Datasheet

CC-PDMA-APB Documentation

1.3

Scope

This document describes the CC-PDMA-APB IP core. Module features and configuration registers are described. The document contains integration guide that covers synthesis options and instantiation example for easy implementation in customer's environment.

Contents

1. Peripheral DMA Controller

1.1 Functionality

- Transmission between peripheral devices and main memory,
- separate configuration for each channel,
- configurable transfer size - 1, 2, 4 bytes,
- configurable number of transfers - 1 to 65535 with reloading,
- ring buffer mode,
- configurable addressing modes - incremental, decremental, constant address,
- separate interrupts for each channel,
- round-robin or channel number priority.

1.2 Overview

The Peripheral Direct Memory Access controller allows data transfer between peripheral devices and the main memory without involving processor attention. The PDMA controller do not support data transfer between two peripherals or two memory regions. Each of PDMA channels has its own configuration registers set including active peripheral, memory address and transfer parameters. The PDMA controller communicates with peripherals using separate, dedicated peripheral bus layer that operates independently from processor bus. Upstream and Downstream PDMA channels operate independently so they act in parallel.

1.3 Block Diagram

Figure 1.1 presents the PDMA block diagram. Upstream and Downstream engines are connected to the dedicated dma channel bus. Upstream engine allows only transfer to the main memory and Downstream engine only transfers from the main memory. Both of them have their own channel arbiter and perform one transfer at a time using selected channel and corresponding peripheral. There can be configured two arbitration modes: round-robin and channel number priority (CH0 > CH1>...>CHn).

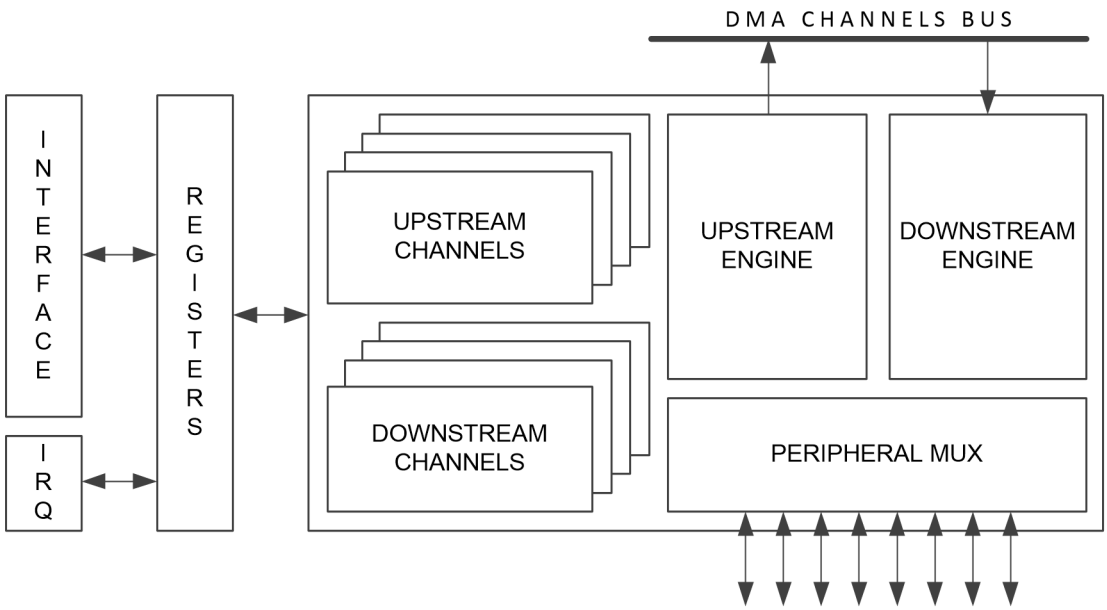


Figure 1.1. PDMA block diagram.

1.4 Transfer Configuration

1.4.1 Overview

PDMA controller consists of two independent Downstream and Upstream engines. Each of them has their own configuration registers. A number of step has to be performed to activate PDMA transfer. The peripheral device participating in the transfer has to be activated and configured properly. Upstream and Downstream engine activation is done using configuration register (CONF, 1.6.2). The peripheral select register (PSELECT, 1.6.7) of PDMA Upstream or Downstream channel has to be written with PID (Peripheral Identity) of desired peripheral module. Single transfer size has to be configured accordingly to the selected peripheral module. The source or destination address has to be written in address register (ADDRESS, 1.6.8). After each transfer address can be incremented or decremented by 1, 2 or 4 depending on the channel configuration. Number of transfers is configured using counter register (COUNTER, 1.6.10). Channel activation is done using transfer enable bit in control register (CTRL, 1.6.6). After each transfer content of the ADDRESS (1.6.8) and COUNTER (1.6.10) registers are updated. These registers can be read in any moment to monitor transfer progress. Transmission status can also be checked using ACT bit in STATUS register (1.6.5). After transmission ends (COUNTER = 0), ACT bit becomes 0. Writing a new value to the counter register automatically starts new transfer.

1.4.2 Memory Pointer

Each of PDMA channels has an address register (ADDRESS, 1.6.8) storing source/destination address of main memory location. After each transfer address can be incremented or decremented by 1, 2 or 4 depending on the channel configuration. PDMA channel has also an address reload register (ADDRESSREL, 1.6.9). Its content is loaded to the address register when transmission counter is zero. Address reload is then cleared unless ring buffer mode is active.

1.4.3 Transmission Counter

Each of PDMA channels has 16-bit counter register (COUNTER, 1.6.10). It has to be programmed with desired number of transfers. After each transfer counter is decremented by 1. PDMA channel has also an counter reload register (COUNTERREL, 1.6.11). Its content is loaded to the counter register when transmission counter is zero. Counter reload is then cleared unless ring buffer mode is active. Together with address reload register it allows alternate transmission to or from two buffers.

1.4.4 Ring Buffer

Ring Buffer mode block clearing of address reload (1.6.9) and counter reload registers (1.6.11) after copying them to address (1.6.8) and counter (1.6.10) registers. This allows continuous alternate transmission to or from two buffers.

1.4.5 Transfer Size

Each channel can be independently configured with desired 1, 2, or 4 byte transfer size. Transfer data and address are automatically adjusted and aligned to be bus with. Data to be transfered is LSB aligned and insignificant MSB bits are cleared.

1.4.6 Channel Activation

Transfer channel is activated in two stages. First, the global configuration is stored in CONF register (1.6.2). It is used to independently activate upstream or downstream engine. Each channel is activated in CTRL register (1.6.6). Transmission status can be checked using ACT bit in STATUS register (1.6.5). Bit is valid only when the transmission is in progress or counter register or counter reload register is greater than 0.

1.4.7 Priorities

PDMA controller allows to configure two arbitration modes: round-robin and channel number priority. Round-robin algorithm to prevent starvation problem. The channel number priority serves channels with lower index first. Arbitration mode is set independently for Upstream and Downstream engines in CONF (1.6.2) register.

