**ID00001001**

**DUMMY IP-CORE USER MANUAL**

# DESCRIPCION

El convulucionador IP-core es un sistema en el que usa el principio matemático de convolución, para el procesamiento de dos memorias. Después de recibir un comando de inicio, el núcleo IP, realiza una convolución desplazando un kernel sobre los datos de entrada y produciendo la salida correspondiente. El tamaño de las memorias se puede configurar a través del software.

## CARACTERISTICAS CONFIGURABLES

|  |  |
| --- | --- |
| **Software configuración** | **Descripción** |
| Tamaño | Tamaño de la memoria |

## APLICACIÓN TIPICA

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figure 1.1 IP Convulucionador conectado al host

# CONTENIDO

[1. DESCRIPCION 1](#_Toc167647338)

[1.1. CARACTERISTICAS CONFIGURABLES 1](#_Toc167647339)

[1.2. APLICACIÓN TIPICA 1](#_Toc167647340)

[2. CONTENIDO 2](#_Toc167647341)

[2.1. Lista de figuras 3](#_Toc167647342)

[2.2. Lista de tablas 3](#_Toc167647343)

[3. INPUT/OUTPUT DESCRIPCION DE DEÑALES 4](#_Toc167647344)

[4. TEORIA DE LA OPERACION 5](#_Toc167647345)

[5. Descripción de registros y memorias de la interfaz AIP 6](#_Toc167647346)

[5.1. Registro de estado 6](#_Toc167647347)

[5.2. Configuration delay register **¡Error! Marcador no definido.**](#_Toc167647348)

[5.3. Input data memory 7](#_Toc167647349)

[5.4. Output data memory **¡Error! Marcador no definido.**](#_Toc167647350)

[6. PYTHON DRIVER 8](#_Toc167647351)

[6.1. Usage example **¡Error! Marcador no definido.**](#_Toc167647352)

[6.2. Methods 9](#_Toc167647353)

[6.2.1. Constructor 9](#_Toc167647354)

[6.2.2. writeData 10](#_Toc167647355)

[6.2.3. readData 10](#_Toc167647356)

[6.2.4. startIP 10](#_Toc167647357)

[6.2.5. enableDelay **¡Error! Marcador no definido.**](#_Toc167647358)

[6.2.6. disableDelay **¡Error! Marcador no definido.**](#_Toc167647359)

[6.2.7. enableINT **¡Error! Marcador no definido.**](#_Toc167647360)

[6.2.8. disableINT 11](#_Toc167647361)

[6.2.9. status 11](#_Toc167647362)

[6.2.10. waitINT 12](#_Toc167647363)

[7. C DRIVER 12](#_Toc167647364)

[7.1. Usage example 12](#_Toc167647365)

[7.2. Driver functions 14](#_Toc167647366)

[7.2.1. id00001001\_init 14](#_Toc167647367)

[7.2.2. id00001001\_writeData 15](#_Toc167647368)

[7.2.3. id00001001\_readData 15](#_Toc167647369)

[7.2.4. id00001001\_startIP 15](#_Toc167647370)

[7.2.5. id00001001\_enableDelay 16](#_Toc167647371)

[7.2.6. id00001001\_disableDelay **¡Error! Marcador no definido.**](#_Toc167647372)

[7.2.7. id00001001\_enableINT 16](#_Toc167647373)

[7.2.8. id00001001\_disableINT 16](#_Toc167647374)

[7.2.9. id00001001\_status 16](#_Toc167647375)

[7.2.10. id00001001\_waitINT 16](#_Toc167647376)

## Lista de figuras

[Figure 1.1 IP Convulucionador conectado al host 1](#_Toc140501005)

[Figure 2.1 IP convulucionador estatus registro………………………………………………………………………………………. **¡Error! Marcador no definido.**](#_Toc140501006)

[Figure 2.2 Configuracion de registro. **¡Error! Marcador no definido.**](#_Toc140501007)

[Figure 5.3 Expected result after processing data with the Ip Dummy. **¡Error! Marcador no definido.**](file:///D:\Vidkar\Google%20Drive\IPDI\Documentación\Plantillas%20limpias\User%20manual%20-%20ID0001001.docx#_Toc140501008)

