# LPP IP Core User’s Manual

## Table of content

# 1 Introduction

This document describes specific IP cores provided wit h the LPPLIB IP library. When applicable, the cores use the GRLIP plug & play configuration method as described in the "GRLIB User 's Manual" [RD1].

## 1.1 Reference documents

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

# 2 apb_lfr_time_management

## 2.1 Overview

## 2.2 Operation

## 2.3 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 1 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 3 coarse_time_load register**

| 31 … 0 |
|---|

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

**Table 4 coarse_time register**

| 31 … 0 |
|---|

- 31:0 current valid coarse time value

**Table 5 fine_time register**

| 31 … 0 |
|---|

- 31:0 current fine time value

## 2.4 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].

## 2.5 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 |

## 2.6 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 | - |

## 2.7 Library dependencies

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

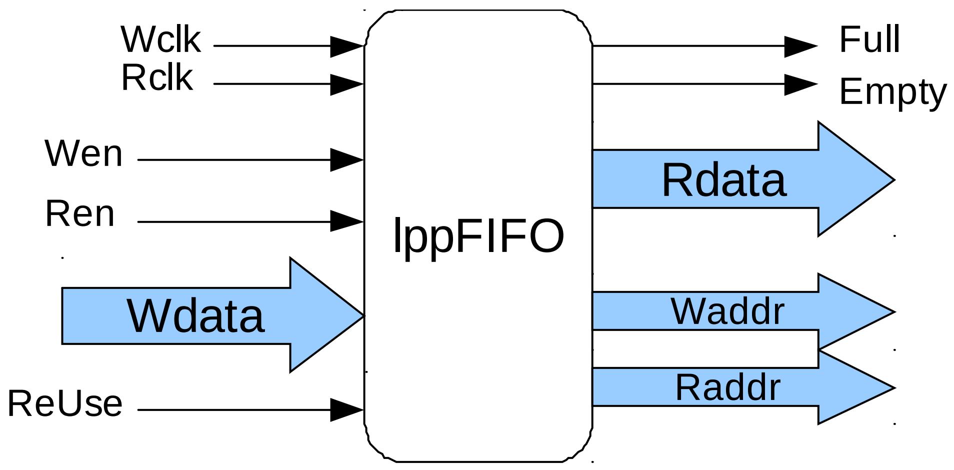
| **Library** | **Package** | **Imported unit(s)** | **Description** |
|---|---|---|---|
| | | | |

## 2.8 Instantiation

# 3 Lpp FIFO

## 3.1 Overview

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



## 3.2 Operation

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

First 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' to use this function.

The lpp fifo can instantiate more than one fifo in the same IP, a VHDL generic 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 and the other one for the second. In the same way the Rdata x bits vector became a 2x bits vector. Etc…

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

## 3.3 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 | Data (fifo 1) |
| 0x0C | FIFO_Ctrl (fifo 2) |
| 0x10 | Data (fifo 2) |
| … | … |
| 0x... | FIFO_Ctrl (fifo X) |

| | |
|---|---|
| 0x... | Data (fifo X) |

**Table 1 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 |

## 3.4 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].

## 3.5 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 | | 1 |
| Data_sz | data size | | 16 |
| Addr_sz | address size | | 8 |
| Enable_ReUse | enable the reuse function | | 0 |
| R | read setup (use apb bus or not) | | 0 |
| W | write setup (use apb bus or not) | | 0 |

