ID1000500C
CONVOLVER IP-CORE USER MANUAL

1. DESCRIPTION

The Convolver IP – core is a basic processing system unit to compute the convolution between two signals. The characteristics of this IP – core allow the user to choose the “full” or “same” mode to compute the full size convolution of the aforementioned two signals or its central convolution.

After receiving a start command, and the size configuration sets and Shape, this IP-core computes the full or central convolution between two signals provided from two memories with 32 slots each.

Depending on the size and the shape of the convolution the Convolution operation can reach till 61 samples at the output at full mode with a word width of 16 bits.

1.1. CONFIGURABLE FEATURES

Software configurations	**Description**
Processing convolution operation	Computes the convolution operation between two signals of different sizes with till 31 samples per signal.
Changing the Shape of the convolution	Allows the convolution operation to change between “Full” mode convolution or “same” mode.
Changing of the signal	With this feature the convolver IP – Core can perform a convolution till 31 samples per signal, changing the read address of at the inputs of the aip interface.

1.2. TYPICAL APPLICATION

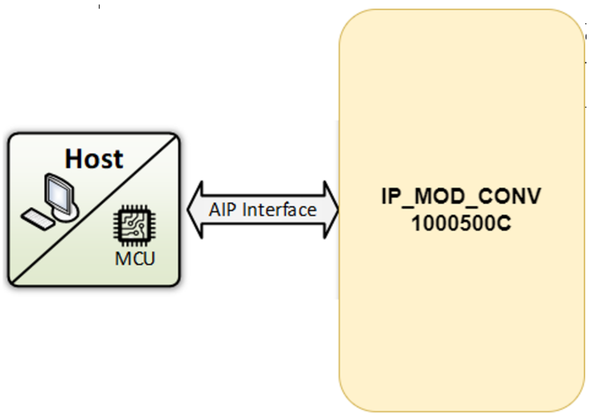


Figure 1.1 IP Convolver connected to a host

2. CONTENTS

2.1. List of figures

2.2. List of tables

3. INPUT/OUTPUT SIGNAL DESCRIPTION

Table 1 IP Convolver input/output signal description

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_in	32	Input	Input data for configuration and processing
data_out	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
write	1	Input	Write indication, data from the data_in bus will be written into the AIP Interface according to the conf_dbus value
read	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.
start	1	Input	Initializes the IP Core process
int_req	1	Output	Interruption request. It notifies certain events according to the configurated interruption bits.
Core signals			
clk	1	Input	System clock
rstn	1	Input	Asynchronous system reset, low active.
start	1	Input	Start flag to initialize the process.
shape	1	Input	Shape flag to choose between full or same convolution mode
sizex	5	Input	Size of the X signal. Define the number of sample in signal X.
sizey	5	Input	Size of the Y signal. Define the number of sample in signal Y.
datax	8	Input	Data input for convolution from X signal.
datay	8	Input	Data input for convolution from Y signal.
memz_addr	6	Output	write pointer to write the result of the process to aip memory.

memx_addr	5	Output	Read pointer to choose the samples at X memory.
memy_addr	5	Output	Read pointer to choose the samples at Y memory.
data_z_o	16	Output	Data output of the IP - core
busy	1	Output	Busy flag.
done	1	Output	Done process flag.
we_z	1	Output	Write Enable Signal to write the result of the output to the aip memory.

4. THEORY OF OPERATION

The convolver IP Core basically, computes the convolution operation between two signal (till 31 samples each) applying the basic algorithm of the discrete convolution [1]. The core also allows the change of the sizes of the signals to compute every convolution in the range of 1 to 31. Once those values are set, the user choose which convolution mode want to use (Full Size or Same size) and once the Done flag is activated the result is ready.
[1] - The convolution of two vectors, u and v, represents the area of overlap under the points when v spans the same length as u. Algebraically, convolution is the same operation as multiplying polynomials whose coefficients are the elements of u and v. :retrieved from: https://la.mathworks.com/help/matlab/ref/conv.html

Suppose m = length(u) and n = length(v). w is the vector of length m+n-1 whose k-th element is:

$$w(k) = \sum_j u(j)v(k - j + 1).$$

As well as the IP-Core of the Dummy the value of the maximun clock frequency was set to 50 MHz, which is commonly used in commercial FPGAs.

