**RPW Instrument**

**LPP IP**

**User’s Manual**

|  |  |  |  |
| --- | --- | --- | --- |
| **Prepared by:** | **Function:** | **Signature:** | **Date** |
| J.-C. PELLION  M. MORLOT | Engineer |  |  |
| **Approved by:** | **Function:** | **Signature:** | **Date** |
|  |  |  |  |
| **For application:** | **Function:** | **Signature:** | **Date** |
|  |  |  |  |

**Change Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ver.** | **Date** | **Authors** | **Modifications** |
| 2.1 | 04 SEP 2013 | JCPE | First Released |

[1 Introduction 4](#_Toc372546781)

[1.1 Reference documents 4](#_Toc372546782)

[2 Basic Module 6](#_Toc372546783)

[2.1 lpp\_FIFO 6](#_Toc372546784)

[2.2 lppFIFOxN 8](#_Toc372546785)

[3 Subsystem/Complex Module 10](#_Toc372546786)

[3.1 MatriceSpectrale 10](#_Toc372546787)

[3.2 HeaderBuilder 13](#_Toc372546788)

[3.3 DEMUX 15](#_Toc372546789)

[3.4 IIR CEL Filter 17](#_Toc372546790)

[3.5 DacDriver 21](#_Toc372546791)

[4 SoC Module 23](#_Toc372546792)

[4.1 apb\_lfr\_time\_management 23](#_Toc372546793)

[4.2 apb\_lfr\_time\_management\_v2 25](#_Toc372546794)

[4.3 APB\_DAC 26](#_Toc372546795)

[4.4 APB\_FIFO 28](#_Toc372546796)

[4.5 Spectral Matrix DMA 34](#_Toc372546797)

# Introduction

This document describes specific IP cores provided with the LPPLIB IP library. When applicable, the cores use the GRLIP plug & play configuration method as described in the “GRLIB User’s Manual” [RD1].

## Reference documents

|  |  |  |
| --- | --- | --- |
| **#** | **Title** | **Version** |
| **RD1** | GRLIB User’s Manual (grlib.pdf) |  |
| **RD2** | GRLIB IP Library User’s Manual (grip.pdf) |  |

# Basic Module

## lpp\_FIFO

### Overview

Here is the lpp fifo IP structure, based on a classical fifo one.



### Operation

Some upgrades are done, in regard to a classical fifo (like the actel one):

Especially a Reuse function, which, via an input bit (ReUse), can lock the fifo in a Full state. So all the same data are available in output, and can be read again and again. The fifo never came to the Empty state, and the writing process is not allowed anymore.

Set to ‘1’ the ReUse signal to use this function.

A generic signal allows or not the use of this function (Enable\_ReUse).

The Mem\_use generic signal controls the memory type used with the fifo:

* Mem\_use = use\_RAM : an external RAM is used.
* Mem\_use = use\_CEL : a VHDL IP simulates the RAM (useful in simulation debug mode)

### Registers

N/A

### Vendor and device identifiers

N/A

### Configuration options

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| tech | target technology |  | apa3 |
| Mem\_use | memory type used |  | use\_RAM |
| Enable\_ReUse | allows the ReUse function |  | ‘0’ |
| DataSz | data size | 1 to 32 | 8 |
| AddrSz | address size | 2 to 12 | 8 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| rstn |  | INPUT | Master reset | low |
| ReUse |  | INPUT | Ask for the reuse function | high |
| rclk |  | INPUT | Read clock |  |
| ren |  | INPUT | Read instruction | low |
| rdata | DataSz-1 downto 0 | OUTPUT | Read data register |  |
| empty |  | OUTPUT | Empty flag | high |
| raddr | AddrSz-1 downto 0 | OUTPUT | Read address register |  |
| wclk |  | INPUT | Write clock |  |
| wen |  | INPUT | Write instruction | low |
| wdata | DataSz-1 downto 0 | INPUT | Write data register |  |
| full |  | OUTPUT | Full flag | high |
| waddr | AddrSz-1 downto 0 | OUTPUT | Write address register |  |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |
| --- | --- |
| **Library** | **Package** |
| ieee | std\_logic\_1164 |
|  | numeric\_std |
| techmap | gencomp |
| lpp | lpp\_memory |
|  | iir\_filter |

### Instantiation

Here is an example of how the fifo can be instantiated.

MEM0 : lpp\_fifo

generic map ( tech => apa3,

Mem\_use => use\_RAM,

Enable\_ReUse => ‘1’,

Data\_sz => 16,

Addr\_sz => 8)

port map (rstn, ReUse, rclk, Ren, RDATA, Empty, RADDR, wclk, Wen, WDATA, Full, WADDR);

## lppFIFOxN

### Overview

Here is the lppFIFOxN IP structure. It’s an instantiation of multiple lpp\_fifo IP.



