Graywolf BIU Design Specification

Document Number	DSP-00094
Date Issued	
Status	In-progress

Revision History

Rev.	Revision Date	Revised Chapter-Section	Revised Content	Author
0.4	2016-03-17	CH3	1. add/reuse the buffers to hold the return information when read data FIFO is full 2. the hllsc_req is only asserted at the valid address phase for LLW & SCW 3. enhance the flow chart of 7-byte access	Ruei-Yuan
0.3	2016-02-18	CH2,CH3	1. Add the descriptions for the timing diagram 2. Add the timing diagram for back to back read case 3. Add the descriptions for the FIFO depth in BIU_SYNC module ~~3.~~4. Support the AHB bus	Ruei-Yuan
0.2	2016-02-05	CH1,CH2	1. Modify the acknowledge signal definition rule 2. Add the timing diagram for read and write transaction	Ruei-Yuan
0.1	2016-01-20	All	Initial Draft	Ruei-Yuan

格式化: 格線被設定時，不要調整右側縮排, 編號 + 階層: 1 + 編號樣式: 1, 2, 3, … + 起始號碼: 1 + 對齊方式: 左 + 對齊: 0 公分 + 縮排: 0.63 公分, 文字對齊方式: 基線

Review History				
Rev.	Review Date	Reviewed Chapter-Section	Comments	Reviewer
0.3	2016-03-16	CH3	1. add/reuse the buffers to hold the return information when read data FIFO is full 2. the hllsc_req is only asserted at the valid address phase for LLW & SCW 3. enhance the flow chart of 7-byte access	Ethan, Justin, Osban, Taco, Wolfson, Yung-Ching
0.2	2016-02-17	CH2, CH3	1. Add the descriptions for the timing diagram 2. Add the timing diagram for back to back read case 3. Add the descriptions for the FIFO depth in BIU_SYNC module	Ding-Kai, Edward, Ethan, Osban, Taco, Wolfson
0.1	2016-02-03	CH1,CH2	1. Modify the acknowledge signal naming rule 2. Add the timing diagram for read and write transaction	Alice, Ding-Kai, Ethan, Feng, Jonathan, Justin, Osban, Taco, Wolfson, Yung-Ching

格式化: 編號 + 階層: 1 + 編號樣式: 1, 2, 3, … + 起始號碼: 1 + 對齊方式: 左 + 對齊: 0 公分 + 縮排: 0.63 公分

Document Sign-off List			
Department	Name	Date	Comment
VLSI		2016-mm-dd	

Table of Contents

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化 ... [1]

格式化 ... [2]

格式化 ... [3]

格式化 ... [4]

格式化 ... [5]

格式化 ... [6]

格式化 ... [7]

格式化 ... [8]

格式化 ... [9]

格式化 ... [10]

格式化 ... [11]

格式化 ... [12]

格式化 ... [13]

格式化 ... [14]

格式化 ... [15]

格式化 ... [16]

格式化 ... [17]

格式化 ... [18]

格式化 ... [19]

格式化 ... [20]

格式化 ... [21]

格式化 ... [22]

格式化 ... [23]

List of Tables

List of Figures

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

格式化: 預設段落字型, 使用拼字與文法檢查

Typographical Convention Index

Document Element	Font	Font Style	Size	Color
Normal	Georgia	Normal	12	Black
`Code`	`Lucida Console`	`Normal`	`11`	`Indigo`
`USER VARIABLE`	`Lucida Console`	`ALL-CAPS`	`11`	`INDIGO`
Note/Warning	Georgia	Normal	12	Red
Hyperlink	**Georgia**	**Bold + Underlined**	**12**	**Blue**

1. Overview

1.1. Introduction

Bus Interface Unit (BIU) is designed for transferring data between the processor core and the system bus. In the core, there are five modules which send requests to BIU: FCU, LSU, MMU, LDMA, and EDM.

1.2. Features

- Support 64-bit AMBA® AXI4™ interface
- Support 64-bit AMBA® AHB2™ interface
- Support the wait state for read data return when the LSU sends the wait state to BIU, BIU needs to hold the read data until the wait state is released

格式化: 字型: (中文) Georgia

1.3. Block Diagram

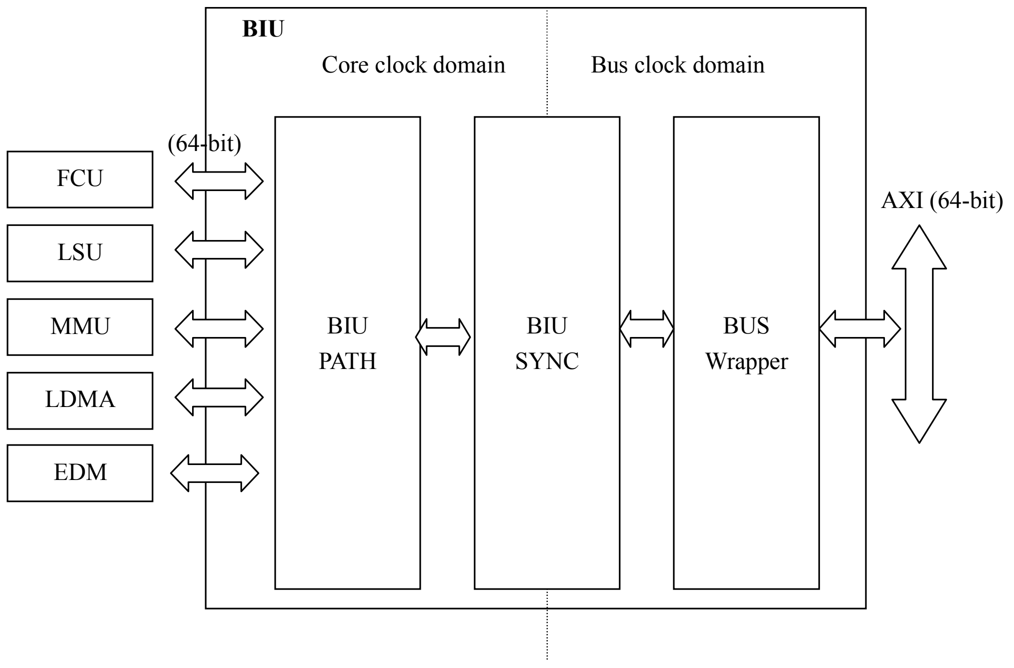


Figure 1. Block Diagram

1.4. Function Description

1.4.1. BIU PATH

Allocate the priority of requests for read and write command.

Sends or receives the command or data from/to BIU_SYNC.

1.4.2. BIU SYNC

Synchronize the command and data between core clock domain and bus clock domain.

1.4.3. BUS Wrapper

Transfer the data from/to the bus.

