

Proyecto integrador clase Metodología de diseño de System-on-chip

Integrantes:

MI. Julio César García Méndez.

Dr. Jaciel David Hernandez Resendiz.

Contenido

1.	Integración de la unidad de procesamiento convolucional a un núcleo RISC-V	2
1.1.	Estructura esperada.....	2
1.3	Desarrollo del firmware.....	8
1.3.1	Configuración del convolucionador a través del firmware.....	8
1.3.2	Registro de configuración.....	9
1.3.3	Escritura de memorias X y Y	10
1.3.4	Lectura de la memoria Z.....	11
1.4	Resultados.....	12
1.4.1	Validación de la solución a nivel de simulación.....	12
1.4.2	Validación implementada en el FPGA.....	13
1.	IMPLEMENTACIÓN CO-PROCESADOR DE CONVOLUCIÓN	16
1.	Descripción del problema	16
2.	Diagrama de caja Negra	16
3.	Pseudocódigo.....	17
4.	Validación en python	18
i.	Cantidad de registros.....	22
ii.	Esquemático TOP LEVEL.....	22
2.	MANUAL DE USUARIO DEL CONVOLUTION IP-CORE	26
3.1	DESCRIPTION.....	26
i.	CARACTERÍSTICAS CONFIGURABLES.....	26

2.	DESCRIPCIÓN DE LA SEÑAL DE ENTRADA/SALIDA	26
3.	Funcionamiento del convolucionador.....	27
4.	Descripción de los registros y memorias de la interfaz AIP	27
4.1	Registro de estado	27
4.2	Registro de configuración.....	28
4.3	Memoria de datos de entrada X.....	28
4.4	Memoria de datos de entrada Y.	28
4.5	Memoria de datos de salida Z.	29

1. Integración de la unidad de procesamiento convolucional a un núcleo RISC-V

El objetivo de este proyecto es la integración de IP core de la operación de Convolución con un IP core RISC-V (PicoRV32) para llevar a cabo las operaciones de convolución entre dos memorias X y Y de manera más rápida precisa, utilizando la flexibilidad y la arquitectura abierta de RISC-V CORE. Para la validación del funcionamiento se usará un ambiente de simulación y el proyecto se bajará a una FPGA leyendo sus salidas usando el GPIO 1 los pines (TX, RX).

En la sección 2 de anexos, se puede encontrar los detalles del diseño y el desarrollo del IP core del Convolucionador. También, en la sección 3 se encuentra el manual de uso del IP core del convolucionador.

1.1. Estructura esperada.

En la siguiente figura se muestra en general la estructura en la comunicación entre IP core PicoRV32 y el IP CORE de convolución. Así como la AIP interface IP que se comunica con el PicoRV32. También se observa los diferentes módulos de memorias, buses de comunicación GPIO, UART entre otros módulos para el funcionamiento correcto.

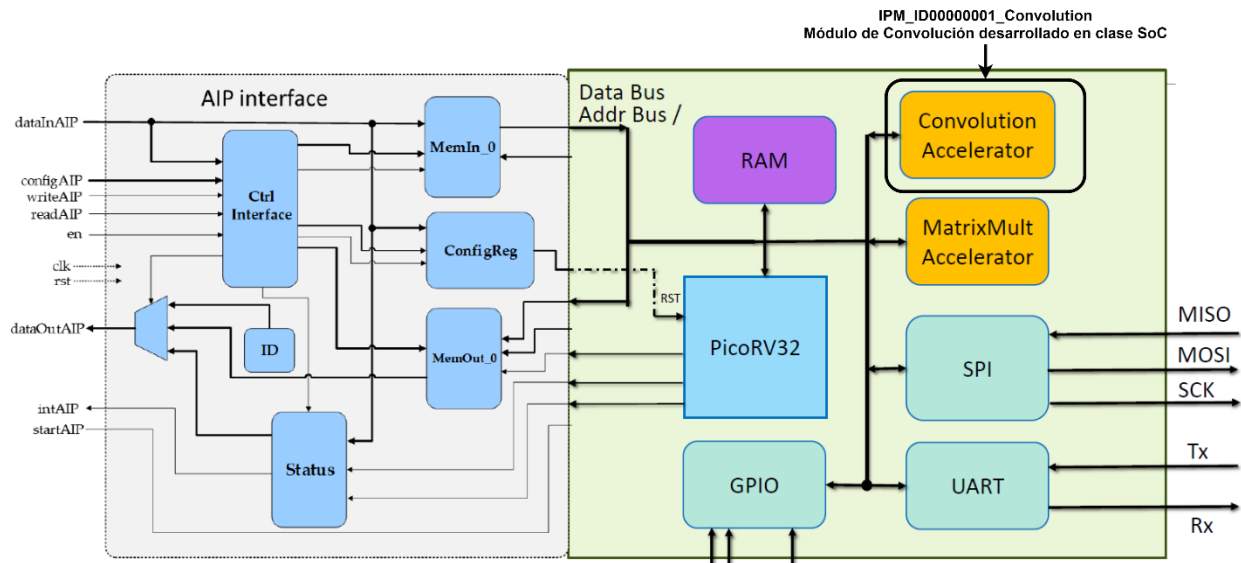
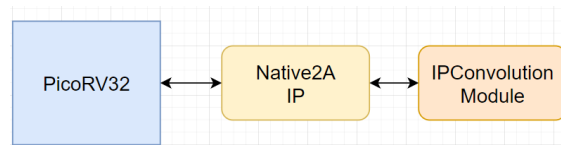


Figure 1 Vista general del AIP Interface for IP RISC V Core (Imagen obtenida de la clase Metodología de Diseño para SoCs - AIP Interface –RISC V with IP Dummy Core)

Para la comunicación entre el PicoRV32 y nuestro convolucionador debe haber una interfaz nativa como la que se muestra en la figura de abajo.



1.2 Integración del convolucionador al RISC V (PicoRV32).

Para este trabajo se nos ha proporcionado un proyecto PicoRV32 (picosoc_AIP) base, el cual incluye los módulos de memoria y UART para la comunicación.

Definimos la dirección de memoria utilizada por nuestro IP CORE Convolution:

```
//IP_Convolution
localparam IP1_BASE_ADDR = 32'h80000300;
localparam IP1_RANGE    = IP1_BASE_ADDR + 32'h00000100;
```

Agregamos el módulo de comunicación nativa para la comunicación de nuestro convolucionador y el módulo PicoRV32

```

native_aip CPU_to_aip1(
    .i_clk      (clk),
    .i_rst      (resetn),

    .i_cpu_mem_valid  (mem_valid),
    .i_cpu_mem_addr   (mem_addr),
    .i_cpu_mem_wdata   (mem_wdata),
    .i_cpu_mem_wen     (mem_valid && !mem_readyAIP1 && ((sel_aip1)? |(mem_wstrb) : 1'b0)),

    .o_cpu_mem_rdata   (mem_rdata_AIP1),
    .o_cpu_mem_ready   (mem_readyAIP1),
    .o_cpu_irq         (),

    // aip interface
    .i_aip_sel         (sel_aip1),
    .i_aip_enable      (1'b1),
    .i_aip_dataOut     (iPdataOut1),
    .o_aip_dataIn      (iPdataIn1),
    .o_aip_config      (iPconf1),
    .o_aip_read        (iPread1),
    .o_aip_write       (iPwrite1),
    .o_aip_start       (iPstart1),
    .i_aip_int         (iPINTstatus1),
    .o_core_int        ()
);

```

Agregamos el módulo de nuestro convolucionador al proyecto del PicoRV32.

