**ID1000500B**

**CONVOLUTION IP-CORE USER MANUAL**

# DESCRIPTION

The Dummy IP-core is the most basic processing system and can be used as a reference example for testing and demonstration purposes.

After receiving a start command, this IP-core only copies data from its input memory to its output memory. In addition, the processing delay can be configurated via software.

## CONFIGURABLE FEATURES

Aquí cambiar por el size, el tamaño de la secuencia

|  |  |
| --- | --- |
| **Software configurations** | **Description** |
| Processing delay | Delay in milliseconds. |

## TYPICAL APPLICATION

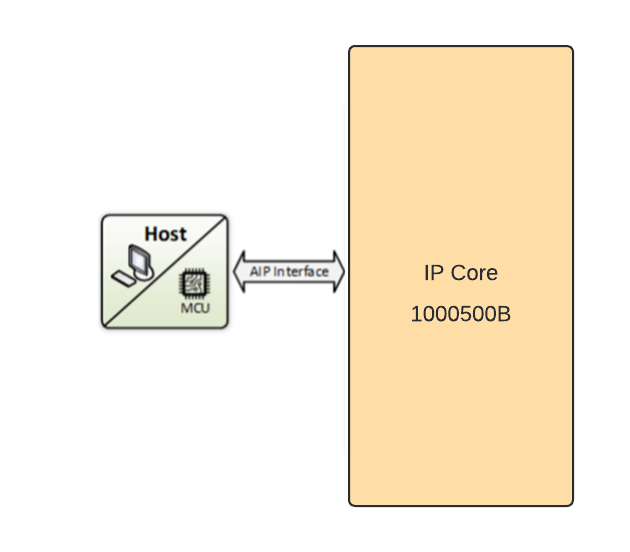


Figure 1.1 IP Core connected to a host

# CONTENTS

[1. DESCRIPTION 1](#_Toc167645310)

[1.1. CONFIGURABLE FEATURES 1](#_Toc167645311)

[1.2. TYPICAL APPLICATION 1](#_Toc167645312)

[2. CONTENTS 2](#_Toc167645313)

[2.1. List of figures 3](#_Toc167645314)

[2.2. List of tables 3](#_Toc167645315)

[3. INPUT/OUTPUT SIGNAL DESCRIPTION 4](#_Toc167645316)

[4. THEORY OF OPERATION 5](#_Toc167645317)

[5. AIP interface registers and memories description 6](#_Toc167645318)

[5.1. Status register 6](#_Toc167645319)

[5.2. Configuration delay register 7](#_Toc167645320)

[5.3. Input data memory 7](#_Toc167645321)

[5.4. Output data memory 7](#_Toc167645322)

[6. PYTHON DRIVER 8](#_Toc167645323)

[6.1. Usage example 8](#_Toc167645324)

[6.2. Methods 9](#_Toc167645325)

[6.2.1. Constructor 9](#_Toc167645326)

[6.2.2. writeData 9](#_Toc167645327)

[6.2.3. readData 9](#_Toc167645328)

[6.2.4. startIP 10](#_Toc167645329)

[6.2.5. enableDelay 10](#_Toc167645330)

[6.2.6. disableDelay 10](#_Toc167645331)

[6.2.7. enableINT 10](#_Toc167645332)

[6.2.8. disableINT 10](#_Toc167645333)

[6.2.9. status 11](#_Toc167645334)

[6.2.10. waitINT 11](#_Toc167645335)

[7. C DRIVER 11](#_Toc167645336)

[7.1. Usage example 11](#_Toc167645337)

[7.2. Driver functions 11](#_Toc167645338)

[7.2.1. ID1000500B \_init 11](#_Toc167645339)

[7.2.2. ID1000500B \_writeData 12](#_Toc167645340)

[7.2.3. ID1000500B \_readData 12](#_Toc167645341)

[7.2.4. ID1000500B \_startIP 12](#_Toc167645342)

[7.2.5. ID1000500B \_enableDelay 13](#_Toc167645343)

[7.2.6. ID1000500B \_disableDelay 13](#_Toc167645344)

[7.2.7. ID1000500B \_enableINT 13](#_Toc167645345)

[7.2.8. ID1000500B \_disableINT 14](#_Toc167645346)

[7.2.9. ID1000500B \_status 14](#_Toc167645347)

[7.2.10. ID1000500B\_waitINT 14](#_Toc167645348)

## List of figures

[Figure 1.1 IP Dummy connected to a host 1](#_Toc140501005)

