Datasheet

CC-PDMA_ARB-AHB Documentation

1.1

Scope

This document describes the CC-PDMA_ARB-AHB 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. CC-PDMA_ARB-AHB IP Core

1.1 Description

The CC-PDMA_ARB-AHB IP core can be used to convert the native CC-PDMA memory interface to AMBA AHB-Lite master port.

1.1.1 Block Diagram

Figure 1.1 shows how the CC-PDMA_ARB-AHB IP core can be uses to convert CC-PDMA upstream and downstream DMA channels into a single AMBA AHB-Lite master port.

1.1.2 Use Cases

CC-PDMA_ARB-AHB IP core can also be used to convert CC-PDMA upstream and downstream DMA channels into separate AMBA AHB-Lite master ports, each dedicated for upstream or downstream transfer (Figure 1.2). Furthermore, it is also possible to connect two or more CC-PDMA cores to a single CC-PDMA_ARB-AHB arbiter (Figure 1.3).

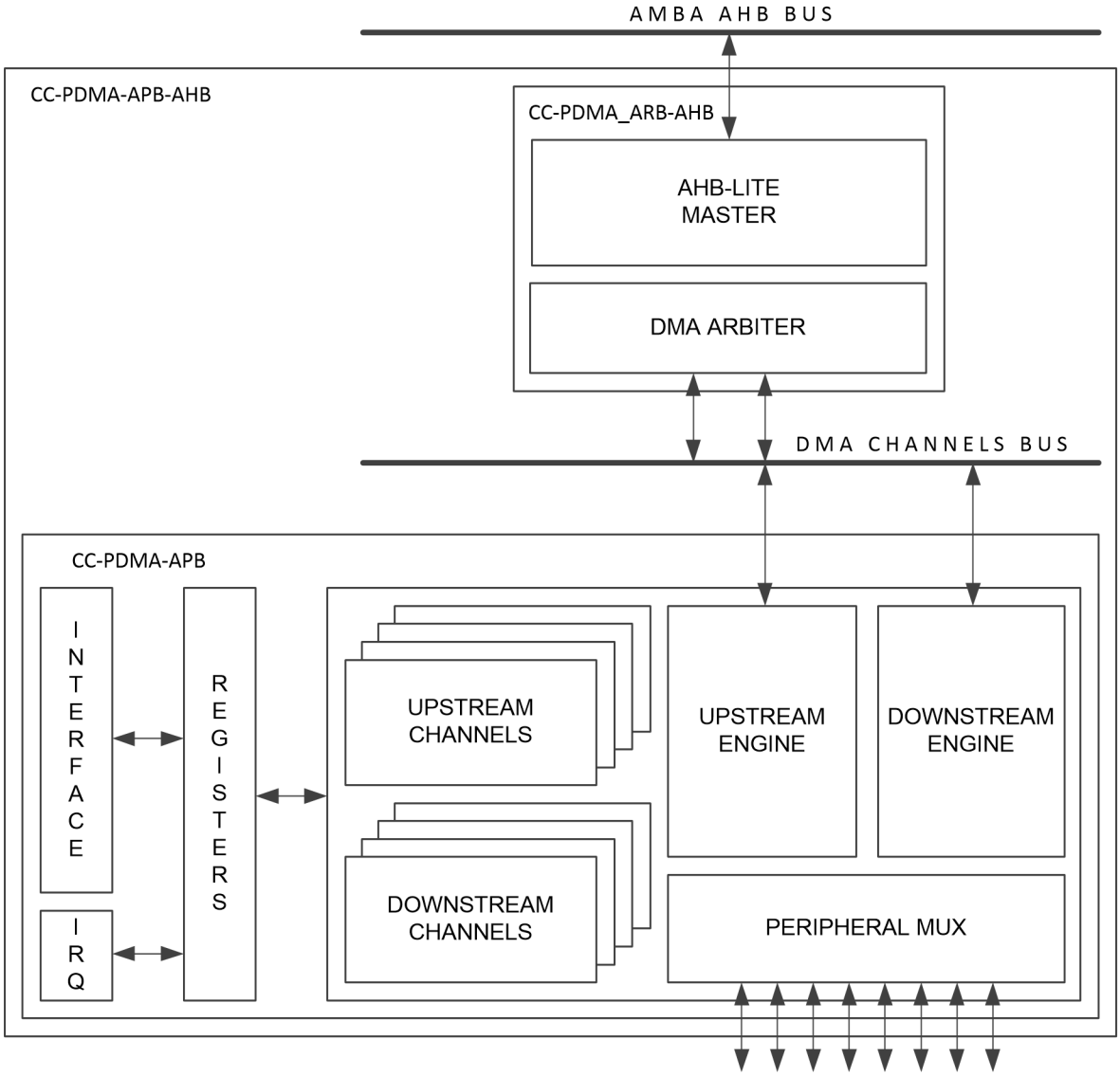

Figure 1.1. Block diagram of CC-PDMA-APB core with CC-PDMA_ARB-AHB wrapper.

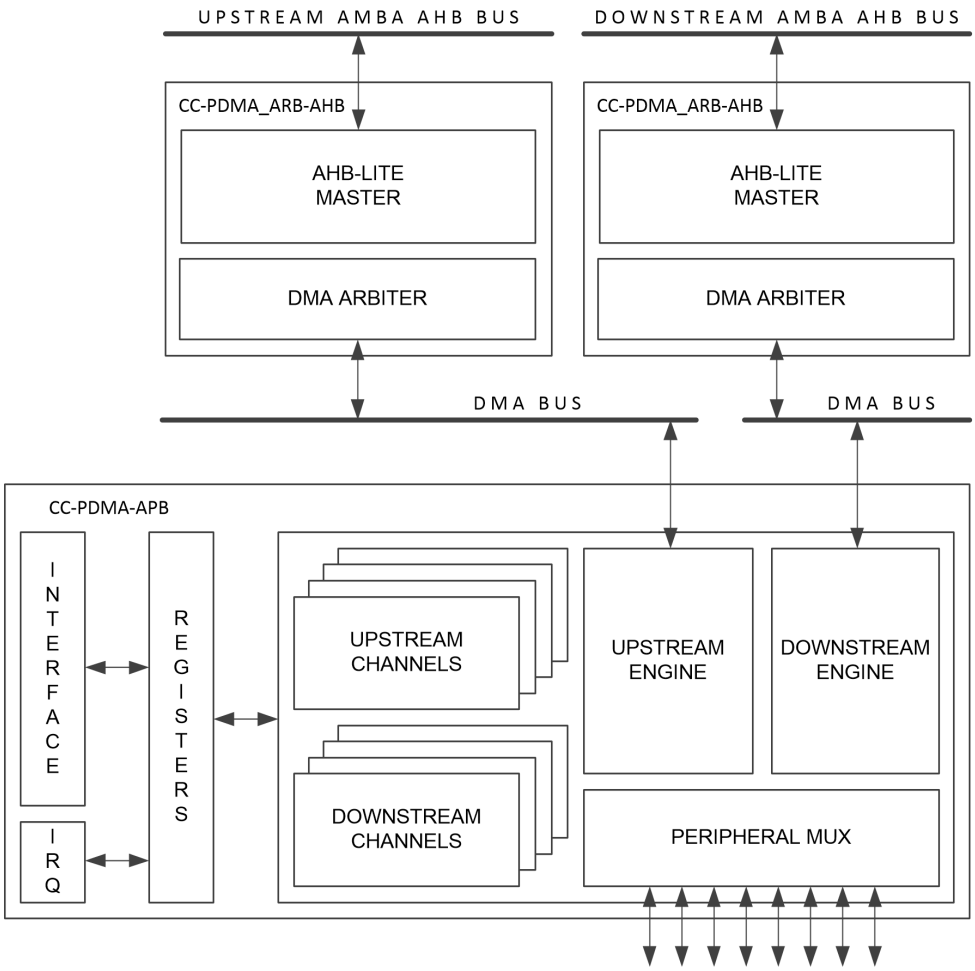

Figure 1.2. Two CC-PDMA_ARB-AHB cores connected to one CC-PDMA-APB core.