1.5 Interrupts

There are four interrupt sources generated by Upstream and Downstream engines.

1.5.1 TCZ - Transfer Counter Zero

Transfer Counter Zero interrupt is signalized when COUNTER (1.6.10) register is zero. Interrupt line reflects TCZ bit in STATUS (1.6.5) register.

1.5.2 RCZ - Transfer Counter Reload Zero

Transfer Counter Reload Zero interrupt is signalized when COUNTER RELOAD (1.6.11) register is zero. Interrupt line reflects RCZ bit in STATUS (1.6.5) register.

1.5.3 MAR - Memory Address Reloaded

Memory Address Reloaded interrupt is signalized when ADDRESS (1.6.8) is loaded with ADDRESSREL (1.6.9) content. Interrupt line reflects MAR bit in STATUS (1.6.5) register.

1.5.4 MERR - Memory Access Error

Memory Access Error is signalized when memory transfer receives error response from memory bus. Interrupt line reflects MERR bit in STATUS (1.6.5) register. When transfer receives error response the channel will be automatically disabled.

1.6 Configuration Registers

1.6.1 Registers List

The core is controlled through registers mapped into memory address space. Not implemented locations are read as zeros.

Address Offset	Register	Name
0x00	CONF	Configuration Register
0x04	IRQMAP	Interrupt Mapping Register
0x08	INFO	Info Register
Downstream channels:		
baseAddrDown = 0x20 channelAddrDown = baseAddrDown + channelIndex * 0x20		
channelAddrDown + 0x00	STATUS	Status Register
channelAddrDown + 0x04	CTRL	Control Register
channelAddrDown + 0x08	PSELECT	Peripheral Select Register
channelAddrDown + 0x0C	ADDRESS	Memory Address Register
channelAddrDown + 0x10	ADDRESSREL	Memory Address Reload Register
channelAddrDown + 0x14	COUNTER	Transfer Counter Register
channelAddrDown + 0x18	COUNTERREL	Transfer Counter Reload Register
channelAddrDown + 0x1C	IRQM	Interrupt Mask Register
Upstream channels:		
baseAddrUp = 0x20 + channelDownNumber * 0x20 channelAddrUp = baseAddrUp + channelIndex * 0x20		
channelAddrUp + 0x00	STATUS	Status Register
channelAddrUp + 0x04	CTRL	Control Register
channelAddrUp + 0x08	PSELECT	Peripheral Select Register
channelAddrUp + 0x0C	ADDRESS	Memory Address Register
channelAddrUp + 0x10	ADDRESSREL	Memory Address Reload Register
channelAddrUp + 0x14	COUNTER	Transfer Counter Register
channelAddrUp + 0x18	COUNTERREL	Transfer Counter Reload Register
channelAddrUp + 0x1C	IRQM	Interrupt Mask Register

1.6.2 Base Configuration Register

Address: 0x00

31	30	...	...	...	...	9	8
DBG_STOP		...	...	...	...		
R/W	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
				USARN	USEN	DSARB	DSEN
R	R	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DSEN *Downstream Enable* Global enable for Downstream engine.

0 Downstream engine inactive.

1 Downstream engine active.

DSARB *Downstream Arbitration Mode*

0 Channel number priority. Channel 0 has the highest priority.

1 Round-robin algorithm.

UPEN *Upstream Enable* Global enable for Upstream engine.