### Operation

The lppFIFOxN can instantiate more than one fifo in the same IP, a VHDL generic signal configure this option (FifoCnt). That means, for two fifo (FifoCnt=2), the Full flag signal became a two bits vector, one for the first fifo (Full(0)) and the other one for the second (Full(1)). In the same way the Rdata x bits vector became a 2x bits vector. Etc…

### Registers

N/A

### Vendor and device identifiers

N/A

### Configuration options

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| tech | target technology |  | apa3 |
| Mem\_use | memory type used |  | use\_RAM |
| Data\_sz | data size | 1 to 32 | 8 |
| Addr\_sz | address size | 2 to 12 | 8 |
| Enable\_ReUse | allows the ReUse function |  | ‘0’ |
| FifoCnt | number of fifo instantiate in the IP |  | 1 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| rstn |  | INPUT | Master reset | low |
| wclk |  | INPUT | Write clock |  |
| rclk |  | INPUT | Read clock |  |
| ReUse |  | INPUT | Ask for the reuse function | high |
| wen |  | INPUT | Write instruction | low |
| ren |  | INPUT | Read instruction | low |
| wdata | (FifoCnt \*Data\_sz)-1 downto 0 | INPUT | Write data register |  |
| rdata | (FifoCnt \*Data\_sz)-1 downto 0 | OUTPUT | Read data register |  |
| full |  | OUTPUT | Full flag | high |
| empty |  | OUTPUT | Empty flag | high |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |
| --- | --- |
| **Library** | **Package** |
| ieee | std\_logic\_1164 |
|  | numeric\_std |
| techmap | gencomp |
| lpp | lpp\_memory |
|  | iir\_filter |

### Instantiation

This example shows how the core can be instantiated.

MEM1 : lppFIFOxN

generic map ( tech => apa3,

Mem\_use => use\_RAM,

Data\_sz => 16,

Addr\_sz => 8,

FifoCnt => 2,

Enable\_ReUse => ‘0’)

port map ( rstn, clk, clk, (others => '0'), Write, Read, DataIn, DataOut, Full, Empty);

# Subsystem/Complex Module

## MatriceSpectrale

### Overview

Here is a synopsis of the MatriceSpectrale IP, with all the different IP modules which build it.



### Operation

Before this Spectral Matrix computation IP a 5 fifo block *“lppFIFOxN”* is instantiate, each one is field by FFT data determine from input waves shape (B1,B2,B3,E1,E2).

These data are drive in a predetermine order to the main IP “*SpectralMatrix*”, wich build the Matrix via an Arithmetic and logic unit (ALU), “*TopSpecMatrix*” IP assume this function.

The “*ReUse\_CTRLR*” IP allowed using data as many time as necessarily via the ReUse signal of lppFifo IP.

After this Spectral Matrix computation IP a 2 fifo block “*lppFIFOxN*” is instantiate, field by spectral matrix results, “*Dispatch*” IP drive these data to the 2 fifos by creating a “pong effect”.

### Registers

N/A

### Vendor and device identifiers

N/A

### Configuration options

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| Input\_SZ | input data size |  | 16 |
| Result\_SZ | output data size |  | 32 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clkm |  | INPUT | Master clock |  |
| rstn |  | INPUT | Master reset | low |
| FifoIN\_Full | 4 downto 0 | INPUT | Full Flag | high |
| SetReUse | 4 downto 0 | INPUT | Set the reuse function (from FFT IP) | high |
| Valid |  | INPUT | Flag, Matrix component received | high |
| Data\_IN | (5\*Input\_SZ)-1 downto 0 | INPUT | Input Data register |  |
| ACK |  | INPUT | Acknowledge flag for the error flag | low |
| SM\_Write |  | OUTPUT | Data write instruction | low |
| FlagError |  | OUTPUT | Error Flag in writing operation | high |
| Statu | 3 downto 0 | OUTPUT | Matrix parameter compute |  |
| Write | 1 downto 0 | OUTPUT | Write instruction | low |
| Read | 4 downto 0 | OUTPUT | Read instruction | low |
| ReUse | 4 downto 0 | OUTPUT | Ask for the reuse function | high |
| Data\_OUT | (2\*Result\_SZ)-1 downto 0 | OUTPUT | Output Data register |  |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |
| --- | --- |
| **Library** | **Package** |
| ieee | std\_logic\_1164 |
|  | numeric\_std |
| lpp | lpp\_matrix |

### Instantiation

This example shows how the core can be instantiated.

MEM\_IN : lppFIFOxN

generic map ( tech => apa3,

Mem\_use => use\_RAM,

Data\_sz => 16,

Addr\_sz => 8,

FifoCnt => 5,

Enable\_ReUse => ‘1’)

port map ( rstn, clkm, clkm, ReUse, Write, Ren, DATA, DataIn, Full, open);

SM0 : MatriceSpectrale

generic map ( Input\_SZ => 16,

Result\_SZ => 32)

Port map ( clkm, rstn, Full, FFT\_ReUse, Valid, DataIn, ack, SM\_Write, FlagError, MatrixParam, Wen, Ren, ReUse, DataOut);

MEM\_OUT : lppFIFOxN

generic map ( tech => apa3,

Mem\_use => use\_RAM,

Data\_sz => 32,

Addr\_sz => 8,

FifoCnt => 2,

Enable\_ReUse => ‘0’)

port map ( rstn, clkm, clkm, (others => '0'), Wen, Read, DataOut, DataResult, open, open);

## HeaderBuilder

### Overview

The purpose of the HeaderBuilder IP is to build the Header register.

A register which contain some information from others IP of the design, regularly updated, it allows users to check where the computation is.