[Figure 6.1 IP Dummy status register 6](#_Toc140501009)

[Figure 6.2 Configuration delay register. 7](#_Toc140501010)

## Lista de tablas

[Table 1 IP Convulucionador input/output descripción de señal 4](#_Toc140488110)

# INPUT/OUTPUT DESCRIPCION DE DEÑALES

Table 1 IP Convulucionador input/output descripción de señal

|  |  |  |  |
| --- | --- | --- | --- |
| Señal | Ancho de bits | Dirección | Descripción |
| **Señal General** | | | |
| clk | 1 | Input | Sistema de reloj |
| rst\_a | 1 | Input | Reinicio del sistema asíncrono, activo bajo |
| en\_s | 1 | Input | Habilita la funcionalidad IP Core |
| **AIP Interfaz** | | | |
| data\_in | 32 | Input | Datos de entrada para configuración y procesamiento. |
| data\_out | 32 | Output | Datos de salida para procesar resultados y estados. |
| conf\_dbus | 5 | Input | Selecciona la configuración del bus para determinar el flujo de información desde/hacia el IP Core |
| write | 1 | Input | Indicación de escritura, los datos del bus data\_in se escribirán en la interfaz AIP de acuerdo con el valor de conf\_dbus |
| read | 1 | Input | Indicación de lectura, los datos de la interfaz AIP se leerán de acuerdo con el valor de conf\_dbus. El bus data\_out muestra los nuevos datos leídos. |
| start | 1 | Input | Inicializa el proceso IP Core |
| int\_req | 1 | Output | Solicitud de interrupción. Notifica determinados eventos según los bits de interrupción configurados. |
| **Core señal** | | | |
| data\_X | 8 | Input | Entrada de datos de la memoria externa que se va a procesar |
| memX\_addr | 5 | Output | Datos de salida para la dirección de memoria de envío para obtener los datos |
| data\_Y | 8 | Input | Entrada de datos de la memoria externa que se va a procesar |
| memY\_addr | 5 | Output | Datos de salida para la dirección de memoria de envío para obtener los datos |
| data\_Z | 16 | Output | Datos de salida para procesar el resultado de la convolución |
| memZ\_addr | 6 | Output | Datos de salida para la dirección de memoria de envío |
| writeZ | 1 | Output | Instrucción de escritura, los datos del proceso de convolución se escribirán en dataZ |
| Config\_in | 32 | Input | Entrada de datos para enviar como datos de data\_X y data\_Y como una sola palabra |
| busy\_out | 1 | Output | Datos de salida para la comprobación de estado. Si el proceso aún está activo, significa busy\_out está activo |
| done\_out | 1 | Output | Datos de salida para la comprobación de estado. Si el proceso ha terminado, significa que busy\_out está activo |

# TEORIA DE LA OPERACION

El convulucionador IP-core, es un módulo especializado para operaciones de convolución de datos. Proporcionando una solución ejecutable en hardwere, ofreciendo ventajas sobre la implementada en sofware. La convolución es una operación matemática que combina dos conjuntos de información. La forma de manejarse los datos se puede describir de la siguiente forma:

* Inicialización: Los datos de entrada y el kernel se cargan en sus respectivas ubicaciones de memoria dentro del núcleo IP.
* Deslizamiento del kernel: El kernel se desliza sobre la matriz de datos de entrada de forma sistemática, normalmente desde la parte superior izquierda hasta la parte inferior derecha. En cada posición, se realiza una multiplicación por elementos entre el kernel y la submatriz correspondiente de los datos de entrada.
* Suma: Los resultados de estas multiplicaciones se suman para producir un único valor de salida para cada posición del núcleo.
* Zancada y relleno: La zancada define el tamaño del paso por el cual el kernel se mueve a través de los datos de entrada. El relleno se puede aplicar a los bordes de los datos de entrada para controlar las dimensiones espaciales de la salida.
* Generación de salida: Los valores calculados se almacenan en la memoria de salida, formando la matriz de datos de salida.

La convolución se expresa matemáticamente de la siguiente forma:

Donde I son los datos de entrada, K es el kernel y m y n son las dimensiones del kernel.