0 Upstream engine inactive.

1 Upstream engine active.

UPARB *Upstream Arbitration Mode*

0 Channel number priority. Channel 0 has the highest priority.

1 Round-robin algorithm.

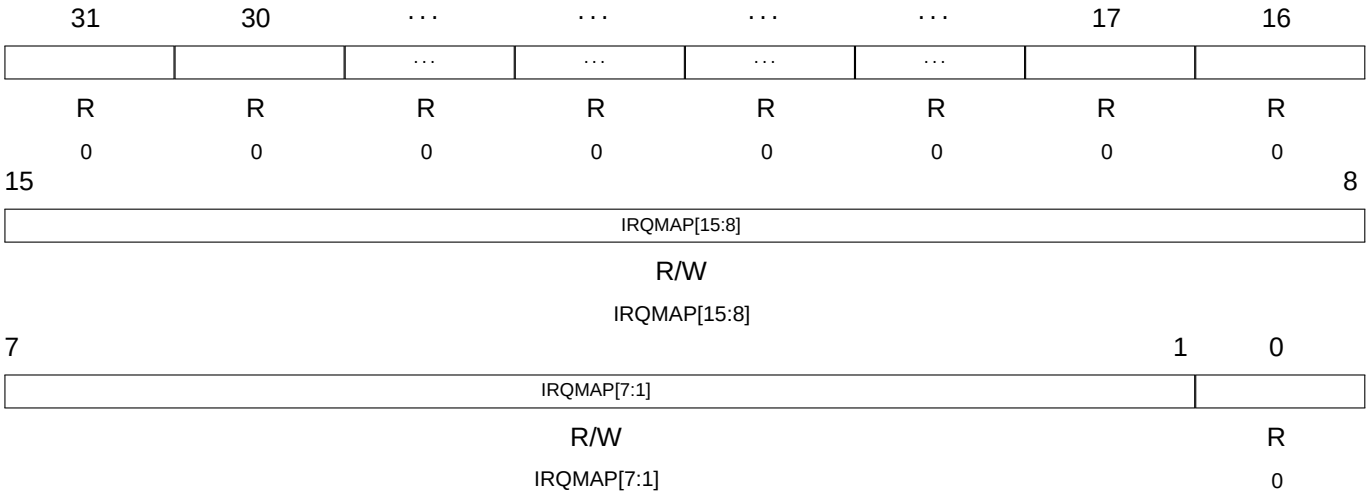
DBG_STOP *Stop in debug mode*

0 PDMA continues to work in debug mode.

1 PDMA stops transfers in debug mode.

1.6.3 Interrupt Mapping Register

Address: 0x04

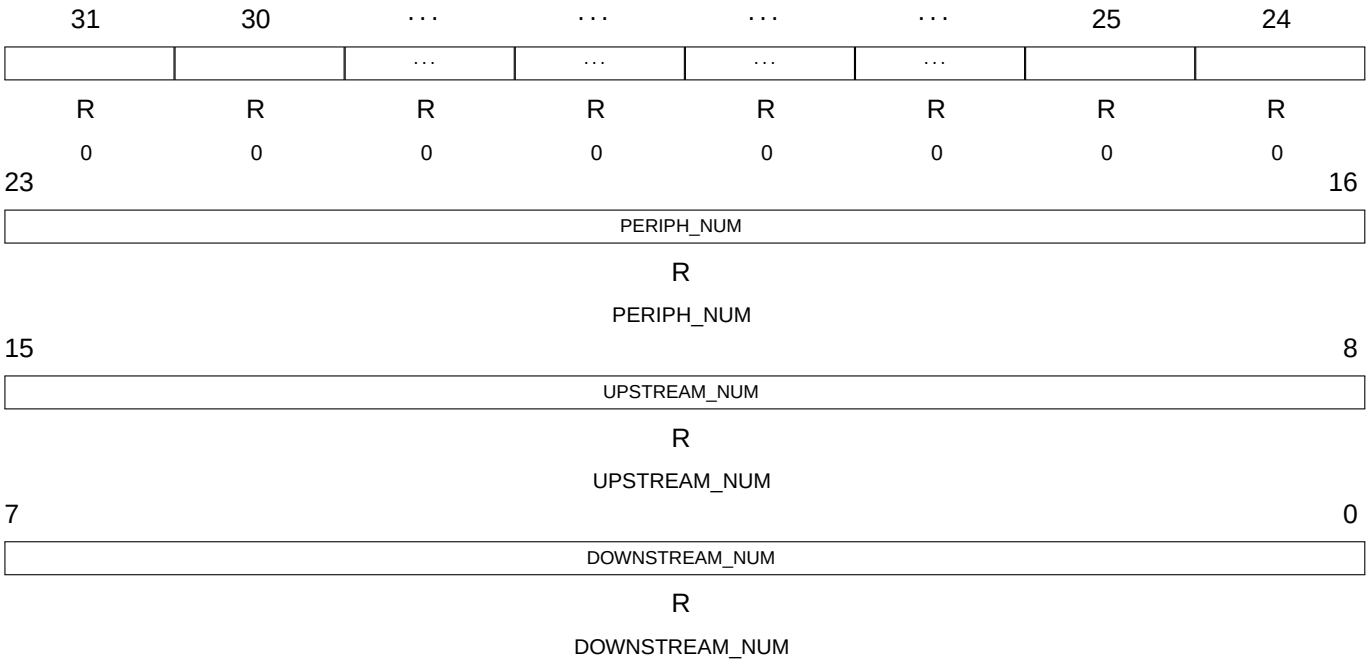


IRQMAP[15:1] *Interrupt Mapping*

Each set bit represents the interrupt number that will be passed to interrupt controller. It is allowed to set more than one bit.

1.6.4 Info Register

Address: 0x08



DOWNSTREAM_NUM *Downstream Channels Number*

Number of PDMA downstream channels.

UPSTREAM_NUM *Upstream Channels Number*

Number of PDMA upstream channels.

PERIPH_NUM *Peripherals Number*

Number of peripheral devices connected to the PDMA controller.

1.6.5 Status Register

Address: channelAddr + 0x00

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
			MERR	MAR	RCZ	TCZ	ACT
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

ACT *Channel Active*

0 Channel is inactive.

1 Channel is active.

Bit reflects the actual channel state. Disabling channel during the transfer will delay the action till the end of the transfer.

TCZ *Transfer Counter Zero*

0 Counter Register is greater than zero.

1 Counter Register is than zero.

RCZ *Counter Reload Zero*

0 Counter Reload Register is greater than zero.

1 Counter Reload Register is zero.

MAR *Memory Address Reload*

0 Bit is cleared after Status Register readout.

1 Bit is set after copying Memory Address Reload to Memory Address register.

MERR *Memory Access Error*

0 Bit is cleared after Status Register readout.

1 Bit is set then memory bus error occurs during the transfer.

1.6.6 Control Register

Address: channelAddr + 0x04

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

7	6	5	4	3	2	1	0
		TRU[1:0]		INC	DEC	RING	EN
R	R	R/W		R/W	R/W	R/W	R/W
0	0	0		0	0	0	0

EN *Transfer Enable*

0 Transfer is disabled.

1 Transfer is enabled.

RING *Ring Buffer*

0 Ring buffer disabled (Counter Reload Register will be cleared after copying to Counter Register).

1 Ring buffer enabled (Counter Reload Register will not be cleared after copying to Counter Register).

DEC *Address Decrement*

0 No action.

1 Address decrementation after each transfer.

INC *Address Increment*

0 No action.

1 Address incrementation after each transfer.

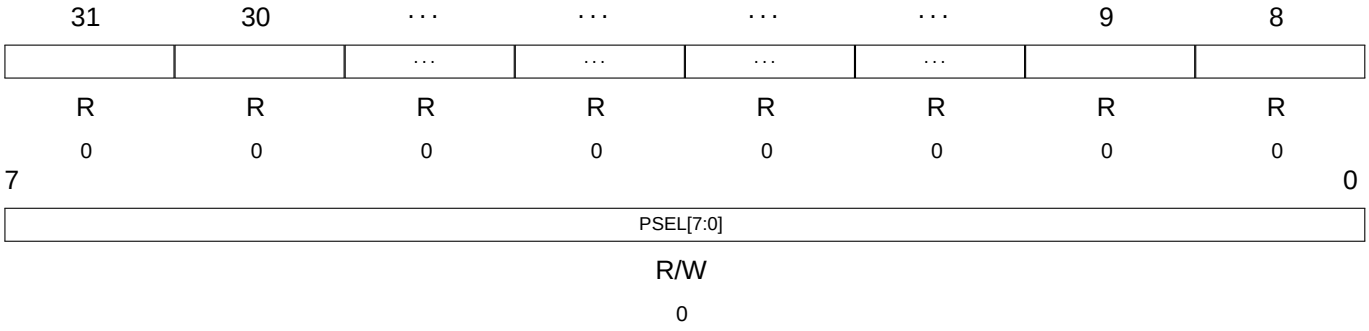
TRU[1:0] *Transfer Unit Size*

Single transfer size.

TRU[1:0]	Transfer unit size
00	32 bits
01	8 bits
10	16 bits
11	32 bits

1.6.7 Peripheral Select Register

Address: channelAddr + 0x08

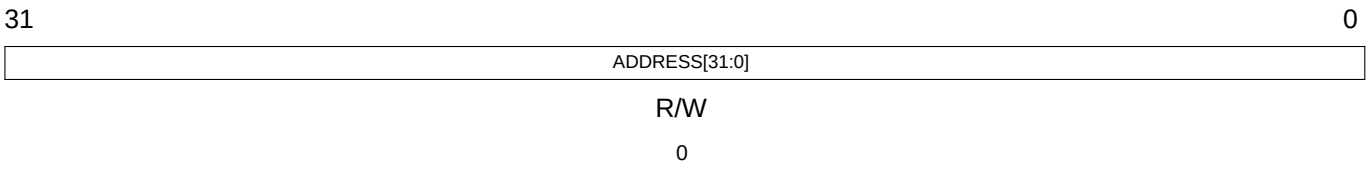


PSEL[7:0] *Peripheral Select*

Value directly decodes the index number of peripheral connected to the channel.