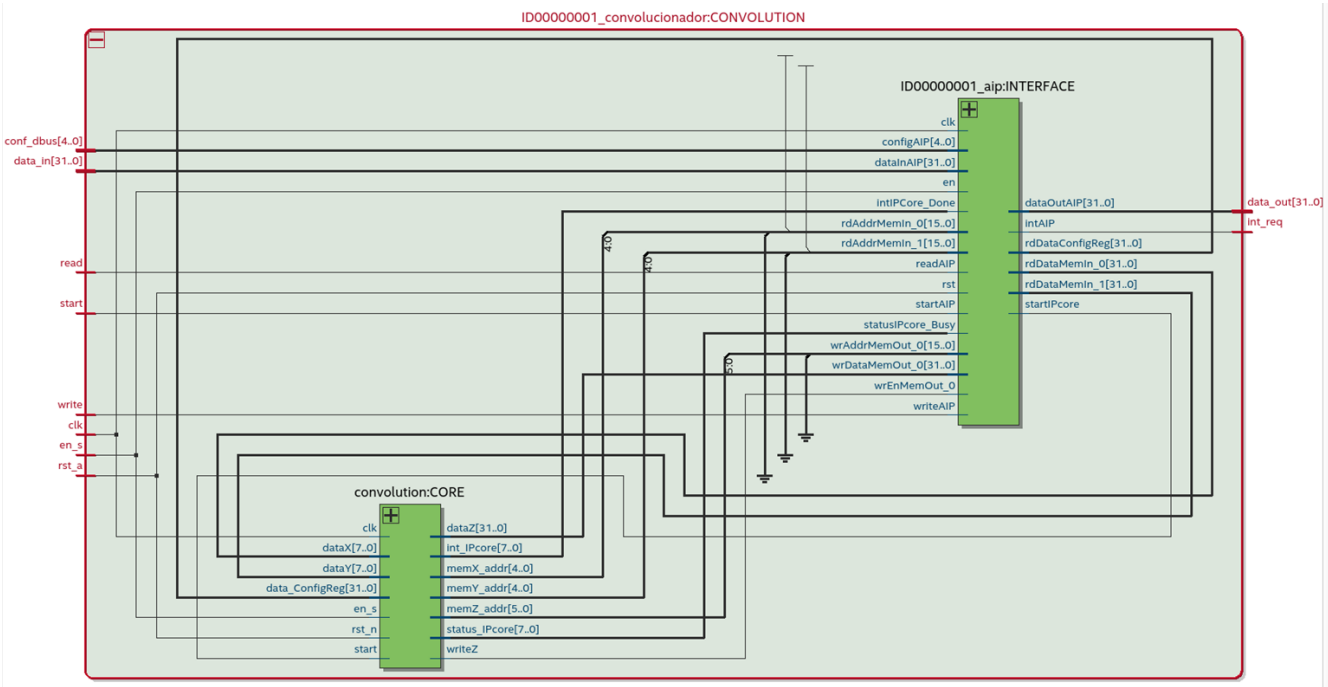
```

ID00000001_convolucionador
CONVOLUTION
(
    .clk      (clk),
    .rst_a    (resetn),
    .en_s     (1'b1),

    .data_in   (iPdataIn1),
    .data_out  (iPdataOut1),
    .write     (iPwrite1),
    .read      (iPread1),
    .start     (iPstart1),
    .conf_dbus (iPconf1),
    .int_req   (iPINTstatus1)
);
endmodule

```

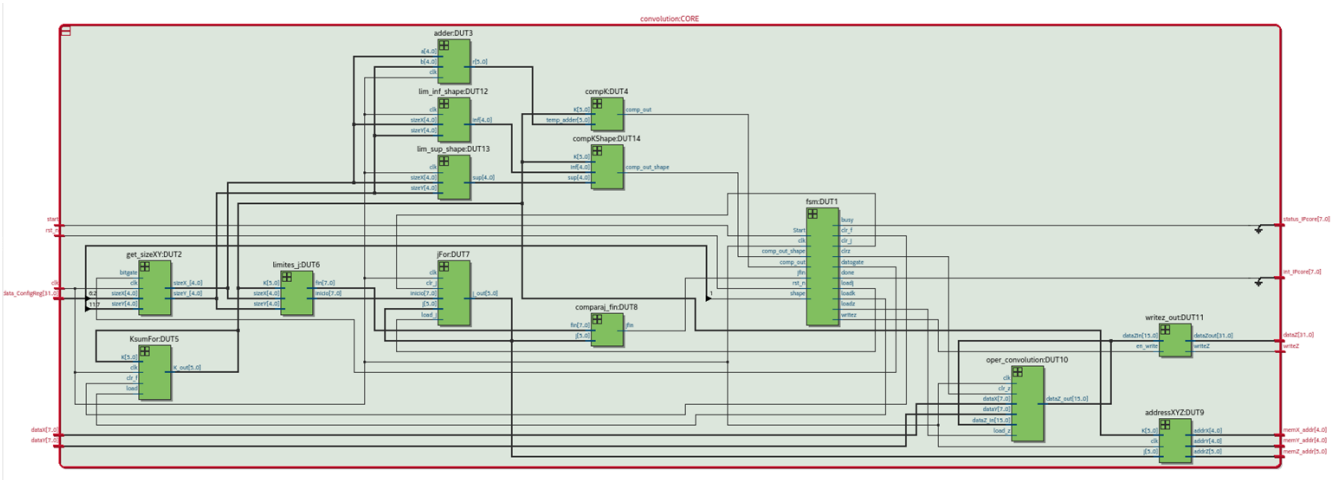
Nuestro módulo de convolucionador ID00000001_convolucionador cuenta con la interfaz AIP y el core de convolucionador, como se muestra en la siguiente figura.



Donde el módulo cconvolution:CORE tiene las siguientes entradas:

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

El convolucionador está diseñado e implementado usando la metodología top-down, esto nos permitió generar 11 módulos para llevar a cabo la operación de convolución.

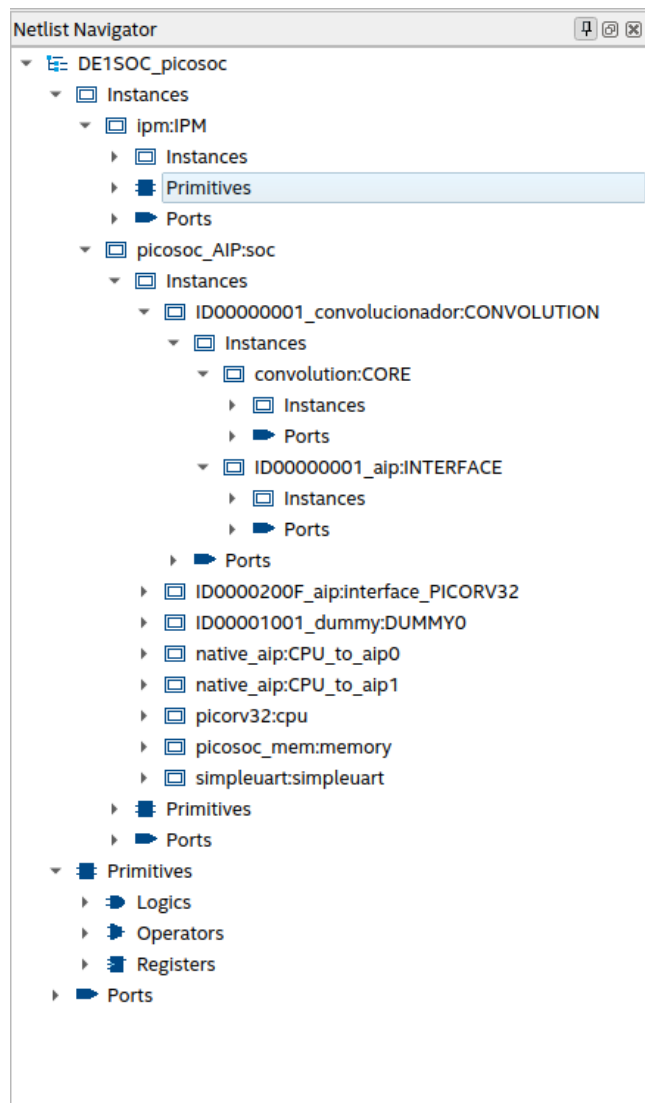


El módulo de interfaz ID0000001_aid:INTERFACE tiene la siguiente configuración:

Type	alphaID	numID	size	mode	description
IPID	IP_MODULE_CONVOLUTION	0x00000001			IP MODULE CONVOLUTION
CFG	MmemX	0b00000	32	W	memoriaX
CFG	MmemY	0b00010	32	W	memoriaY
CFG	MmemZ	0b00100	32	R	memoriaZ
CFG	AmemX	0b00001	1	W	Set Write Address Pointer of MmemX
CFG	AmemY	0b00011	1	W	Set Write Address Pointer of MmemY
CFG	AmemZ	0b00101	1	W	Set Read Address Pointer of MmemZ
CFG	CONF_REG	0b00110	1	W	registro de configuración
CFG	AONF_REG	0b00111	1	W	Set Write Address of CONF_REG
CFG	STATUS	0b11110	1	B	IP status register
CFG	IPID	0b11111	1	R	IP ID

Para mayor detalle del diseño e implementación de módulo de convolución puede dirigirse a la sección 2 y 3 de este documento.

Una vez agregado nuestro módulo de convolución y haber realizado correspondientes con el core de PicoRV32, el documento top quedó de la siguiente forma:



1.3 Desarrollo del firmware.

Una vez que nuestro convolucionador y PicoRV32 puede comunicarse, es necesario desarrollar firmware que es el software que se ejecuta directamente en el procesador RISC-V. Este software es el encargado de ejecutar los comandos del convolucionador.

El procesador PicoRV32 es un núcleo de procesador RISC-V que puede ejecutar instrucciones de forma independiente del firmware. Este núcleo se encarga del flujo de control del sistema y ejecutar código de aplicaciones (firmware).

El firmware que desarrollamos configura y realiza operaciones de escritura y lectura de direcciones de memoria que interactúan con nuestro módulo IP convolucionador para realizar la operación de convolución entre los datos de las memorias X y Y. Esto nos permite que PicoRV32 y el firmware puedan coordinarse para asegurar que las operaciones del convolucionador se ejecuten en el orden correcto y que los resultados se manejen de forma adecuada.

1.3.1 Configuración del convolucionador a través del firmware.

Primero definimos la memoria en la que va a trabajar nuestro convolucionador (0x80000300), también definimos nuestras memorias X, Y y Z, y su dirección de memoria y el registro de configuración, etc. Esta configuración se realiza con base a la interfaz generada para nuestro convolucionador.

Type	alphaID	numID	size	mode	description
IPID	IP_MODULE_CONVOLUTION	0x00000001			IP MODULE CONVOLUTION
CFG	MmemX	0b00000	32	W	memoriaX
CFG	MmemY	0b00010	32	W	memoriaY
CFG	MmemZ	0b00100	32	R	memoriaZ
CFG	AmemX	0b00001	1	W	Set Write Address Pointer of MmemX
CFG	AmemY	0b00011	1	W	Set Write Address Pointer of MmemY
CFG	AmemZ	0b00101	1	W	Set Read Address Pointer of MmemZ
CFG	CONF_REG	0b00110	1	W	registro de configuracion
CFG	AONF_REG	0b00111	1	W	Set Write Address of CONF_REG
CFG	STATUS	0b11110	1	B	IP status register
CFG	IPID	0b11111	1	R	IP ID


```

#define AIP_IP_IPCONVOLUTION 0x80000300
#define AIP_MEMX 0
#define OFFSET 0
#define AIP_ADDRMEMX 1
#define AIP_MEMY 2
#define AIP_ADDRMEMY 3
#define AIP_MEMZ 4
#define AIP_ADDRMEMZ 5
#define AIP_CONFIG1 6
#define AIP_AonfReg1 7
#define AIP_STATUS 30
#define AIP_ID 31
#define SIZEMEMZ 4

```

Definimos la velocidad para simulación o implementación en FPGA.

```

//reg_uart_clkdiv = 104; //12 MHz/ 115200 Baud
reg_uart_clkdiv = 434; //50 MHz/ 115200 Baud
//print("Booting..\n");

```

1.3.2 Registro de configuración.

Posteriormente, configuramos el convolucionador escribiendo a la dirección de memoria el número decimal del registro de configuración, por ejemplo:

1. El tamaño de la memoria X = 4, el tamaño de la memoria Y = 3, enable = 0 y el tipo de convolución 0 (Full).

00011 00100 00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
- Y/Sizes[11:7] - X/Sizes[6:2]-Shape[1]- En[0]																																
w	w	W	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	
																					ZY	ZY	ZY	ZY	ZY	ZX	ZX	ZX	ZX	ZX	Shape	En
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	

```

print("write CONFIG!\n");
//CONF
*(ptr + AIP_CONFIG) = AIP_AonfReg1;
*(ptr + AIP_DATAIN) = OFFSET; //0;
*(ptr + AIP_CONFIG) = AIP_CONFIG1;
*(ptr + AIP_DATAIN) = 400; //CONFIO

```

2. El tamaño de la memoria X = 3, el tamaño de la memoria Y = 2, enable = 0 y el tipo de convolución 1 (shape).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
- Y/Sizes[11:7] - X/Sizes[6:2]-Shape[1]- En[0]																															
w	w	W	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	W	w	
																				ZY	ZY	ZY	ZY	ZY	ZX	ZX	ZX	ZX	ZX	Shape	En
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0

1.3.3 Escritura de memorias X y Y

Para poder realizar la operación de convolución es necesario llenar las memorias X y Y utilizadas por el convolucionador. A continuación, se muestra un ejemplo donde la memoria X = {1,2,3,4} y Y={1,2,3},

```

//MEMORIA X
*(ptr + AIP_CONFIG) = AIP_ADDRMEMX; // Posiciona
*(ptr + AIP_DATAIN) = OFFSET; //POSICIONARSE EN
*(ptr + AIP_CONFIG) = AIP_MEMX; //POSICIONARSE
*(ptr + AIP_DATAIN) = 1; //ESCRIBIR EN RAFAGA
*(ptr + AIP_DATAIN) = 2; //ESCRIBIR EN RAFAGA
*(ptr + AIP_DATAIN) = 3; //ESCRIBIR EN RAFAGA
*(ptr + AIP_DATAIN) = 4; //ESCRIBIR EN RAFAGA

print("write MemY!\n");
//MEMORIA Y
*(ptr + AIP_CONFIG) = AIP_ADDRMEMY; //3; // Pos
*(ptr + AIP_DATAIN) = OFFSET; //POSICIONARSE EN
*(ptr + AIP_CONFIG) = AIP_MEMY; //POSICIONARSE
*(ptr + AIP_DATAIN) = 1; //ESCRIBIR EN RAFAGA
*(ptr + AIP_DATAIN) = 2; //ESCRIBIR EN RAFAGA
*(ptr + AIP_DATAIN) = 3; //ESCRIBIR EN RAFAGA