# Descripción de registros y memorias de la interfaz AIP

## Registro de estado

Config: ESTATUS

Tamaño: 32 bits

Modo: Lectura/Escritura.

Este registro se divide en tres secciones:

* Bits de estado: Estos bits indican el estado actual del núcleo.
* Indicadores de interrupción: Estos bits se utilizan para generar una solicitud de interrupción en la señal int\_req de la interfaz AIP.
* Bits de máscara: Cada uno de estos bits puede habilitar o deshabilitar las banderas de interrupción.



Figure 2.1 IP convulucionador estatus registro

Bits 31:24 – Reservado, debe mantenerse despejado.

Bits 23:17 - Bits de máscara reservados para uso futuro y deben mantenerse desactivados.

Bit 16 – MSK: bit de máscara para el indicador de interrupción DN (Done). Si es necesario habilitar el indicador de interrupción DN, este bit debe escribirse en 1.

Bits 15:9 – Bits de estado reservado para uso futuro y se leen como 0.

Bit 8 – BSY: bit de estado "Ocupado".

La lectura de este bit indica el estado actual del convulucionador IP:

0: El convulucionador IP no está ocupado y listo para iniciar un nuevo proceso.

1: El convulucionador IP está ocupado y no está disponible para un nuevo proceso.

Bits 7:1 – Reservado Indicadores de interrupción/borrado para uso futuro y deben mantenerse desactivados.

Bit 0 – DN: indicador de interrupción/borrado "Hecho"

La lectura de este bit indica si el convulucionador IP ha generado una interrupción:

0: interrupción no generada.

1: el convulucionador IP ha finalizado con éxito su procesamiento.

Al escribir este bit en 1, se borrará el DN del indicador de interrupción.

## REGISTRO DE CONFIGURACION

Configuración: CConfig

Tamaño: 32 bits

Modo: Lectura-Escritura

Este registro se utiliza para establecer una matriz de cadenas a partir de las memorias de datos X e Y antes de que el núcleo inicie el proceso de convolución.



Figure 2.2 configuración de registro.

Bits 31:10 – Entradas de memoria de datos reservados. Debe mantenerse despejado.

Bits 4:0 – Tamaño X: Entrada X de memoria de datos. Se puede configurar con un tamaño mayor.

Bits 9:5 – Tamaño Y: Entradas de memoria de datos Y. Debe mantenerse despejado.

## Input data memoria (Mem X)

Config: MMemX

Tamaño: 32 bits

Modo: Escritura

Esta memoria se utiliza para almacenar datos de una memoria externa para ser procesados por el núcleo del convulucionador IP. El tamaño de esta memoria se establece como un parámetro de hardware antes de la síntesis.

## Input data memoria (Mem Y)

Config: MMemY

Tamaño: 32 bits

Modo: Escritura

Esta memoria se utiliza para almacenar datos de una memoria externa para ser procesados por el núcleo del convulucionador IP. El tamaño de esta memoria se establece como un parámetro de hardware antes de la síntesis.

## Output data memory (Mem Z)

Config: MMemZ

Tamaño: 32 bits

Modo: Lectura

Esta memoria se utiliza para almacenar los datos procesados por el convolutioner IP. Después de que el convolutioner IP complete su procesamiento, los datos almacenados en esta memoria serán el resultado de la convolución almacenada por Mem X y Mem Y. El tamaño de esta memoria se establece como un parámetro de hardware antes de la síntesis.

# PYTHON DRIVER

El archivo id1000500A.py contiene la definición de la clase convulucionador. Esta clase se utiliza para controlar el núcleo del convolutioner IP para aplicaciones de Python.