1.6.8 Address Register

Address: channelAddr + 0x0C

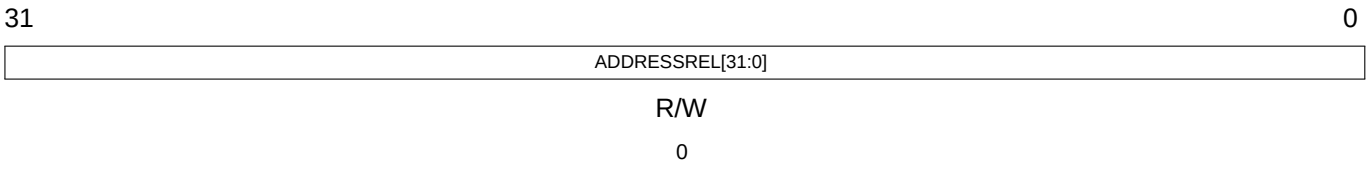


ADDRESS[31:0] *Memory Address*

Source or destination memory address. After each transfer can be incremented or decremented by 1, 2 or 4 depending on the channel configuration.

1.6.9 Memory Address Reload Register

Address: channelAddr + 0x10

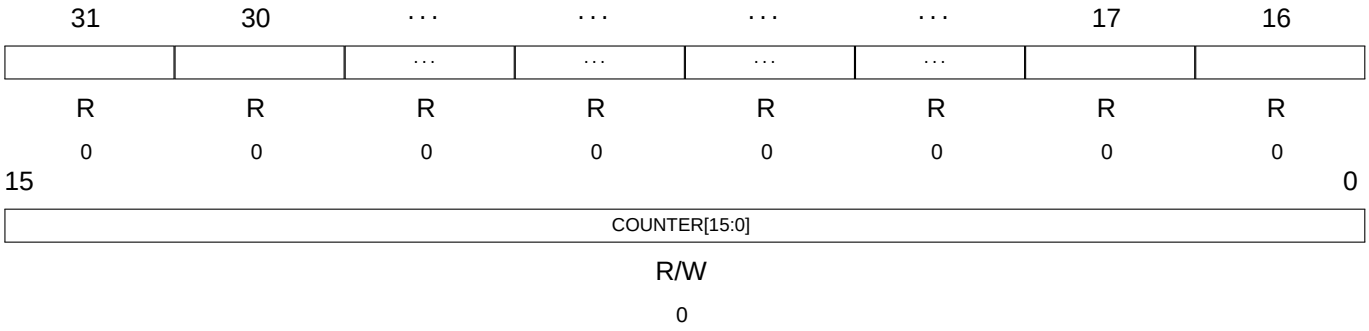


ADDRESSREL[31:0] *Memory Address Reload*

Source or destination memory address. Register content will be copied to Memory Address Register then Counter Register become zero.

1.6.10 Counter Register

Address: channelAddr + 0x14

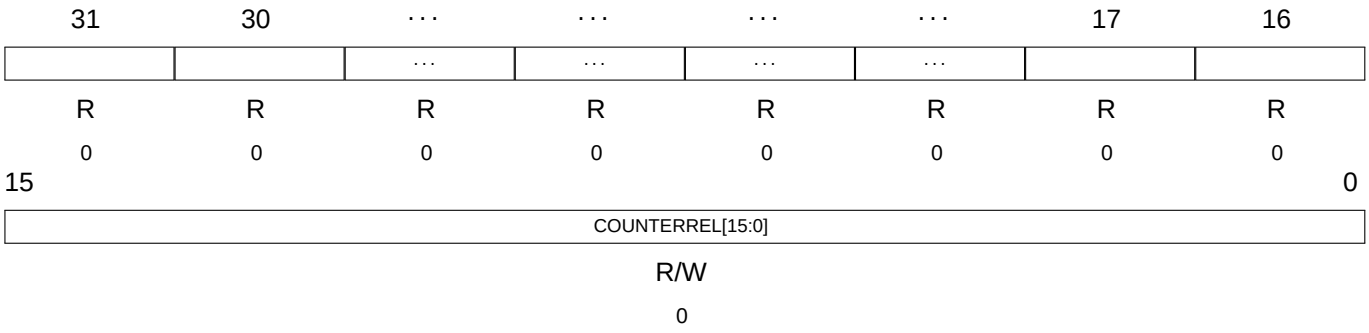


COUNTER[15:0] *Transfer Counter*

Number of transfers to be performed. Number is decremented after each transfer.

1.6.11 Counter Reload Register

Address: channelAddr + 0x18

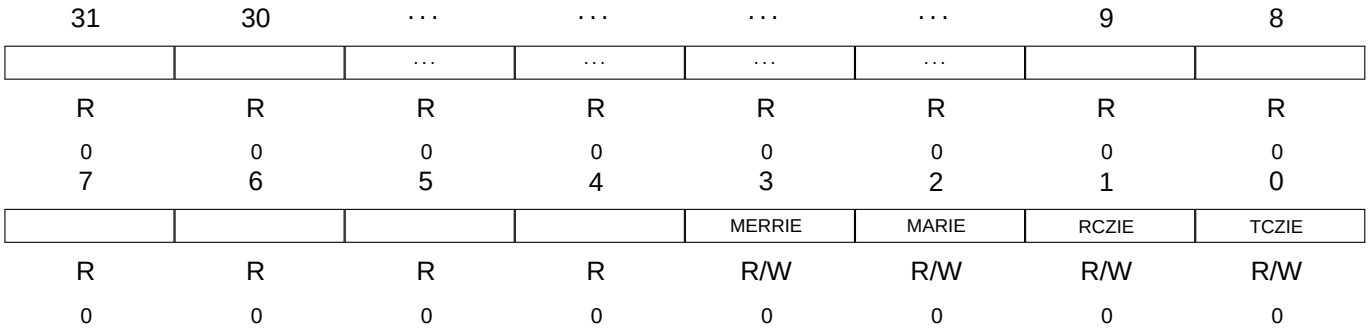


COUNTERREL[15:0] *Transfer Counter Reload*

Number of transfers to be performed. Register content will be copied to Counter Register then Counter Register become zero. Register is then cleared unless ring buffer mode is active.