Here is the data layout in the Header register:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 | 6 | 5 | 2 | 1 | 0 |

* 31:6 n/a (todo => timer)
* 5:2 matrix parameter
* 1:0 matrix type

### Operation

Matrix type: issue from the demultiplexer IP *“DEMUX”*, the working frequency of the current spectral matrix computation.

Matrix parameter: issue from the Spectral Matrix computation IP *“MatriceSpectrale”*, identifier of the current spectral matrix parameter computation.

### Registers

N/A

### Vendor and device identifiers

N/A

### Configuration options

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| Data\_sz | data size |  | 32 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clkm |  | INPUT | Master clock |  |
| rstn |  | INPUT | Master reset | low |
| Statu | 3 downto 0 | INPUT | Matrix parameter |  |
| Matrix\_Type | 1 downto 0 | INPUT | Matrix type |  |
| Matrix\_Write |  | INPUT | Data write instruction from MatriceSpectrale IP | low |
| Valid |  | OUTPUT | Flag, Matrix parameter received for MatriceSpectrale IP | high |
| dataIN | (2\*Data\_sz)-1 downto 0 | INPUT | data register from output fifo |  |
| emptyIN | 1 downto 0 | INPUT | empty flag from output fifo | high |
| RenOUT | 1 downto 0 | OUTPUT | data read instruction | low |
| dataOUT | Data\_sz-1 downto 0 | OUTPUT | output data register |  |
| emptyOUT |  | OUTPUT | empty flag | high |
| RenIN |  | INPUT | Read instruction from DMA | low |
| header | 31 downto 0 | OUTPUT | header register |  |
| header\_val |  | OUTPUT | Flag, Matrix parameter received for DMA IP | high |
| header\_ack |  | INPUT | Acknowledge flag for the header\_val flag | high |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |
| --- | --- |
| **Library** | **Package** |
| ieee | std\_logic\_1164 |
|  | numeric\_std |
| lpp | lpp\_Header |

### Instantiation

HEAD0 : HeaderBuilder

generic map ( Data\_sz => 32)

port map ( clkm => clkm,

rstn => rstn,

Statu => Matrix\_Param,

Matrix\_Type => Demux\_Type,

Matrix\_Write => Matrix\_Wen,

Valid => Head\_Valid,

dataIN => Fifo\_Data,

emptyIN => Fifo\_Empty,

RenOUT => Head\_Ren,

dataOUT => Head\_Data,

emptyOUT => Head\_Empty,

header => Head\_Header,

header\_val => Head\_Valid,

header\_ack => Dma\_ACK);

## DEMUX

### Overview

A classic Demultiplexer use to field the FFT with data issue from 3 different fifo block “*lppFIFOxN*”, one for each working frequency.



### Operation

TODO => Vérifier fréquence !!!!!!

### Registers

N/A

### Vendor and device identifiers

N/A

### Configuration options

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| Data\_sz | data size | 1 to 32 | 16 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clk |  | INPUT | Master clock |  |
| rstn |  | INPUT | Master reset | low |
| Read | 4 downto 0 | INPUT | Data read instruction from FFT | low |
| Load |  | INPUT | FFT ask for data loading | high |
| EmptyF0 | 4 downto 0 | INPUT | Empty flag for F0 fifos | high |
| EmptyF1 | 4 downto 0 | INPUT | Empty flag for F1 fifos | high |
| EmptyF2 | 4 downto 0 | INPUT | Empty flag for F2 fifos | high |
| DataF0 | (5\*Data\_sz)-1 downto 0 | INPUT | Input data for F0 fifos |  |
| DataF1 | (5\*Data\_sz)-1 downto 0 | INPUT | Input data for F1 fifos |  |
| DataF2 | (5\*Data\_sz)-1 downto 0 | INPUT | Input data for F2 fifos |  |
| WorkFreq | 1 downto 0 | OUTPUT | Working frequency (f0, f1 or f2) |  |
| Read\_DEMU | 14 downto 0 | OUTPUT | Read instruction for FIFO | low |
| Empty | 4 downto 0 | OUTPUT | Empty flag | high |
| Data | (5\*Data\_sz)-1 downto 0 | OUTPUT | Output Data register |  |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |
| --- | --- |
| **Library** | **Package** |
| ieee | std\_logic\_1164 |
|  | numeric\_std |
| lpp | lpp\_demux |

### Instantiation

DMUX0: DEMUX

generic map (Data\_sz => 16)