## 3.6 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 | |

## 3.7 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 |

## 3.8 Instantiation

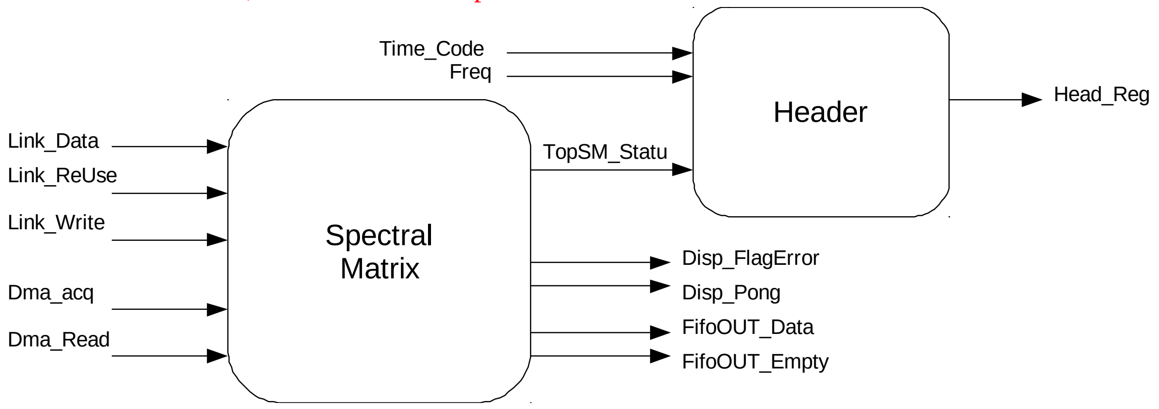
This example shows how the core can be instantiated.

```
MEM0 : APB_FIFO
        generic map (pindex => 15, paddr => 15, FifoCnt => 1, Data_sz => 32, Addr_sz => 8,
        Enable_ReUse => '0', R => 1, W => 0)
         port map (clkm, rstn, clkm, clkm, ReUse, (others => '1'), Write, Empty, Full, open, Results,
        Waddr, Raddr, apbi, apbo(15));
```

# 4 Spectral Matrix Computation

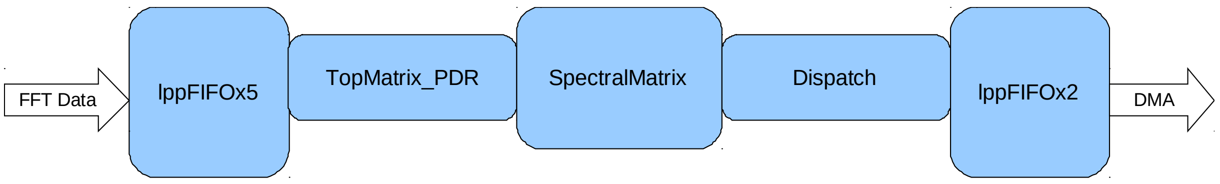
## 4.1 Overview

Add a few comments, see RD2 for example.



## 4.2 Operation

Here are all the different modules which build the Spectral Matrix IP.



| | |
|---|---|
| *lppFIFOx5* | A 5 Fifo block, each one is field by FFT data determine from input waves shape (B1,B2,B3,E1,E2) |
| *TopMatrix_PDR* | Driver which provide the good data for the Matrix Calculator block |
| *SpectralMatrix* | Spectral Matrix Calculator via Arithmetic and logic unit (ALU) |
| *Dispatch* | Driver for the « pong effect » on output fifos |
| *LppFIFOx2* | A 2 Fifo block, each one is field by spectral matrix results |

## 4.3 Registers

N/A

## 4.4 Vendor and device identifiers

N/A

## 4.5 Configuration options

N/A

No configuration option? No VHDL generic to set?

