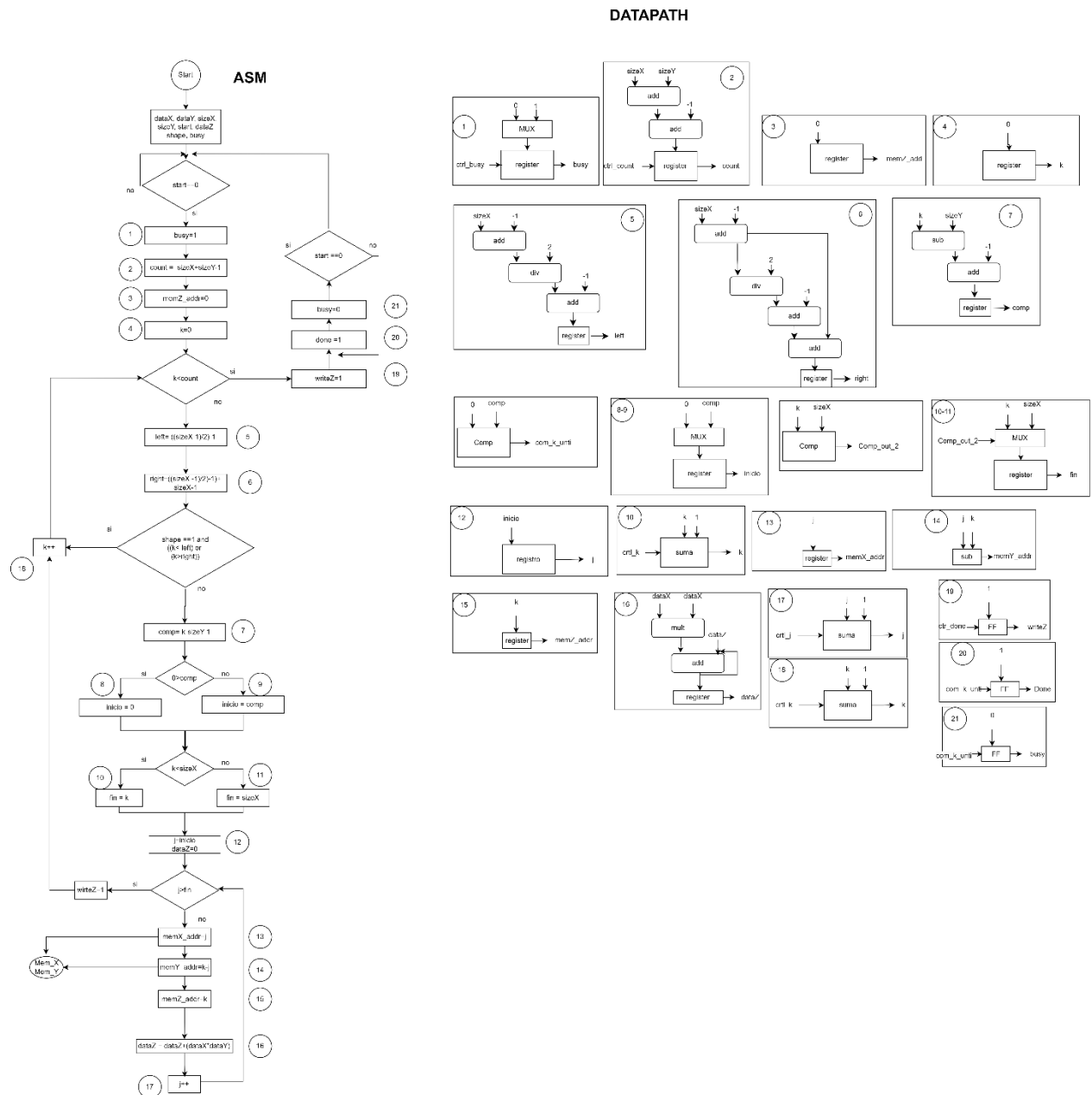


Figura 8 Diagrama datapath optimizando bloques.

En la figura 9 se aprecia los bloques conectados de acuerdo con los datos de entrada, salida y banderas para su funcionamiento.