```

1.3.4 Lectura de la memoria Z.

Para iniciar la convolución le escribimos a la dirección de memoria del registro AIP_START=1

```
*(ptr + AIP_START) = 1;
```

Podemos saber información del convolucionador, por ejemplo, el estatus. Para ello es necesario acceder a la dirección de memoria que guarda dicha información:

```
print("Reading STATUS !\n");
uint32_t STATUS = 0;

*(ptr + AIP_CONFIG) = 30;
STATUS = *(ptr + AIP_DATAOUT);
print_hex(STATUS, 8);
```

Para acceder a la información de la tarea de convolución, es necesario acceder a la dirección de memoria designada para la memoria Z donde se guarda el resultado de la convolución.

```
uint32_t VALZ = 0;
print("Reading MEMZ !\n");
*(ptr + AIP_CONFIG) = 5;
*(ptr + AIP_DATAIN) = 0;
*(ptr + AIP_CONFIG) = 4;

for (i = 0; i < SIZEMEMZ; i++){
    VALZ = *(ptr + AIP_DATAOUT);
    print_hex(VALZ, 8);
}
```

1.4 Resultados.

1.4.1 Validación de la solución a nivel de simulación.

A continuación, se presenta un ejemplo a nivel de simulación usando los siguientes datos.

$X = \{1, 2, 3, 4\}$

$Y = \{1, 2, 3\}$

Tipo de convolución: Full.

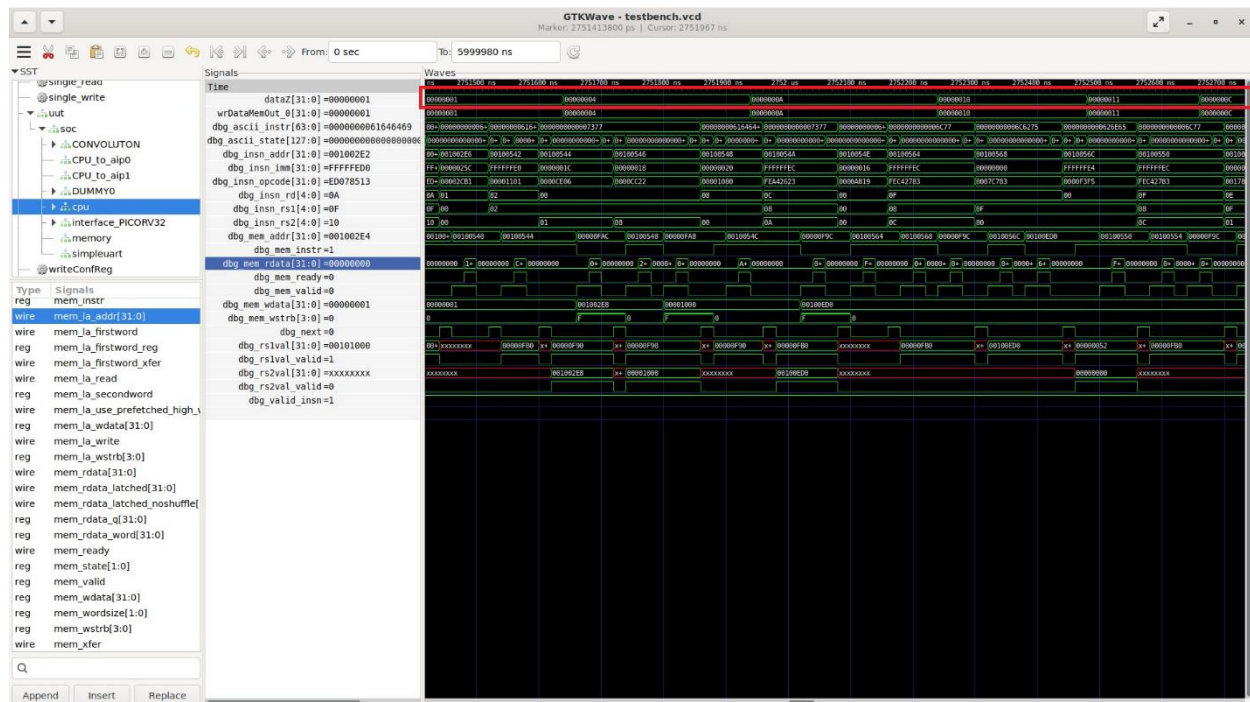
En la siguiente figura se muestra la solución de la convolución para estos conjuntos de datos utilizando MATLAB.

```
>> X
X =
     1     2     3     4

>> Y
Y =
     1     2     3

>> conv(X,Y)
ans =
     1     4    10    16    17    12
```

En la siguiente imagen se muestra el resultado de la simulación de la convolución.



Podemos ver que los resultados mostrados en la memoria Z (data Z) = 1 (1), 4(4), A(10), 10(16), 11(17), C(12), coinciden con lo que obtenemos usando MATLAB.

1.4.2 Validación implementada en el FPGA.

Para la validación de la implementación en hardware, la solución se implementó en un FPGA. Para la configuración del FPGA se usó la siguiente asignación de pines:

Pin Planner - D:/Download/picorv32_InterfaceAIP_quartus_I/picosoc_AIP/picorv32_quartus - picorv32_quartus

File Edit View Processing Tools Window Help

Report Report not available

Groups Report

Tasks

- Early Pin Planning
 - Early Pin Plan
 - Run I/O Assig
 - Export Pin As
- Highlight Pins
- I/O Banks
- VREF Groups
- Edges
- Clock Pins
 - Clock

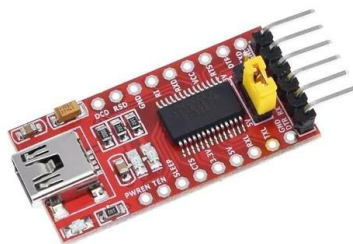
Top View - Wire Bond
Cyclone V - 5CSEMA5F31C6