2. Signal Description

2.1. Signal Definition

The rule of acknowledge signal definition is as following descriptions:

xxx_yyy_read_ack (read data phase ack)

xxx_yyy_write_ack (write data phase ack)

xxx_yyy_accept_write_ack (accept write data ack)

xxx: source, yyy: destination

Table 1. Signal Definition

Signal Name	I/O Type	Description
FCU signals		
fcu_biu_addr[`NDS_ADDR_MSB:0]	I	The FCU request address
fcu_biu_length[1:0]	I	2'b00: single, 2'b01: 2 2'b10: 4, 4'b11: 8
fcu_biu_size[1:0]	I	2'b00: byte. 2'b01: halfword. 2'b10: word. 2'b11: double word(for cache line)
fcu_biu_cacheability[2:0]	I	FCU read cacheability; [2]: wt; [1]: noncache; [0]: nonbuf;
fcu_biu_req	I	FCU request to BIU
fcu_biu_req_kill	I	FCU kill request to BIU
biu_fcu_read_error	O	An error happens in read transactions
biu_fcu_read_ack	O	Read data acknowledge
biu_fcu_grant	O	Address grant
biu_fcu_last	O	Indicate the last read data in read transactions
biu_fcu_rdata[63:0]	O	Read data from bus
LSU signals		

Signal Name	I/O Type	Description
lsu_biu_addr[`NDS_ADDR_ MSB:0~~2~~]	I	LSU request address
lsu_biu_write	I	1'b0: read. 1'b1: write
lsu_biu_isync	I	Current instruction is "isync". BIU should block request from FCU
lsu_biu_length[1:0]	I	2'b00: single, 2'b01: 2 2'b10: 4, 4'b11: 8
lsu_biu_size[1:0]	I	2'b00: byte. 2'b01: halfword. 2'b10: word. 2'b11: double word(for cache line)
lsu_biu_lock	I	Indicate this transaction is an exclusive access
lsu_biu_lock_clear	I	Indicate the exclusive access fails in core
lsu_biu_msync	I	Current instruction is "msync". BIU should block the request from LDMA and EDM
lsu_biu_cacheability[2:0]	I	LSU read cacheability; [2]: wt; [1]: noncache; [0]: nonbuf;
lsu_biu_rdata_wait	I	Read data wait to BIU
lsu_biu_req	I	LSU request to BIU
lsu_biu_req_kill	I	The request is killed
lsu_biu_wb_addr[`NDS_AD DR_MSB:~~2~~0]	I	LSU wb request address
lsu_biu_wb_bwe[7:0]	I	LSU wb byte write enable
lsu_biu_wb_last	I	LSU wb last data
lsu_biu_wb_length[1:0]	I	2'b00: single, 2'b01: 2 2'b10: 4, 4'b11: 8
lsu_biu_wb_size[1:0]	I	2'b00: byte. 2'b01: halfword. 2'b10: word. 2'b11: double word(for cache line) Note: if the lsu_biu_wb_bwe is 0x07 (3-byte access), the lsu_biu_wb_size should be greater than the access bytes (in this case is word).
lsu_biu_wb_cacheability[2:0]	I	LSU wb write cacheability; [2]: wt; [1]: noncache; [0]: nonbuf;
lsu_biu_wb_req	I	LSU wb request
lsu_biu_wb_wdata[63:0]	I	LSU wb write data

Signal Name	I/O Type	Description
lsu_biu_wdata[63:0]	I	Write data
lsu_biu_wb_dbgacc	I	LSU wb debug access
biu_lsu_grant	O	Address grant
biu_lsu_lock_fail	O	The exclusive access is failed
biu_lsu_rdata[63:0]	O	Read data
biu_lsu_wb_grant	O	The LSU wb address grant
biu_lsu_wdata_fifo_empty	O	Indicate the write data FIFO in BIU is empty
biu_lsu_write_error	O	An error happens in the write transaction (imprecise signal)
biu_lsu_read_error	O	An error happens in the read transactions
biu_lsu_read_ack	O	Read data acknowledge
biu_lsu_write_ack	O	Write data phase acknowledge
biu_lsu_wb_accept_write_ack	O	Write data address acknowledge
MMU signals		
mmu_biu_addr[`NDS_ADDR_MSB~~:~~:0~~2~~]	I	MMU request address
mmu_biu_length[1:0]	I	2’b00: single, 2’b01: 2 2’b10: 4, 4’b11: 8
mmu_biu_size[1:0]	I	2’b00: byte. 2’b01: halfword. 2’b10: word. 2’b11: double word(for cache line)
mmu_biu_req	I	MMU read request to BIU
mmu_biu_req_kill	I	MMU Kill request to BIU
biu_mmu_read_error	O	An error happens in the read transactions
biu_mmu_read_ack	O	Read data acknowledge
biu_mmu_grant	O	Address grant
biu_mmu_rdata[63:0]	O	Read data from bus
LDMA signals		

Signal Name	I/O Type	Description
ldma_biu_addr[`NDS_ADDR_MSB:0~~2~~]	I	LDMA request address
ldma_biu_bwe[7:0]	I	Byte enable
ldma_biu_write	I	1'b0: read. 1'b1: write
ldma_biu_length[1:0]	I	2'b00: single word, 2'b01: 2 words 2'b10: 4 words, 4'b11: 8 words
ldma_biu_size[1:0]	I	2'b00: byte. 2'b01: halfword. 2'b10: word. 2'b11: double word(for cache line) Note: if the ldma_biu_bwe is 0x07 (3-byte access), the ldma_biu_size should be greater than the access bytes (in this case is word).
ldma_biu_cacheability[2:0]	I	LDMA data cacheability; [2]: wt; [1]: noncache; [0]: nonbuf;
ldma_biu_req	I	LDMA request to BIU
ldma_biu_wdata[63:0]	I	LDMA Write data to BIU
ldma_biu_wlast	I	The last write data to BIU
biu_ldma_read_error	O	An error happens in the read transactions
biu_ldma_write_error	O	An error happens in the write transactions
biu_ldma_read_ack	O	Read data phase acknowledge
biu_ldma_write_ack	O	Write data phase acknowledge
biu_ldma_accept_write_ack	O	Write data address acknowledge
biu_ldma_grant	O	Address grant
biu_ldma_rdata[63:0]	O	Read data from bus
EDM signals		
edm_biu_addr[`NDS_ADDR_MSB:0]	I	EDM request address
edm_biu_bwe[7:0]	I	Byte enable
edm_biu_size[1:0]	I	2'b00: byte. 2'b01: halfword. 2'b10: word. 2'b11: double word(no use)
edm_biu_write	I	1'b0: read. 1'b1: write

Signal Name	I/O Type	Description
edm_biu_req	I	EDM request to BIU
edm_biu_wdata[63:0]	I	EDM write data to BIU
biu_edm_read_ack	O	Read data acknowledge
biu_edm_accept_write_ack	O	Write data address acknowledge
biu_edm_grant	O	Address grant
biu_edm_rdata[63:0]	O	Read data from bus
AXI write address channel signals		
awid[3:0]	I	Write address ID tag
awaddr[31:0]	I	Write Address
awlen[7:0]	I	Write Length
awsize[2:0]	I	Write burst size
awburst[1:0]	I	Write burst type
Awlock	I	Write atomic access
awcache[3:0]	I	Cache type
awprot[2:0]	I	Protection type
awvalid	I	Write address valid
awready	O	Write address ready
AXI write data channel signals		
wdata[63:0]	I	Write data bus
wstrb[7:0]	I	Write enable
wlast	I	Last write in a burst
wvalid	I	Write data valid
wready	O	Write data ready
AXI write response channel signals		
bid[3:0]	O	Write Response ID tag