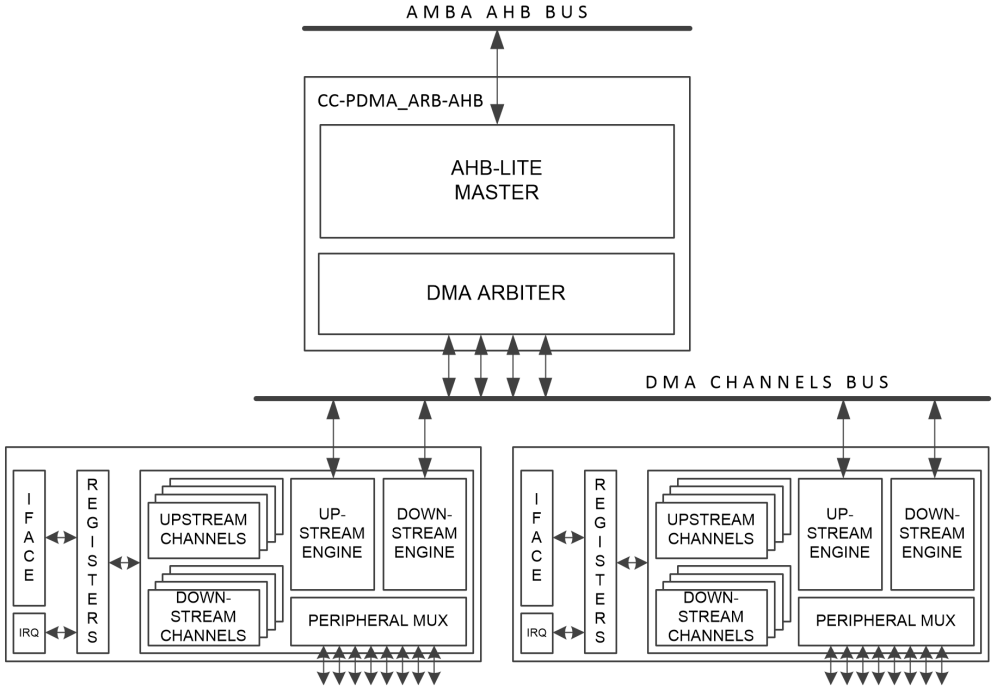

Figure 1.3. One CC-PDMA_ARB-AHB core connected to two CC-PDMA-APB cores.

1.2 Operation

1.2.1 Aligned Access

Each aligned single transfer on DMA bus passes through the internal arbiter and is converted into single transfer on AMBA AHB-Lite port. Channel arbiter works with fixed round-robin priority (Figure 1.4).