All Pins

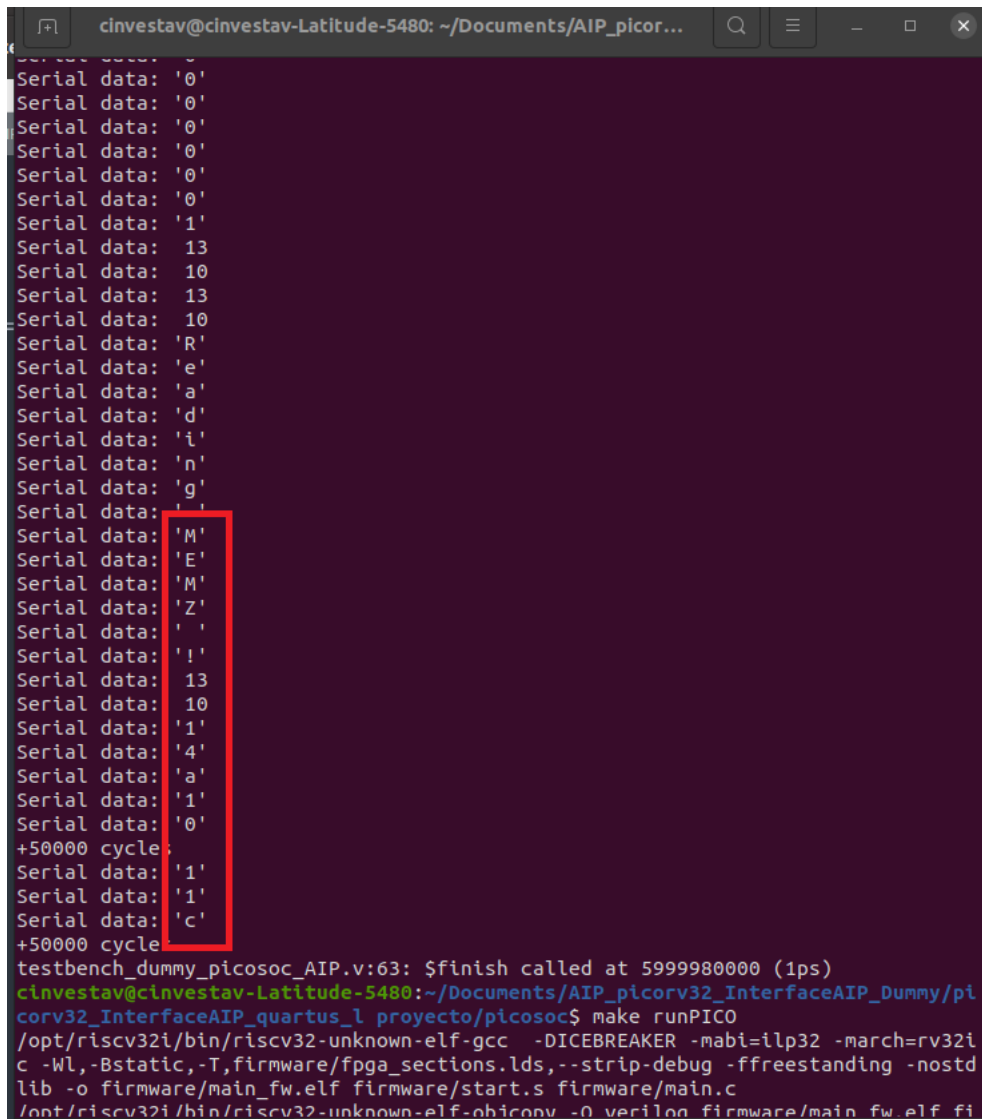
Node Name	Direction	Location	I/O Bank
addressMCU[3]	Input	PIN_AA18	4A
addressMCU[2]	Input	PIN_AF16	4A
addressMCU[1]	Input	PIN_AG16	4A
addressMCU[0]	Input	PIN_AH18	4A
clk	Input	PIN_AF14	3B
dataMCU[7]	Bidir	PIN_AD20	4A
dataMCU[6]	Bidir	PIN_AK21	4A
dataMCU[5]	Bidir	PIN_AJ20	4A
dataMCU[4]	Bidir	PIN_AC20	4A
dataMCU[3]	Bidir	PIN_AA19	4A
dataMCU[2]	Bidir	PIN_AG17	4A
dataMCU[1]	Bidir	PIN_AE16	4A
dataMCU[0]	Bidir	PIN_AH17	4A
ena	Input	PIN_AB12	3A
iniMCU	Output	PIN_AD19	4A
led1	Output	PIN_V16	4A
led2	Output	PIN_W16	4A
led3	Output	PIN_V17	4A
led4	Output	PIN_V18	4A
led5	Output	PIN_W17	4A
ledg_n	Output	PIN_W19	4A
ledr_n	Output	PIN_Y19	4A
rdMCU	Input	PIN_AE17	4A
rst	Input	PIN_AA14	3B
rstMCU	Input	PIN_AH20	4A
ser_rx	Input	PIN_AB17	4A
ser_tx	Output	PIN_AB21	4A
wrMCU	Input	PIN_AH19	4A

<<new node>>

Los pines de salida fijados en el pin planer corresponden a los pines GPIO 1 [2] (leer la salida) y el GPIO[11] para tierra. Para la lectura del resultado se usó un Convertidor Usb Serial y la aplicación PUTTY con la configuración (serial, 115200 bauds).



En la siguiente figura se muestra el resultado del comando `make sym`, se puede observar que el resultado obtenido es similar a la figura de simulación y de la validación usando MATLAB.



```
cinvestav@cinvestav-Latitude-5480: ~/Documents/AIP_picor...
Serial data: '0'
Serial data: '0'
Serial data: '0'
Serial data: '0'
Serial data: '0'
Serial data: '0'
Serial data: '1'
Serial data: 13
Serial data: 10
Serial data: 13
Serial data: 10
Serial data: 'R'
Serial data: 'e'
Serial data: 'a'
Serial data: 'd'
Serial data: 'i'
Serial data: 'n'
Serial data: 'g'
Serial data: 'M'
Serial data: 'E'
Serial data: 'M'
Serial data: 'Z'
Serial data: '!'
Serial data: 13
Serial data: 10
Serial data: '1'
Serial data: '4'
Serial data: 'a'
Serial data: '1'
Serial data: '0'
+50000 cycle
Serial data: '1'
Serial data: '1'
Serial data: 'c'
+50000 cycle
testbench_dummy_picosoc_AIP.v:63: $finish called at 5999980000 (1ps)
cinvestav@cinvestav-Latitude-5480:~/Documents/AIP_picorv32_InterfaceAIP_Dummy/pi
corv32_InterfaceAIP_quartus_l proyecto/picosoc$ make runPICO
/opt/riscv32i/bin/riscv32-unknown-elf-gcc -DICEBREAKER -mabi=ilp32 -march=rv32i
c -WL,-Bstatic,-T,firmware/fpga_sections.lds,--strip-debug -ffreestanding -nostd
lib -o firmware/main_fw.elf firmware/start.s firmware/main.c
/opt/riscv32i/bin/riscv32-unknown-elf-obiconv -O verilog firmware/main_fw.elf fi
```

En la siguiente figura se muestra el resultado del comando `make runPICO` para ejecutar las operaciones del convolucionador usando la FPGA a través del firmware desarrollado. Se puede observar que la lectura del identificador y del estatus del convolucionador se puede leer de forma correcta. Por otro lado, se observa que la salida de la memoria Z no se obtiene de forma correcta. Siendo que al momento de hacer la simulación los valores de la memoria Z existen.

[illegible]

ANEXOS.

1. IMPLEMENTACIÓN CO-PROCESADOR DE CONVOLUCIÓN

1. Descripción del problema

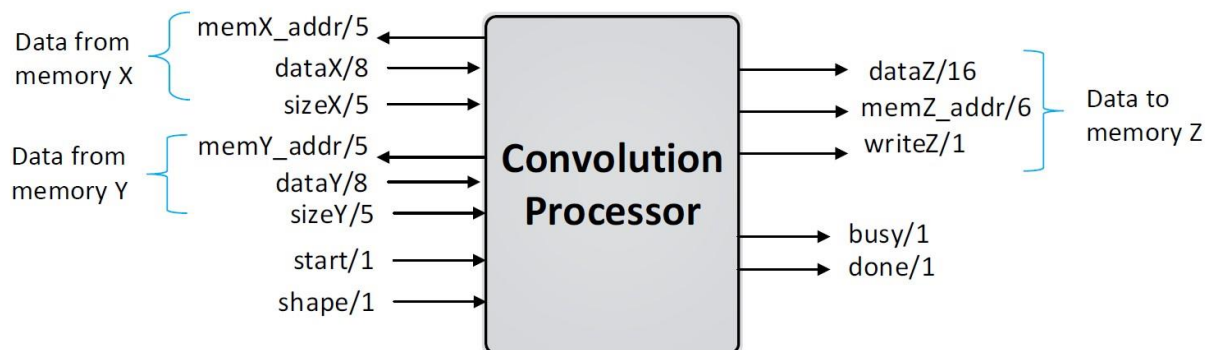
Implementar un co-procesador de convolución que contribuya a mejorar el desempeño de un sistema de procesamiento que involucra esta tarea en particular. Dentro de las características más relevantes de este co-procesador están, i) Los datos son recibidos uno a uno para realizar dichas operaciones y ii)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

El desarrollo de esta implementación considera lo siguientes puntos:

- El diseño e implementación debe regirse por la metodología top-down.
- Una descripción estructural.
- Considerar los dos tipos de convolución FULL y SAME.