1.6.12 Interrupt Mask Register

Address: channelAddr + 0x1C



TCZIE *Transfer Counter Zero Interrupt Enable*

0 Transfer Counter Zero Interrupt is disabled.

1 Transfer Counter Zero Interrupt is enabled.

RCZIE *Reload Counter Zero Interrupt Enable*

0 Reload Counter Zero Interrupt is disabled.

1 Reload Counter Zero Interrupt is enabled.

MARIE *Memory Address Reload Interrupt Enable*

0 Memory Address Reload Interrupt is disabled.

1 Memory Address Reload Interrupt is enabled.

MERRIE *Memory Access Error Interrupt Enable*

0 Memory Access Error Interrupt is disabled.

1 Memory Access Error Interrupt is enabled.

1.7 Implementation

1.7.1 Design Structure

The synthesible RTL IP core part (*COMMON/rtl* and *PDMA/rtl* folder) utilizes Verilog 2005 HDL. The testbench part (*PDMA/tb* folder) uses SystemVerilog language.

```
COMMON
├── rtl
│   ├── DFF_en.v
│   ├── edge_detector.v
│   └── ones_counter.v
PDMA
├── beh
├── rtl
│   ├── APB_PDMA.v
│   ├── PDMA_arbiter.v
│   ├── PDMA_channel_configuration.v
│   ├── PDMA_config.v
│   ├── PDMA_defines.v
│   ├── PDMA_downstream_channel.v
│   ├── PDMA_downstream_engine.v
│   ├── PDMA_downstream_FIFO.v
│   ├── PDMA_upstream_aggregator.v
│   ├── PDMA_upstream_engine.v
│   └── PDMA_upstream_FIFO.v
└── tb
    ├── common
    │   └── timescale.v
    ├── run
    │   ├── irun_apb_pdma_all.sh
    │   ├── irun_apb_pdma_all_downstream_spi_master_tests.sh
    │   ├── irun_apb_pdma_all_downstream_tests.sh
    │   ├── irun_apb_pdma_all_downstream_uart_tests.sh
    │   ├── irun_apb_pdma_all_upstream_downstream_spi_master_tests.sh
    │   ├── irun_apb_pdma_all_upstream_spi_master_tests.sh
    │   ├── irun_apb_pdma_all_upstream_tests.sh
    │   ├── irun_apb_pdma_all_upstream_uart_tests.sh
    │   ├── irun_apb_pdma_downstream.sh
    │   ├── irun_apb_pdma_downstream_spi_master.sh
    │   ├── irun_apb_pdma_downstream_uart.sh
    │   ├── irun_apb_pdma_flash.sh
    │   ├── irun_apb_pdma_upstream.sh
    │   ├── irun_apb_pdma_upstream_downstream_spi_master.sh
    │   ├── irun_apb_pdma_upstream_spi_master.sh
    │   └── irun_apb_pdma_upstream_uart.sh
    ├── tasks
    │   ├── APB
    │   │   ├── tb_APB_PDMA_init.v
    │   │   └── tb_APB_PDMA_reg_access_tasks.v
    │   └── common
    │       ├── tb_APB_PDMA_config_tasks.v
    │       └── tb_APB_PDMA_data_transfer_tasks.v
    └── tests
        └── tb_downstream_enable_test.sv
```

```
    ├── tb_downstream_enable_with_tcounter_reload_test_interrupts.sv
    ├── tb_downstream_enable_with_tcounter_reload_test.sv
    ├── tb_downstream_general_test_interrupts.sv
    ├── tb_downstream_general_test.sv
    ├── tb_downstream_hsi_error.sv
    ├── tb_downstream_tcounter_reload_resume_test_interrupts.sv
    ├── tb_downstream_tcounter_reload_resume_test.sv
    ├── tb_downstream_tcounter_resume_test_interrupts.sv
    ├── tb_downstream_tcounter_resume_test.sv
    ├── tb_interrupt_MAPPING_test.sv
    ├── tb_upstream_enable_test.sv
    ├── tb_upstream_enable_with_tcounter_reload_test_interrupts.sv
    ├── tb_upstream_enable_with_tcounter_reload_test.sv
    ├── tb_upstream_general_test_interrupts.sv
    ├── tb_upstream_general_test.sv
    ├── tb_upstream_hsi_error.sv
    ├── tb_upstream_tcounter_reload_resume_test_interrupts.sv
    ├── tb_upstream_tcounter_reload_resume_test.sv
    ├── tb_upstream_tcounter_resume_test_interrupts.sv
    └── tb_upstream_tcounter_resume_test.sv
├── virtual_components
    ├── virtual_ahb_lite_mem_ctrl.sv .4 virtual_dma.sv
    ├── virtual_downstream_peripheral.sv
    ├── virtual_mem_ctrl.sv
    ├── virtual_upstream_peripheral_array.sv
    └── virtual_upstream_peripheral.sv
├── tb_PDMA_arbiter.sv
├── tb_PDMA_downstream_and_UART.sv
├── tb_PDMA_downstream_SPI_master.sv
├── tb_PDMA_downstream.sv
├── tb_PDMA_SPI_FLASH.sv
├── tb_PDMA_upstream_and_UART.sv
├── tb_PDMA_upstream_SPI_master.sv
└── tb_PDMA_upstream.sv
└── compile.list
```

1.7.2 Simulation Flow

The IP Core is provided with self-checking testbench to verify the correct operation of the IP prior to use in a design. To run the simulation using Cadence® Incisive® Enterprise Simulator run *irun_apb_pdma_all.sh* script located in *PDMA/tb/run* folder. The script will run multiple simulations with different configuration options. Aside from verification using virtual components, the verification suite will use CC-SPI-APB and CC-UART-APB components connected to CC-PDMA-APB peripheral side. The script will also verify CC-PDMA_ARB-AHB component that is used to convert the native CC-PDMA-APB memory interface to AMBA AHB-Lite master port.

1.7.3 Clock and Reset

The CC-PDMA-APB utilizes a fully synchronous design with one positive edge clocking domain and negative asynchronous reset assertion. External reset synchronizer has to be used to ensure synchronous reset deassertion.

1.7.4 Constraints

In most cases only module output ports are registered. Therefore, it is a good practice to reserve the entire clock cycle for module inputs combinational logic and set minimal input delay (*set_input_delay* command). Registered outputs leave the entire clock cycle for external logic (*set_output_delay* command).

1.7.5 Configuration Options

The table below shows the generic parameters of the core.

Generic name	Description	Range	Default
reg_ADDR_width	APB PADDR signal width	1:32	7
peripherals_number	Number of peripherals	1:32	6
peripheral_select_width	Width of the peripheral select register	1:5	3
downstream_channel_number	Number of downstream channels	1:32	4
downstream_channel_number_width	Downstream channels number width	1:5	2
upstream_channel_number	Number of upstream channels	1:32	4
upstream_channel_number_width	Upstream channels number width	1:5	2
downstream_data_aggregation_and_FIFO_enable	Enable downstream data aggregation	0,1	0
downstream_fifo_depth	Downstream FIFO depth	0:32	0
downstream_fifo_depth_width	Downstream FIFO depth width	0:5	0
upstream_data_aggregarion_enable	Enable upstream data aggregation	0,1	0
upstream_fifo_depth	Upstream FIFO depth	0:32	0
upstream_fifo_depth_width	Upstream FIFO depth width	0:5	0
upstream_fifo_water_level	Water level of upstream FIFO	0:32	0
default_interrupt_MAPPING	Reset value of interrupt_MAPPING register	1:32767	0
big_endian	Selects big-endian or little-endian byte ordering of memory controller interface	0, 1	1

1.7.6 Configuration Options Description

reg_ADDR_width *APB PADDR signal width*

Width (in bits) of the PADDR input (must be set accordingly to the number of channels).