port map ( clk => clkm,

rstn => rstn,

Read => FFT\_Read,

Load => FFT\_Load,

EmptyF0 => FifoF0\_Empty,

EmptyF1 => FifoF1\_Empty,

EmptyF2 => FifoF2\_Empty,

DataF0 => FifoF0\_Data,

DataF1 => FifoF1\_Data,

DataF2 => FifoF2\_Data,

WorkFreq => Demux\_Type,

Read\_DEMUX => Demux \_Read,

Empty => Demux\_Empty,

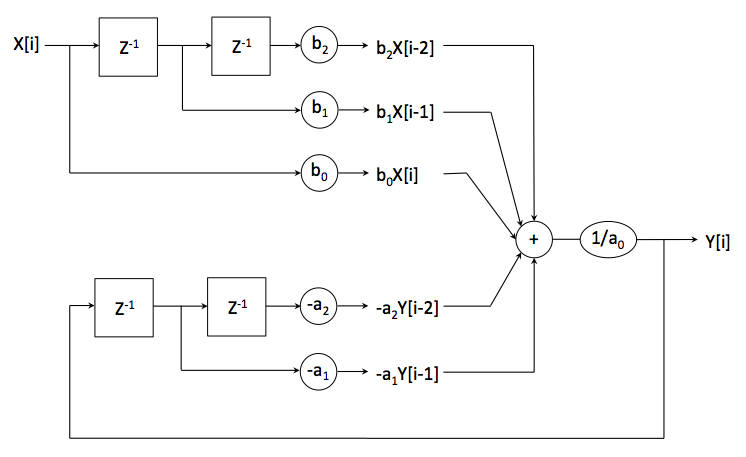
Data => Demux\_Data);

## IIR CEL Filter

### Overview

### Operation

IIR Filter is a sub-system which computes IIR CEL on sample Data. The IIR CEL is a succession of IIR of order 2 (as shown below):



Digital filter of order two

The IIR CEL can compute C channel “in parallel”. The data is set in parallel (Data channel0, channel1, …, channelC) and output in parallel. For each CEL, all channels use the same coefficients (b2, b1, b0, a2, a1). The coefficients are constants define at instantiation.

As shown is the next figure, IIR\_CEL sub system is composed of 2 blocks:

* Dataflow which receives the sample, compute, stock and output the data
* Control which controls the dataflow part

The dataflow:

And its ALU:

The control part and the pipeline:

### Registers

There is no AMBA registers.

### Vendor and device identifiers

There is no vendor identifier or device identifier.

### Configuration options

The following table shows the configuration options of the core (VHDL generics).

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| tech | target technology |  | apa3 |
| Mem\_use | Memory Type Use :   * use\_RAM => hard-macro * use\_CEL => logic ram |  | 2 |
| Sample\_SZ | Sample Size in bit |  | 18 |
| Coef\_size | Coefficient size in bit |  | 9 |
| Coef\_Nb | Coefficient Number |  | 25 |
| Coef\_sel\_SZ | Coefficient selection Size (log2(Coef\_Nb)) |  | 5 |
| Cels\_count | Number of cellule of order 2 |  | 5 |
| ChanelsCount | Number of Chanels |  | 8 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Signal name** | **Field** | **Type** | **Function** | **Active** |
| clk | - | INPUT | Clock | - |
| Rstn | - | INPUT | Reset | low |
| Virg\_pos | - | INPUT | Virgule position for coefficients field | - |
| Coefs | - | INPUT | IIR Coefficient | - |
| Sample\_in\_val | - | INPUT | Sample in valid bit | high |
| Sample\_in | - | INPUT | Sample Vector in data | - |
| Sample\_out\_val | - | OUTPUT | Sample out valid bit | high |
| Sample\_out | - | OUTPUT | Sample Vector out data | - |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |  |  |
| --- | --- | --- | --- |
| **Library** | **Package** | **Imported unit(s)** | **Description** |
|  |  |  |  |

### Component

COMPONENT IIR\_CEL\_CTRLR\_v2

GENERIC (

tech : INTEGER;

Mem\_use : INTEGER;

Sample\_SZ : INTEGER;

Coef\_SZ : INTEGER;

Coef\_Nb : INTEGER;

Coef\_sel\_SZ : INTEGER;

Cels\_count : INTEGER;

ChanelsCount : INTEGER);

PORT (

rstn : IN STD\_LOGIC;

clk : IN STD\_LOGIC;

virg\_pos : IN INTEGER;

coefs : IN STD\_LOGIC\_VECTOR(

(Coef\_SZ\*Coef\_Nb)-1 DOWNTO 0);

sample\_in\_val : IN STD\_LOGIC;

sample\_in : IN samplT(

ChanelsCount-1 DOWNTO 0,

Sample\_SZ-1 DOWNTO 0);

sample\_out\_val : OUT STD\_LOGIC;

sample\_out : OUT samplT(

ChanelsCount-1 DOWNTO 0,

Sample\_SZ-1 DOWNTO 0));

END COMPONENT;

### Instantiation

IIR\_CEL\_CTRLR\_v2\_i: IIR\_CEL\_CTRLR\_v2

GENERIC MAP (

tech => tech,

Mem\_use => Mem\_use,

Sample\_SZ => Sample\_SZ,

Coef\_SZ => Coef\_SZ,

Coef\_Nb => Coef\_Nb,

Coef\_sel\_SZ => Coef\_sel\_SZ,

Cels\_count => Cels\_count,

ChanelsCount => ChanelsCount)

PORT MAP (

rstn => rstn,

clk => clk,

virg\_pos => virg\_pos,

coefs => coefs,

sample\_in\_val => sample\_in\_val,

sample\_in => sample\_in,

sample\_out\_val => sample\_out\_val,

sample\_out => sample\_out);

## DacDriver

### Overview

Here is a synopsis of the DacDriver IP, with all the different IP modules which build it.



### Operation

This IP is a driver for the DAC 121S101 used to convert the calibration signal.

“*Systeme\_Clock”* and *“Gene\_SYNC”* IP provide respectively the SCLK and SYNC signals used by the converter, and the data register (Data\_C) is serialized by *“serialize”* IP.