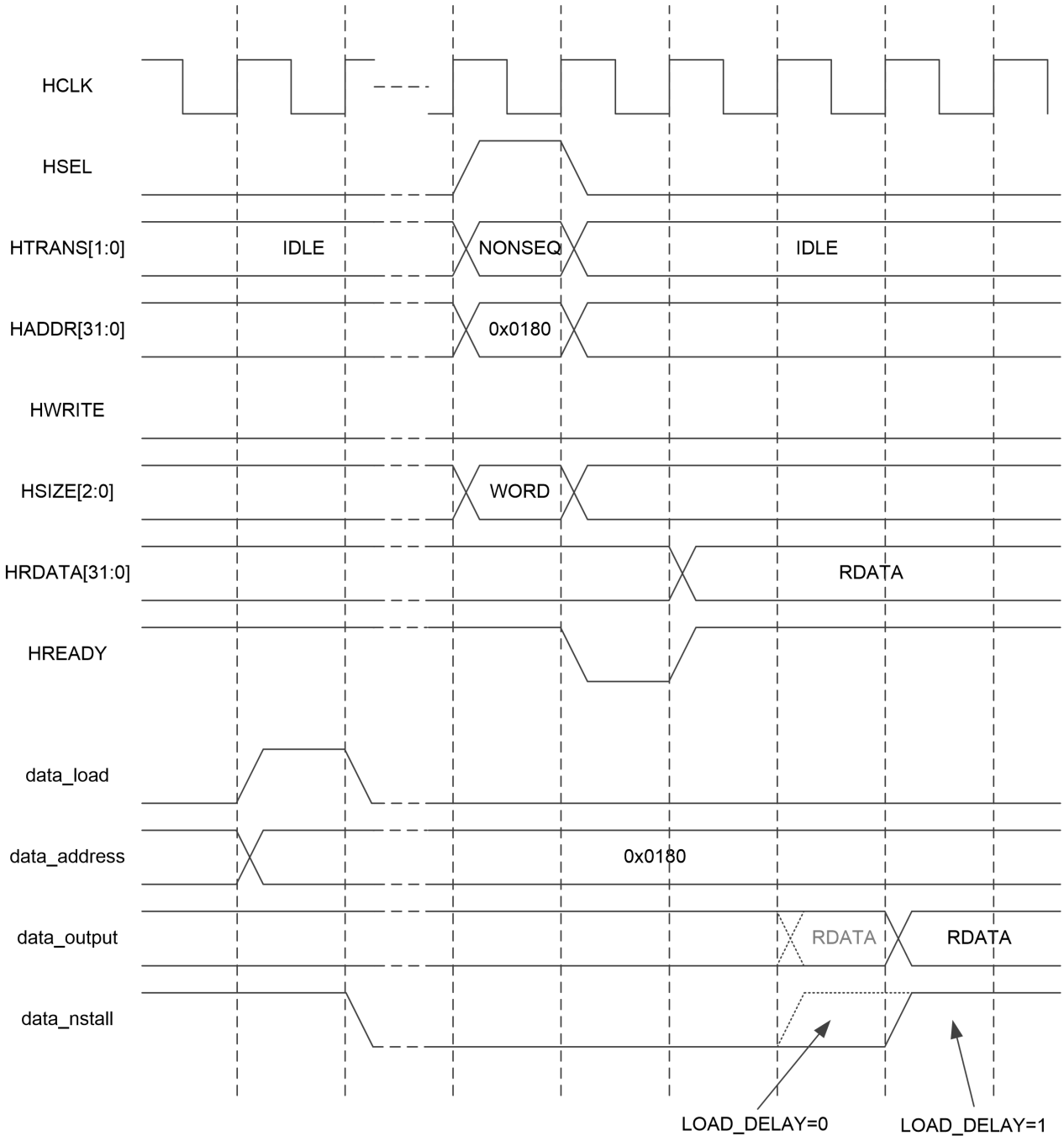

Figure 1.4. CC-PDMA_ARB-AHB single transfer (big-endian byte order).

1.2.2 Unaligned Access

CC-PDMA_ARB-AHB IP core allows CC-PDMA core to perform also unaligned store transfers. In such situation each transfer on the DMA bus is converted into two aligned single transfers on AMBA AHB-Lite port. CC-PDMA core always reads full 32-bit data words (Figure 1.5).

