ID1000500B
CONVOLUTION COPROCESSOR IP-CORE USER MANUAL

1. DESCRIPTION

The convolution coprocessor is a system that performs the convolution operation between two discrete signals, one of the signals is internal, so it has a fixed value and size, and the other signal is part of an external memory, so this signal must be introduced to the system to enter the size of this signal by means of a configuration register.

1.1. CONFIGURABLE FEATURES

Software configurations	**Description**
sizeY	Size of the input signal.

1.2. TYPICAL APPLICATION

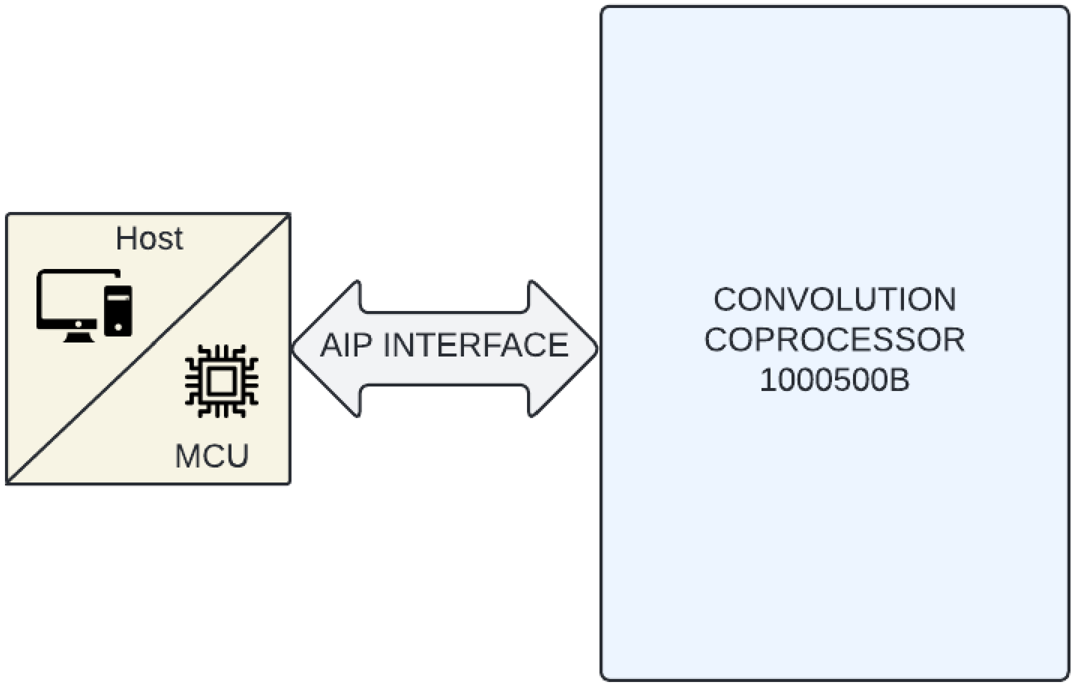

Figure 1.1 IP Convolution coprocessor connected to a host

2. CONTENTS

2.1. List of figures

2.2. List of tables

3. INPUT/OUTPUT SIGNAL DESCRIPTION

Table 1 IP Convolution coprocessor 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			
start	1	Input	Start the process of convolution using the actual values stored in the memories.
busy	1	Output	Indicates the state of the process and when the core is occupied.
done	1	Output	Interruption that indicates when the process has finished successfully.
sizeY	5	Input	Input data that indicate the size of the input signal.
dataY	8	Input	Input data, data from the external memory that stores the signal convolution input.
addrY	5	Output	Adress of the external memory that is wanted to be accessed in that moment.
dataZ	16	Output	Data output of the result of the convolution.
adrrZ	6	Output	Adress of the data result that is meant to write in the external memory.
writeEnZ	1	Output	Signal that enable the writing in the external memory.

4. THEORY OF OPERATION

The convolution coprocessor receives an input signal from an external memory to be able to perform the main operation, however it needs to know how big this signal is, in order to know how far to stop reading data from the input memory, so the configuration register has the task of introducing this data to the core.

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.

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

Figure 5.1 IP Convolution coprocessor 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 Convolution coprocessor state:
0: The IP Convolution coprocessor is not busy and ready to start a new process.
1: The IP Convolution coprocessor 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 Convolution coprocessor has generated an interruption:
0: interruption not generated.
1: the IP Convolution coprocessor has successfully finished its processing.
Writing this bit to 1 will clear the interruption flag DN.