5. AIP interface registers and memories description

5.1. Status register

Config: STATUS

Size: 32 bits

Mode: Read/Write.

This register is divided in 3 sections, see Figure 5.1:

- **Status Bits**: These bits indicate the current state of the core.
- **Interruption Flags:** These bits are used to generate an interruption request in the *int_req* signal of the AIP interface.
- **Mask Bits**: Each one of these bits can enable of disable the interruption flags.

Status Register

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

Figure 5.1 IP Convolver status register

Bits 31:24 – Reserved, must be kept cleared.

Bits 23:17 – Reserved Mask Bits for future use and must be kept cleared.

Bit 16 – **MSK:** mask bit for the DN (Done) interruption flag. If it is required to enable the DN interruption flag, this bit must be written to 1.

Bits 15:9 – Reserved Status Bits for future use and are read as 0.

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

Reading this bit indicates the current IP Dummy state:

0: The IP Dummy is not busy and ready to start a new process.

1: The IP Dummy is busy, and it is not available for a new process.

Bits 7:1 – Reserved Interrupt/clear flags for future use and must be kept cleared.

Bit 0 – **DN**: interrupt/clear flag “**Done**”

Reading this bit indicates if the IP Dummy has generated an interruption:

0: interruption not generated.

1: the IP Dummy has successfully finished its processing.

Writing this bit to 1 will clear the interruption flag DN.

5.2. Configuration delay register

Config: CCONF_REG
Size: 32 bits
Mode: Write

This register is used to set the sizes of X and Y signal sizes and the shape of the convolution operation See Figure 5.2

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
Shape/Sizes[30:0]																															
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

Figure 5.2 Configuration Shape/sizes Register. The yellow color corresponds to the size of the X signal. The blue one to the Y signal and the remaining bit (bit 10) to the Shape bit.

Bits 4:0 – **X signal size:** Size of the X signal. The size of the X signal can be since 0 to 31 and depending on that size the core will take the size for the resulting signal.

Bits 9:5 – **Y signal size:** Size of the Y signal. The size of the Y signal can be since 0 to 31 and depending on that size the core will take the size for the resulting signal.

Bit 10 – **Shape Bit:** Shape bit to Choose Full or Same convolution mode.

5.3. Input data memories

Config: MMEM_X
Size: Nx32 bits (N=32)
Mode: Write

This memory is used to store data to X Signal to be processed by the IP Convolver. The size of this memory is set as a hardware parameter before the synthesis. It has support for storing 32 Due the Characteristics of the IP Core.

Config: MMEM_Y
Size: Nx32 bits (N=32)
Mode: Write

This memory is used to store data to Y Signal to be processed by the IP Convolver. The size of this memory is set as a hardware parameter before the synthesis. It has support for storing 32 Due the Characteristics of the IP Core.