### Registers

N/A

### Vendor and device identifiers

N/A

### Configuration options

N/A

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clk |  | INPUT | Master clock |  |
| rst |  | INPUT | Master reset | low |
| enable |  | INPUT | Allow the use of the DAC | high |
| Data\_C | 15 downto 0 | INPUT | Input Data register |  |
| SYNC |  | OUTPUT | Frame synchronization for the data input | high |
| SCLK |  | OUTPUT | Serial Clock |  |
| flag\_sd |  | OUTPUT | Data serialization over flag | high |
| Data |  | OUTPUT | Serialized data |  |

### Library dependencies

N/A

### Instantiation

CONV0 : DacDriver

port map ( clk => clkm,

rst => rstn,

enable => Cal\_EN,

Data\_C => DATA,

SYNC => SYNC,

SCLK => SCLK,

flag\_sd => Sended,

Data => SDIN);

# SoC Module

## apb\_lfr\_time\_management

### Overview

### Operation

### Registers

The core is programmed through registers mapped into APB address space.

|  |  |
| --- | --- |
| **APB address offset** | **Register** |
| 0x00 | ctrl |
| 0x04 | coarse\_time\_load |
| 0x08 | coarse\_time |
| 0x0C | fine\_time |
| 0x10 | next\_commutation |

Table ctrl register

|  |  |  |  |
| --- | --- | --- | --- |
| 31 |  | 1 | 0 |

* 31:1 Reserved for further usages
* 0 Force tick bit. If set to ‘1’, load the coarse\_time\_load value in the coarse\_time register and resets the fine time counter. Automatically reset to ‘0’.

Table 2 coarse\_time\_load register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 coarse time value to load at the next tick out emitted by the grspw module

Table 3 coarse\_time register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 current valid coarse time value

Table 4 fine\_time register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 current fine time value

Table 6 next\_commutation

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 next commutation time

### Vendor and device identifiers

The core has vendor identifier 0x00 (TBD LPP) and device identifier 0x00 (TBD). For description of vendor and device identifiers see GRLIB IP Library User’s Manual [RD2].

### Configuration options

The following table shows the configuration options of the core (VHDL generics).

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| pindex | APB index |  | 0 |
| paddr | ADDR field of the APB BAR |  | 0 |
| pmask | MASK field of the APB BAR |  | 0xFFF |
| masterclk | master clock in Hz |  | 50000000 |
| finetimeclk | divided clock used for the fine time counter |  | 65536 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clk | - | INPUT | general clock | - |
| resetn | - | INPUT | master reset | LOW |
| grspw\_tick | - | INPUT | synchronization signal | HIGH |
| apbi |  | INPUT | APB input signals | - |
| apbo |  | OUTPUT | APB output signals | - |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |  |  |
| --- | --- | --- | --- |
| **Library** | **Package** | **Imported unit(s)** | **Description** |
|  |  |  |  |

### Instantiation

## apb\_lfr\_time\_management\_v2

### Overview

### Operation

### Registers

The core is programmed through registers mapped into APB address space.

|  |  |
| --- | --- |
| **APB address offset** | **Register** |
| 0x00 | ctrl |
| 0x04 | coarse\_time\_load |
| 0x08 | coarse\_time |
| 0x0C | fine\_time |

Table ctrl register

|  |  |  |  |
| --- | --- | --- | --- |
| 31 |  | 1 | 0 |

* 31:1 Reserved for further usages
* 0 Force tick bit. If set to ‘1’, load the coarse\_time\_load value in the coarse\_time register and resets the fine time counter. Automatically reset to ‘0’.

Table 7 coarse\_time\_load register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 coarse time value to load at the next tick out emitted by the grspw module

Table 8 coarse\_time register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 current valid coarse time value

Table 9 fine\_time register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 current fine time value

### Vendor and device identifiers

The core has vendor identifier 0x00 (TBD LPP) and device identifier 0x00 (TBD). For description of vendor and device identifiers see GRLIB IP Library User’s Manual [RD2].

### Configuration options

The following table shows the configuration options of the core (VHDL generics).

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| pindex | APB index |  | 0 |
| paddr | ADDR field of the APB BAR |  | 0 |
| pmask | MASK field of the APB BAR |  | 0xFFF |
| pirq |  |  | 0 |
| masterclk | master clock in Hz |  | 50 000 000 |
| otherclk | other clock in Hz |  | 49 152 000 |
| finetimeclk | divided clock used for the fine time counter in Hz |  | 65 536 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clk25MHz | - | INPUT | general clock | - |
| clk49\_152MHz | - | INPUT | secondary clock | - |
| resetn | - | INPUT | master reset | LOW |
| grspw\_tick | - | INPUT | synchronization signal | HIGH |
| apbi | - | INPUT | APB input signals | - |
| apbo | - | OUTPUT | APB output signals | - |
| coarse\_time | - | OUTPUT | coarse time value (32 bits) | - |
| fine\_time | - | OUTPUT | fine time value (32 bits) | - |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |  |  |
| --- | --- | --- | --- |
| **Library** | **Package** | **Imported unit(s)** | **Description** |
|  |  |  |  |

### Instantiation

## APB\_DAC

### Overview

Here is the APB\_DAC IP structure.