5.2. Configuration input memory size register

Config: CSIZE_Y
Size: 32 bits
Mode: Write

This register is used to configure the size of the input signal. See Figure 5.2

31 … 5	4	3	2	1	0
NA	sizeY [4:0]				
NA	w	w	w	w	w

Figure 5.2 Configuration delay register.

Bits 31:5 – **NA:** These bits have no use in the system

Bits 4:0 – **sizeY**: Size of the input signal, it can take values from 0 to 31.

5.3. Input data memory

Config: MMEM_Y
Size: Nx32 bits (N=32,64,128,256)
Mode: Write

This memory stores the system input data with which the convolution operation will be performed in the core.

5.4. Output data memory

Config: MMEM_Z
Size: Nx32 bits (N=32,64,128,256)
Mode: Read

This memory stores the output data resulting from the convolution operation executed by the core.

6. PYTHON DRIVER

The file *id1000500B.py* contains the convolution coprocessor class definition. This class is used to control the IP Convolution coprocessor core for python applications.

6.1. Usage example

In the following code a basic test of the IP Convolution coprocessor core is presented. First, it is required to create an instance of the convolution_coprocessor object class. The constructor of this class requires the network address and port where the IP Convolution coprocessor is connected, the communication port, and the path where the configs csv file is located. Thus, the communication with the IP convolution coprocessor will be ready. In this code, the input memory is written with random data by using the writeData method. Then, the size of the input signal is entered into the configuration register whit the function writeConfigReg, for which a start is then executed. Finally, the waitINT method is used to wait the activation of the DONE flag, and after that, the output data is read with the readData method.

```
import sys, random, time, os

    logging.basicConfig(level=logging.DEBUG)
    connector = '/dev/ttyACM0'
    csv_file = '/home/patricio/Descargas/conv_temp/ID1000500B_config.csv'
    aip_mem_size = 10
    addr = 1
    port = 0
    try:
        conv_cop = convoution_coprocessor(connector, addr, port, csv_file)
        logging.info("Test convoution_coprocessor: Driver created")
    except:
        logging.error("Test convoution_coprocessor: Driver not created")
        sys.exit()

    random.seed(time.time())

    WR = [random.randrange(2 ** 8) for i in range(0, aip_mem_size)]

    #WR = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    result = conv_cop.conv(WR)

    expected = get_convolution(WR)
    compare_results(result, expected)

    logging.info("The End")

def conv(self, X):

    conv_cop.setSize(len(X))

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

    conv_cop.startIP()
    conv_cop.waitInt()

    RD = conv_cop.readData(len(X)+4)

    conv_cop.status()
    conv_cop.__clearStatus()
    conv_cop.status()

    conv_cop.finish()
    return RD
```

```
# gets convolution from the golden model to get expected results
def get_convolution(X):
    sizeH = 5
    size  = len(X)
    H = [0x04, 0x30, 0x13, 0x0A, 0x26]
    Z = []
    i = 0
    while i < (size + sizeH - 1):
        currentZ = 0
        j = 0
        while j < size:
            if (i-j) >= 0 and (i-j) < sizeH:
                currentZ += H[i-j] * X[j]
            j += 1
        Z.append(currentZ)
        i += 1
    return Z


# compares results between the results gotten by the hw accelerator
# and the expected results
def compare_results(result, rExpected):
    for x, y in zip(result, rExpected):
         logging.info(f"Got: {x:08x} | Expected: {y:08x} | {'TRUE' if x == y else 'FALSE'}")
```

The driver uses two independent functions that perform the printing and comparison of the convolution result with the expected result of the gold model.

The get_convolution function returns the value of the convolution operation of the external signal with the internal function using the same process as the gold model.

The compare_results function receives the expected result returned by the get_convolution function and compares it with the result of this system and prints if both results are equal, indicating it one by one of output signal.

6.2. Methods

6.2.1. Constructor

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

Creates an object to control the IP Convolution coprocessor 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 Convolution coprocessor csv file location.

6.2.2. writeData

```
def writeData(self, data):
```

Write data in the IP Convolution coprocessor input memory.

Parameters:

- data (List[int]): Data to be written.