Signal Name	I/O Type	Description
bresp[1:0]	O	Write response status
bvalid	O	Write response valid
bready	I	Write response ready
AXI read address channel signals		
arid[3:0]	I	Read address ID tag
araddr[31:0]	I	Read address
arlen[7:0]	I	Read length
arsize[2:0]	I	Read burst size
arburst[1:0]	I	Read burst type
arlock	I	Read atomic access
arcache[3:0]	I	Cache type
arprot[2:0]	I	Protection type
arvalid	I	Read address valid
arready	O	Read address ready
AXI read data channel signals		
rid[3:0]	O	Read ID tag
rdata[63:0]	O	Read data bus
rresp[1:0]	O	Read response
rlast	O	Last read in a burst
rvalid	O	Read data valid
rready	I	Read data ready
Performance monitor signals		
event_dma_biu_cycle	O	LDMA access cycles
event_dma_biu_request	O	LDMA request
event_external_event	O	1'b0

Signal Name	I/O Type	Description
event_hptwk_biu_cycle	O	HPTWK access cycles
event_hptwk_biu_request	O	HPTWK request
event_icache_fill_biu_cycle	O	FCU fill data cycles
event_icache_fill_biu_reques t	O	FCU fill request
event_lsu_biu_cycle	O	LSU access cycles
event_lsu_biu_request	O	LSU request
Other signals		
scan_enable	I	Scan enable
ucore_standby_stall_wait	I	Standby request
ice_debug_session	I	debug access
reg_psw_pom	I	Privileged mode
aclk	I	AXI bus clock
aresetn	I	AXI bus reset
core_clk	I	Core clock
core_reset_n	I	Core reset
cpu_estrb	I	Clock enable from BUS to CPU
sync_mode	I	SYNC or ASYNC mode
biu_ucore_standby_ready	O	BIU Standby ready
biu_pcu_ready_to_gck	O	BIU to PCU ready to gck
ardebug_access	O	Read debug access
awdebug_access	O	Write debug access
hclk	I	AHB bus system clock
hreset_n	I	AHB bus reset
AHB general signals		
haddr[`NDS_ADDR_MSB:0]	O	AHB system address bus

Signal Name	I/O Type	Description
hburst [2:0]	O	Burst transfer indication signals
hprot [3:0]	O	Transfer protection control signals
hsize [2:0]	O	Size of the transfer
htrans [1:0]	O	AHB transfer type
hwrite	O	AHB read/write transfer signal
hwdata[63:0]	O	AHB write data
hready	I	AHB bus ready
hgrant	I	AHB bus grant
hrdata [63:0]	I	AHB read data
hlock	O	AHB lock bus
hbusreq	O	AHB bus request
hresp[1:0]	O	AHB response
AHB sideband signals		
hllsc_req	O	LLW & SCW request
hllsc_error	I	LLW & SCW status

格式化: 靠左

格式化: 靠右

2.2. Timing Diagram for Interface Protocol

~~Note~~Assumption:~~–~~

~~Assume the t~~Two bubbles are always encountered for the data access cycles. The bubbles are introduced by the ~~if BUS has zero wait cycle. One of the bubbles is~~ command FIFO ~~delay~~ and the ~~other is~~ read data FIFO ~~delay~~.

格式化: 字型: (中文) +本文中文字型 (新細明體)

2.2.1. Read transactions

格式化: 字型: (中文) +本文中文字型 (新細明體)