### Operation

The data register is drive through the DacDriver via the apb bus, it is serialized here then send to the DAC with the others inputs synchronization signals.

### Registers

The core is programmed through registers mapped into APB address space.

|  |  |
| --- | --- |
| **APB address offset** | **Register** |
| 0x00 | DAC\_Cfg |
| 0x04 | DAC\_Data |

Table 1 DAC\_Cfg register

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 0 |

* 31:2 n/a
* 1 Sended flag
* 0 Enable flag

Table 2 DAC\_Data register

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 | 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |

* 31:16 n/a
* 15:0 Data

### Vendor and device identifiers

The core has vendor identifier 0x19 (VENDOR\_LPP) and device identifier 0x7 (LPP\_CNA).

For description of vendor and device identifiers see GRLIB IP Library User’s Manual [RD2].

### Configuration options

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| pindex | APB index |  | 0 |
| paddr | APD address |  | 0 |
| pmask | APB Mask address |  | 0xFFF |
| pirq | output AHB interruption number |  | 0 |
| abits | address size for APD address |  | 8 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clk |  | INPUT | Master clock |  |
| rst |  | INPUT | Master reset | low |
| apbi |  | INPUT | APB input interface |  |
| apbo |  | OUTPUT | APB output interface |  |
| Cal\_EN |  | OUTPUT | Allow the use of the DAC | high |
| SYNC |  | OUTPUT | Frame synchronization for the data input | high |
| SCLK |  | OUTPUT | Serial Clock |  |
| DATA |  | OUTPUT | Digital serialized data |  |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |
| --- | --- |
| **Library** | **Package** |
| ieee | std\_logic\_1164 |
|  | numeric\_std |
| grlib | amba |
|  | stdlib |
|  | devices |
| lpp | lpp\_amba |
|  | apb\_devices\_list |
|  | lpp\_cna |

### Instantiation

This example shows how the core can be instantiated.

CAL0 : APB\_DAC

generic map ( pindex => 4,

paddr => 4)

port map ( clk => clkm,

rst => rstn,

apbi => apbi,

apbo => apbo(4),

Cal\_EN => DAC\_Cal\_EN

SYNC => DAC\_SYNC,

SCLK => DAC\_SCLK,

DATA => DAC\_DATA);

## APB\_FIFO

### Overview

Here is the APB\_FIFO IP structure.



### Operation

This IP works like an lppFIFOxN IP but with an APB bus communication level.

The Write and the Read process can work on the APB bus or in hard via another VHDL IP (the usual way, like lppFIFOxN operation). Its VHDL generics signals (R and W) which configure these options, set to ‘1’ to work on the APB bus, else you work like an usual fifo.

### Registers

The core is programmed through registers mapped into APB address space.

|  |  |
| --- | --- |
| **APB address offset** | **Register** |
| 0x00 | FIFO\_ID |
| 0x04 | FIFO\_Ctrl (fifo 1) |
| 0x08 | FIFO\_Data (fifo 1) |
| 0x0C | FIFO\_Ctrl (fifo 2) |
| 0x10 | FIFO\_Data (fifo 2) |
| … | … |
| 0x... | FIFO\_Ctrl (fifo X) |
| 0x... | FIFO\_Data (fifo X) |

Table FIFO\_ID register

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  |  |  | 24 | 23 |  |  |  |  |  |  | 16 | 15 |  |  |  |  |  |  | 8 | 7 | 6 | 5 | 4 | 3 |  |  | 0 |

* 31:24 n/a
* 23:16 address size value (Addr\_sz)
* 15:8 data size value (Data\_sz)
* 7:6 n/a
* 5 R generic value
* 4 W generic value
* 3:0 fifo counter value (FifoCnt)

Table 2 FIFO\_Ctrl register

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  | a | b |  |  |  |  | 20 | 19 |  | 17 | 16 | 15 |  |  |  |  | x | y |  |  |  |  | 4 | 3 | 2 | 1 | 0 |

* 31:a n/a
* b:20 Write address
* 19:17 n/a
* 16 Full flag
* 15:x n/a
* y:4 Read address
* 3:2 n/a
* 1 ReUse flag
* 0 Empty flag

Table 3 FIFO\_Data register

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | y |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |

* 31:x n/a
* y:0 data

### Vendor and device identifiers

The core has vendor identifier 0x19 (VENDOR\_LPP) and device identifier 0x11 (LPP\_FIFO).

For description of vendor and device identifiers see GRLIB IP Library User’s Manual [RD2].