Returns:

- bool An indication of whether the operation has been completed successfully.

6.2.3. readData

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

Read data from the IP Convolution coprocessor output memory.

Parameters:

- size (int): Communications port used by the host.

Returns:

- List[int] Data read from the output memory.

6.2.4. startIP

```
def startIP(self):
```

Start processing in IP Convolution coprocessor.

Returns:

- bool An indication of whether the operation has been completed successfully.

6.2.5. enableINT

```
def enableINT(self):
```

Enable IP Convolution coprocessor interruptions (bit DONE of the STATUS register).

Returns:

- bool An indication of whether the operation has been completed successfully.

6.2.6. disableINT

```
def disableINT(self):
```

Disable IP Convolution coprocessor interruptions (bit DONE of the STATUS register).

Returns:

- bool An indication of whether the operation has been completed successfully.

6.2.7. status

```
def status(self):
```

Show IP Convolution coprocessor status.

Returns:

- bool An indication of whether the operation has been completed successfully.

6.2.8. waitINT

```
def waitINT(self):
```

Wait for the completion of the process.

Returns:

- bool An indication of whether the operation has been completed successfully.

6.2.9. conv

```
def conv(self, X):
```

It performs the whole convolution process described in the above example of use, so it writes to the configuration register, writes to the input memory, uses the start, reads the output and clears the interrupt, the convolution is stored in the result pointer.

Parameters:

- X (List[int]): Data input for the convolution operation.

Returns:

- List[int] Result of the convolution operation.

6.2.10. setSize

```
def conv(self, size):
```

This function writes to the configuration register the data passed as a parameter, thus setting the size of the input signal.

Parameters:

- size: Data input for the convolution operation.

Returns:

- bool An indication of whether the operation has been completed successfully.

7. C DRIVER

In order to use the C driver, it is required to use the files: *id100500b.h, id1000500b.c* that contain the driver functions definition and implementation. The functions defined in this library are used to control the IP convolution coprocessor core for C applications.

7.1. Usage example

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

```
int main()
{
    uint8_t nic_addr  = 1;
    uint8_t port = 0;
    uint8_t aip_mem_size = 10; //Size of the input and output memories

    id1000500b_init("/dev/ttyACM0",nic_addr,port,
"/home/patricio/Descargas/conv_temp/ID1000500B_config.csv");

    srand(time(0));

    uint16_t* RD = (uint16_t*) malloc(sizeof(uint16_t)*(aip_mem_size+SizeH-Adjust));
    uint8_t* WR = (uint8_t*) malloc(sizeof(uint8_t)*(aip_mem_size));

    printf("\nData generated with %i\n",aip_mem_size);
    printf("\nTX Data\n");

    for(uint32_t i=0; i<aip_mem_size; i++){
        WR[i] = rand() %0XFFFFFFFF;
        printf("%08X\n", WR[i]);
    }

    uint16_t* Expected = (uint16_t*) malloc(sizeof(uint16_t)*(aip_mem_size+SizeH-Adjust));

    id1000500b_conv(WR, aip_mem_size, RD);

    printf("\n");

    get_convolution(WR, aip_mem_size, Expected);
    compare_results(RD, aip_mem_size, Expected);

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

}



void get_convolution(uint8_t* X, uint8_t size, uint16_t* rExpected){
    const int sizeH = SizeH;
    const uint8_t H[5] = {0x04, 0x30, 0x13, 0x0A, 0x26};
    int i, j, currentZ;

    i = 0;
    while(i < (size + sizeH - Adjust)){
        currentZ = 0;
        j = 0;
        while(j < size){
            if(((i - j) >= 0) && ((i - j) < sizeH)){
                currentZ += H[i - j] * X[j];
            }
            j++;
        }
        rExpected[i] = currentZ;
```

```
        i++;
    }
}

void compare_results(uint16_t* result, uint8_t size, uint16_t* rExpected){
    printf("Memory \t Expected \t Got \t\t Status\n");
    for(int i = 0; i < (size + SizeH - Adjust); i++){
        printf("%i    \t    %08X    \t    %08X    \t    %s    \n",    i,    rExpected[i],    result[i],
(result[i]==rExpected[i])?"OK":"ERROR" );
    }
}


/* Execute convolution*/
uint32_t id1000500b_conv(uint8_t* X, uint8_t size, uint16_t* result){

    uint16_t sizeZ = size+SizeH-Adjust;

    id1000500b_setSize(size);

    //Cast to uint32_t
    uint32_t* Xdata32 = (uint32_t*) malloc(sizeof(uint32_t)*size);
    for (uint32_t i = 0; i < size; i++){
        Xdata32[i]=(uint32_t)X[i];
    }

    id1000500b_writeData(Xdata32, size);

    id1000500b_startIP();

    id1000500b_waitINT();

    printf("\n\n Done detected \n\n");

    uint32_t* RD = (uint32_t*) malloc(sizeof(uint32_t)*(sizeZ));
    id1000500b_readData(RD, sizeZ);

    id1000500b_status();
    id1000500b_clearStatus();
    id1000500b_status();
    id1000500b_finish();

    //Cast to uint16_t
    for (uint32_t i = 0; i < sizeZ; i++){
        result[i]=(uint16_t)RD[i];
    }
    free(RD);
    free(Xdata32);

    return 0;
}
```