## Ejemplo de uso

En el siguiente código se presenta una prueba básica del núcleo del convulucionador IP. En primer lugar, es necesario crear una instancia de la clase de objeto convulucionador. El constructor de esta clase requiere la dirección de red y el puerto donde está conectado el convulucionador IP, el puerto de comunicación y la ruta donde se encuentra el archivo csv de configs. Por lo tanto, la comunicación con el convolutioner IP estará lista. En este código, se llama a una función 'conv'. La función escribe en ambas memorias con datos aleatorios mediante los métodos writeDataX y writeDataY. A continuación, se usa el método configurationRegister para establecer los datos en una sola palabra que se va a procesar y, a continuación, se usa el método startIP para iniciar el procesamiento principal. Finalmente, el método waitINT se usa para esperar la activación de la marca DONE y, después de eso, los datos de salida se leen con el método readData.

|  |
| --- |
| **import sys**  **try:**  **connector = '/dev/ttyACM0'**  **nic\_addr = 1**  **port = 0**  **csv\_file = '/home/a/Documents/HDL/ID1000500A\_AlejandroPardo\_LeonelGallo/config/ID1000500A\_config.csv'**  **aip = pyaip\_init(connector, nic\_addr, port, csv\_file)**  **aip.reset()**  **#==========================================**  **# Code generated with IPAccelerator**  **ID = aip.getID()**  **print(f'Read ID: {ID:08X}\n')**  **STATUS = aip.getStatus()**  **print(f'Read STATUS: {STATUS:08X}\n')**  **MemX = [0x00000001, 0x00000002, 0x00000003, 0x00000004, 0x00000003, 0x00000007]**  **print('Write memory: MdataX')**  **aip.writeMem('MdataX', MemX, 6, 0)**  **print(f'MemX Data: {[f"{x:08X}" for x in MemX]}\n')**  **MemY = [0x00000003, 0x00000003, 0x00000005, 0x00000006, 0x00000007]**  **print('Write memory: MdataY')**  **aip.writeMem('MdataY', MemY, 5, 0)**  **print(f'MemY Data: {[f"{x:08X}" for x in MemY]}\n')**  **Size = [0x000000A6]**  **print('Write configuration register: Csize')**  **aip.writeConfReg('Csize', Size, 1, 0)**  **print(f'Size Data: {[f"{x:08X}" for x in Size]}\n')**  **print('Start IP\n')**  **aip.start()**  **STATUS = aip.getStatus()**  **print(f'Read STATUS: {STATUS:08X}\n')**  **print('Read memory: MdataZ')**  **MemZ = aip.readMem('MdataZ', 10, 0)**  **print(f'MemZ Data: {[f"{x:08X}" for x in MemZ]}\n')**  **print('Clear INT: 0')**  **aip.clearINT(0)**  **STATUS = aip.getStatus()**  **print(f'Read STATUS: {STATUS:08X}\n')**  **#==========================================**  **aip.finish()**  **except:**  **e = sys.exc\_info()**  **print('ERROR: ', e)** |

## Metodos

### Constructor

**def** \_\_init\_\_**(**self**,** connector**,** nic\_addr**,** port, csv\_file**):**

Crea un objeto para controlar el convulucionador IP en la dirección de red especificada.

**Parámetros:**

* connector (string): Puerto de comunicaciones utilizado por el host.
* nic\_addr (int): Dirección de red donde está conectado el núcleo.
* port (int): Puerto donde se conecta el núcleo.
* csv\_file (string): Ubicación del archivo csv del convolutioner IP.

### writeDataX

**def** writeDataX**(**self**,** data**):**

Escriba la memoria de formulario de datos X en la memoria de entrada del convolutioner IP MMemX.

**Parámetros:**

* data (List[int]): Matriz de cadenas con los datos que se van a escribir.

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### writeDataY

**def** writeDataY**(**self**,** data**):**

Escriba la memoria de formulario de datos X en la memoria de entrada del convolutioner IP MMemX.

**Parámetros:**

* data (List[int]): Matriz de cadenas con los datos que se van a escribir.

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### readData

**def** readData**(**self**,** size**):**

Leer datos de la memoria de salida del convolutioner IP.

**Parámetros:**

* size (int): Puerto de comunicaciones utilizado por el host.

**Retorno:**

* List[int] Datos leídos de la memoria de salida.

### startIP

**def** startIP**(**self**):**

Inicie el procesamiento en IP Convolutioner.

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### ConfigurationRegister

**def** configurationRegister**(**self**,** dateSizeX, dataSizeY**):**

Establezca una matriz de cadenas asignada a partir de los métodos writeDataX y writeDataY en el procesamiento del convolutioner IP.

