

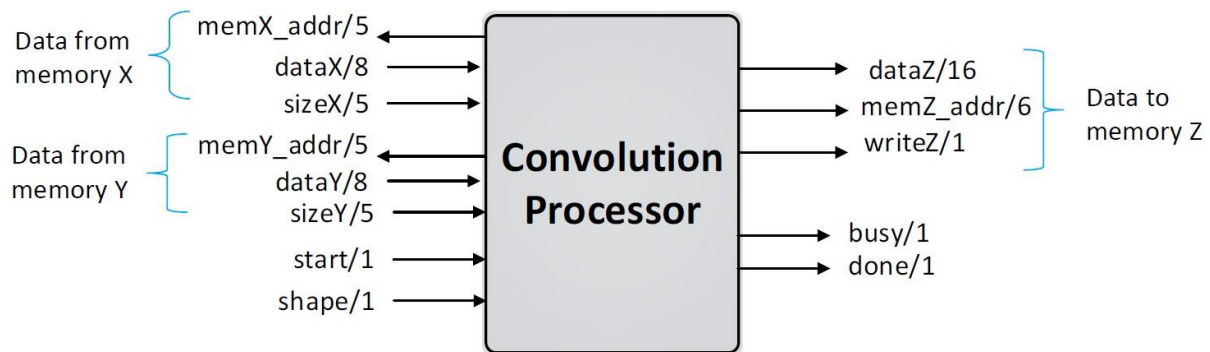
## REPORTE IMPLEMENTACIÓN CO-PROCESADOR DE CONVOLUCIÓN

### 1 Descripción del problema

Implementar un co-procesador de convolución que contribuya a mejorar el desempeño de un sistema de procesamiento que involucre esta tarea en particular. Dentro de las características más relevantes de este co-procesador están, i) Los datos son recibidos uno a uno para realizar dichas operaciones y ii) cada resultado también es enviado uno a uno, esto significa que en este diseño no se cuenta con una memoria de almacenamiento .

### 2 Diagrama de caja Negra

Abajo se identifican las señales que se intercambian con el procesador principal. El co-procesador es capaz de realizar las tareas de convolución “Full” y “Same”.



Señal	E/S	bits	Descripción
memX_addr	Salida	5	Dirección memoria datos X
dataX	Entrada	8	Dato X
sizeX	Entrada	5	Tamaño del vector X
memY_addr	Salida	5	Dirección memoria datos Y
dataY	Entrada	8	Dato Y
sizeY	Entrada	5	Tamaño del vector Y
memZ_addr	Salida	6	Dirección memoria datos Z
dataZ	Salida	16	Dato Z
Start	Entrada	1	Señal de inicio de convolución
Shape	Entrada	1	Tipo de convolución FULL=0 , SAME =1
Busy	Salida	1	Estado del co-procesador 1=ocupado, 0= disponible
Done	Salida	1	Estado de la tarea del co- procesador 1= terminado, 0= en proceso