5.4. Output data memory

Config: MMEM_Z
Size: Nx32 bits (N=64)
Mode: Read

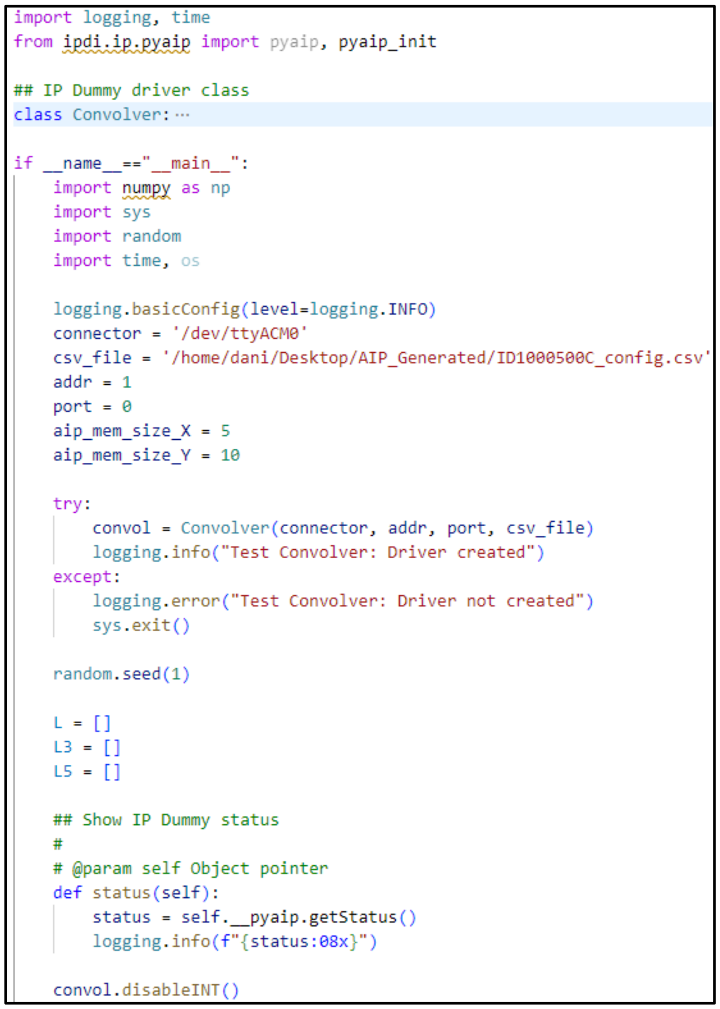
This memory is used to store processed data by the IP Convolver. After the IP Convolver completes its processing, the data stored in this memory will be the size of the X+M-1 or only X size depending on the Shape bit. The size of this memory is set as a hardware parameter before the synthesis. It has support for storing 64.

6. PYTHON DRIVER

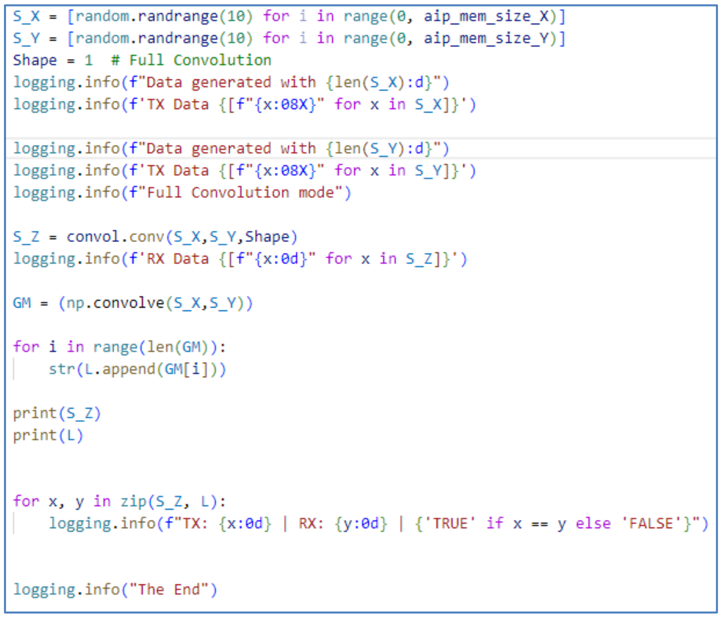
The file *id1000500C.py* contains the class *Convolver* definition. This class is used to control the IP convolver core for python applications.