### Configuration options

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| tech | target technology |  | apa3 |
| pindex | APB index |  | 0 |
| paddr | APD address |  | 0 |
| pmask | APB Mask address |  | 0xFFF |
| pirq | output AHB interruption number |  | 0 |
| abits | address size for APD address |  | 8 |
| FifoCnt | number of fifo instantiate in the IP |  | 2 |
| Data\_sz | data size |  | 16 |
| Addr\_sz | address size |  | 9 |
| Enable\_ReUse | enable the reuse function |  | ‘0’ |
| Mem\_use | memory type used |  | use\_RAM |
| R | read setup (use apb bus or not) |  | 1 |
| W | write setup (use apb bus or not) |  | 1 |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal name | Field | Type | Function | Active |
| clk |  | INPUT | Master clock |  |
| rst |  | INPUT | Master reset | low |
| rclk |  | INPUT | Read clock |  |
| wclk |  | INPUT | Write clock |  |
| ReUse | FifoCnt-1 downto 0 | INPUT | Ask for the reuse function | high |
| REN | FifoCnt-1 downto 0 | INPUT | Read instruction | low |
| WEN | FifoCnt-1 downto 0 | INPUT | Write instruction | low |
| Empty | FifoCnt-1 downto 0 | OUTPUT | Empty flag | high |
| Full | FifoCnt-1 downto 0 | OUTPUT | Full Flag | high |
| RDATA | (FifoCnt\*Data\_sz)-1 downto 0 | OUTPUT | Read Data register |  |
| WDATA | (FifoCnt\*Data\_sz)-1 downto 0 | INPUT | Write Data register |  |
| WADDR | (FifoCnt\*Addr\_sz)-1 downto 0 | OUTPUT | Write address register |  |
| RADDR | (FifoCnt\*Addr\_sz)-1 downto 0 | OUTPUT | Read address register |  |
| apbi |  | INPUT | APB input interface |  |
| apbo |  | OUTPUT | APB output interface |  |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |
| --- | --- |
| **Library** | **Package** |
| ieee | std\_logic\_1164 |
|  | numeric\_std |
| techmap | gencomp |
| grlib | amba |
|  | stdlib |
|  | devices |
| lpp | lpp\_amba |
|  | apb\_devices\_list |
|  | lpp\_memory |
|  | iir\_filter |

### Instantiation

This example shows how the core can be instantiated.

MemIn : APB\_FIFO

generic map ( pindex => 4,

paddr => 4,

FifoCnt => 1,

Data\_sz => 16,

Addr\_sz => 8,

Enable\_ReUse => '0',

R => 0,

W => 1)

port map ( clk => clk,

rst => rstn,

rclk => clk,

wclk => clk,

ReUse => (others => '0'),

REN => Read,

WEN => (others => '1'),

Empty => Empty,

Full => open,

RDATA => Data,

WDATA => (others => '0'),

WADDR => open,

RADDR => open,

apbi => apbi,

apbo => apbo(4));

Bridge0 : Link

port map (clk, rstn, Empty, Full, Write, Read);

MemOut : APB\_FIFO

generic map ( pindex => 5,

paddr => 5,

FifoCnt => 1,

Data\_sz => 16,

Addr\_sz => 8,

Enable\_ReUse => '0',

R => 1,

W => 0)

port map ( clk => clk,

rst => rstn,

rclk => clk,

wclk => clk,

ReUse => (others => '0'),

REN => (others => '1'),

WEN => Write,

Empty => open,

Full => Full,

RDATA => open,

WDATA => Data,

WADDR => open,

RADDR => open,

apbi => apbi,

apbo => apbo(5));

## Spectral Matrix DMA

### Overview

LPP

Spectral Matrix

LPP\_SP\_DMA

**AHB**

**APB**

DMA2AHB  
(GRlib)

**M**

APBReg

**S**

LPP FIFO

(Lpp IP)

**DATA**

ren

empty

**HEADER**

valid

ack

Latency

Correction

### Operation

LPP\_SP\_DMA is a sub-system. His function is the transfer of ‘matrix’ from a FIFO to a ‘memory’ connected on an AHB bus.

The LPP Spectral Matrix pushes the data of a component Matrix (there is 15 types of component and 4 types of matrix). When all data of a component are into the FIFO, the LPP Spectral Matrix indicates to LPP\_SP\_DMA that the component C of Matrix M is ready (Header interface).

If the bit Matrix M is not set (into APB register status), the FIFOs data are transferred at address “Matrix M address” (APB register) through AHB bus.

LPP\_SP\_DMA checked the length of the current component C, and the sequence of component (component 0 of M, 1 of M, C+1 of M…., 15 of M, 0 of Mi, …). If an error occurs, all data remaining for the current Matrix are trashed and an error flag is set.

LPP\_SP\_DMA implements the DMA2AHB IP. DMA2AHB permits to transfer the data by burst of 64B (16\*32b) and the header by 4B. There is 3 FSM in LPP\_SP\_DMA which connect FIFO to DMA2AHB:

* transfers of 64B of data
* transfers of 4B of Header
* controls and checked

### Registers

The core is programmed through registers mapped into APB address space.

|  |  |
| --- | --- |
| **APB address offset** | **Register** |
| 0x00 | config |
| 0x04 | status |
| 0x08 | matrix f0 address 0 |
| 0x0C | matrix f0 address 1 |
| 0x10 | matrix f1 address |
| 0x14 | matrix f2 address |

Table config register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 | 2 |  | 1 | 0 |

* 31:2 Reserved for further usages
* 1 Active interruption on “Error”
* 0 Active interruption on “new Ready Matrix”

Table status register

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

* 31:6 Reserved for further usages
* 5 Error – anticipating empty FIFO
* 4 Error – bad component header
* 3 Matrix f2 is ready
* 2 Matrix f1 is ready
* 1 Matrix0 f0 is ready
* 0 Matrix1 f0 is ready

Table 13 matrix f0 address 0 register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 matrix f0 address 0 register

Table 14 matrix f0 address 1 register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 matrix f0 address 1 register

Table 15 matrix f1 address register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 matrix f0 address0 register

Table 16 matrix f2 address register

|  |  |
| --- | --- |
| 31 | 0 |

* 31:0 matrix f0 address0 register

### Vendor and device identifiers

The core has vendor identifier 0x00 (TBD LPP) and device identifier 0x00 (TBD). For description of vendor and device identifiers see GRLIB IP Library User’s Manual [RD2].