## 4.6 Signal descriptions

| Signal name | Field | Type | Function | Active |
|---|---|---|---|---|
| clkm | | INPUT | Master clock | |

| | | | | |
|---|---|---|---|---|
| rstn | | INPUT | Master reset | low |
| Link_Data | | INPUT | Input data, 80 bits (5x16) | |
| Link_ReUse | | INPUT | Fifo lock state flag, for write step (5 bits) | high |
| Link_Write | | INPUT | Fifo write statement (5 bits) | low |
| Dma_Read | | INPUT | Fifo read statement (2 bits) | low |
| Dma_acq | | INPUT | Allows "pong" on output fifos | high |
| Disp_Pong | | OUTPUT | Ask for "pong" on output fifos | high |
| Disp_FlagError | | OUTPUT | Writing error, fifo not empty (DMA side) | high |
| FifoOUT_Data | | OUTPUT | Matrix data, 64 bits | |
| TopSM_Statu | | SIGNAL | Curent component to fix, 4 bits | |
| Time_Code | | INPUT | Time Code, 26 bits | |
| Freq | | INPUT | Frequency f0,f1,f2, 2 bits | |
| Head_Reg | coarse_time | OUTPUT | 10 bits | |
| | fine_time | | 16 bits | |
| | component type | | 4 bits | |
| | f0, f1 or f2 | | 2 bits | |

## 4.7 Library dependencies

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

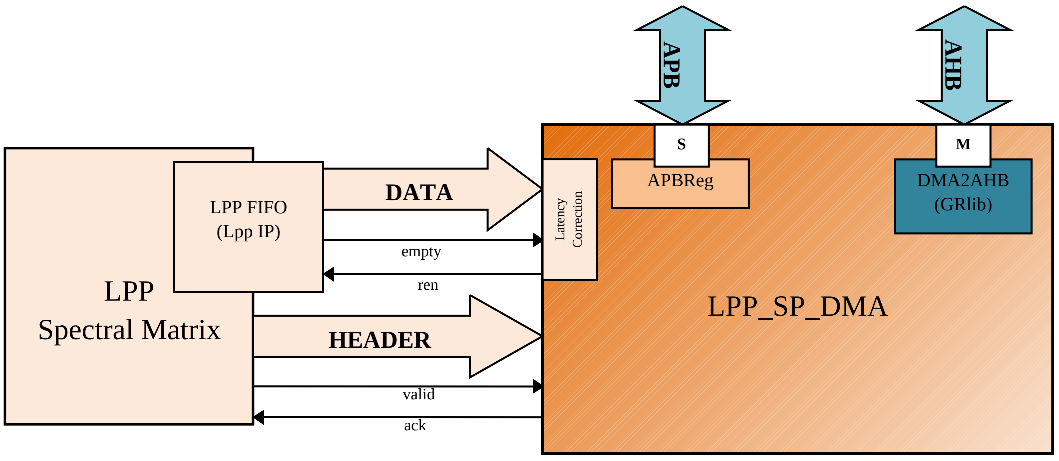
| Library | Package | Imported unit(s) | Description |
|---|---|---|---|
| | | | |

## 4.8 Instantiation

This example shows how the core can be instantiated.

# 5 Spectral Matrix DMA

## 5.1 Overview



## 5.2 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

## 5.3 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 1 config register**

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

| | | |
|---|---|---|
| 31:2 | Reserved for further usages | |
| 1 | Active interruption on "Error" | (TO FIX : Not supported actually) |
| 0 | Active interruption on "new Ready Matrix" | (TO FIX : Not supported actually) |

**Table 2 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 3 matrix f0 address 0 register**

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

- 31:0 matrix f0 address 0 register

**Table 4 matrix f0 address 1 register**

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

- 31:0 matrix f0 address 1 register

**Table 5 matrix f1 address register**

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

- 31:0 matrix f0 address0 register

**Table 6 matrix f2 address register**

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

- 31:0 matrix f0 address0 register

## 5.4 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].

## 5.5 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 |

## 5.6 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<br>00 - Matrix at f0 frequency<br>01 - Matrix at f1 frequency<br>10 - Matrix at f2 frequency | - |
| | | | ndicates the current component type in the FIFO | |

| | | | | |
|---|---|---|---|---|
| | component_type | | 0000 - S11<br>0001 - S12<br>0010 - S13<br>0011 - S14<br>0100 - S15<br>0101 - S22<br>0110 - S23<br>0111 - S24<br>1000 - S25<br>1001 - S33<br>1010 - S34<br>1011 - S35<br>1100 - S44<br>1101 - S45<br>1110 - S55<br>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 |