[Figure 5.1 Basic schematic of the IP Dummy block with the ipm block. 5](#_Toc140501006)

[Figure 5.2 IP Accelerator initialization and selection of the configs file. 6](#_Toc140501007)

[Figure 5.3 Expected result after processing data with the Ip Dummy. 6](file:///D:\Vidkar\Google%20Drive\IPDI\Documentación\Plantillas%20limpias\User%20manual%20-%20ID0001001.docx#_Toc140501008)

[Figure 6.1 IP Dummy status register 7](#_Toc140501009)

[Figure 6.2 Configuration delay register. 8](#_Toc140501010)

## List of tables

[Table 1 IP Dummy input/output signal description 4](#_Toc140488110)

# INPUT/OUTPUT SIGNAL DESCRIPTION

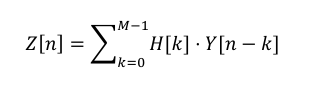
Table 1 IP Core input/output signal description

|  |  |  |  |
| --- | --- | --- | --- |
| 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** | | | |
|  |  |  |  |
|  |  |  |  |

# THEORY OF OPERATION

Implementation of a convolution coprocessor, where the data from the Y-signal (shifted signal) are from an external 32-address the Y signal (shifted signal) is from an external 32-address memory and the H signal (fixed signal) is from an internal memory in the data from the H signal (fixed signal) is from an internal memory in the convolution.

Manually the convolution resolution uses the following formula:



Where:

Convolution result at position **“n”.**

Fixed signal at position **“k”.**

The signal of the input shifted by **“k”** positions with respect to **“n”.**

Convolution in the coprocessor involves multiplication and addition of products between the values of the fixed signal and the shifted values of the input signal using the above formula. This process is repeated at each **“n”** position of the output signal.

The result of the convolution will have **𝑛 + 𝑚 - 1** product sums. That is to say will be obtained at the output, a vector of this size where:

Number of data in fixed signal **“H”.**

Number of data in the input signal **“Y”.**

The process is responsible for shifting the input signal, using its positions in the vector, and multiplying it with the fixed signal, in the same way using the positions, it only the multiplication of the numbers in position that crosses, as shown in the following image, and performs the in the following image, and the corresponding sum of products is performed.



# AIP interface registers and memories description

## Status register

Config: STATUS

Size: 32 bits

Mode: Read/Write.

This register is divided in 3 sections, see Figure 6.1:

* **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.



Figure 6.1 IP Dummy 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.

## Configuration delay register

Config: CDELAY

Size: 32 bits

Mode: Write

This register is used to configure a delay time in milliseconds before the core starts copying data from the input to the output data memory. See Figure 6.2



Figure 6.2 Configuration delay register.

Bits 31:1 – **DELAY:** DELAY count value. The DELAY value can be any value in the range 0x00000001-0xFFFFFFFF. A value of 0 is possible, meaning that no delay will occur.

Bit 0 – **ENA**: delay enable. When ENA is set to 1, a delay of DELAY milliseconds elapses before the core starts its processing.

0: delay counter disabled.

1: delay counter enabled.

## Input data memory

Config: MDATAIN

Size: Nx32 bits (N=32,64,128,256)

Mode: Write

This memory is used to store data to be processed by the IP Dummy core. The size of this memory is set as a hardware parameter before the synthesis. It has support for storing 32, 64, 128, and 256 32-bit words.

## Output data memory

Config: MDATAOUT

Size: Nx32 bits (N=32,64,128,256)

Mode: Read

This memory is used to store processed data by the IP Dummy. After the IP Dummy completes its processing, the data stored in this memory will be the same as the input data memory. The size of this memory is set as a hardware parameter before the synthesis. It has support for storing 32, 64, 128, and 256 32-bit words.

# PYTHON DRIVER

The file *id00001001.py* contains the dummy class definition. This class is used to control the IP Dummy core for python applications.