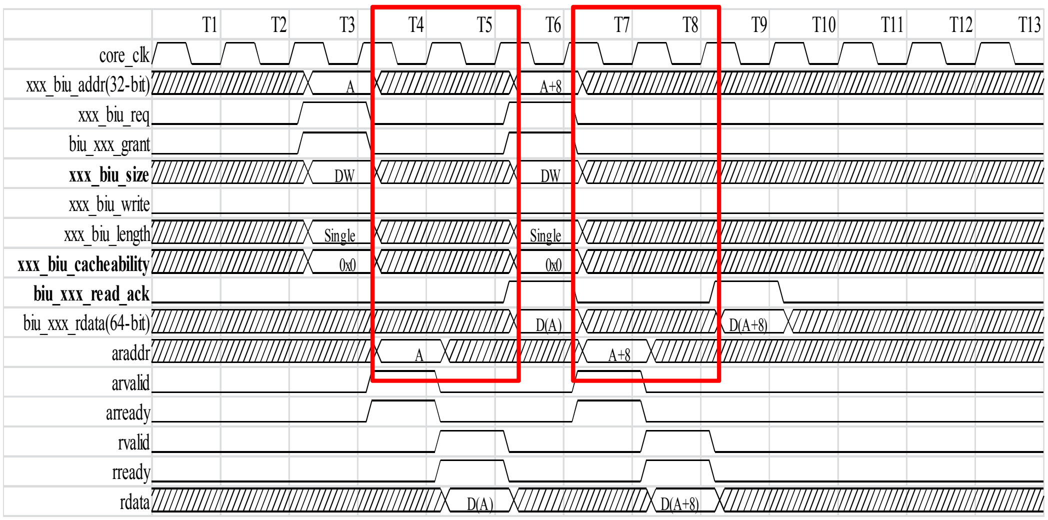


Figure 2. The waveform of single read data transaction without wait cycles

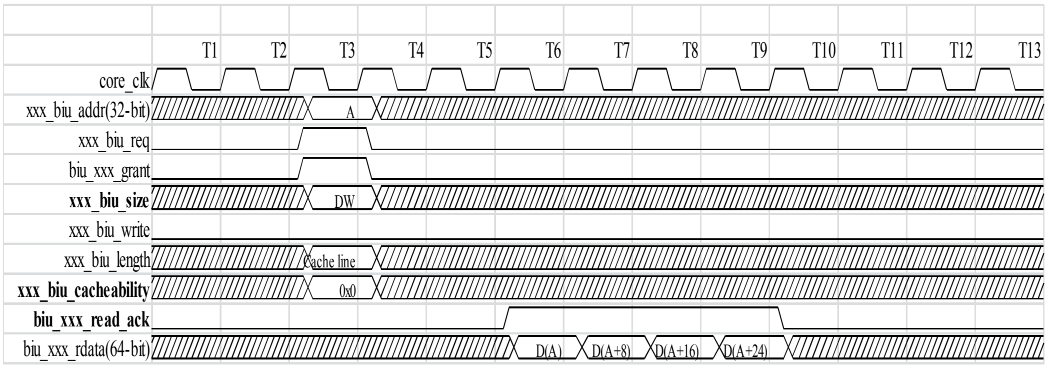


Figure 3. The waveform of burst read transaction without wait cycles

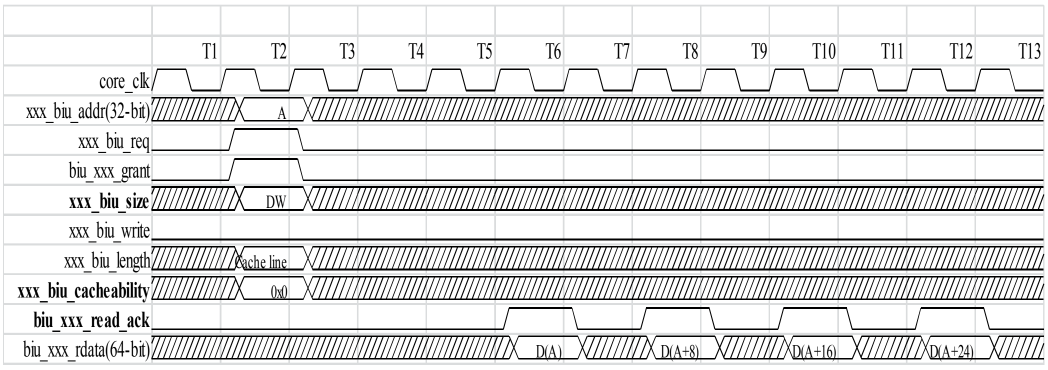


Figure 4. The waveform of burst read data transaction with wait cycles

Figure 4 錯誤! 找不到參照來源。~~Figure 4~~ shows the read data with the one bus wait cycle.

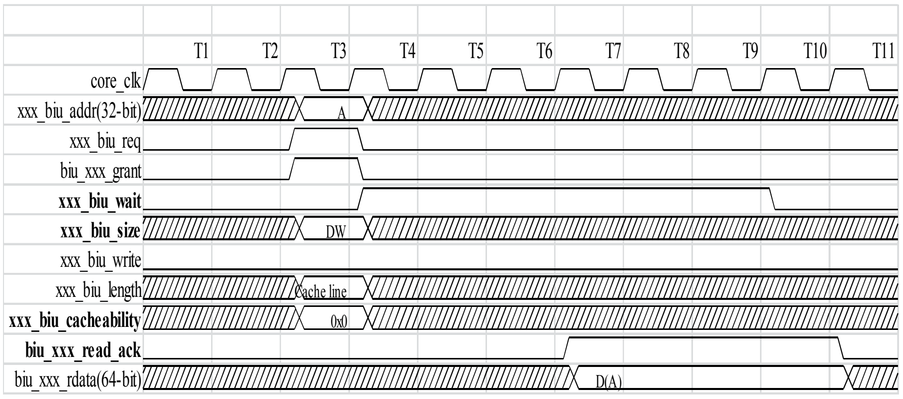


Figure 5. The waveform of read data return with the wait state

Figure 5~~Figure 5~~ shows the return data encounters the wait state at T7. During T8 ~ T10, BIU needs to hold the return data until the wait state is released (T10). This feature is only supported for LSU.

2.2.2. Write transactions

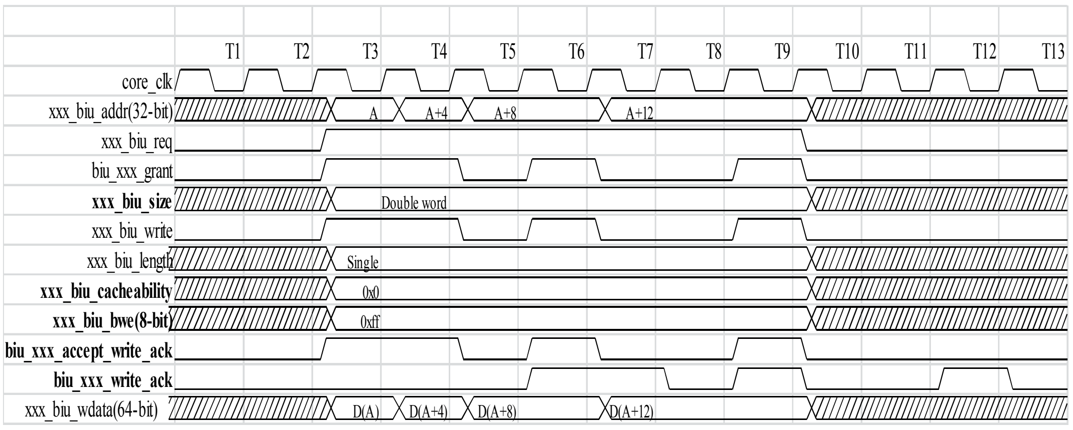


Figure 6. The waveform of single write transaction without the wait cycles

Figure 6~~Figure 6~~ shows the sequential write transactions with the non-sequential grants. At T5, the write request is asserted but the write command FIFO is full, therefore, the write request is not granted until the command FIFO not full (T6).

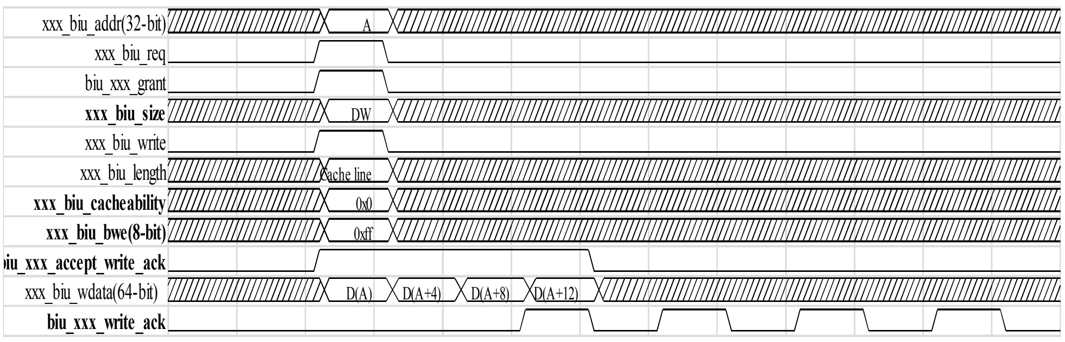


Figure 7. The waveform of burst write transaction with the wait cycles

Figure 7~~Figure 7~~ shows the write data with the one bus wait cycle.

2.2.3. MISC

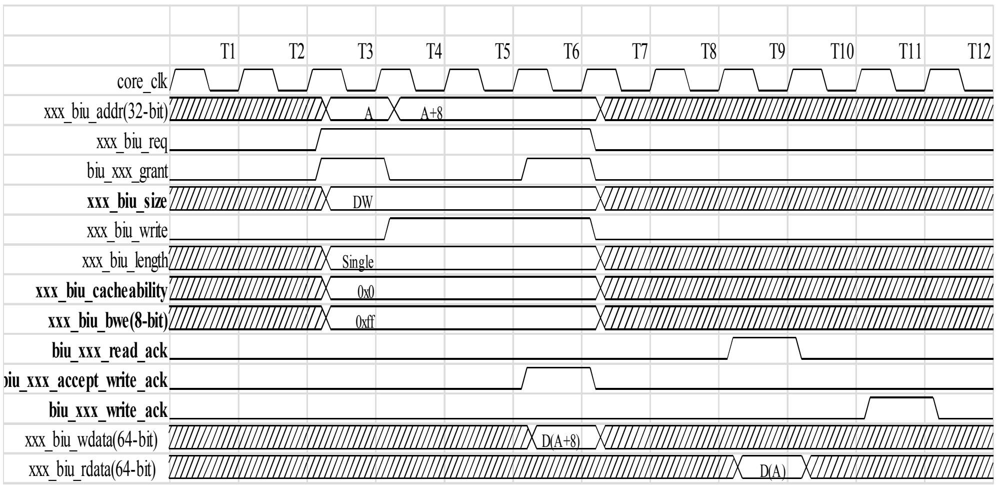


Figure 8. The waveform of two sequential read-write transactions with non-sequential grant and ack