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
sizeX	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
```

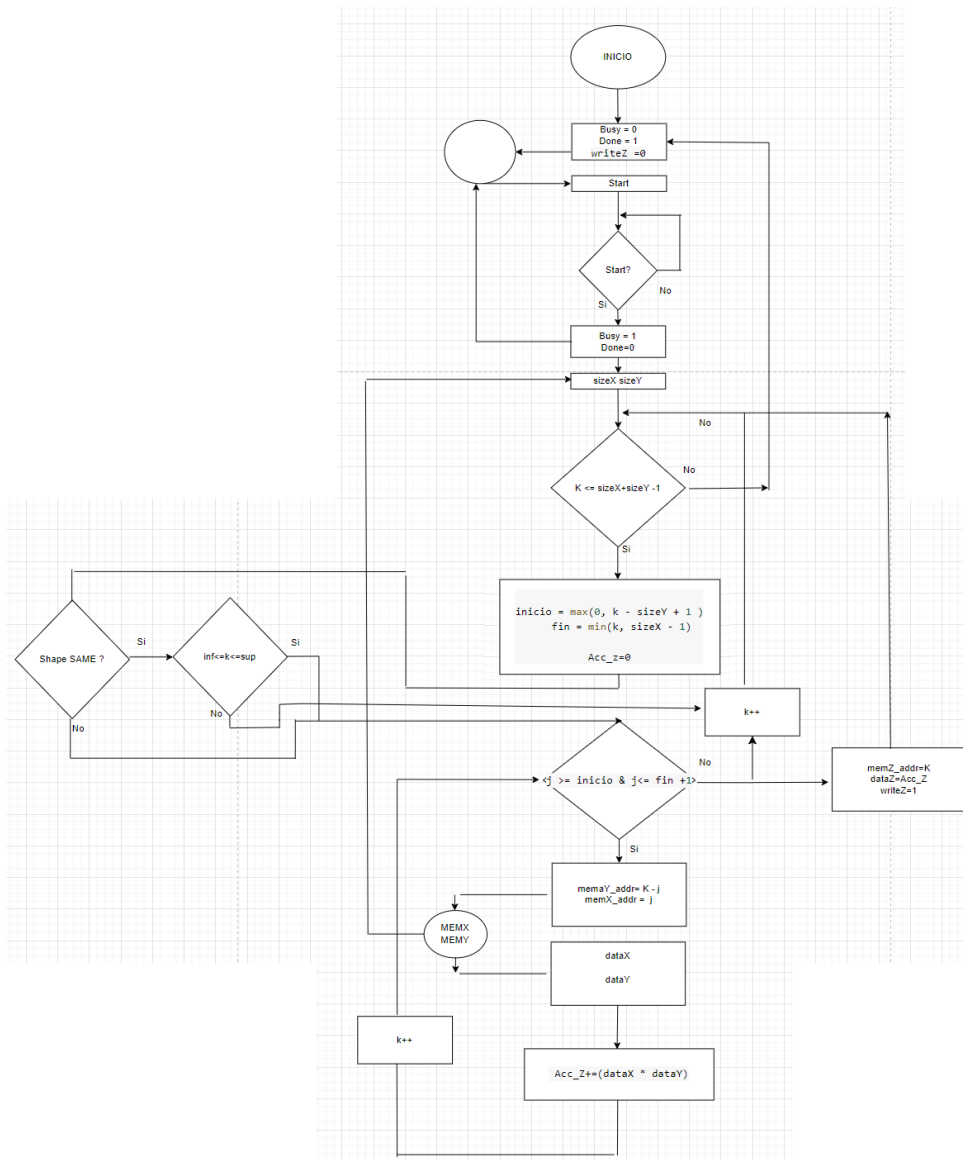
caso full
convolution
[1. 9. 30. 72. 137. 224. 227. 380. 324. 424. 401. 199. 178. 120.]

```
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.]

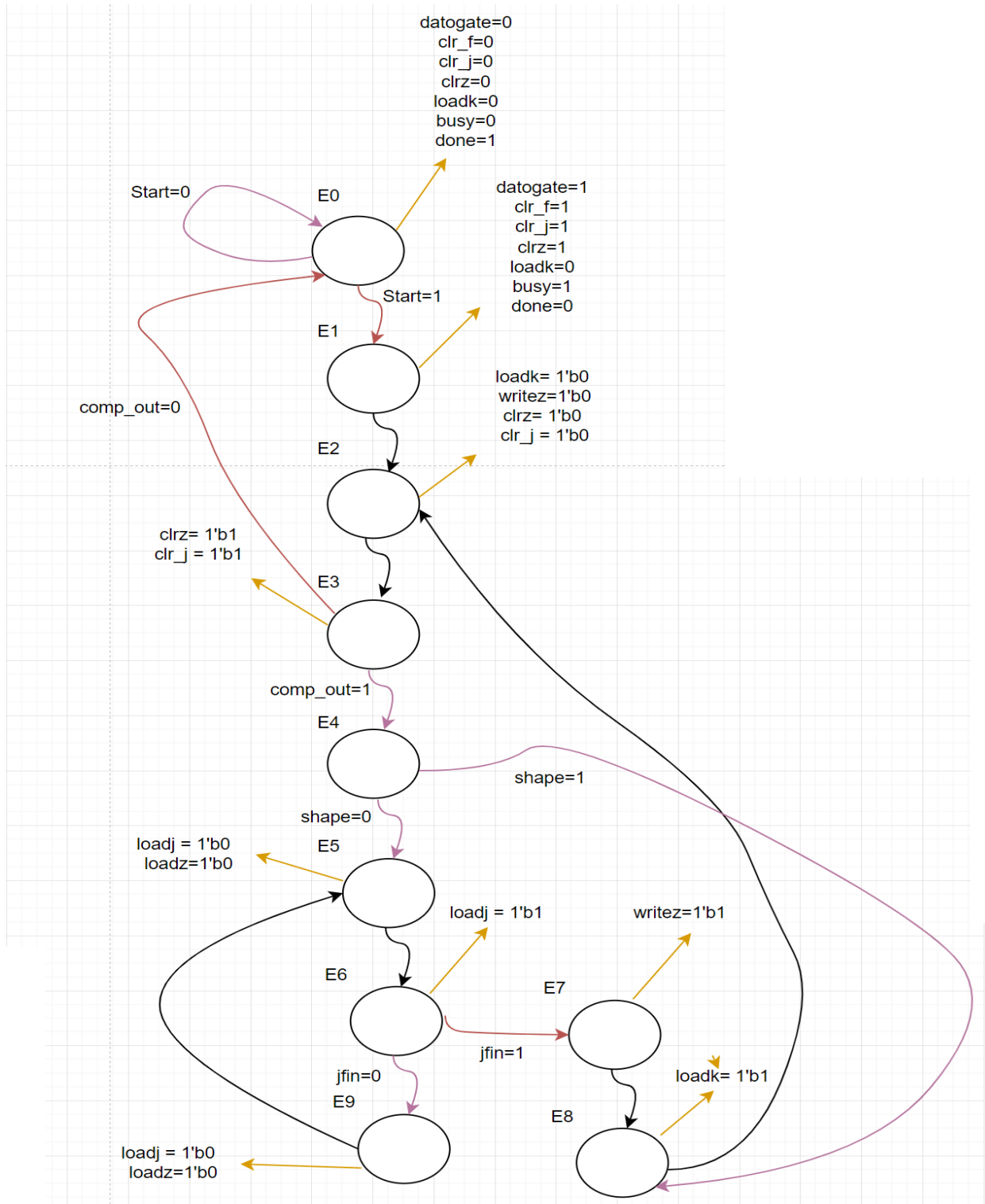
5. Diagrama ASM

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.