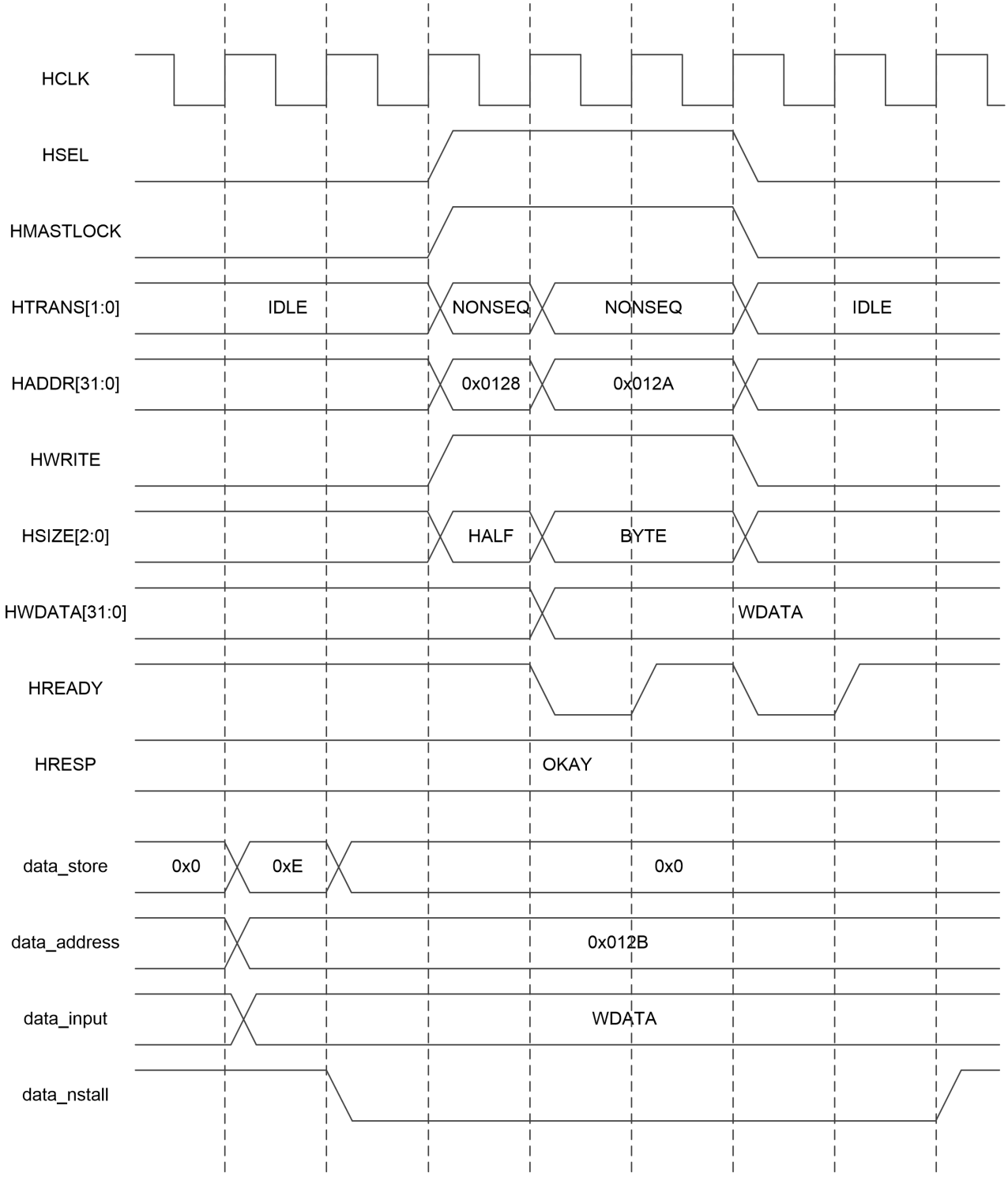

Figure 1.5. CC-PDMA_ARB-AHB unaligned transfer (big-endian byte order).

1.2.3 Error Response

Figure 1.6 shows the ERROR response case on AMBA AHB-Lite port. If ERROR response occurs during the first transfer of unaligned access, the second part will be discarded (Figure 1.6).