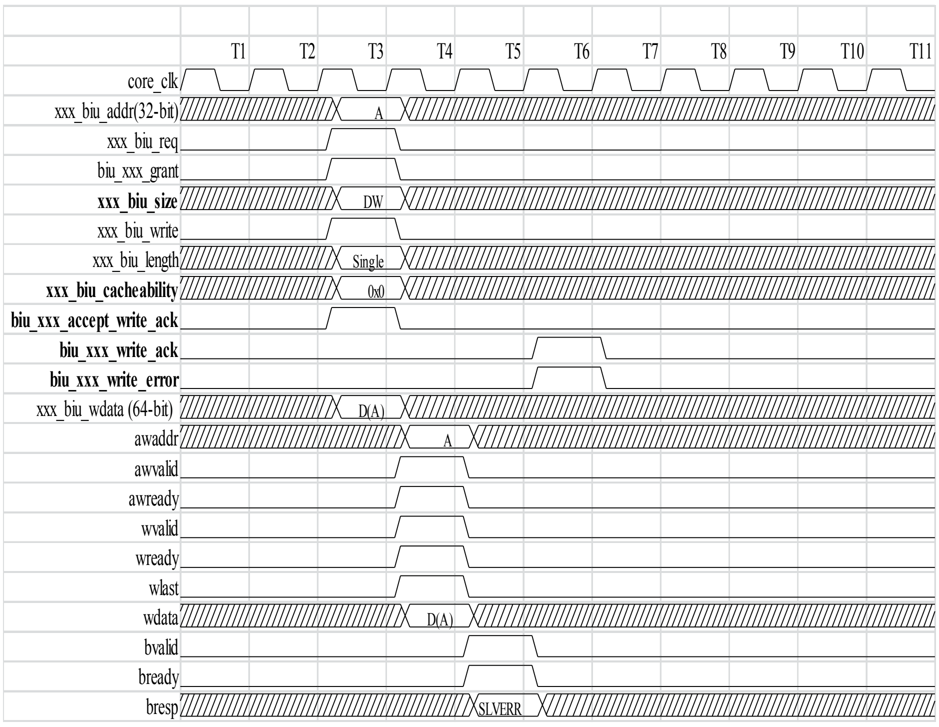


Figure 9. The waveform of write transaction with the bus error

Figure 9~~Figure 10~~ shows the write transaction with the bus error. The bus error is injected at T5, then, the error is received at T6.

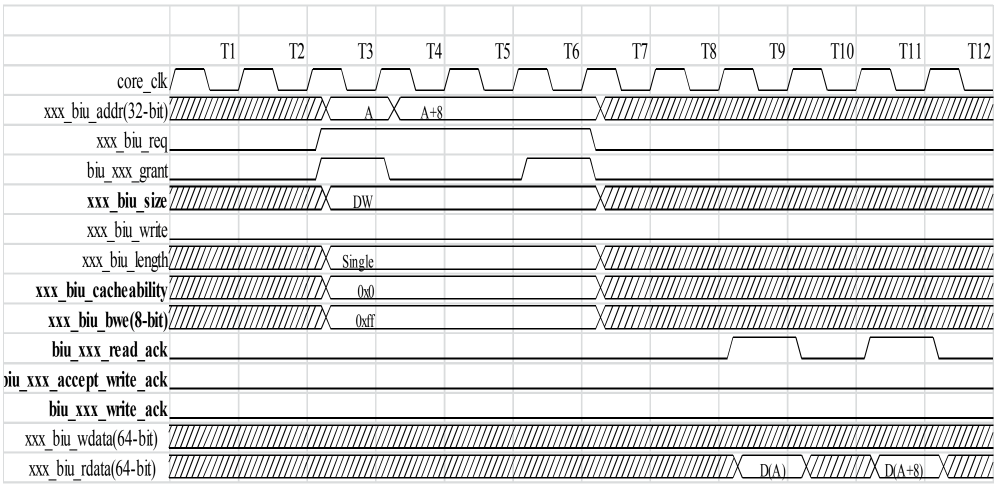


Figure 10. The waveform of two sequential read transaction with non-sequential ack and grant

3. Micro-Architecture

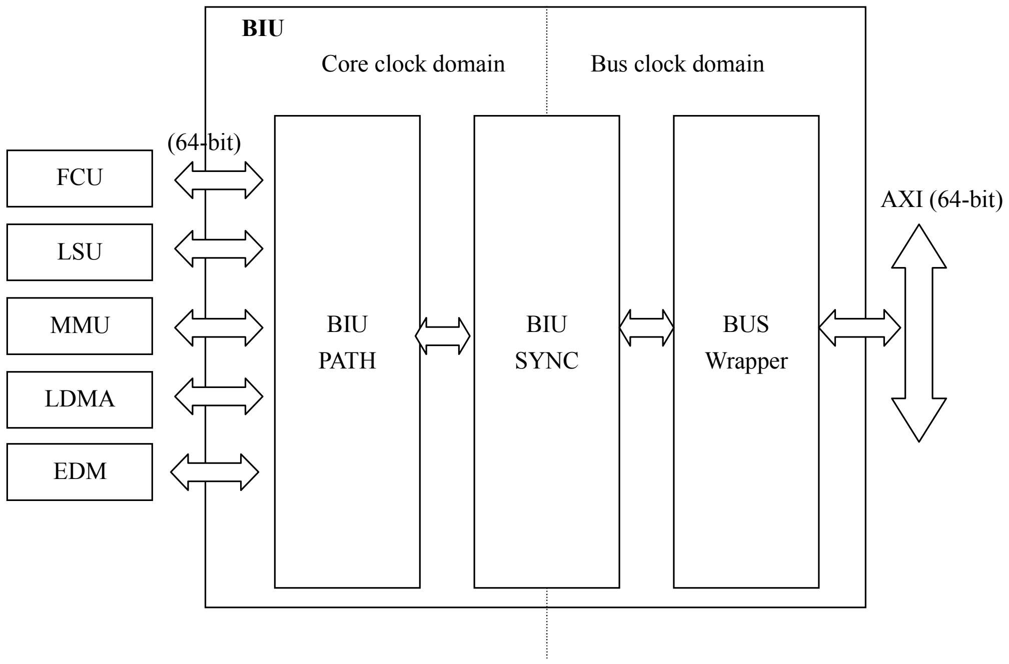


Figure 11. The block diagram of BIU

3.1. BIU_PATH

- The biu_path is shown in Figure 12. According to the arbitration result, the multiplex select the address and control. When an address phase is completed, BIU asserts the enable of selection register. There are five modules request to BIU. They are FCU, LSU, MMU, LDMA, and EDM. The priority of arbitration is LSU > MMU > FCU > EDM > LDMA. If the hold_arbitration is asserted, the arbiter will hold the grant. The hold_arbitration signal prevents the request signal changes before BIU returns ACK. Support the wait state when the LSU sends the wait state to BIU, BIU needs to hold the read data until the wait state is released.