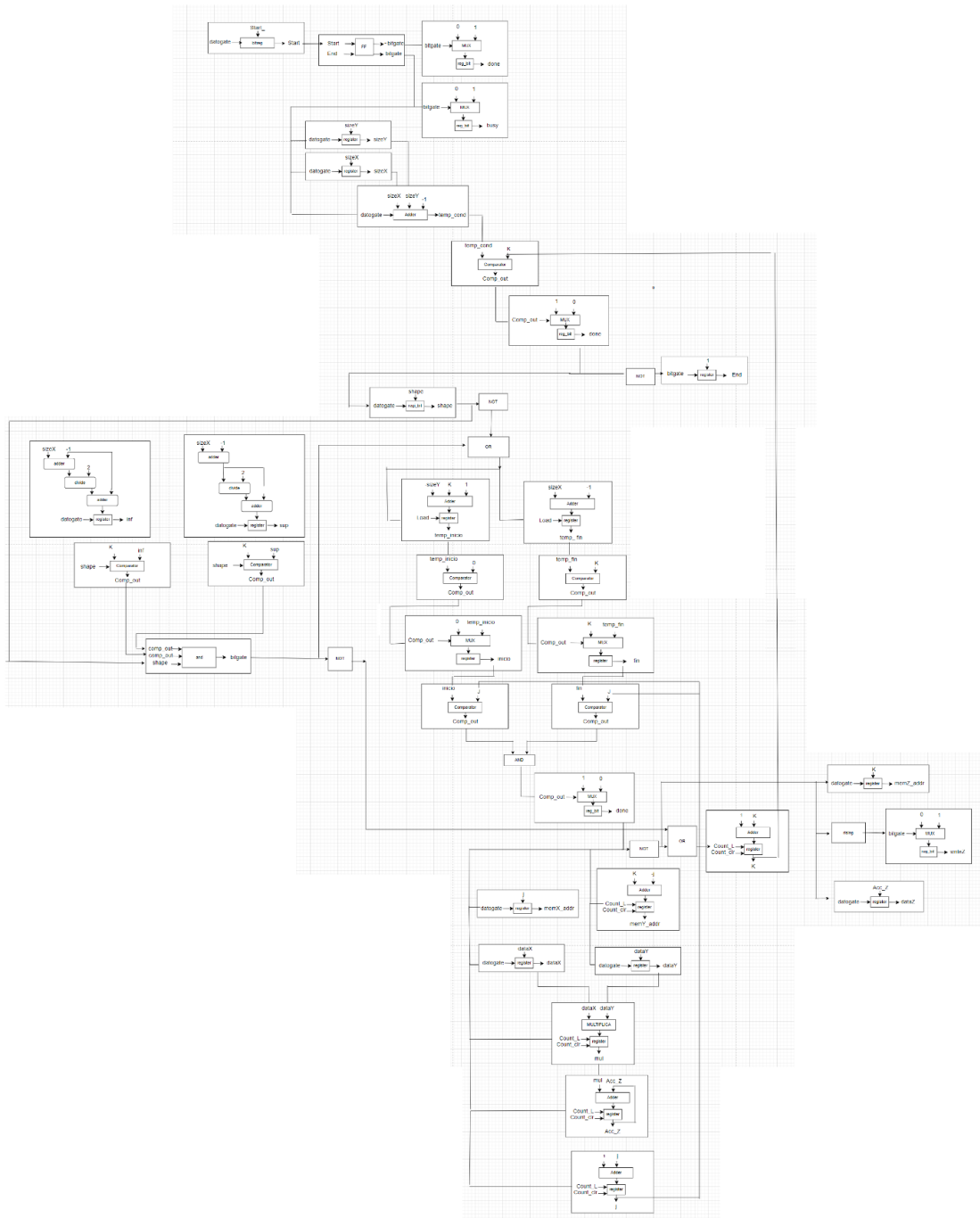
6. Máquina de estados



La máquina de estado se conformó por 10 instancias.


7. Datapath

Abajo se muestra el datapath:

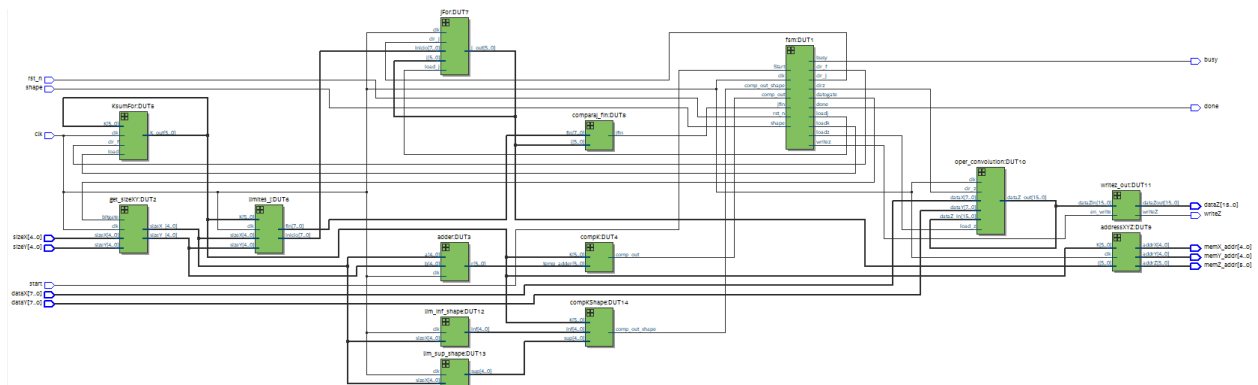


8. Resultados de simulación y síntesis en FPGA.

i. 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 %)

ii. Esquemático TOP LEVEL.