### Configuration options

The following table shows the configuration options of the core (VHDL generics).

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic** | **Function** | **Allowed range** | **Default** |
| tech | target technology |  | apa3 |
| hindex | AHB index |  | 2 |
| pirq | output AHB interruption number for “Matrix Ready” |  | 0 |
| msize | Matrix size in 4Bytes |  | 0xF00 |
| pindex | APB index |  | 0 |
| paddr | APD address |  | 0 |
| pmask | APB Mask address |  | 0xFFF |

### Signal descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Signal name** | **Field** | **Type** | **Function** | **Active** |
| HCLK | - | INPUT | Amba clock | - |
| HRESETn | - | INPUT | Amba reset | low |
| AHB\_Master\_In | hgrant[0:15] | INPUT | AHB Master Input Interface | high |
| hready | high |
| hresp[1:0] | - |
| hrdata[31:0] | - |
| hcache | high |
| hirq[31:0] | high |
| testen | high |
| testrst | low |
| scanen | hig |
| testoen | low |
| AHB\_Master\_Out | hbusreq | INPUT | AHB Master Output Interface | hig |
| hlock | hig |
| htrans[1:0] | - |
| haddr[31:0] | - |
| hwrite | hig |
| hsize[2:0] | - |
| hburst[2:0] | - |
| hprot[3:0] | - |
| hwdata[31:0] | - |
| hirq[31:0] | high |
| hconfig[7:0] | - |
| hindex[3:0] | - |
| apbi | psel[0:15] | INPUT | APB Slave Input Interface | hig |
| penable | hig |
| paddr[31:0] | - |
| pwdata[31:0] | - |
| pwrite | high |
| pirq[31:0] | high |
| testen | high |
| testrst | low |
| scanen | high |
| testoen | low |
| apbo | prdata[31:0] | INPUT | APB Slave Output Interface | - |
| pirq[31:0] | high |
| pconfig[7:0] | - |
| pindex[3:0] | - |
| fifo\_data | - | INPUT | Fifo data | - |
| fifo\_empty | - | INPUT | Fifo occupancy ( number of data available) | - |
| Fifo\_ren | - | OUTPUT | Fifo read enable | Low |
| Header | matrix\_type | INPUT | indicates the current matrix type in the FIFO 00 - Matrix at f0 frequency  01 - Matrix at f1 frequency  10 - Matrix at f2 frequency | - |
| component\_type | ndicates the current component type in the FIFO 0000 - S11 0001 - S12 0010 - S13 0011 - S14 0100 - S15 0101 - S22 0110 - S23 0111 - S24 1000 - S25 1001 - S33 1010 - S34 1011 - S35 1100 - S44 1101 - S45 1110 - S55 1111 - Unused | - |
| Coarse\_time | Coarse time of the current matrix |  |
| Fine\_time | Fine time of the current matrix |  |
| Header\_Valid | - | INPUT | Valid bit. It is set when the Header is updated and unset when the header\_ack is asserted | high |
| Header\_ack | - | OUPUT | Acknowledge | high |

### Library dependencies

The following table shows libraries used when instantiating the core (VHDL libraries).

|  |  |  |  |
| --- | --- | --- | --- |
| **Library** | **Package** | **Imported unit(s)** | **Description** |
|  |  |  |  |

### Component

COMPONENT lpp\_dma

GENERIC (

tech : INTEGER;

hindex : INTEGER;

pindex : INTEGER;

paddr : INTEGER;

pmask : INTEGER;

pirq : INTEGER);

PORT (

HCLK : IN STD\_ULOGIC;

HRESETn : IN STD\_ULOGIC;

apbi : IN apb\_slv\_in\_type;

apbo : OUT apb\_slv\_out\_type;

AHB\_Master\_In : IN AHB\_Mst\_In\_Type;

AHB\_Master\_Out : OUT AHB\_Mst\_Out\_Type;

fifo\_data : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

fifo\_empty : IN STD\_LOGIC;

fifo\_ren : OUT STD\_LOGIC;

header : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

header\_val : IN STD\_LOGIC;

header\_ack : OUT STD\_LOGIC);

END COMPONENT;

### Instantiation

lpp\_dma\_1: lpp\_dma

GENERIC MAP (

tech => tech,

hindex => hindex,

pindex => pindex,

paddr => paddr,

pmask => pmask,

pirq => pirq)

PORT MAP (

HCLK => HCLK,

HRESETn => HRESETn,

apbi => apbi,

apbo => apbo,

AHB\_Master\_In => AHB\_Master\_In,

AHB\_Master\_Out => AHB\_Master\_Out,

fifo\_data => fifo\_data,

fifo\_empty => fifo\_empty,

fifo\_ren => fifo\_ren,

header => header,

header\_val => header\_val,

header\_ack => header\_ack);