## 5.7 Library dependencies

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

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

## 5.8 Component

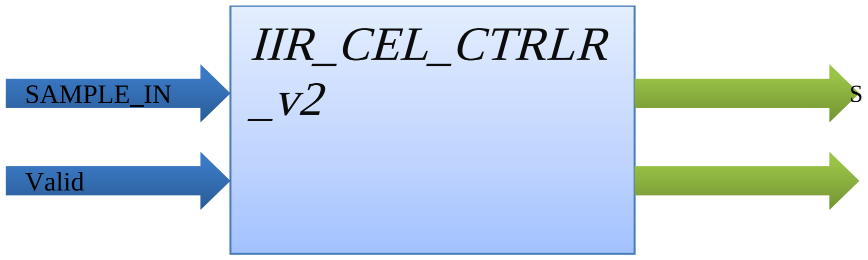
```
COMPONENT lpp_dma
        GENERIC (
                tech     : INTEGER;
                hindex   : INTEGER;
                pindex   : INTEGER;
                paddr    : INTEGER;
                pmask    : INTEGER;
                pirq     : INTEGER);
        PORT (
                HCLK              : 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;
```

## 5.9 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);
```

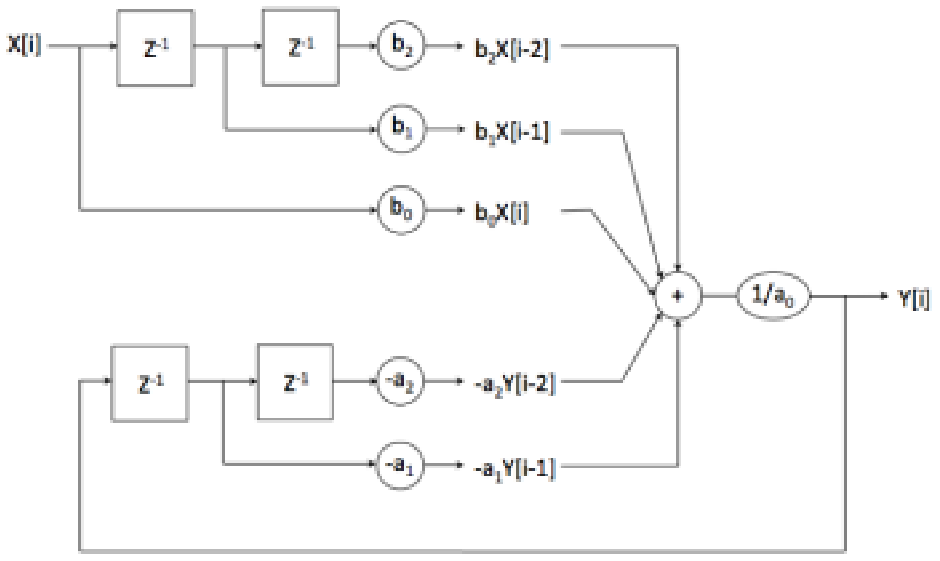
# 6 IIR CEL Filter

## 6.1 Overview



## 6.2 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):

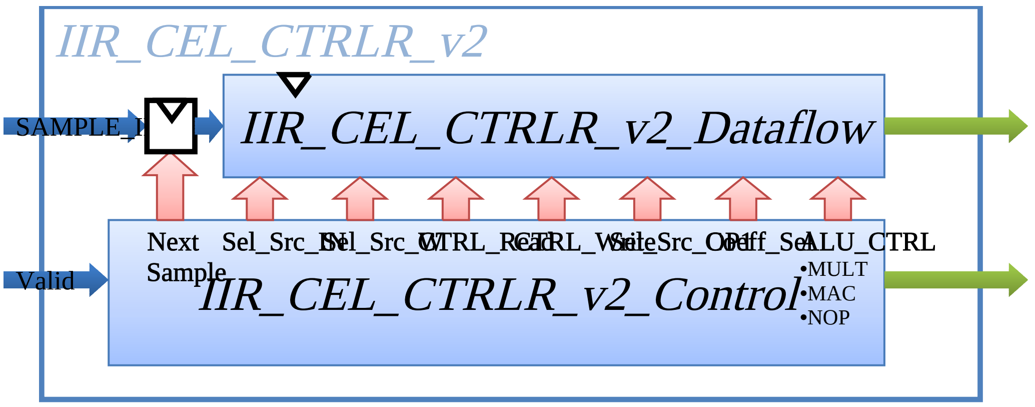


Digital filter of order two

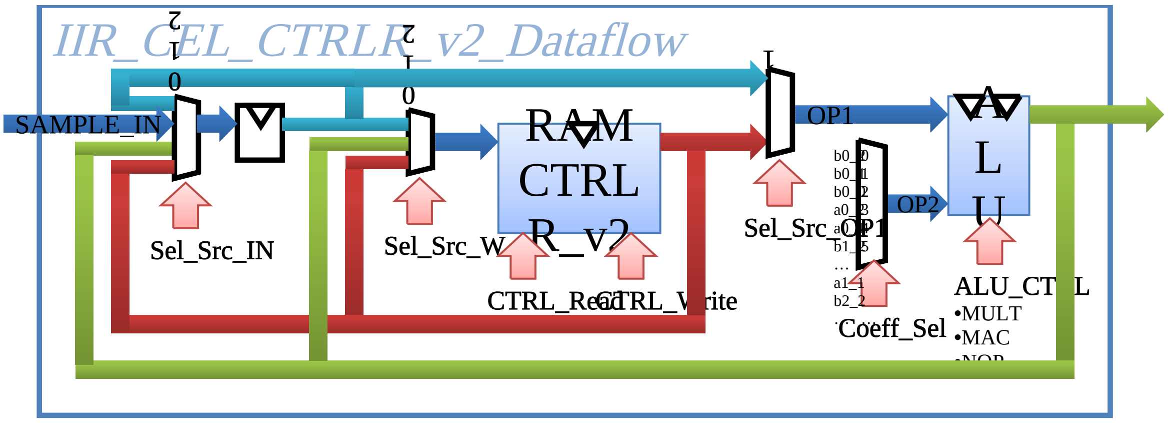