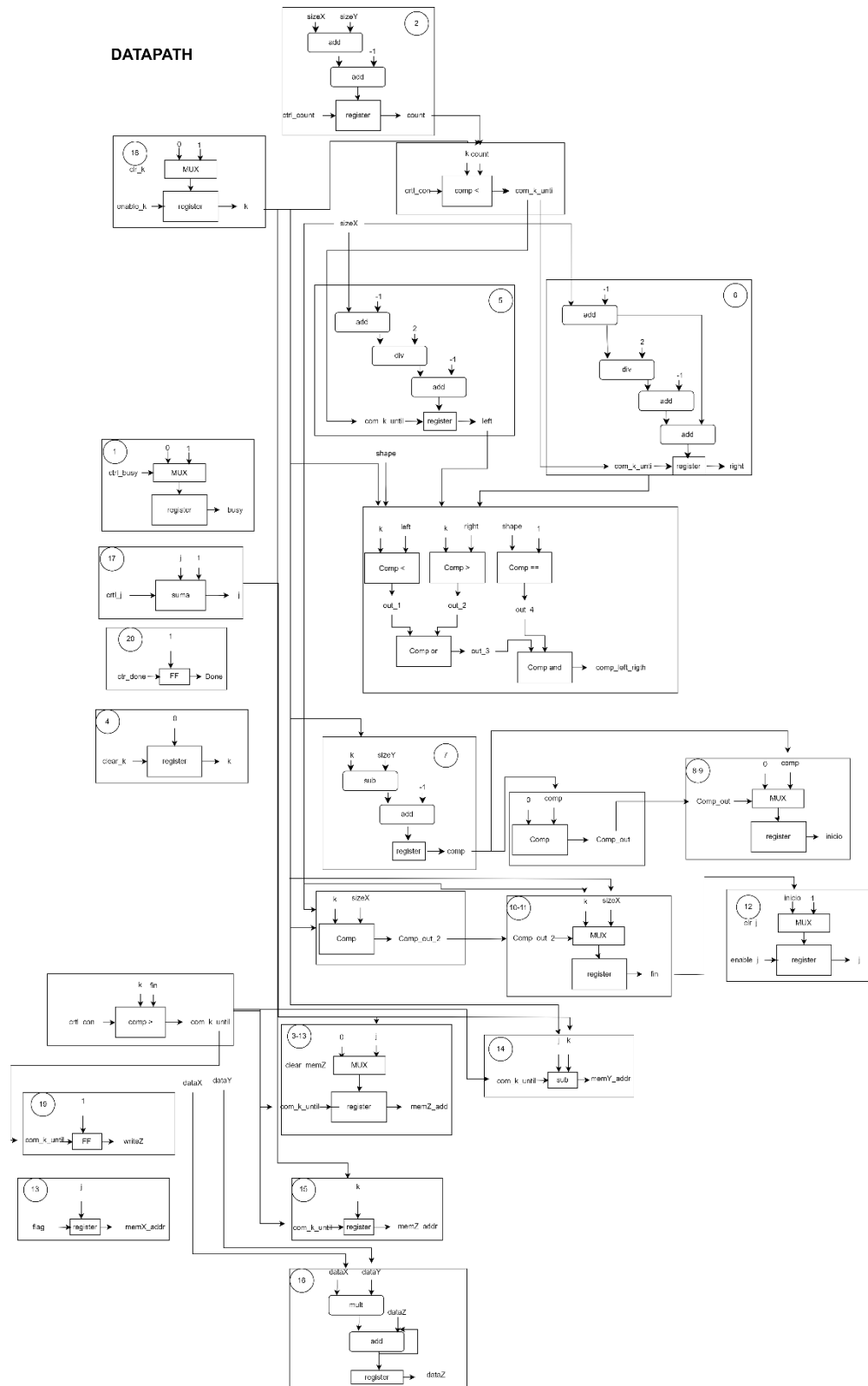


Figura 9 Datapath conectado



A continuación, se muestra el diagrama de máquina de estados (FSM) de la propuesta, en la figura 6 se muestra el diagrama ASM con 21 estados inicialmente identificados, al momento de hacer la máquina de estados se indientificaron 10 estados que pueden cubrir todas las tareas identificadas en el diagrama ASM (ver figura 6). En la figura 11 se muestra los estados generales identificados del diagrama ASM (enmarcados en cuadros punteados de color rojo). Mientras que en la figura 10 se muestra la máquina de estados con sus entradas y salidas.