## Usage example

In the following code a basic test of the IP Dummy core is presented. First, it is required to create an instance of the dummy object class. The constructor of this class requires the network address and port where the IP Dummy is connected, the communication port, and the path where the configs csv file is located. Thus, the communication with the IP Dummy will be ready. In this code, the input memory is written with random data by using the writeData method. Then, the enableDelay method is used to set a delay of 2000 milliseconds, and then the startIP method is used to start core processing. 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  **from** id00001001 **import** dummy  **from** ipdi.ip.pyaip **import** pyaip**,** pyaip\_init**,** Callback  logging**.**basicConfig**(**level**=**logging**.**INFO**)**  connector **=** 'COM7'  csv\_file **=** 'E:/id00001001.csv'  addr **=** 1  port **=** 0  aip\_mem\_size **=** 32  **try:**  dmy **=** dummy**(**connector**,** addr**,** port**,** csv\_file**)**  logging**.**info**(**"Test Dummy: Driver created"**)**  **except:**  logging**.**error**(**"Test Dummy: Driver not created"**)**  sys**.exit()**  random**.**seed**(**1**)**  dmy**.**disableINT**()**  WR **=** **[**random**.**randrange**(**2**\*\***32**)** **for** i **in** **range(**0**,** aip\_mem\_size**)]**  dmy**.**writeData**(**WR**)**  logging**.**info**(**f"Data generated with {**len(**WR**)**:d}"**)**  logging**.**info**(**f'TX Data {**[**f"{x:08X}" **for** x **in** WR**]**}'**)**  dmy**.**enableDelay**(**2000**)**  dmy**.**startIP**()**  dmy**.**waitInt**()**  RD **=** **[]**  RD = dmy**.**readData**(**aip\_mem\_size**)**  logging**.**info**(**f'RX Data {**[**f"{x:08X}" **for** x **in** RD**]**}'**)**  **for** x**,**y **in** **zip(**WR**,** RD**):**  logging**.**info**(**f"TX: {x:08x} | RX: {y:08x} | {'TRUE' **if** x**==**y **else** 'FALSE'}"**)**  dmy**.**disableDelay**()**  dmy**.**enableINT**()**  dmy**.**status**()**  dmy**.**disableINT**()**  dmy**.**status**()**  dmy**.**finish**()**  logging**.**info**(**"The End"**)** |

## Methods

### 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.

### writeData

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

Write data in the IP Dummy input memory.

**Parameters:**

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

**Returns:**

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

### readData

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

Read data from the IP Dummy output memory.

**Parameters:**

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

**Returns:**

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

### startIP

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

Start processing in IP Dummy.

**Returns:**

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

### enableDelay

**def** enableDelay**(**self**,** msec**):**

Set and enable delay in IP Dummy processing.

**Parameters:**

* msec (int): Number of milliseconds of delay.

**Returns:**

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

### disableDelay

**def** disableDelay**(**self**):**

Disable delay in IP Dummy processing.

**Returns:**

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

### enableINT

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

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

**Returns:**

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

### disableINT

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

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

**Returns:**

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

### status

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

Show IP Dummy status.

**Returns:**

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

### waitINT

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

Wait for the completion of the process.

**Returns:**

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

# C DRIVER

In order to use the C driver, it is required to use the files: *ID1000500B.h, ID1000500B.c* that contain the driver functions definition and implementation. The functions defined in this library are used to control the IP Dummy core for C applications.

## Usage example

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

|  |
| --- |
| CODE SAMPLE: |

## Driver functions

### 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 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.

### 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 Dummy 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.

### 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 Dummy 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.

### ID1000500B \_startIP

int32\_t ID1000500B \_startIP**(**uint\_8 nic\_addr**,** uint\_8 port**)**

Start processing in IP Dummy.

**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.

### ID1000500B \_enableDelay

int32\_t ID1000500B \_enableDelay**(**uint\_32 msec, int\_8 nic\_addr**,** uint\_8 port**)**

Set and enable delay in IP Dummy processing.

**Parameters:**

* msec: Number of milliseconds of delay.
* 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.

### ID1000500B \_disableDelay

int32\_t ID1000500B \_disableDelay**(**int\_8 nic\_addr**,** uint\_8 port**)**

Disable delay in IP Dummy processing.

**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.

### ID1000500B \_enableINT

int32\_t ID1000500B \_enableINT**(**int\_8 nic\_addr**,** uint\_8 port**)**

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

**Returns:**

* int32\_t Return 0 whether the function has been completed successfully.

### ID1000500B \_disableINT

int32\_t ID1000500B \_disableINT**(**int\_8 nic\_addr**,** uint\_8 port**)**

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

**Returns:**

* int32\_t Return 0 whether the function has been completed successfully.

### ID1000500B \_status

int32\_t ID1000500B \_status**(**int\_8 nic\_addr**,** uint\_8 port**)**

Show IP Dummy status.

**Returns:**

* int32\_t Return 0 whether the function has been completed successfully.

### 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.