The IIR CEL can compute C channel “in parallel”. The data is set in parallel (Data channel0, channel1, …, channelC) and output in parallel. For each CEL, all channels use the same coefficients (b2, b1, b0, a2, a1). The coefficients are constants define at instantiation.

As shown is the next figure, IIR_CEL sub system is composed of 2 blocks:

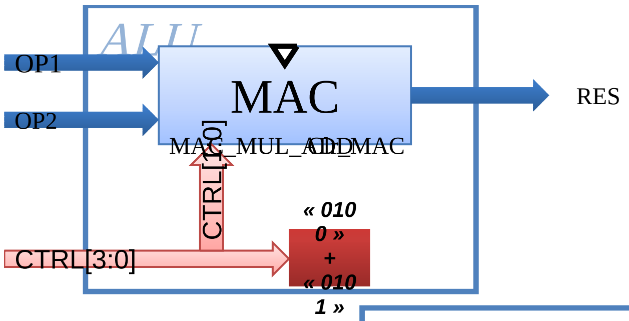
- Dataflow which receives the sample, compute, stock and output the data
- Control which controls the dataflow part

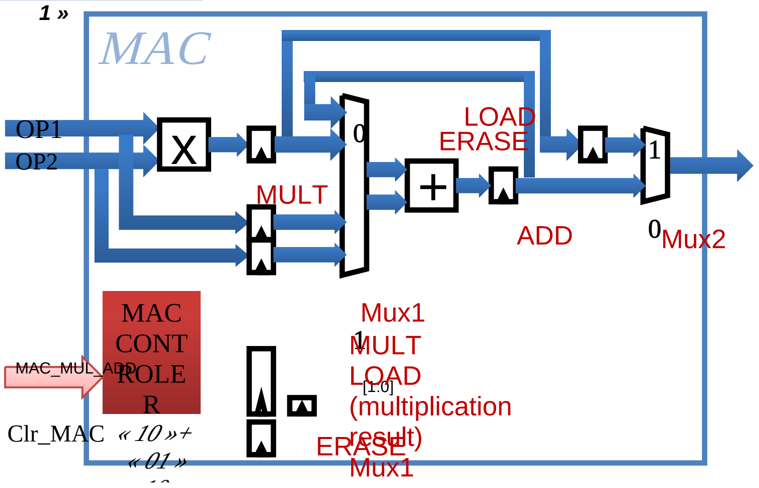


The dataflow :

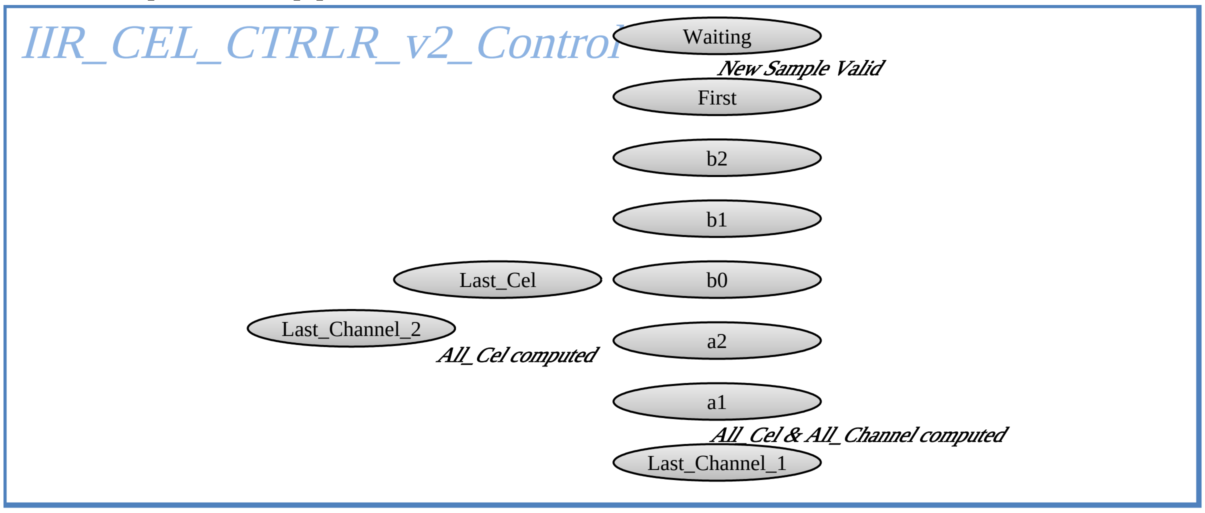


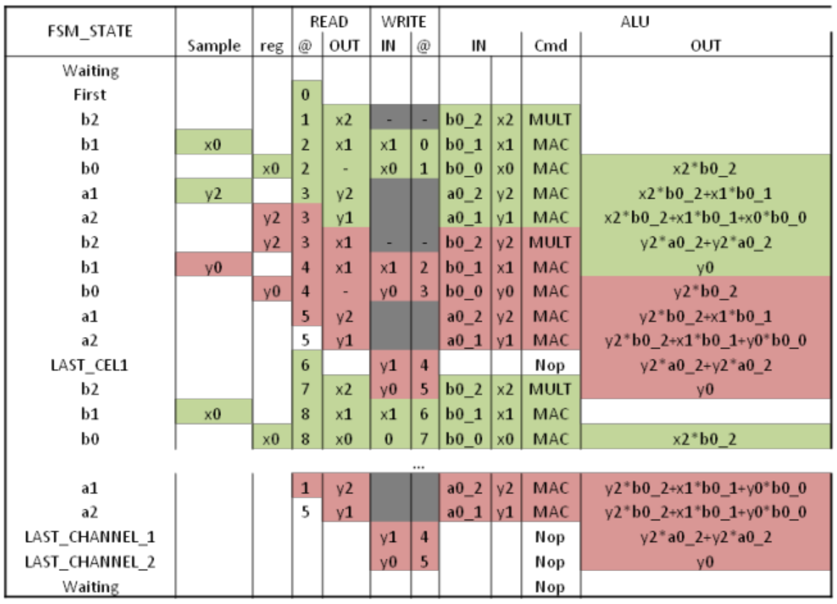
And its ALU :





The control part and the pipeline:





| FSM_STATE | | | READ | | WRITE | | ALU | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Sample | reg | @ | OUT | IN | @ | IN | | Cmd | OUT |
| Waiting | | | | | | | | | | |