The driver uses two independent functions that perform the printing and comparison of the convolution result with the expected result of the gold model.

The get_convolution function returns the value of the convolution operation of the external signal with the internal function using the same process as the gold model.

The compare_results function receives the expected result returned by the get_convolution function and compares it with the result of this system and prints if both results are equal, indicating it one by one of output signal.

7.2. Driver functions

7.2.1. id1000500b_init

```
int32_t id1000500b_init(const char *connector, uint_8 nic_addr, uint_8 port,
const char *csv_file)
```

Configure and initialize the connection to control the IP Convolution coprocessor 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 Convolution coprocessor csv file location.

Returns:

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

7.2.2. id1000500b_writeData

```
int32_t id1000500b_writeData(uint32_t *data, uint32_t data_size, uint_8
nic_addr, uint_8 port)
```

Write data in the IP Convolution coprocessor input memory.

Parameters:

- data: Pointer to the first element to be written.
- data_size: Number of elements to be written.
- nic_addr: Network address where the core is connected.
- port: Port where the core is connected.

Returns:

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

7.2.3. id1000500b_readData

```
int32_t id1000500b_readData(uint32_t *data, uint32_t data_size, uint_8
nic_addr, uint_8 port)
```

Read data from the IP Convolution coprocessor output memory.

Parameters:

- data: Pointer to the first element where the read data will be stored.
- data_size: Number of elements to be read.
- nic_addr: Network address where the core is connected.
- port: Port where the core is connected.

Returns:

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

7.2.4. id1000500b_startIP

```
int32_t id1000500b_startIP(uint_8 nic_addr, uint_8 port)
```

Start processing in IP Convolution coprocessor.

Parameters:

- nic_addr: Network address where the core is connected.
- port: Port where the core is connected.

Returns:

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

7.2.5. id1000500b_enableINT

```
int32_t id1000500b_enableINT(int_8 nic_addr, uint_8 port)
```

Enable IP Convolution coprocessor interruptions (bit DONE of the STATUS register).

Returns:

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

7.2.6. id1000500b_disableINT

```
int32_t id1000500b_disableINT(int_8 nic_addr, uint_8 port)
```

Disable IP Convolution coprocessor interruptions (bit DONE of the STATUS register).

Returns:

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

7.2.7. id1000500b_status

```
int32_t id1000500b_status(int_8 nic_addr, uint_8 port)
```

Show IP Convolution coprocessor status.

Returns:

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

7.2.8. id1000500b_waitINT

```
int32_t id1000500b_status(int_8 nic_addr, uint_8 port)
```

Wait for the completion of the process.

Returns:

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

7.2.9. id1000500b_conv

```
int32_t id1000500b_conv(uint_8* X, uint_8 sizeX, uint16_t* result)
```

It performs the whole convolution process described in the above example of use, so it writes to the configuration register, writes to the input memory, uses the start, reads the output and clears the interrupt, the convolution is stored in the result pointer.

Parameters:

- X: The input signal generated by the user.
- sizeX: Size of the input signal.
- result: Result of the convolution process.

Returns:

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

7.2.10. id1000500b_setSize

```
int32_t id1000500b_setSize(uint_32 size)
```

This function writes to the configuration register the data passed as a parameter, thus setting the size of the input signal.

Parameters:

- size: Size of the input signal.

Returns:

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