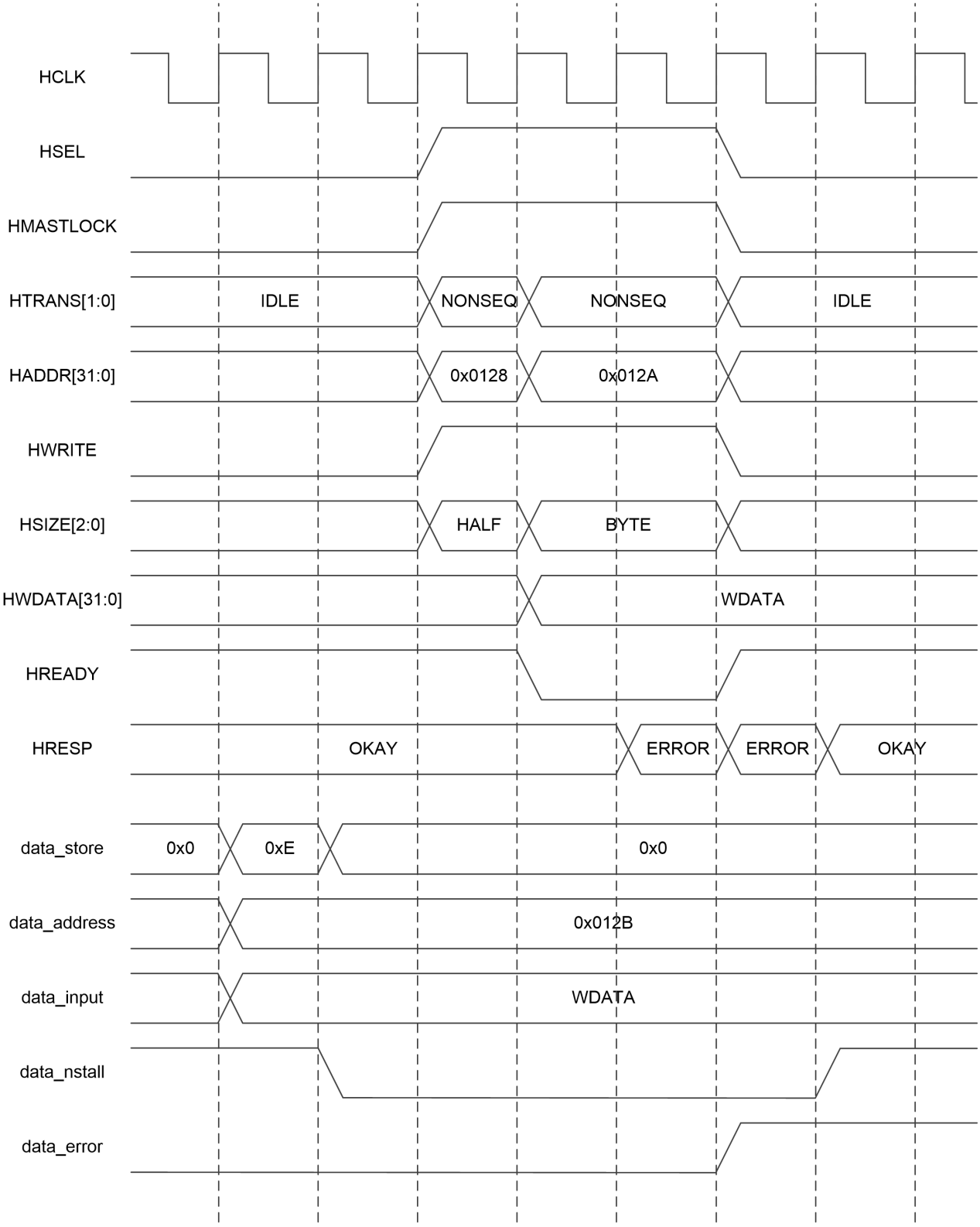

Figure 1.6. CC-PDMA_ARB-AHB unaligned transfer with error response (big-endian byte order).

1.2.4 Clock Scaling

CC-PDMA_ARB-AHB IP core allows the HCLK clock to be integer multiplication of DMA clock.