### 3 Pseudocódigo

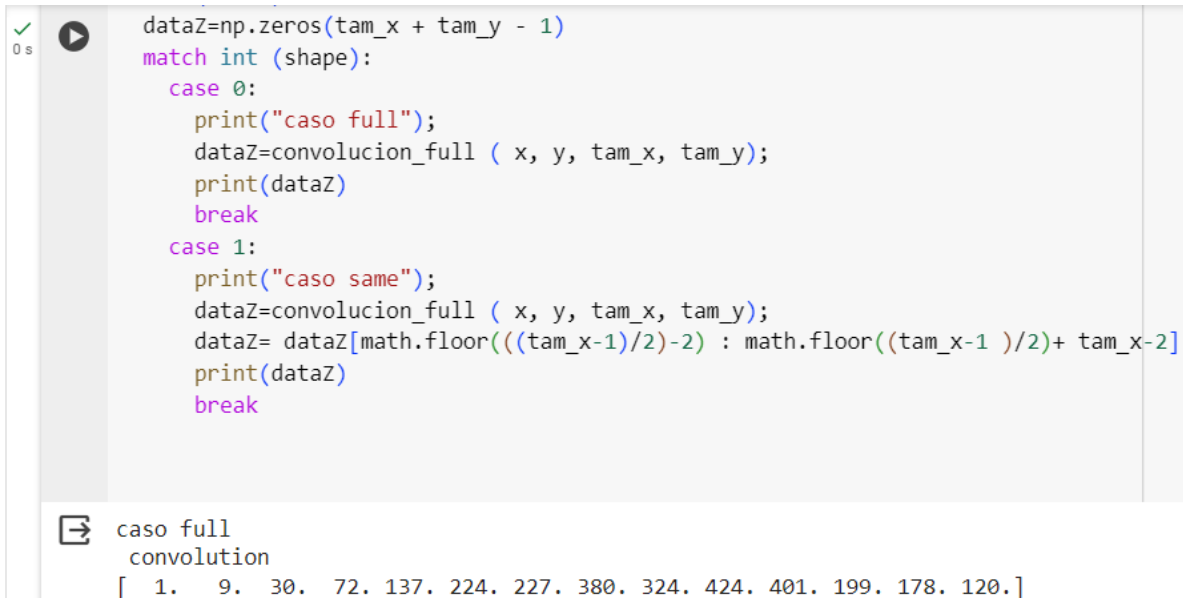
La solución propuesta está en base al cálculo o determinación de los índices de cada arreglo de datos que van a estar involucrados en la convolución. Tres cálculos son claves para este pseudocódigo, i) El del límite de K ,ii) Los límites de j y iii) Los rangos de K para las opciones Shape =“Full” o “Same”.

```
// PSEUDO CODE CONVOLUTION:
1.- Busy = 0 // Estado disponible
2.- Done = 1 // Tarea anterior realizada
3.- While start =0 // Espera start = 1 para iniciar
4.- End while
5.- Busy = 1 // Una vez iniciado su estado es ocupado
6.- Done = 0 // Tarea en proceso
7.-define convolucion(dataX,dataY,sizeX,sizeY) // función de convolución
8.- for k=0 ; k<= sizeX+ sizeY -1; k++ // ciclo de K términos de la convolución
9.- inicio = max(0,k-sizeY+1) // Cálculo límite superior de iteraciones j
10.- fin = min(k,sizeX -1) // Cálculo límite inferior de iteraciones j
11.- for j=0;j<=inicio+fin+1; j++ // Convolución de términos en j iteraciones
12.- z[k]+= dataX[j] * dataY[k-j] // Sumatoria
13.- return z
14.- Case Shape = 0 // Shape Full
15.- dataZ = convolucion(dataX,dataY,sizeX,sizeY) //Llamado a función de convolución
16.- print(dataZ)
17.- Done = 1
18.- break
19.- Case Shape = 1 //Shape Same
20.- dataZ = convolucion(dataX,dataY,sizeX,sizeY) Llamado a función de convolución
21 dataZ= dataZ[Math.floor((tam_x-1)/2)-1 : Math.floor((tam_x-1)/2)+ tam_x-1]
22.- print(dataZ) Se imprimen los valores centrales de la convolución Full
23.- Done = 1
24.- break
25.- End Case
27.- goto 1 //Reinicio
```

#### 4 Validación en python

El pseudocódigo se validó en python sin ningún inconveniente. Abajo se muestran las evidencias.

```
[10] x= [1,5,1,10,3,9,14,4,5,6]
      y= [1,4,9,13,20]
      tam_x= len(x)
      tam_y= len(y)
      start=1
      shape=0 # 0 full / 1 same
```