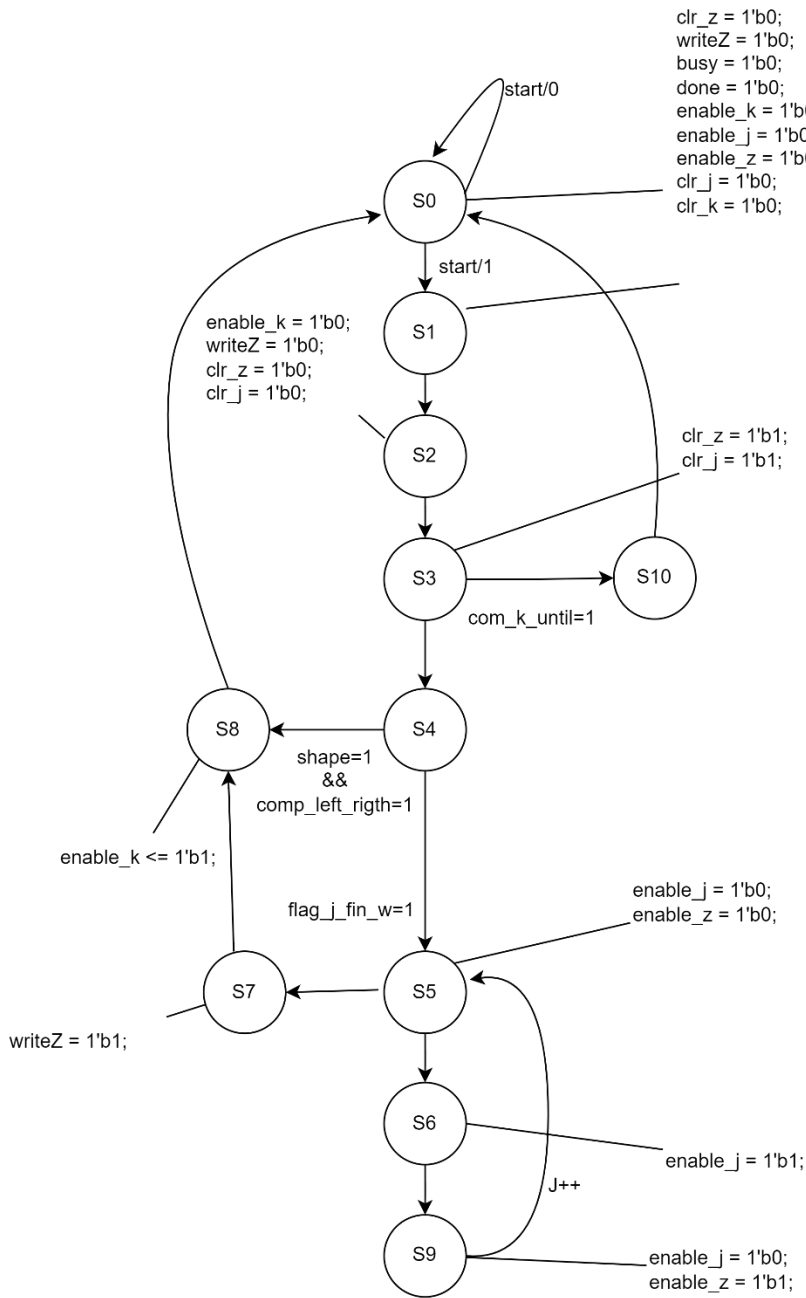


Figura 10 Diagrama FSM

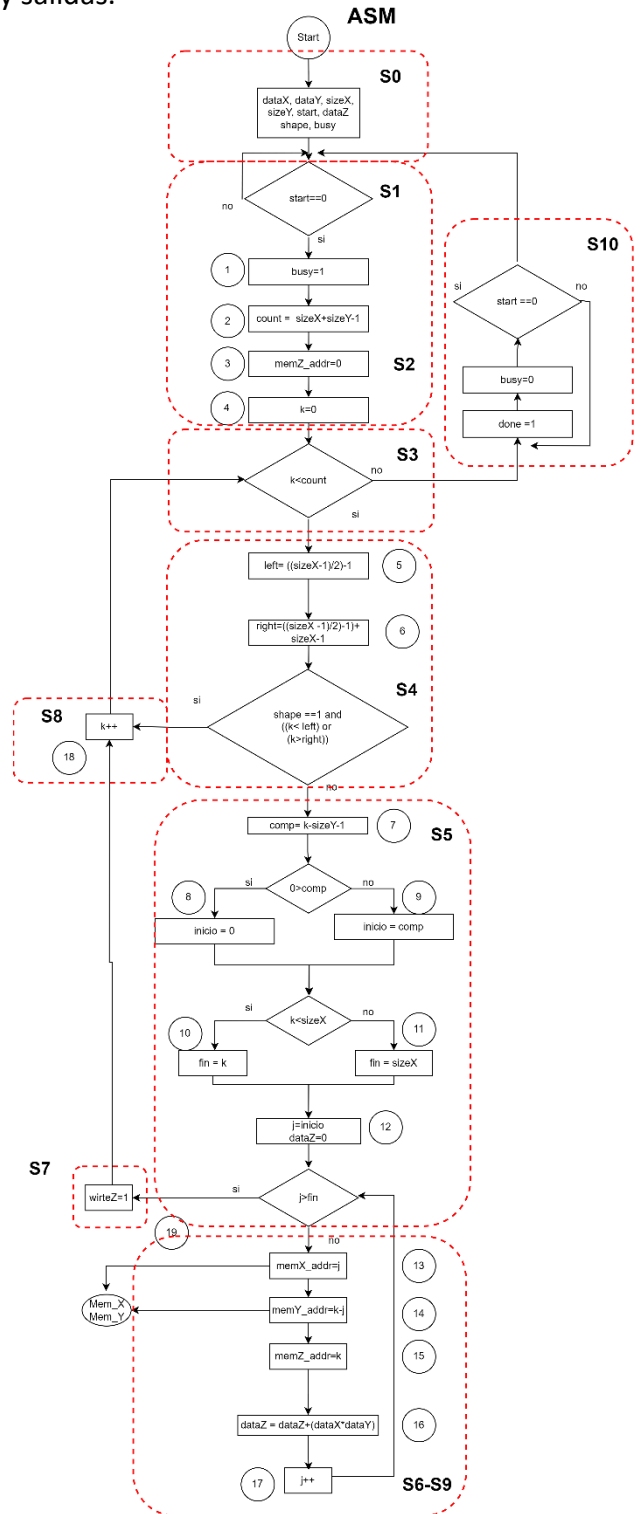


Figura 11 Diagrama ASM optimizado

## 7. (Opcional) Alguna mejora al diseño.

## 8. Resultados de simulación y síntesis en FPGA.