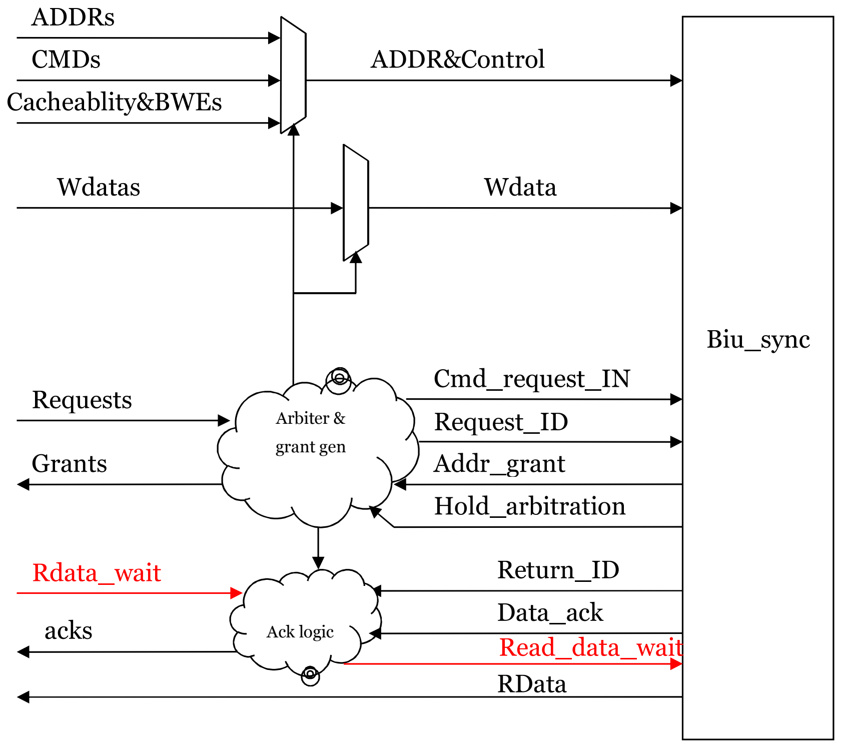


Figure 12. The block diagram of the biu_path

3.2. BIU_SYNC

The biu_sync uses three sync fifos to handle the cross-clock domain signals. The three fifos are used to store the command and write date from core clock to bus clock and read data from bus clock to core clock. If the Read_data_wait is asserted, the data is held in read data fifo until the Read_data_wait is deasserted.