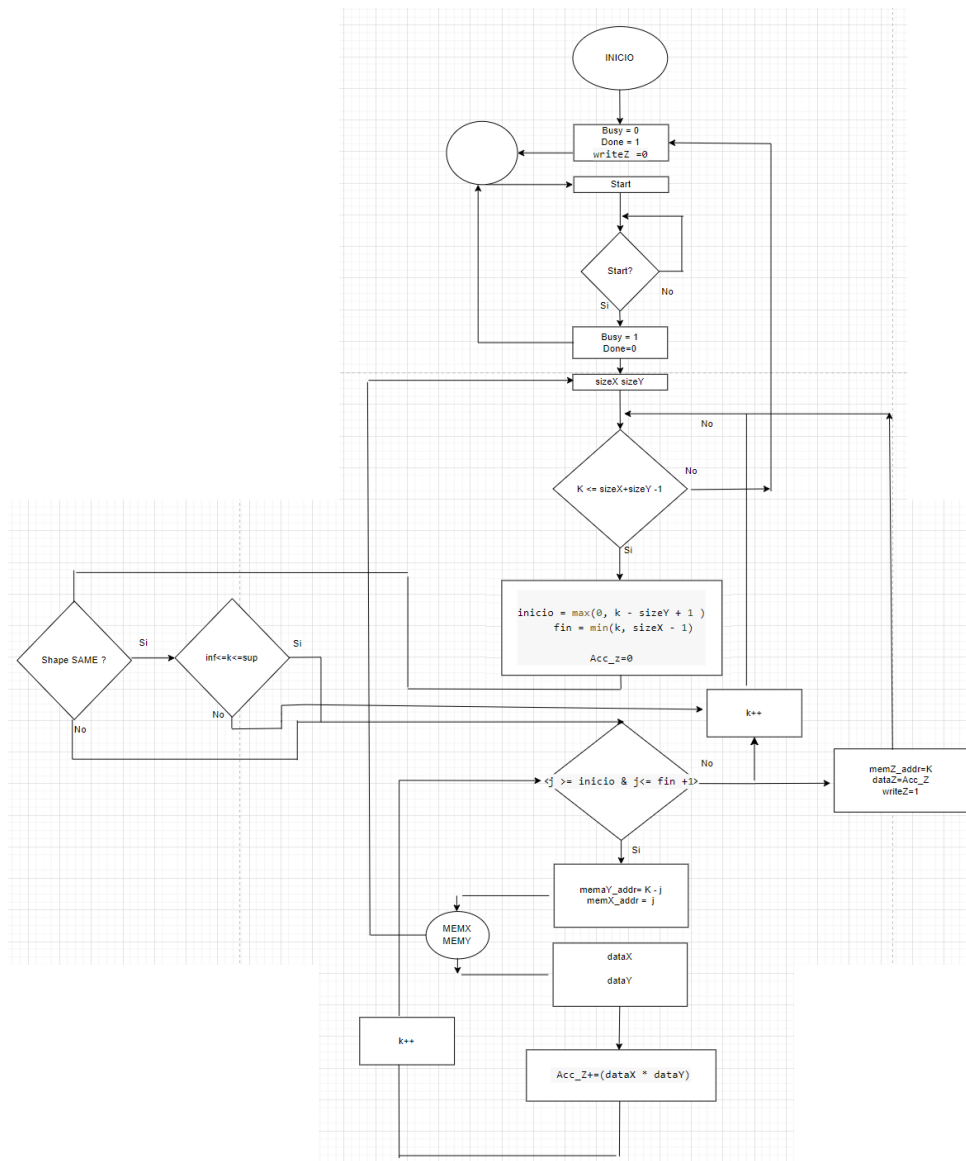


```
dataZ=np.zeros(tam_x + tam_y - 1)
match int (shape):
    case 0:
        print("caso full");
        dataZ=convolucion_full ( x, y, tam_x, tam_y);
        print(dataZ)
        break
    case 1:
        print("caso same");
        dataZ=convolucion_full ( x, y, tam_x, tam_y);
        dataZ= dataZ[math.floor(((tam_x-1)/2)-2) : math.floor((tam_x-1)/2)+ tam_x-2]
        print(dataZ)
        break
```

caso full  
convolution  
[ 1. 9. 30. 72. 137. 224. 227. 380. 324. 424. 401. 199. 178. 120.]