$reg\_ADDR\_width = clog2(8 + 8 * downstream\_channel\_number + 8 * upstream\_channel\_number)$.

peripherals_number *Number of peripherals*

Number of supported peripherals.

peripheral_select_width *Width of the peripheral select register*

Width of the peripheral select register.

$peripheral\_select\_width = clog2(peripherals\_number)$.

downstream_channel_number *Number of downstream channels*

Number of downstream channels.

downstream_channel_number_width *Downstream channels number width*

Downstream channels number width.

$downstream\_channel\_number\_width = clog2(downstream\_channel\_number)$.

upstream_channel_number *Number of upstream channels*

Number of upstream channels.

upstream_channel_number_width *Upstream channels number width*

Downstream channels number width.

$upstream\_channel\_number\_width = clog2(upstream\_channel\_number)$.

downstream_data_aggregation_and_FIFO_enable *Enable downstream data aggregation*

Enable/disable downstream FIFO and/or aggregator functionality. With aggregator functionality enabled, each channel will access memory controller to get full 32-bit data word. This data will then be sent to the peripheral according to the configured data width (i.e. one byte at a time, or one 16-bit word at a time). Each channel may also have a FIFO queue just after the aggregator module.

downstream_fifo_depth *Downstream FIFO depth*

Depth of the downstream FIFO (in 32-bit data words). Each channel may have a FIFO queue placed after the aggregator module (note, that aggregator is required in this configuration, so downstream_data_aggregation_and_ _FIFO_enable must be set to 1 in order for downstream_fifo_depth setting to take effect). The actual depth of the FIFO depends on the configured transfer data width. For instance with downstream_fifo_depth set to 1, FIFO may hold data for single 32-bit data transfer, two 16-bits transfers or four 8-bit transfers. When downstream_fifo_depth is set to 0, a pseudo-FIFO is created that can store data for single transfer (independent of the configured transfer size).

downstream_fifo_depth_width *Downstream FIFO depth width*

Downstream FIFO depth width.

$downstream\_fifo\_depth\_width = clog2(downstream\_fifo\_depth)$.

upstream_data_aggregarion_enable *Enable upstream data aggregation*

Enable/disable upstream aggregator functionality. Each channel may have data aggregator attached just after the upstream engine module. Its role is to aggregate data transferred from the peripheral in order to decrease the number of memory accesses. This module will buffer transferred data (of lower bit width) until full 32-bit worth of data payload is aggregated. Only then, memory (or FIFO) data transfer will be initialized. Note, that in contrast to the similar downstream aggregator module, for upstream, both aggregator and FIFO may be enabled independently.

upstream_fifo_depth *Upstream FIFO depth*

Upstream path may be enhanced with the FIFO queue, used to buffer data before sending it to the memory controller. In contrast to the downstream FIFO, upstream FIFO count specify the absolute number of implemented entries (independent of the transfer length configuration). Note also, that upstream can have only 1 FIFO for all the channels. Setting upstream_fifo_depth to 0 effectively removed FIFO module from the system.

upstream_fifo_depth_width *Upstream FIFO depth width*

Upstream FIFO depth width.

$upstream\_fifo\_depth\_width = clog2(upstream\_fifo\_depth)$.

upstream_fifo_water_level *Water level of upstream FIFO*

Water level of assigned to upstream FIFO. Upstream module is able to assert hsi_lock signal. This signal is asserted, whenever upstream FIFO becomes full and will be held high as long as the number of data within FIFO exceeds upstream_fifo_water_level (when this value is reached, hsi_lock signal will be deasserted). Setting this value equal to upstream_fifo_depth effectively disables handling of the hsi_lock signal.

default_interrupt_MAPPING *Reset value of interrupt_MAPPING register*

Reset value of interrupt_MAPPING register.

big_endian *Memory controller interface endianness*

Set one for big-endian or zero for little-endian byte ordering of memory controller interface.

1.7.7 Signals Description

Signal name	Description	I/O	Active	Type
PCLK	Synchronous clock	I	rising	clock
PRESETn	Asynchronous reset	I	low	reset
PSEL	APB peripheral select	I	high	comb.
PENABLE	APB bus enable	I	high	comb.
PADDR[reg_ADDR_width+1:2]	APB bus address	I	data	comb.
PWRITE	APB bus write	I	high	comb.
PWDATA[31:0]	APB bus write data	I	data	comb.
PREADY	APB bus ready	O	high	reg.
PRDATA[31:0]	APB bus read data	O	data	reg.
downstream_interrupt_TCZ [downstream_channel_number-1:0]	Transfer counter zero interrupt	O	high	reg.
downstream_interrupt_RCZ [downstream_channel_number-1:0]	Reload counter zero interrupt	O	high	reg.
downstream_interrupt_MAR [downstream_channel_number-1:0]	Memory address reloaded interrupt	O	high	reg.
downstream_interrupt_MERR [downstream_channel_number-1:0]	Memory access error interrupt	O	high	reg.
upstream_interrupt_TCZ [upstream_channel_number-1:0]	Transfer counter zero interrupt	O	high	reg.
upstream_interrupt_RCZ [upstream_channel_number-1:0]	Reload counter zero interrupt	O	high	reg.
upstream_interrupt_MAR [upstream_channel_number-1:0]	Memory address reloaded interrupt	O	high	reg.
upstream_interrupt_MERR [upstream_channel_number-1:0]	Memory access error interrupt	O	high	reg.
interrupt_MAPPING[15:1]	Interrupt mapping vector	O	data	reg.
downstream_hsi_address[31:0]	Address for memory controller	O	data	reg.
downstream_hsi_data_in[31:0]	Data from memory controller	I	data	comb.
downstream_hsi_load	Load signal for memory controller	O	high	reg.
downstream_hsi_nstall	Stall input from memory controller	I	low	comb.
downstream_hsi_error	Memory access error indication	I	high	comb.
downstream_hsi_lock	Interface lock signal	O	data	const.
upstream_hsi_address[31:0]	Address for memory controller	O	data	reg.
upstream_hsi_data_out[31:0]	Data to memory controller	O	data	reg.
upstream_hsi_store[3:0]	Store signal for memory controller	O	data	reg.
upstream_hsi_nstall	Stall input from memory controller	I	low	comb.
upstream_hsi_error	Memory access error indication	I	high	comb.
upstream_hsi_lock	Interface lock signal	O	high	const./reg.[1]

Downstream_enable [peripherals_number-1 : 0]	Enable signal for PDMA interface	O	high	reg.
Downstream_busy [peripherals_number-1 : 0]	Busy signal from PDMA interface	I	high	comb.
Downstream_request [peripherals_number-1 : 0]	Request from peripheral's PDMA interface	I	high	comb.
Downstream_ack [peripherals_number-1 : 0]	Acknowledge signal for PDMA interface request	O	high	reg.
Downstream_data[31:0]	Data to peripheral's PDMA interface; Shared for all peripheral devices	O	data	reg.
Upstream_enable [peripherals_number-1 : 0]	Enable signal for PDMA interface	O	high	reg.
Upstream_busy [peripherals_number-1 : 0]	Busy signal from PDMA interface	I	high	comb.
Upstream_request [peripherals_number-1 : 0]	Request from peripheral's PDMA interface	I	high	comb.
Upstream_ack [peripherals_number-1 : 0]	Acknowledge signal for PDMA interface request	O	high	reg.
Upstream_data[31:0]	Data from peripheral's PDMA interface; Ored Upstream_data output of all peripheral devices	I	data	comb.
clock_request	Clock request signal	O	high	reg.
debug_mode	Debug mode indicator (1 - core is halted)	I	high	comb.
debug_ack	Confirm debug mode	O	high	reg.