6.1. Usage example

In the following code a basic test of the IP Convolver core is presented. First, it is required to create an instance of the *Convolver* object class. The constructor of this class requires the network address and port where the IP Convolver is connected, the communication port, and the path where the configs csv file is located. Thus, the communication with the IP Convolver will be ready.



```
import logging, time
from ipdi.ip.pyaip import pyaip, pyaip_init

## IP Dummy driver class
class Convolver:···

if __name__=="__main__":
    import numpy as np
    import sys
    import random
    import time, os

    logging.basicConfig(level=logging.INFO)
    connector = '/dev/ttyACM0'
    csv_file = '/home/dani/Desktop/AIP_Generated/ID1000500C_config.csv'
    addr = 1
    port = 0
    aip_mem_size_X = 5
    aip_mem_size_Y = 10

    try:
        convol = Convolver(connector, addr, port, csv_file)
        logging.info("Test Convolver: Driver created")
    except:
        logging.error("Test Convolver: Driver not created")
        sys.exit()

    random.seed(1)

    L = []
    L3 = []
    L5 = []

    ## Show IP Dummy status
    #
    # @param self Object pointer
    def status(self):
        status = self.__pyaip.getStatus()
        logging.info(f"{status:08x}")

    convol.disableINT()
```



```
S_X = [random.randrange(10) for i in range(0, aip_mem_size_X)]
S_Y = [random.randrange(10) for i in range(0, aip_mem_size_Y)]
Shape = 1  # Full Convolution
logging.info(f"Data generated with {len(S_X):d}")
logging.info(f'TX Data {[f"{x:08X}" for x in S_X]}')

logging.info(f"Data generated with {len(S_Y):d}")
logging.info(f'TX Data {[f"{x:08X}" for x in S_Y]}')
logging.info(f"Full Convolution mode")

S_Z = convol.conv(S_X,S_Y,Shape)
logging.info(f'RX Data {[f"{x:0d}" for x in S_Z]}')

GM = (np.convolve(S_X,S_Y))

for i in range(len(GM)):
    str(L.append(GM[i]))

print(S_Z)
print(L)


for x, y in zip(S_Z, L):
    logging.info(f"TX: {x:0d} | RX: {y:0d} | {'TRUE' if x == y else 'FALSE'}")


logging.info("The End")
```

After that the Driver is created and then the samples for the X an Y memories are randomly created, the Shape is setted and these values are set into the function called conv. The function conv is defined inside the Convolver class and make the necessary to establish all the values into the memories of the interface, write the sizes at the inputs of the convolver and give the core a start flag to initialize its process. Then, the result of the convolution is returned to the main code and we can compare the result with the result of the convolution of the same signals of the convolution function took it from python.

6.2. Methods

6.2.1. Constructor

```
def __init__(self, connector, nic_addr, port, csv_file):
```

Creates an object to control the IP Dummy in the specified network address.

Parameters:

• connector (string):	Communications port used by the host.
• nic_addr (int):	Network address where the core is connected.
• port (int):	Port where the core is connected.
• csv_file (string):	IP Dummy csv file location.

6.2.2. Conv Function

```
def conv(self,S_X,S_Y,Shape):
```

Convolution Operation, the operation involves get the ID of the Convolver core with the interface, the Status, write the memory values from the X and Y signals, define the sizes and the final size of the result. Also, write the Configuration register to finally send an start flag. After that, a polling process was made, when the Done flag is set the process is finished. The Int flag is cleared, and the result is read, returning this to the main code.

Parameters:

- S_X (List[int]): Data to be written at X signal.
- S_Y (List[int]): Data to be written at Y signal.
- Shape(integer)

Returns:

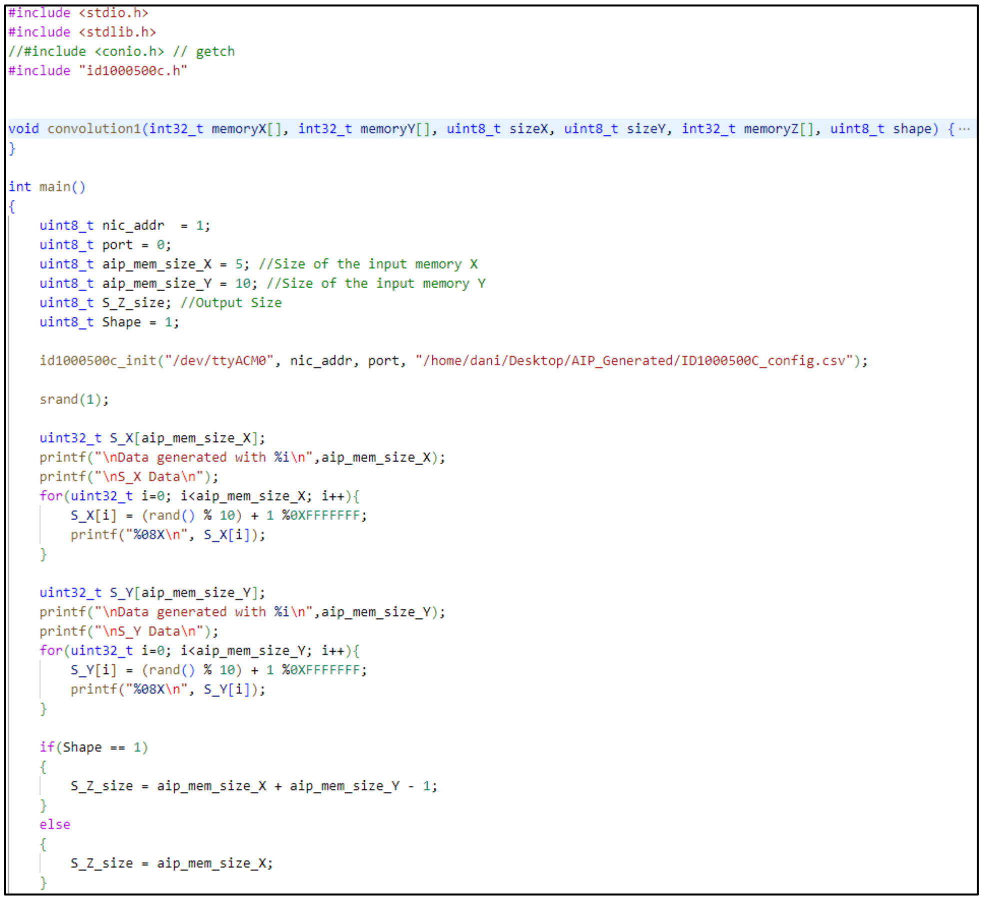
- S_Z (List[int]): the result of the convolution process.

7. C DRIVER

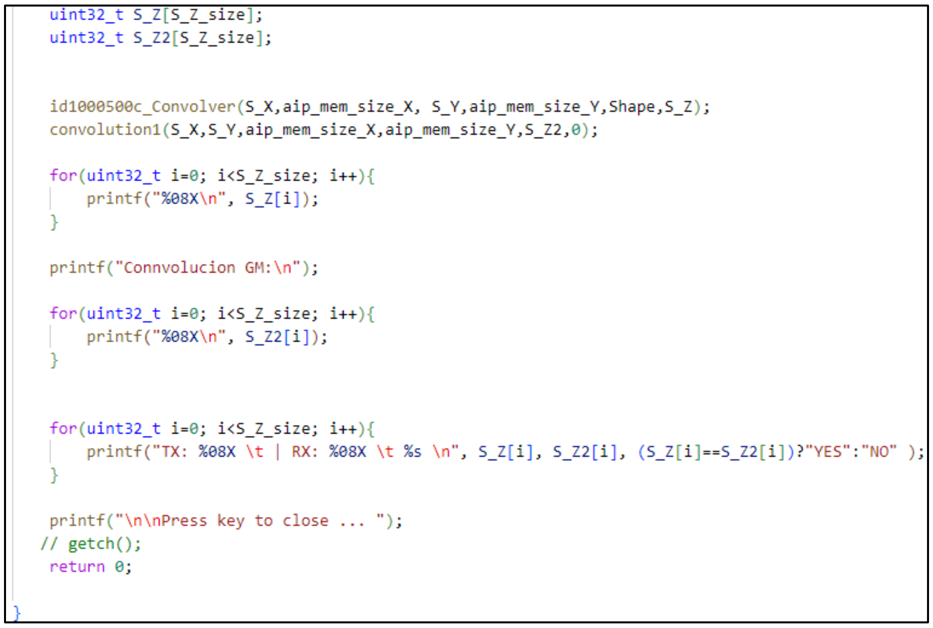
In order to use the C driver, it is required to use the files: *id1000500c.h, id1000500c.c* that contain the driver functions definition and implementation. The functions defined in this library are used to control the IP Convolver core for C applications.