### a. Esquemático Top Level generado por la herramienta.

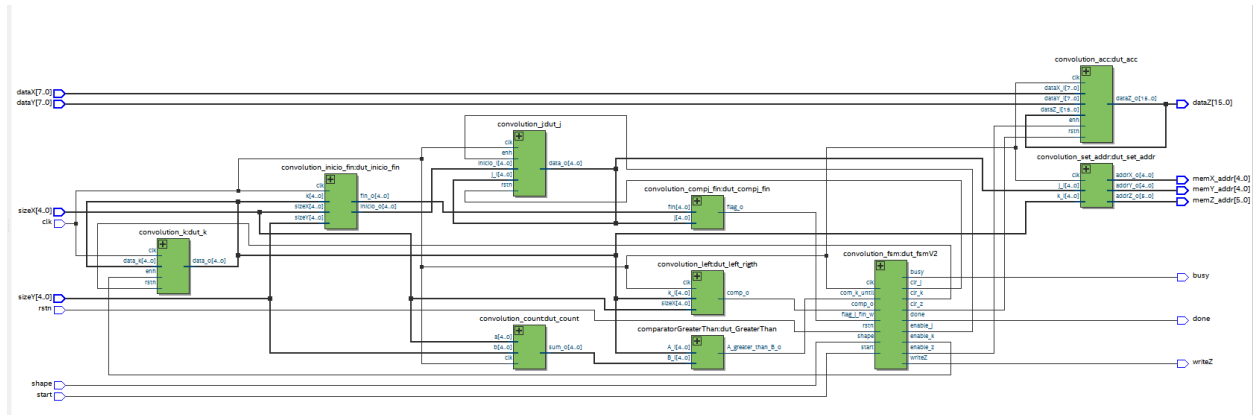


Figura 12 Esquemático TOP level.

### b. Escribir un programa que genere archivos para que puedan ser cargados en las memorias de entrada de su cama de prueba.