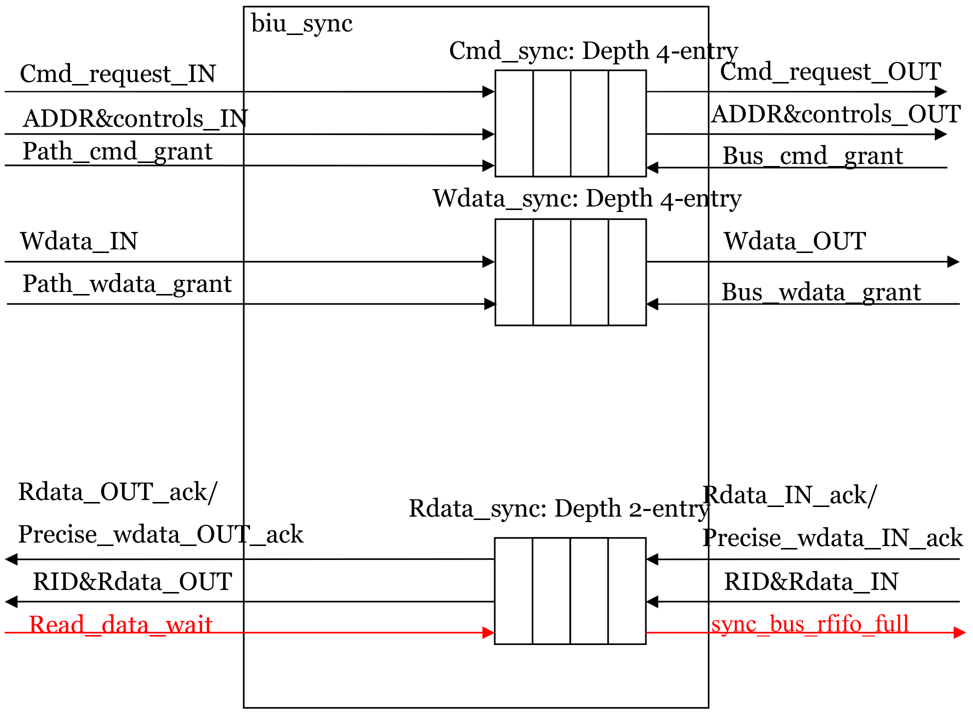


Figure 13 the data path in synchronization module

3.3. AXI bus wrapper

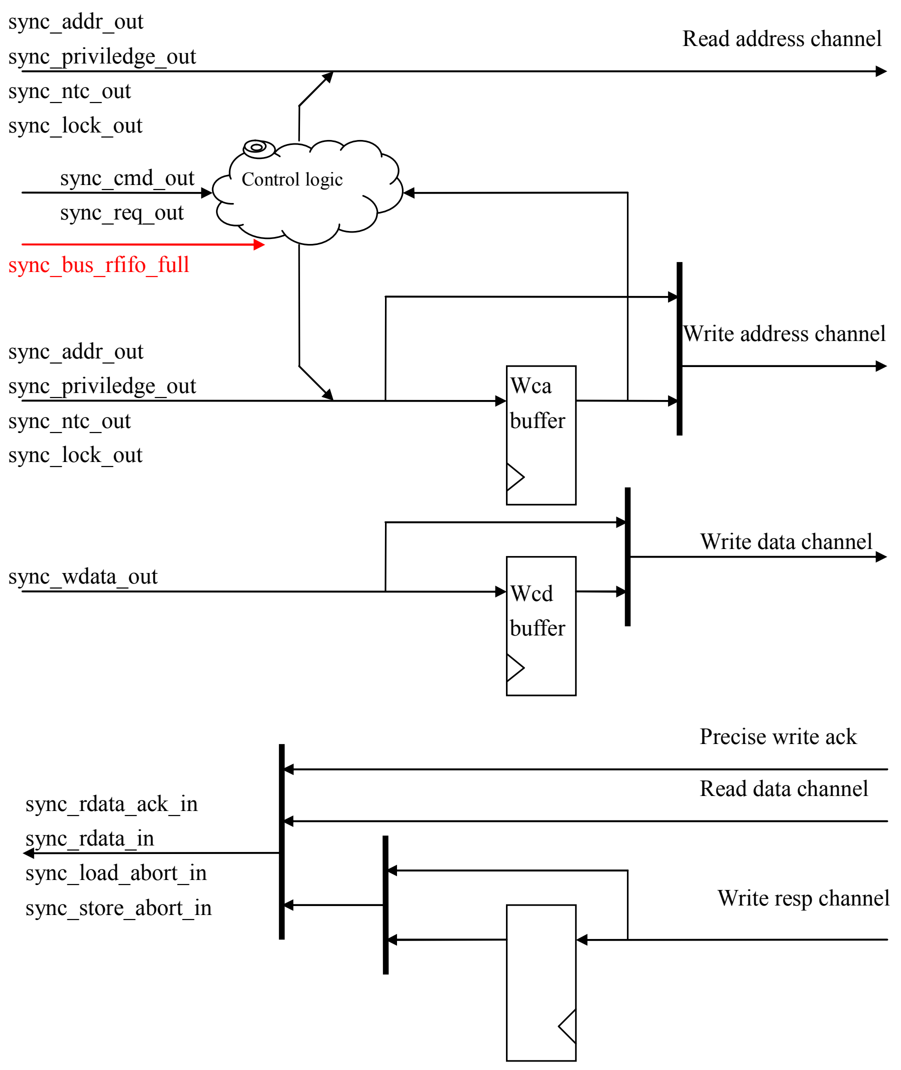


Figure 14. AXI wrapper block diagram

3.4. Write Transfer

There are three channels used in the write transfer. They are the write address channel, the write data channel and the write response channel. If the write transfer in the write address channel and the write data channel is not completed in one cycle, the AXI wrapper read the write transfer from sync queue and stores it into a buffer. Thus, the uncompleted write transfer does not block the next read transfer.

When the AXI wrapper read a write transfer from sync fifo, it increases the wc_count by 1. The wc_count means the uncompleted write transfer count. When the AXI wrapper receives a write channel response, it decreases the wc_count by 1. When the wc_count is 3'b110, the AXI wrapper stops the write transfer.

The biu_axi_wrapper only supports 2/4/8 wrap burst transfer. When the command is write burst, the biu_axi_wrapper uses the wca buffer to store the information for write burst. When the biu_axi_wrapper transfers write wrap data, it blocks the next write address transfer in the write address channel. If the sync_bus_rfifo_full is asserted, the wrapper will not receive the any write response.

3.5. Read Transfer

When the AXI complete a read transfer, it read the read transfer from the sync fifo. The read transfer addess, control signals directly connects with the sync fifo. When the AXI wrapper read a read transfer from the sync fifo, it increases the rc_count by 1. The rc_count means the uncompleted read transfer count. When the rc_count is 3'b111, the AXI wrapper stops the read transfer.

When the read address match the uncompleted write transfer in the wca buffer, the AXI wrapper stops the read transfer.

When the bus error happens in the read wrap transfer, the biu_axi_wrapper only returns the first bus error to BIU_SYNC. Then it blocks the read response with the same RID until completing the last read transaction. If the sync_bus_rfifo_full is asserted, the wrapper will not receive the any read data.

3.6. Lock Transfer

The AXI protocol can support exclusive access. The AXI wrapper does not transfer the write lock transfer until the wc_count is zero. When the AXI wrapper is waiting for the write lock transfer response, it does not transfer other write transfer.

3.7. Write Acknowledge

When the write transfer needs to return acknowledge to core, the sync_bus_write_ack is asserted. In the biu_axi_wrapper, there is a queue to store the information which the write channel response should acknowledge to core. Figure 15 shows the bid_wirte_ack logic. When BIU_AXI wrapper receives a write response, it look up the queue to check this write response return an acknowledge to core.

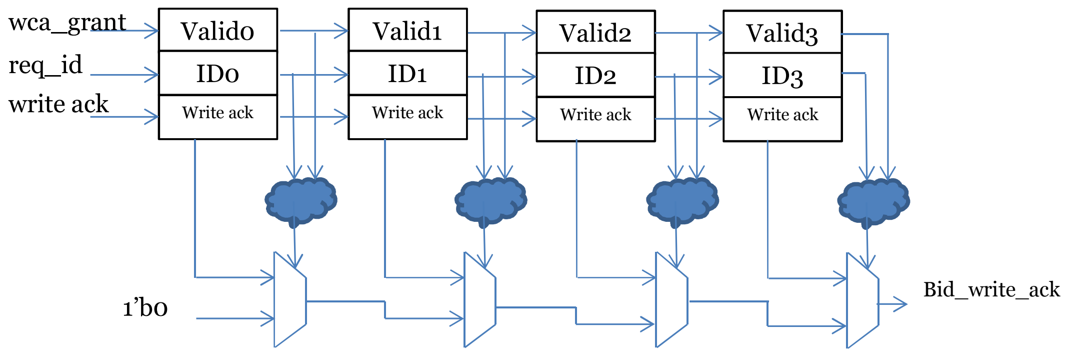


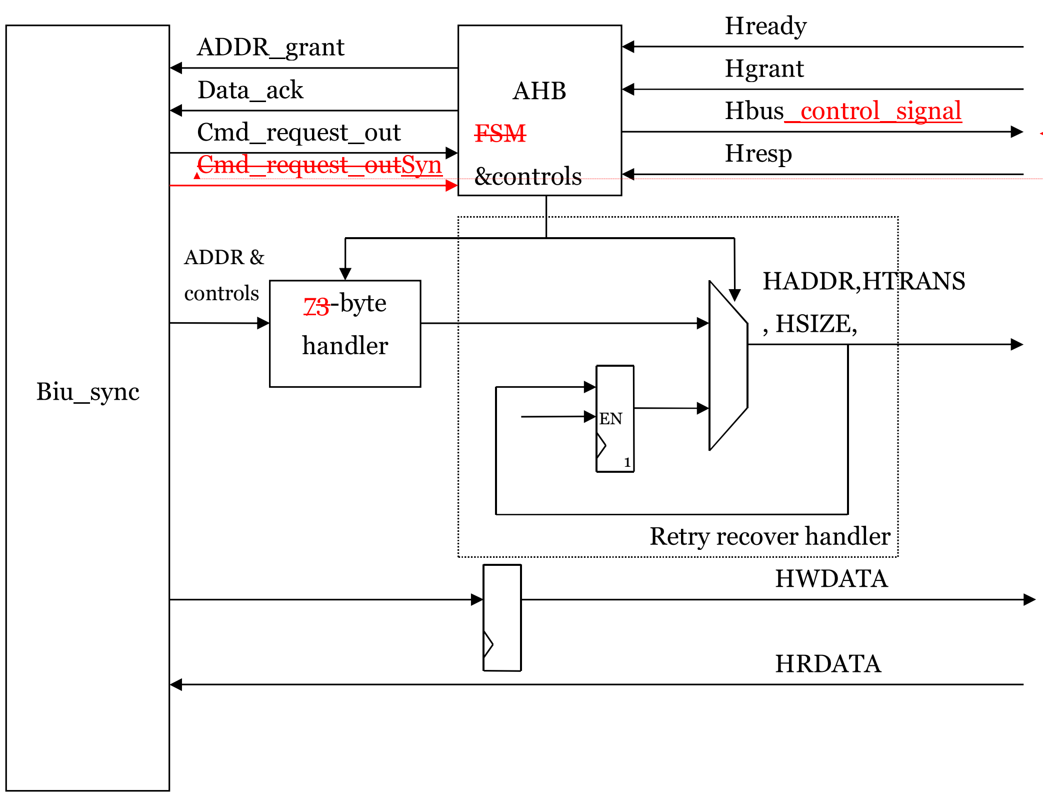
Figure 15. BID write ack logic

3.8. Conservative write order

The axi wrapper does not keep the order of write address and write data. This means the AXI bus matrix or slave might take write data without any write address. To reduce bus matrix and slave design complexity, there is a macro “NDS_AXI_CONSERVATIVE_WRITE_ORDER” to restrict the order of write address and write data channel. If the NDS_AXI_CONSERVATIVE_WRITE_ORDER is defined, the AXI wrapper always sends out write data after write address.