7.1. Usage example

In the following code a basic test of the IP Convolver core is presented.



```c
#include <stdio.h>
#include <stdlib.h>
//#include <conio.h> // getch
#include "id1000500c.h"


void convolution1(int32_t memoryX[], int32_t memoryY[], uint8_t sizeX, uint8_t sizeY, int32_t memoryZ[], uint8_t shape) {⋯
}

int main()
{
    uint8_t nic_addr  = 1;
    uint8_t port = 0;
    uint8_t aip_mem_size_X = 5; //Size of the input memory X
    uint8_t aip_mem_size_Y = 10; //Size of the input memory Y
    uint8_t S_Z_size; //Output Size
    uint8_t Shape = 1;

    id1000500c_init("/dev/ttyACM0", nic_addr, port, "/home/dani/Desktop/AIP_Generated/ID1000500C_config.csv");

    srand(1);

    uint32_t S_X[aip_mem_size_X];
    printf("\nData generated with %i\n",aip_mem_size_X);
    printf("\nS_X Data\n");
    for(uint32_t i=0; i<aip_mem_size_X; i++){
        S_X[i] = (rand() % 10) + 1 %0XFFFFFFF;
        printf("%08X\n", S_X[i]);
    }

    uint32_t S_Y[aip_mem_size_Y];
    printf("\nData generated with %i\n",aip_mem_size_Y);
    printf("\nS_Y Data\n");
    for(uint32_t i=0; i<aip_mem_size_Y; i++){
        S_Y[i] = (rand() % 10) + 1 %0XFFFFFFF;
        printf("%08X\n", S_Y[i]);
    }

    if(Shape == 1)
    {
        S_Z_size = aip_mem_size_X + aip_mem_size_Y - 1;
    }
    else
    {
        S_Z_size = aip_mem_size_X;
    }
```



```
    uint32_t S_Z[S_Z_size];
    uint32_t S_Z2[S_Z_size];


    id1000500c_Convolver(S_X,aip_mem_size_X, S_Y,aip_mem_size_Y,Shape,S_Z);
    convolution1(S_X,S_Y,aip_mem_size_X,aip_mem_size_Y,S_Z2,0);

    for(uint32_t i=0; i<S_Z_size; i++){
        printf("%08X\n", S_Z[i]);
    }

    printf("Connvolucion GM:\n");

    for(uint32_t i=0; i<S_Z_size; i++){
        printf("%08X\n", S_Z2[i]);
    }


    for(uint32_t i=0; i<S_Z_size; i++){
        printf("TX: %08X \t | RX: %08X \t %s \n", S_Z[i], S_Z2[i], (S_Z[i]==S_Z2[i])?"YES":"NO" );
    }

    printf("\n\nPress key to close ... ");
   // getch();
    return 0;

}
```

In order to explain the functionality of the Core, it is necessary to say that the C algorithm do the same process that the python one driver. Basically, the Python Driver was translated to C.

7.2. Driver functions

7.2.1. id00001001_init

```
int32_t id00001001_init(const char *connector, uint_8 nic_addr, uint_8 port,
const char *csv_file)
```

Configure and initialize the connection to control the IP Dummy in the specified network address.

Parameters:

• connector:	Communications port used by the host.
• nic_addr:	Network address where the core is connected.
• port:	Port where the core is connected.
• csv_file:	IP Dummy csv file location.

Returns:

• int32_t	Return 0 whether the function has been completed successfully.

7.2.2. Id1000500c_Convolver

```
int32_t id00001001_Convolver((uint32_t S_X[],uint8_t size_x, uint32_t
S_Y[],uint8_t size_y, uint8_t Shape, uint32_t S_Z[])
```

Convolution Operation, the operation involves get the ID of the Convolver core with the interface, the Status, write the memory values from the X and Y signals, define the sizes and the final size of the result. Also, write the Configuration register to finally send an start flag. After that, a polling process was made, when the Done flag is set the process is finished. The Int flag is cleared, and the result is read, returning this to the main code.

Parameters:

• S_X :	Data to be written at X signal.
• S_Y :	Data to be written at Y signal.
• Shape:	Shape of the convolution process.
• Size_X:	Size of the X signal.
• Size_Y:	Size of the Y signal.

Returns:

• S_Z:	Result of the convolution process.