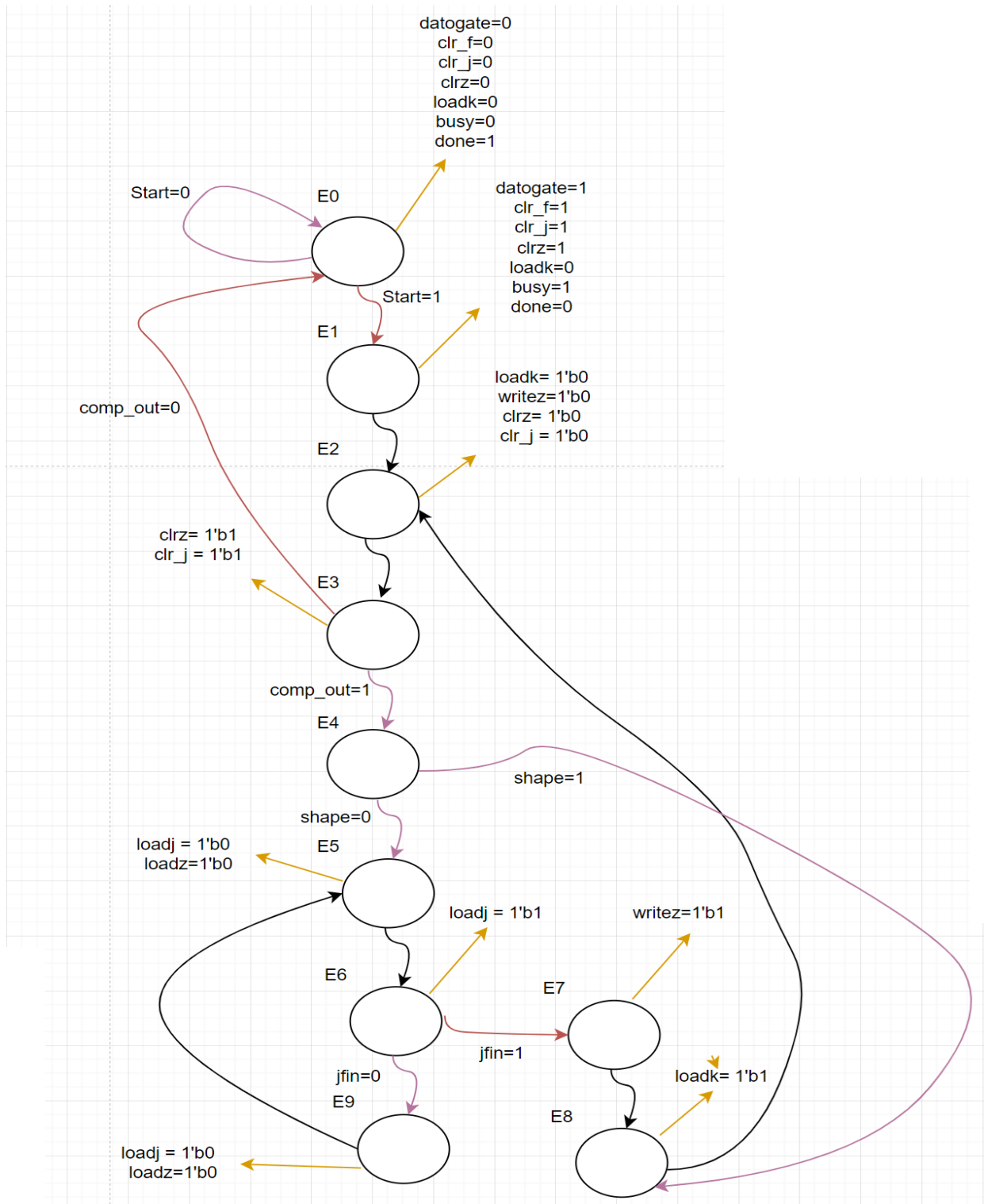
## 5 Diagrama ASM

El diagrama ASM se muestra abajo:



El diagrama de bloques muestra la ruta del algoritmo, inicia con la señal de Start y la salida del estado del co-procesador principalmente, una vez que se da la orden de inicio, se obtienen las informaciones de tamaño para determinar los límites de K, y se pregunta por la opción de trabajo “Full” o “Same”, lo anterior representa el número de iteraciones para calcular cada valor de convolución, posteriormente al ingresar al ciclo “for” de K donde se calculan los límites del ciclo j donde se realiza la sumatoria de cada término para obtener el valor de salida Z.

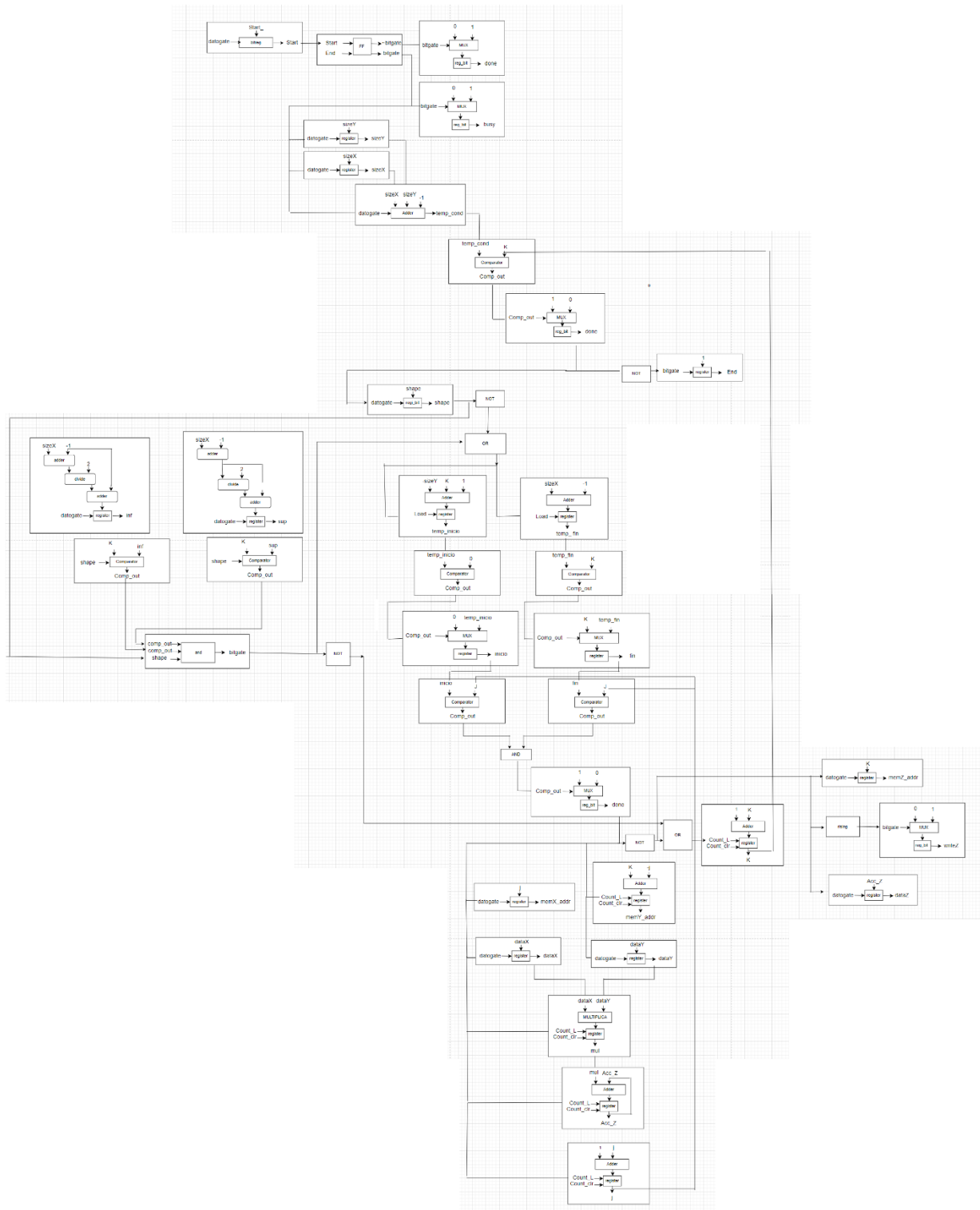
## 6 Máquina de estados




La máquina de estado se conformó por 10 instancias.

## 7 Datapath

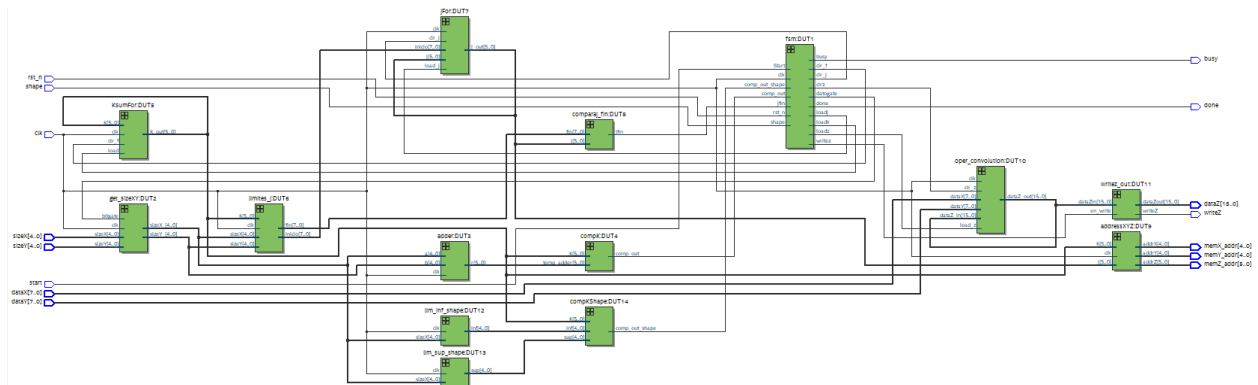
Abajo se muestra el datapath:



**8 Cantidad de registros.**

Fitter Summary	
 <<Filter>>	
Top-level Entity Name	convolution
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	88 / 56,480 ( < 1 % )
Total registers	80
Total pins	65 / 268 ( 24 % )
Total virtual pins	0
Total block memory bits	0 / 7,024,640 ( 0 % )
Total RAM Blocks	0 / 686 ( 0 % )
Total DSP Blocks	1 / 156 ( < 1 % )