1.7.8 Peripheral Connection

Figure 1.2 shows the connection between the PDMA Downstream Engine and the peripheral devices. Each peripheral device has its own set of *Downstream_enable*, *Downstream_busy*, *Downstream_request* and *Downstream_ack* signals. *Downstream_data* bus is shared for all peripheral devices.

Figure 1.3 shows the connection between the PDMA Upstream Engine and the peripheral devices. Each peripheral device has its own set of *Upstream_enable*, *Upstream_busy*, *Upstream_request* and *Upstream_ack* signals. *Upstream_data* buses of all peripheral devices are OR-ed together and passed to the PDMA controller. Only one peripheral device is allowed to drive *Upstream_data* bus at a time.

[1] Constant when upstream_fifo_depth = 0 and upstream_data_aggregarion_enable = 0. Otherwise registered.

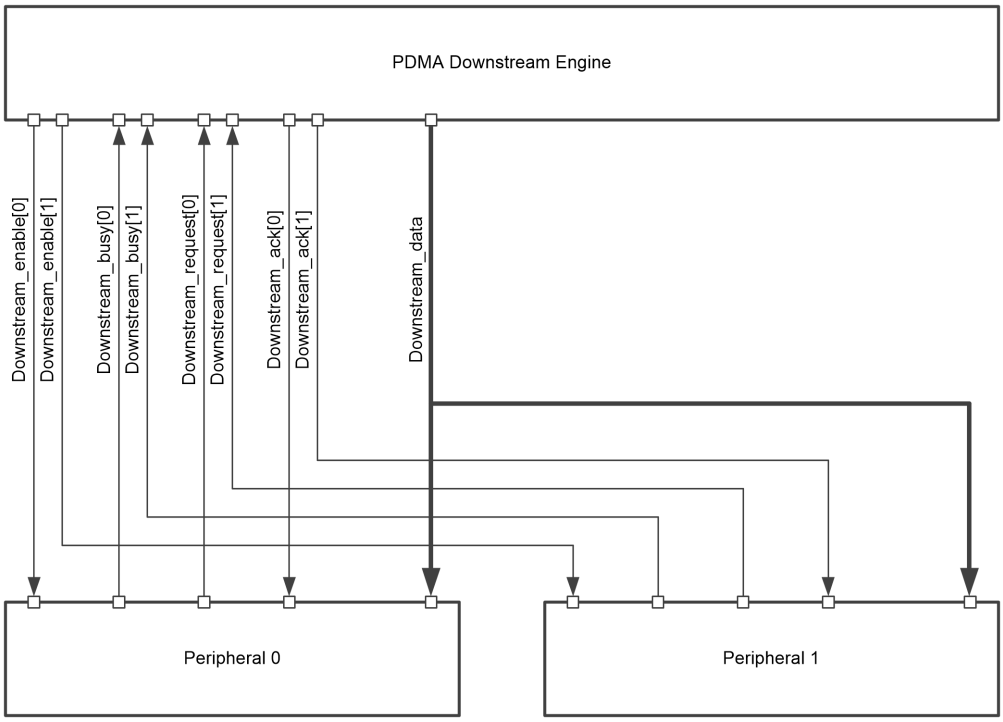


Figure 1.2. Connection of a peripheral device downstream signals.

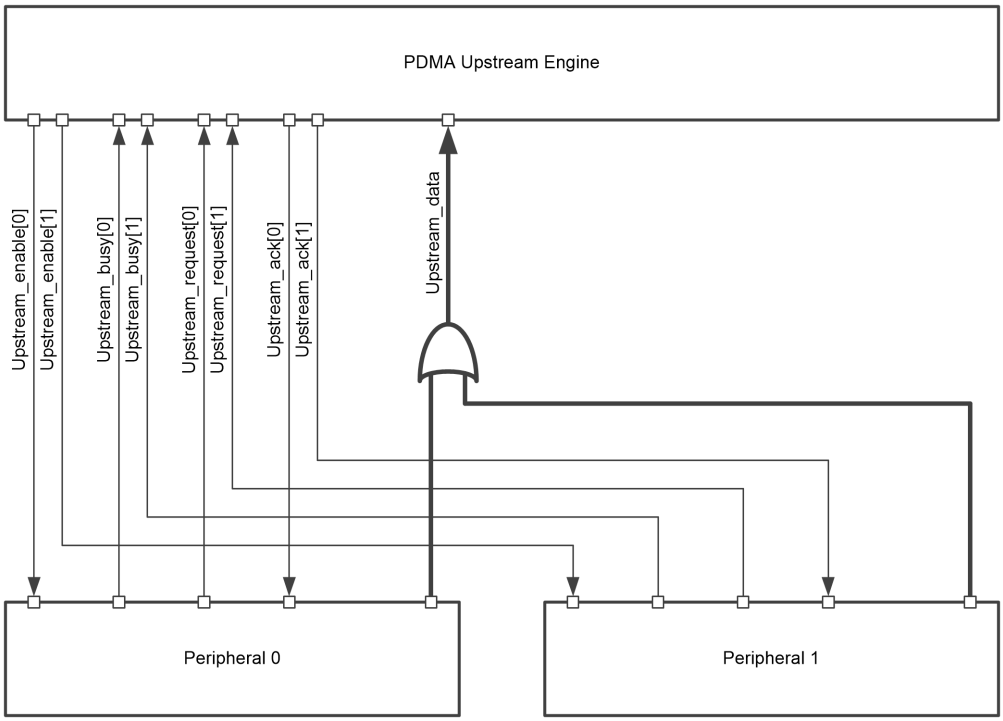


Figure 1.3. Connection of a peripheral device upstream signals.