**Parámetros:**

* dataSizeX Datos asignados de MemX.
* dataSizeY Datos asignados de MemY.

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### enableINT

**def** enableINT**(**self**):**

Habilite las interrupciones del convulucionador IP (bit DONE del registro ESTATUS).

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### disableINT

**def** disableINT**(**self**):**

Deshabilite las interrupciones del convulucionador IP (bit DONE del registro ESTATUS).

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### status

**def** status**(**self**):**

Mostrar el estado del convulucionador IP.

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### waitINT

**def** waitINT**(**self**):**

Espere a que se complete el proceso.

**Retorno:**

* bool Una indicación de si la operación se ha completado correctamente.

### conv

**def** waitINT**(**self, X, Y**):**

El núcleo de funcionalidad del convolutioner IP. Lleva a cabo el proceso de convolución en los datos de entrada X e Y.

**Parámetros:**

* X Datos de entrada de Mem X.
* Y Datos de entrada de Mem Y.

**Retorno:**

* conv\_res(List[int]) Resultado del proceso de convolución.

# C DRIVER

Para utilizar el controlador C, es necesario utilizar los archivos: id1000500A.h, id01000500A.c que contienen la definición e implementación de las funciones del controlador. Las funciones definidas en esta biblioteca se utilizan para controlar el núcleo del convolutioner IP para aplicaciones C.

## Usage example

En el siguiente código se presenta una prueba básica del núcleo del convolutioner IP.

|  |
| --- |
| CODE SAMPLE:  int main(){  const char \*connector = "/dev/ttyACM0";  uint8\_t nic\_addr = 1;  uint8\_t port = 0;  const char \*csv\_file = "/home/a/Documents/HDL/ID1000500A\_AlejandroPardo\_LeonelGallo/config/ID1000500A\_config.csv";  caip\_t \*aip = caip\_init(connector, nic\_addr, port, csv\_file);  aip->reset();  /\*========================================\*/  /\* Code generated with IPAccelerator \*/  uint32\_t ID[1];  aip->getID(ID);  printf("Read ID: %08X\n\n", ID[0]);  uint32\_t STATUS[1];  aip->getStatus(STATUS);  printf("Read STATUS: %08X\n\n", STATUS[0]);  uint32\_t MemX[6] = {0x00000001, 0x00000002, 0x00000003, 0x00000004, 0x00000003, 0x00000007};  uint32\_t MemX\_size = sizeof(MemX) / sizeof(uint32\_t);  printf("Write memory: MdataX\n");  aip->writeMem("MdataX", MemX, 6, 0);  printf("MemX Data: [");  for(int i=0; i<MemX\_size; i++){  printf("0x%08X", MemX[i]);  if(i != MemX\_size-1){  printf(", ");  }  }  printf("]\n\n");  uint32\_t MemY[5] = {0x00000003, 0x00000003, 0x00000005, 0x00000006, 0x00000007};  uint32\_t MemY\_size = sizeof(MemY) / sizeof(uint32\_t);  printf("Write memory: MdataY\n");  aip->writeMem("MdataY", MemY, 5, 0);  printf("MemY Data: [");  for(int i=0; i<MemY\_size; i++){  printf("0x%08X", MemY[i]);  if(i != MemY\_size-1){  printf(", ");  }  }  printf("]\n\n");  uint32\_t Size[1] = {0x000000A6};  uint32\_t Size\_size = sizeof(Size) / sizeof(uint32\_t);  printf("Write configuration register: Csize\n");  aip->writeConfReg("Csize", Size, 1, 0);  printf("Size Data: [");  for(int i=0; i<Size\_size; i++){  printf("0x%08X", Size[i]);  if(i != Size\_size-1){  printf(", ");  }  }  printf("]\n\n");  printf("Start IP\n\n");  aip->start();  aip->getStatus(STATUS);  printf("Read STATUS: %08X\n\n", STATUS[0]);  uint32\_t MemZ[10];  uint32\_t MemZ\_size = sizeof(MemZ) / sizeof(uint32\_t);  printf("Read memory: MdataZ\n");  aip->readMem("MdataZ", MemZ, 10, 0);  printf("MemZ Data: [");  for(int i=0; i<MemZ\_size; i++){  printf("0x%08X", MemZ[i]);  if(i != MemZ\_size-1){  printf(", ");  }  }  printf("]\n\n");  printf("Clear INT: 0\n");  aip->clearINT(0);  aip->getStatus(STATUS);  printf("Read STATUS: %08X\n\n", STATUS[0]);  /\*========================================\*/  aip->finish();  printf("\n\nPress key to close ... ");  getch();  return 0;  } |