Para este punto, se desarrolló un programa en el lenguaje Python, que recibe como entrada las longitudes de las memorias X y Y, y se generan números aleatorios entre 1 y 100, y estos valores se vacían en un archivo txt usando su representación hexadecimal.

```
import random

def generate(sizeX, sizeY):
    memX = [random.randint(1, 100) for _ in range(sizeX)]
    memY = [random.randint(1, 100) for _ in range(sizeY)]
    with open("memX.txt", "w") as archivo:
        for val in memX:
            val_hex = hex(val)[2:]
            archivo.write(val_hex + '\n')
    print("Se ha generado el archivo de la memoria X")
    with open("memY.txt", "w") as archivo:
        for val in memY:
            val_hex = hex(val)[2:]
            archivo.write(val_hex + '\n')
    print("Se ha generado el archivo de la memoria Y")
generate(10,5)
```

Figura 13 Programa para la generación de archivos de memorias X y Y

### c. Waveform de la simulación del punto (f).

Para este punto se utilizó las siguientes memorias X y Y, y su resultado fue comprobado en Matlab usando la configuración FULL.

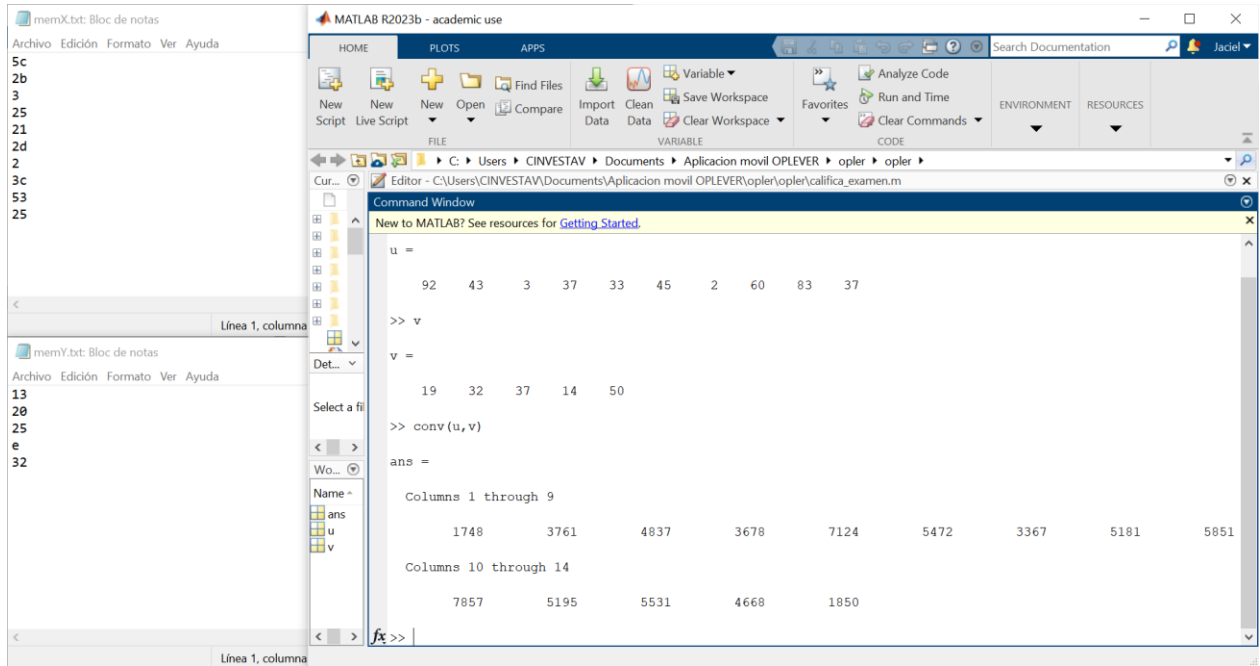


Figura 14 Memorias X y Y con longitudes 10 y 5.

En las siguientes figuras se muestra la forma de onda con los resultados obtenidos de la operación de convolución diseñado e implementado.