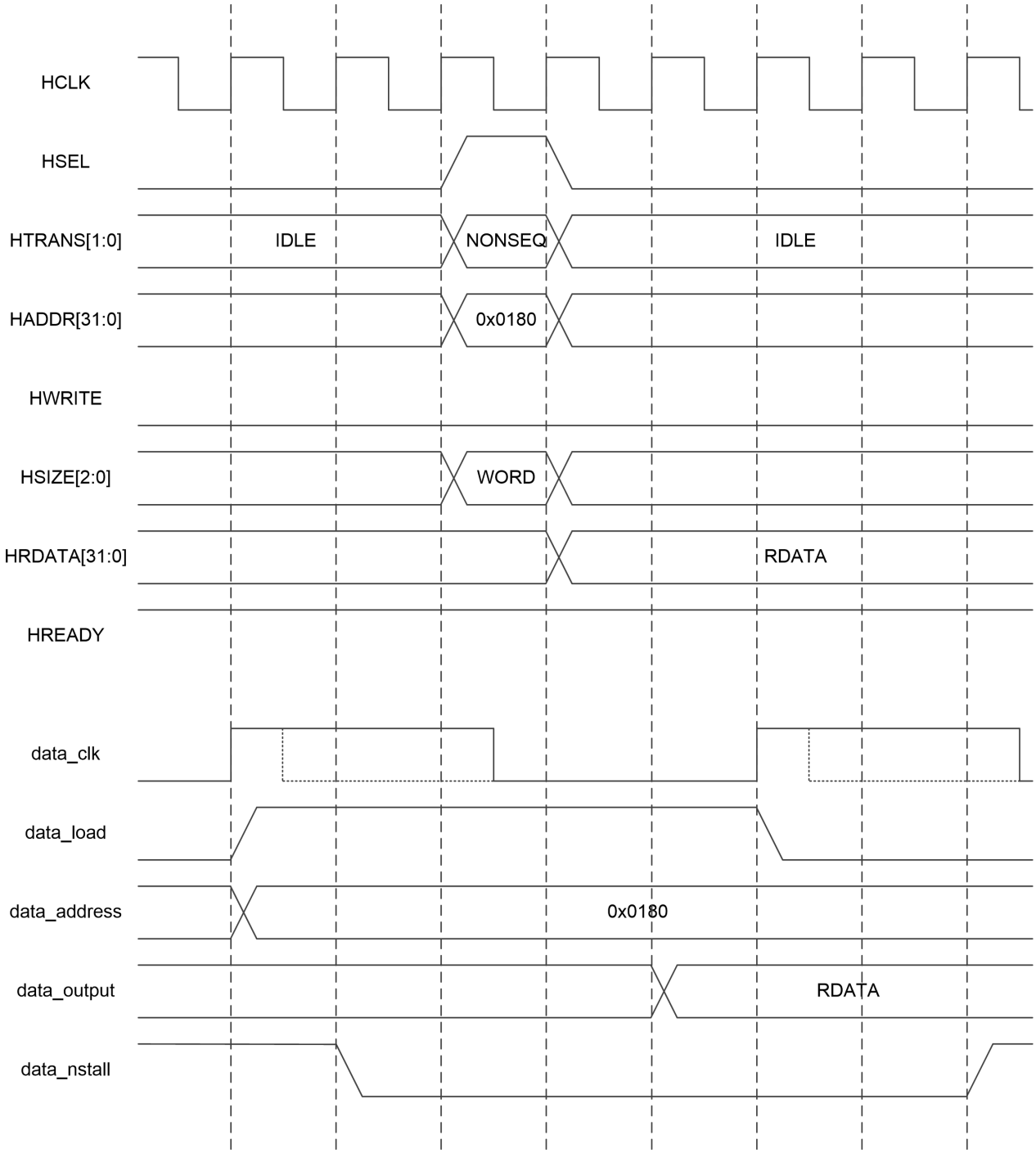

Figure 1.7. CC-PDMA_ARB-AHB with scaled DMA clock (big-endian byte order).

1.3 Implementation

1.3.1 Design Structure

The IP core utilizes Verilog 2005 HDL.

```
UTILS
├── clog2.vh
├── pack_unpack_inout.vh
AHB
├── rtl
    ├── AHB_DMA
        ├── amba_ahb_master_dma_arbiter.v
```

1.3.2 Simulation Flow

CC-PDMA_ARB-AHB IP core is not considered as a stand-alone entity. The self-checking testbench and simulation scripts to verify the correct operation of the IP prior to use in a design are delivered with CC-PDMA-APB IP core.

1.3.3 Clock and Reset

The CC-PDMA_ARB-AHB utilizes a fully synchronous design with positive edge clocking domain and negative asynchronous reset assertion. External reset synchronizer has to be used to ensure synchronous reset deassertion. CC-PDMA_ARB-AHB IP core allows the HCLK clock to be integer multiplication of DMA clock.

1.3.4 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).

1.3.5 Configuration Options

The table below shows the generic parameters of the core.