## Driver functions

### Id1000500A\_init

int32\_t id1000500A\_init**(**const char \*connector**,** uint\_8 nic\_addr**,** uint\_8 port, const char \*csv\_file**)**

Configure e inicialice la conexión para controlar el convolutioner IP en la dirección de red especificada.

**Parámetros:**

* connector: Puerto de comunicaciones utilizado por el host.
* nic\_addr: Dirección de red donde está conectado el núcleo.
* port: Puerto donde se conecta el núcleo.
* csv\_file: IP Convolutioner csv file location.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### Id1000500A\_writeData

int32\_t id1000500A\_writeData**(**uint32\_t **\***dataX**,** uint32\_t sizeX, uint32\_t **\***dataY**,** uint32\_t sizeY**)**

Escriba datos en ambas memorias de entrada del convolutioner IP.

**Parámetros:**

* dataX: Puntero al primer elemento que se escribirá desde Mem X.
* sizeX: Número de elementos que se escribirán para Mem Y.
* dataY: Puntero al primer elemento que se escribirá desde Mem Y.
* sizeY: Número de elementos que se escribirán para Mem Y.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_readData

int32\_t id1000500A\_readData**(**uint32\_t **\***data**,** uint32\_t data\_size, uint\_8 nic\_addr**,** uint\_8 port**)**

Leer datos de la memoria de salida del convolutioner IP.

**Parámetros:**

* data: Puntero al primer elemento donde se almacenarán los datos leídos.
* data\_size: Número de elementos a leer.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_startIP

int32\_t id1000500A\_startIP**()**

Inicie el procesamiento en IP Convulucionador.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_configurationRegister

int32\_t id1000500A\_ configurationRegister**(**int\_8 sizeX**,** uint\_8 sizeY**)**

Establezca una matriz de cadenas asignada desde id1000500A\_writeData en el procesamiento del convolutioner IP.

**Parámetros:**

* sizeX: Datos asignados de Mem X.
* sizeY: Datos asignados de Mem Y.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_enableINT

int32\_t id1000500A\_enableINT**()**

Habilite las interrupciones del convolutioner IP (bit DONE del registro STATUS).

### id1000500A\_disableINT

int32\_t id1000500A\_disableINT**(**int\_8 nic\_addr**,** uint\_8 port**)**

Deshabilite las interrupciones del convolucionador IP (bit DONE del registro STATUS)

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_status

int32\_t id1000500A\_status**()**

Mostrar el estado del convolutioner IP.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_waitINT

int32\_t id1000500A\_status**()**

Espere a que se complete el proceso.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_finish

int32\_t id1000500A\_finish**()**

Finalice la conexión de convolución IP.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_clearStatus

int32\_t id1000500A\_ clearStatus**()**

Borre el registro de estado.

**Retorno:**

* int32\_t Devuelve 0 si la función se ha completado correctamente.

### id1000500A\_conv

int32\_t id1000500A\_conv **(**uint\_8\* X, uint\_8 sizeX, uint\_8\* Y, uint\_8 sizeY, uint\_16\* result**)**

Núcleo de funcionalidad principal. Lleva a cabo el proceso de convolución.

**Parámetros:**

dataX: Puntero al primer elemento que se escribirá desde Mem X.

sizeX: Número de elementos que se escribirán para Mem X.

dataY: Puntero al primer elemento que se escribirá desde Mem Y.

sizeY: Número de elementos que se escribirán para Mem Y.

result: Resultado del proceso de convolución.

**Retorno:**

* int16\_t Devuelve el resultado si la función se ha completado correctamente.