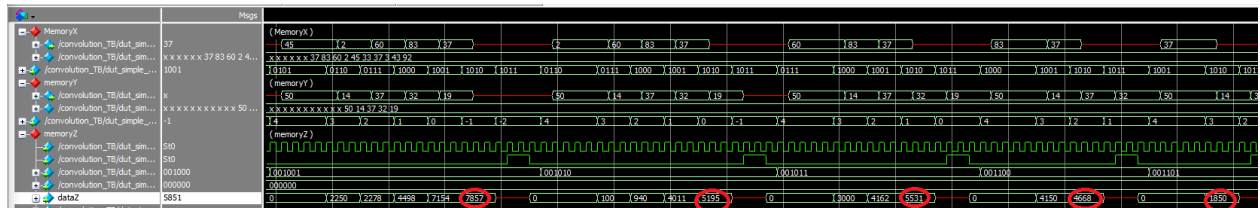


Figura 15 Resultados de la simulación modelSim(Waveform)

Para la siguiente prueba se usaron las mismas memorias X y Y, y utilizando la configuración SAME del procesador de convolución. Los resultados obtenidos para esta prueba se aprecian en las siguientes imágenes.

#### Validación en Matlab.

```
>> conv(u,v,'SAME')

ans =

    4837     3678     7124     5472     3367     5181     5851     7857     5195     5531
```

#### Resultados del coprocesador de convolución.



En ambas pruebas FULL y SAME los resultados obtenidos por el procesador de convolución diseñado e implementado son los mismo que se reportan usando la herramienta MATLAB.

#### d. Área.

Flow Summary	
<<Filter>>	
Flow Status	Successful - Thu May 9 13:24:50 2024
Quartus Prime Version	23.1std.0 Build 991 11/28/2023 SC Lite Edition
Revision Name	convolution
Top-level Entity Name	convolution
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	66 / 56,480 (< 1 %)
Total registers	68
Total pins	65 / 268 (24 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	1 / 156 (< 1 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Figura 16 Información de Área

e. **Máxima Frecuencia.**

Slow 1100mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	114.39 MHz	114.39 MHz	clk	

Figura 17 Frecuencia máxima

- 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, una de 5 muestras y la otra de 10 muestras. (Pueden poner los datos que sean, pero que sean esas longitudes).**

Si consideramos que nuestra cama de prueba está configurada para que cada pulso de reloj se genere en 20ns y que los resultados y la bandera DONE en alto se obtiene hasta los 5800 ns, por lo cual el total de pulsos de reloj que le toma procesador de convolución es de 290 pulso de reloj.

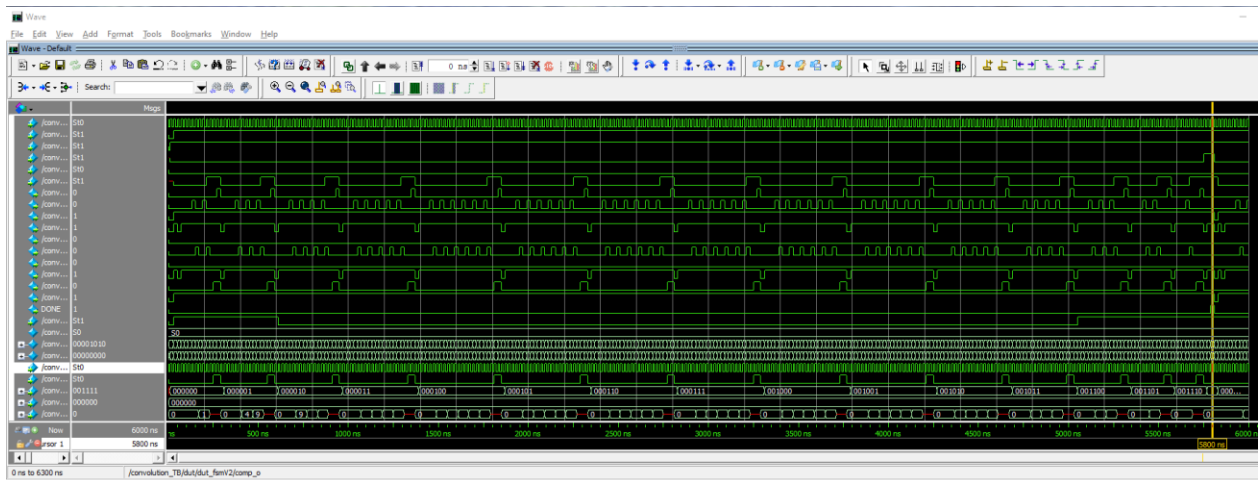


Figura 18 Señal DONE en alto.

Consideraciones (problemas que me faltan por resolver):

- Se tiene problemas al momento de escribir el resultado dataZ a la memoria Z en momento de que writeZ está en alto (Tengo un desfase). Me falta sincronizar bien esta parte, las pruebas que se hicieron indican que los resultados obtenidos en dataZ son los correctos.