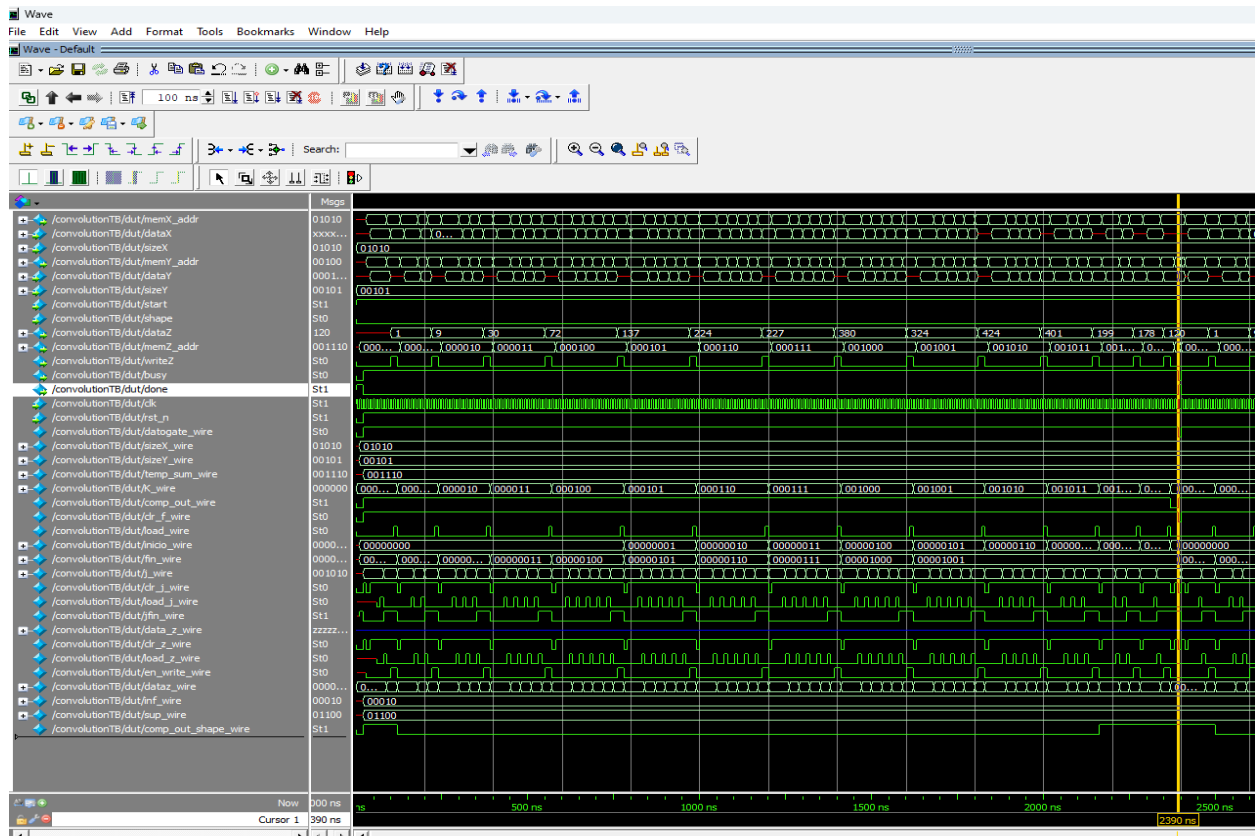
iii. Programa para generar archivos txt

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()
```

iv. Waveform de la simulación




239 ciclos de reloj

v. 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 %

vi. 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	

vii. Número de ciclos de reloj para hacer la convolución

239 ciclos de reloj

viii. Registro de configuración

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DELAY[30:0]																															ENA
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bit 1 – Shape

Bit 2:6 -Size X

Bit 7:11 Size Y

ix. Registro de configuración

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Mask Bits				Status Bits						Interrupt/Clear Flags													
Reserved								Reserved				MSK	Reserved						BSY	Reserved						DN					
												rw							r							rw					

2. MANUAL DE USUARIO DEL CONVOLUTION IP-CORE

3.1 DESCRIPTION

La tarea de convolución entre dos vectores u y v es una operación matemática que representa el área de superposición bajo los puntos cuando v abarca la misma longitud que u .

i. CARACTERÍSTICAS CONFIGURABLES

Software configurations	Description
Shape	Convolution type FULL or SAME
sizeX	Memory size X
sizeY	Memory size Y

2. DESCRIPCIÓN DE LA SEÑAL DE ENTRADA/SALIDA

Signal	Bitwidth	Direction	Description
General signals			
clk	1	Input	System clock
rst_a	1	Input	Asynchronous system reset, low active
en_s	1	Input	Enables the IP Core functionality
AIP Interface			
data_inAIP	32	Input	Input data for configuration and processing
data_outAIP	32	Output	Output data for processing results and status
conf_dbus	5	Input	Selects the bus configuration to determine the information flow from/to the IP Core
WriteAIP	1	Input	Write indication, data from the data_in bus will be written into the AIP Interface according to the conf_dbus value
readAIP	1	Input	Read indication, data from the AIP Interface will be read according to the conf_dbus value. The data_out bus shows the new data read.
startAIP	1	Input	Initializes the IP Core process
int_req	1	Output	Interruption request. It notifies certain events according to the configured interruption bits.
Core signals			
memX_addr	5	output	Data memory address X
dataX	8	Input	Data X

sizeX	5	Input	Size of X
memY_addr	5	output	Data memory address Y
dataY	8	Input	Data Y
sizeX	5	Input	Size of Y
memZ_addr	6	output	Data memory address z
dataZ	16	Input	Data Z
Start	1	Input	Convolution start signal
Shape	1	Input	Convolution type FULL=0 , SAME =1
Busy	1	output	Status of co-processor 1=busy, 0= available
Done	1	output	Co-processor task status 1= completed, 0= in process

3. Funcionamiento del convolucionador.

El núcleo de convolución recibe como entrada los datos de dos memorias de entrada X e Y con tamaño configurable y el tipo de convolución FULL o SAME, después de recibir el comando de arranque, según los datos obtenidos de la prueba realizada, tiene un área de 88 registros, y trabaja a una frecuencia máxima de 106,25 MHz. La operación de dos memorias X de tamaño 10 y Z de tamaño 5, la convolución tarda 230 ciclos de reloj.

4. Descripción de los registros y memorias de la interfaz AIP

4.1 Registro de estado

Config: STATUS

Size: 32 bits

Modo: Read/Write.

Este registro se divide en 3 secciones.:

- **Status Bits:** Estos bits indican el estado actual del núcleo.
- **Interruption Flags:** Estos bits se utilizan para generar una petición de interrupción en la señal *int_req* de la interfaz AIP.
- **Mask Bits:** Cada uno de estos bits puede activar o desactivar las banderas de interrupción.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Mask Bits				Status Bits								Interrupt/Clear Flags											
Reserved								Reserved				MSK	Reserved				BSY	Reserved				DN									
												rw					r					rw									

Bits 31:24 – Reservado.

Bits 23:17 – Reservado..

Bit 16 – **MSK**: bit de máscara para la bandera de interrupción DN (DONE). Si se requiere habilitar la bandera de interrupción DN, este bit debe escribirse a 1.

Bits 15:9 – Reservado.

Bit 8 – **BSY**: status bit “**Busy**”.

La lectura de este bit indica el estado actual de IP Dummy:

0: La Convolución IP no está ocupada y está lista para iniciar un nuevo proceso.

1: La Convolución IP está ocupada y no está disponible para un nuevo proceso.

Bits 7:1 – Reservado.

Bit 0 – **DN**: bandera para interrupción/despeje “**Done**”

La lectura de este bit indica si la Convolución IP ha generado una interrupción:

0: interrupción no generada.

1: la Convolución IP ha finalizado correctamente su procesamiento.

4.2 Registro de configuración

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																- Y/Sizes[11:7] - X/Sizes[6:2]-Shape[1]- En[0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
																				ZY	ZY	ZY	ZY	ZY	ZX	ZX	ZX	ZX	ZX	Shape	En

Bit 0- Está reservado para el enable.

Bit 1- Está reservado para el tipo de operación de convolución FULL o SAME.

1: SAME

0: FULL

Bits 2:6- Reservado para el tamaño de la memoria X.

Bits 7:11- Reservado para el tamaño de la memoria Y.

4.3 Memoria de datos de entrada X.

Config: memX

Size: Nx32 bits

Modo: Write

Es la memoria X utilizada para la operación de convolución entre las memorias X e Y.

4.4 Memoria de datos de entrada Y.

Config: memY

Size: Nx32 bits

Modo: Write

Es la memoria Y utilizada para la operación de convolución entre las memorias X e Y.

4.5 Memoria de datos de salida Z.

Config: memZ

Size: Nx32 bits

Modo: Read

Es la memoria Z utilizada para almacenar el resultado de la operación de convolución entre las memorias X e Y.