1.7.9 Instantiation

```
icg
    icg_pdma   ( .E(pdma_PSEL|pdma_clock_request|debug_mode),
                 .clk(PCLK),
                 .gclk(pdma_clk),
                 .scan_enable(scan_enable));

APB_PDMA      #( .reg_ADDR_width(CFG_DMA_REG_LOG),
                 .peripherals_number(CFG_DMA_PERIPH_NUM),
                 .peripheral_select_width(CFG_DMA_PERIPH_NUM_LOG),
                 .downstream_channel_number(CFG_DMA_DWN),
                 .downstream_channel_number_width(CFG_DMA_DWN_LOG),
                 .upstream_channel_number(CFG_DMA_UP),
                 .upstream_channel_number_width(CFG_DMA_UP_LOG),
                 .default_interrupt_MAPPING(CFG_DEF_INT_MAPPING),
                 .downstream_data_aggregation_and_FIFO_enable(CFG_DMA_AGGR_EN),
                 .downstream_fifo_depth(CFG_DMA_DWN_FIFO_DEPTH),
                 .downstream_fifo_depth_width(DMA_DWN_FIFO_LOG),
                 .upstream_data_aggregarion_enable(CFG_DMA_AGGR_EN),
                 .upstream_fifo_depth(CFG_DMA_UP_FIFO_DEPTH),
                 .upstream_fifo_depth_width(DMA_UP_FIFO_LOG),
                 .upstream_fifo_water_level(CFG_DMA_WATER_LEVEL))

    APB_PDMA_u ( .PCLK(pdma_clk),
                 .PRESETn(pdma_rst),
                 .PSEL(pdma_PSEL),
                 .PENABLE(PENABLE),
                 .PADDR(PADDR[CFG_DMA_REG_LOG+1:2]),
                 .PWRITE(PWRITE),
                 .PWDATA(PWDATA),
                 .PREADY(pdma_PREADY),
                 .PRDATA(pdma_PRDATA),
                 .downstream_interrupt_TCZ(downstream_interrupt_TCZ),
                 .downstream_interrupt_RCZ(downstream_interrupt_RCZ),
                 .downstream_interrupt_MAR(downstream_interrupt_MAR),
                 .downstream_interrupt_MERR(downstream_interrupt_MERR),
                 .upstream_interrupt_TCZ(upstream_interrupt_TCZ),
                 .upstream_interrupt_RCZ(upstream_interrupt_RCZ),
                 .upstream_interrupt_MAR(upstream_interrupt_MAR),
                 .upstream_interrupt_MERR(upstream_interrupt_MERR),
```


```
                .interrupt_MAPPING(pdma_interrupt_MAPPING),
                .downstream_hsi_address(downstream_hsi_address),
                .downstream_hsi_data_in(downstream_hsi_data_in),
                .downstream_hsi_load(downstream_hsi_load),
                .downstream_hsi_nstall(downstream_hsi_nstall),
                .downstream_hsi_error(downstream_hsi_error),
                .downstream_hsi_lock(downstream_hsi_lock),
                .upstream_hsi_address(upstream_hsi_address),
                .upstream_hsi_data_out(upstream_hsi_data_out),
                .upstream_hsi_store(upstream_hsi_store),
                .upstream_hsi_nstall(upstream_hsi_nstall),
                .upstream_hsi_error(upstream_hsi_error),
                .upstream_hsi_lock(upstream_hsi_lock),
                .Downstream_enable(  {periph0_downstream_enable,
                                      periph1_downstream_enable}),
                .Downstream_busy(    {periph0_downstream_busy,
                                      periph1_downstream_busy}),
                .Downstream_request( {periph0_downstream_request,
                                      periph1_downstream_request}),
                .Downstream_ack({    periph0_downstream_ack,
                                      periph1_downstream_ack}),
                .Downstream_data(downstream_data),
                .Upstream_enable(    {periph0_upstream_enable,
                                      periph1_upstream_enable}),
                .Upstream_busy(      {periph0_upstream_busy,
                                      periph1_upstream_busy}),
                .Upstream_request(   {periph0_upstream_request,
                                      periph1_upstream_request}),
                .Upstream_ack(       {periph0_upstream_ack,
                                      periph1_upstream_ack}),
                .Upstream_data(upstream_data),
                .clock_request(pdma_clock_request),
                .debug_mode(debug_mode),
                .debug_ack(debug_ack));

assign upstream_data             = periph0_upstream_data |
                                   periph1_upstream_data;

assign pdma_downstream_interrupt = (|downstream_interrupt_TCZ) |
                                   (|downstream_interrupt_RCZ) |
                                   (|downstream_interrupt_MAR) |
                                   (|downstream_interrupt_MERR);
```

```
assign pdma_upstream_interrupt   = (|upstream_interrupt_TCZ) |
                                   (|upstream_interrupt_RCZ) |
                                   (|upstream_interrupt_MAR) |
                                   (|upstream_interrupt_MERR);

assign pdma_irq                  = pdma_downstream_interrupt |
                                   pdma_upstream_interrupt;

assign pdma_irq_vector           = pdma_interrupt_MAPPING & {15{pdma_irq}};

APB_PERIPH0_u  ( .PCLK(periph0_clk),
                 .PRESETn(rst),
                 .PSEL(periph0_PSEL),
                 .PENABLE(periph0_PENABLE),
                 .PADDR(PADDR),
                 .PWRITE(PWRITE),
                 .PWDATA(PWDATA),
                 .PREADY(periph0_PREADY),
                 .PRDATA(periph0_PRDATA),
                 ...
                 .Downstream_enable(periph0_downstream_enable),
                 .Downstream_busy(periph0_downstream_busy),
                 .Downstream_request(periph0_downstream_request),
                 .Downstream_ack(periph0_downstream_ack),
                 .Downstream_data(downstream_data),
                 .Upstream_enable(periph0_upstream_enable),
                 .Upstream_busy(periph0_upstream_busy),
                 .Upstream_request(periph0_upstream_request),
                 .Upstream_ack(periph0_upstream_ack),
                 .Upstream_data(periph0_upstream_data));

APB_PERIPH1_u  ( .PCLK(periph1_clk),
                 .PRESETn(rst),
                 .PSEL(periph1_PSEL),
                 .PENABLE(periph1_PENABLE),
                 .PADDR(PADDR),
                 .PWRITE(PWRITE),
                 .PWDATA(PWDATA),
                 .PREADY(periph1_PREADY),
                 .PRDATA(periph1_PRDATA),
                 ...
```


```
.Downstream_enable(periph1_downstream_enable),
.Downstream_busy(periph1_downstream_busy),
.Downstream_request(periph1_downstream_request),
.Downstream_ack(periph1_downstream_ack),
.Downstream_data(downstream_data),
.Upstream_enable(periph1_upstream_enable),
.Upstream_busy(periph1_upstream_busy),
.Upstream_request(periph1_upstream_request),
.Upstream_ack(periph1_upstream_ack),
.Upstream_data(periph1_upstream_data));
```

1.8 Revision History

Doc. Rev.	Date	Comments
1.3	12-2017	Added missing interrupt_MAPPING signal description in 1.7.7 Signals Description section.
1.2	09-2017	Added endianness selection parameter in 1.7.5 Configuration Options section.
1.1	08-2017	Added 1.7.8 Peripheral Connection section. Updated clock gating cell instantiation in 1.7.9 Instantiation section.
1.0	05-2017	First Issue.

ChipCraft Sp. z o.o. Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND www.chipcraft-ic.com

 CC-PDMA-APB-Doc_122017.