Generic name	Description	Range	Default
CHANNEL_NUM	Number of DMA channels	1:32	2
LOAD_DELAY	Additional register DMA channel output data	0,1	1
AHB_HPROT	The value passed to the AHB HPROT output	0:15	3
BIG_ENDIAN	Endianness of AHB-Lite port	0,1	1

1.3.6 Signals Description

Signal name	Description	I/O	Active	Type
HCLK	Synchronous clock	I	rising	clock
HRESETn	Asynchronous reset	I	low	reset
HSEL	AHB peripheral select	O	high	reg.
HADDR[31:0]	AHB bus address	O	data	reg.
HWRITE	AHB bus write	O	high	reg.
HSIZE[2:0]	AHB bus transfer size	O	data	reg.
HBURST[2:0]	AHB bus burst type	O	data	reg.
HPROT[3:0]	AHB bus burst protection indication	O	data	reg.
HTRANS[1:0]	AHB bus transfer type	O	data	reg.
HMASTLOCK	AHB bus locked transfer	O	high	reg.
HRDATA[31:0]	AHB bus read data	I	data	comb.
HREADY	AHB bus ready signal	I	high	comb.
HRESP	AHB bus response	I	data	comb.
HWDATA[31:0]	APB bus write data	O	data	reg.
data_load[CHANNEL_NUM-1:0]	DMA bus load	I	high	comb.
data_store[((CHANNEL_NUM)*(4))-1:0]	DMA bus store	I	high	comb.
data_lock[CHANNEL_NUM-1:0]	DMA bus lock	I	high	comb.
data_error[CHANNEL_NUM-1:0]	DMA bus error	O	high	reg.
data_address[((CHANNEL_NUM)*(32))-1:0]	DMA bus address	I	data	comb.
data_input[((CHANNEL_NUM)*(32))-1:0]	DMA bus data input	I	data	comb.
data_output[((CHANNEL_NUM)*(32))-1:0]	DMA bus data output	O	data	reg.
data_nstall[CHANNEL_NUM-1:0]	DMA bus stall signal	O	low	reg.

1.3.7 Instantiation

```
amba_ahb_master_dma_arbiter
              #( .CHANNEL_NUM(2),
                 .LOAD_DELAY(1),
                 .BIG_ENDIAN(1))

 ahb_dma_arbiter_u
               ( .HCLK(HCLK),
                 .HRESETn(HRESETn),
                 .HSEL(HSEL),
                 .HADDR(HADDR),
                 .HWRITE(HWRITE),
```


```
                .HSIZE(HSIZE),
                .HBURST(HBURST),
                .HPROT(HPROT),
                .HTRANS(HTRANS),
                .HMASTLOCK(HMASTLOCK).
                .HRDATA(HRDATA),
                .HREADY(HREADY),
                .HRESP(HRESP),
                .HWDATA(HWDATA),
                .data_load(    {1'b0,
                                downstream_hsi_load}),
                .data_store(   {upstream_hsi_store,
                                4'd0}),
                .data_address( {upstream_hsi_address,
                                downstream_hsi_address}),
                .data_input(   {upstream_hsi_data_out,
                                32'd0}),
                .data_output(  {upstream_hsi_data_in,
                                downstream_hsi_data_in}),
                .data_nstall(  {upstream_hsi_nstall,
                                downstream_hsi_nstall}),
                .data_lock(    {upstream_hsi_lock,
                                downstream_hsi_lock}),
                .data_error(   {upstream_hsi_error,
                                downstream_hsi_error}));

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_aggregation_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(pdma_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));
```

1.4 Revision History

Doc. Rev.	Date	Comments
1.1	08-2017	Added 1.1.2 Use Cases section.
1.0	05-2017	First Issue.

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

©2017 ChipCraft Sp. z o.o. CC-PDMA_ARB-AHB-Doc_082017.

ChipCraft®, ChipCraft logo and combination of thereof are registered trademarks or trademarks of ChipCraft Sp. z o.o. All other names are the property of their respective owners.

Disclaimer: ChipCraft makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. ChipCraft does not make any commitment to update the information contained herein.