| First | | | 0 | | | | | | | |
| b2 | | | 1 | x2 | - | - | b0_2 | x2 | MULT | |
| b1 | x0 | | 2 | x1 | x1 | 0 | b0_1 | x1 | MAC | |
| b0 | | x0 | 2 | - | x0 | 1 | b0_0 | x0 | MAC | x2*b0_2 |
| a1 | y2 | | 3 | y2 | | | a0_2 | y2 | MAC | x2*b0_2+x1*b0_1 |
| a2 | | y2 | 3 | y1 | | | a0_1 | y1 | MAC | x2*b0_2+x1*b0_1+x0*b0_0 |
| b2 | | y2 | 3 | x1 | - | - | b0_2 | y2 | MULT | y2*a0_2+y2*a0_2 |
| b1 | y0 | | 4 | x1 | x1 | 2 | b0_1 | x1 | MAC | y0 |
| b0 | | y0 | 4 | - | y0 | 3 | b0_0 | y0 | MAC | y2*b0_2 |
| a1 | | | 5 | y2 | | | a0_2 | y2 | MAC | y2*b0_2+x1*b0_1 |
| a2 | | | 5 | y1 | | | a0_1 | y1 | MAC | y2*b0_2+x1*b0_1+y0*b0_0 |
| LAST_CEL1 | | | 6 | | y1 | 4 | | | Nop | y2*a0_2+y2*a0_2 |
| b2 | | | 7 | x2 | y0 | 5 | b0_2 | x2 | MULT | y0 |
| b1 | x0 | | 8 | x1 | x1 | 6 | b0_1 | x1 | MAC | |
| b0 | | x0 | 8 | x0 | 0 | 7 | b0_0 | x0 | MAC | x2*b0_2 |
| | | | | | ... | | | | | |
| a1 | | | 1 | y2 | | | a0_2 | y2 | MAC | y2*b0_2+x1*b0_1+y0*b0_0 |
| a2 | | | 5 | y1 | | | a0_1 | y1 | MAC | y2*b0_2+x1*b0_1+y0*b0_0 |
| LAST_CHANNEL_1 | | | | | y1 | 4 | | | Nop | y2*a0_2+y2*a0_2 |
| LAST_CHANNEL_2 | | | | | y0 | 5 | | | Nop | y0 |
| Waiting | | | | | | | | | Nop | |

## 6.3 Registers

There is no AMBA registers.

## 6.4 Vendor and device identifiers

There is no vendor identifier or device identifier.

## 6.5 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 :<br>- use_RAM => hard-macro<br>- 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 |

## 6.6 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 | - |

## 6.7 Library dependencies

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

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

## 6.8 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;
```

## 6.9 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);
```