## 9 Esquemático TOP LEVEL.



## 10 Programa para generar archivos txt

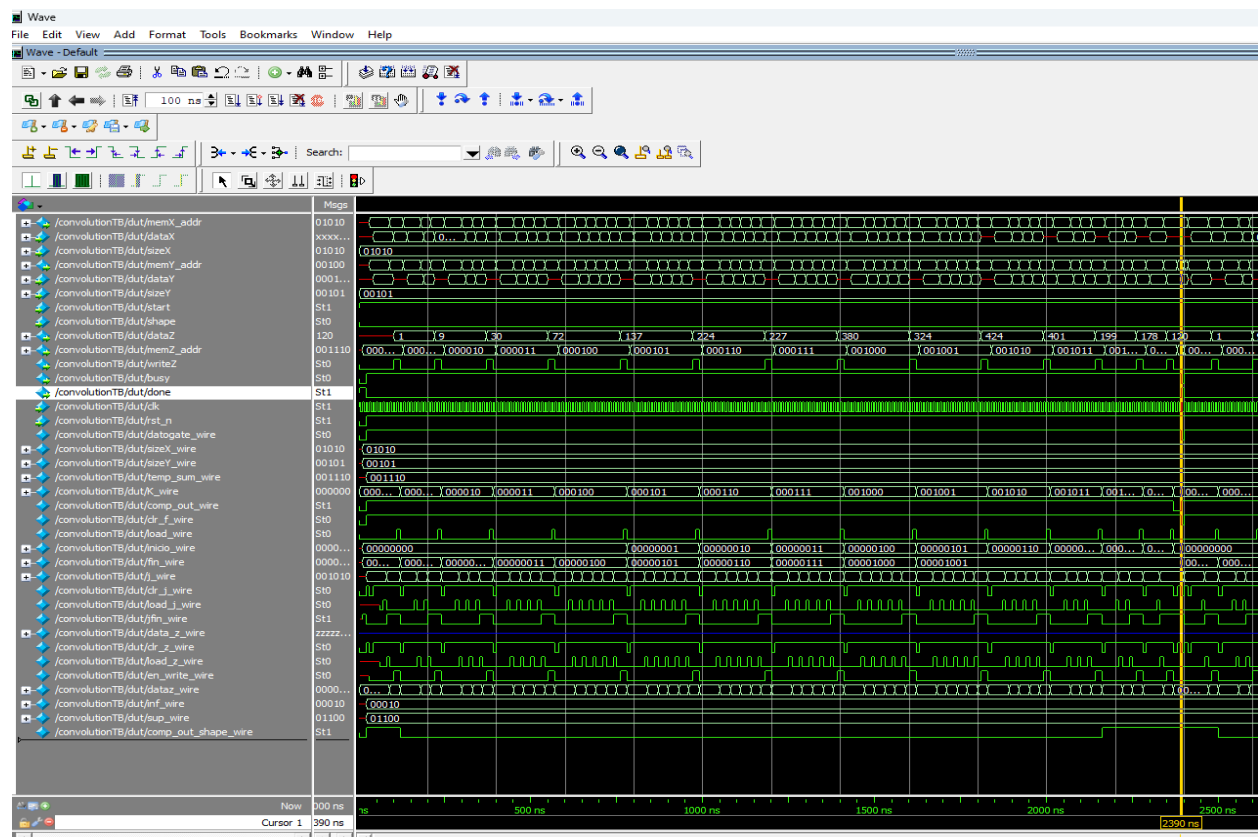
Python:

```
f = open("dataX.txt", "w")
for i in range(10):

    if (i <10-1 ):
        randomlist
= hex(random.randint(1,100))[2:]
        f.write(str(randomlist)+ '\n')
    else:
        randomlist
= hex(random.randint(1,100))[2:]
        f.write(str(randomlist))
f.close()
```


## 11 Waveform de la simulación

239 ciclos de reloj






## 12 Área

Fitter Resource Usage Summary			
 <<Filter>>			
	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	88 / 56,480	< 1 %
2	> ALMs needed [=A-B+C]	88	
3			
4	Difficulty packing design	Low	
5			
6	> Total LABs: partially or completely used	12 / 5,648	< 1 %
7			
8	> Combinational ALUT usage for logic	158	
9	Combinational ALUT usage for route-throughs	0	
10			
11	> Dedicated logic registers	80	
12			
13	Virtual pins	0	
14	> I/O pins	65 / 268	24 %

## 13 Frecuencia máxima de operación.

Slow 1100mV 85C Model Fmax Summary				
 <<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	106.26 MHz	106.26 MHz	clk	

## 14 Número de ciclos de reloj para hacer la convolución

239 ciclos de reloj