3.9. AHB wrapper

AHB wrapper handles bus request and returns data through biu_sync. To fit the AHB bus protocol AHB wrapper needs a 7-byte handler (un-align access) which divides 7-byte write access into low-4 byte and high-4byte write access.



格式化: 字型色彩: 紅色

格式化: 靠左

格式化: 字型色彩: 紅色

Figure 16 the data path of AHB wrapper

格式化: 靠左, 縮排: 左: 5.1 公分, 第一行: 0.85 公分

3.9.1. 7-byte command handle

7-byte command handler can handle 7-byte write access which is generated by LSMW/LS un-align command. The 7-byte write access definition rule is the write transaction (64-bit) needs to be divided to 2~4 write transactions. (ex: bwe = 8'b01100110, the transaction will be divided to 4 transactions as the following descriptions.)

1'st transaction => bwe = 8'b00000010 (hsize = byte)

2'nd transaction => bwe = 8'b00000100 (hsize = byte)

3'rd transaction => bwe = 8'b00100000 (hsize = byte)

4'th transaction => bwe = 8'b01000000 (hsize = byte)

The figure 17 shows the flow chart for 7-byte access.

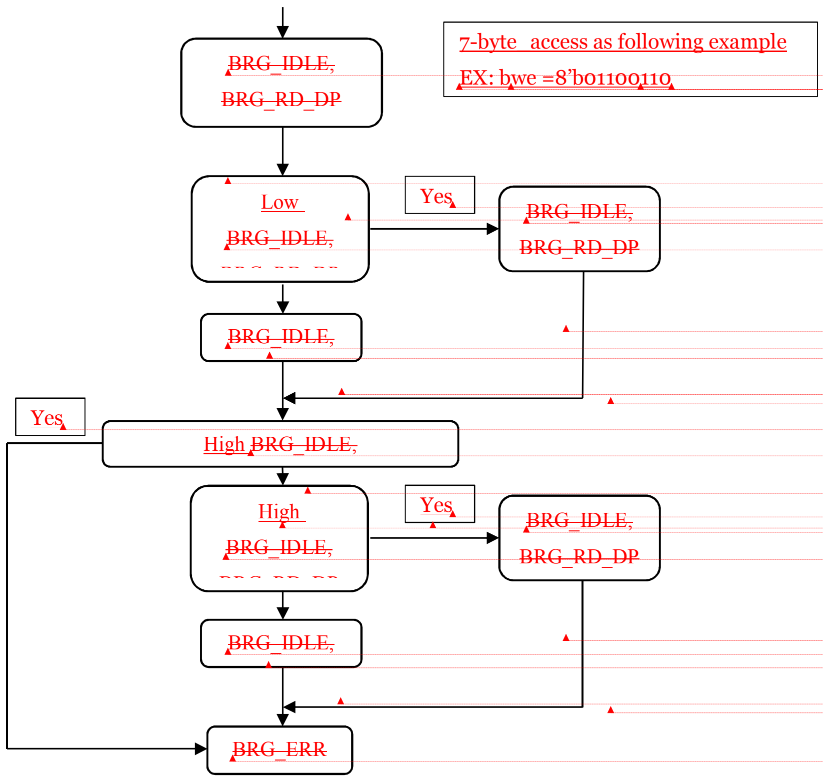


Figure 17 flow chart of 7-byte access

格式化: 字型: (中文) +本文中文字型 (新細明體), 字型色彩: 紅色
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: (中文) +本文中文字型 (新細明體), 10 點
格式化: 字型: 10 點
格式化: 字型: (中文) +本文中文字型 (新細明體), 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: (中文) +本文中文字型 (新細明體), 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: (中文) +本文中文字型 (新細明體), 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點
格式化: 字型: 10 點

When 7-byte access is detected, the low 4-byte will be serviced first. If the 3-byte access is encountered in the low 4-byte (ex: bwe[3:0] is 4'b0110), the data will be processed in 3-byte access handler. Else, the low 4-byte will be processed in align access.
If the 3-byte access is encountered in the high 4-byte (ex: bwe[7:4] is 4'b0110), the data also be processed in 3-byte access handler. Else, the high 4-byte will be processed in align access.
(Note: the process of high 4-byte is the same with the process of the low 4-byte.)

3.9.2. 3-byte command handle

3-byte command handler can handle 3-byte write access. There are three cases of 3-byte access. As Figure 18 shows, the 3-byte handler divides 3-bytes access into three write access.

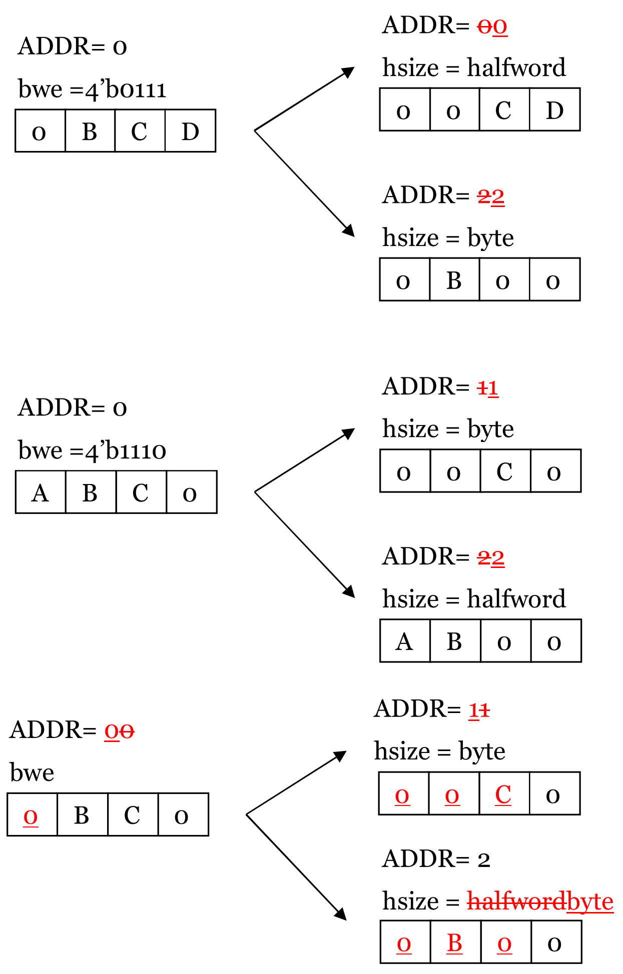


Figure 18 three cases of 3-byte access

3.9.3. Retry recover handler

Retry recover handler recovers the bus access when bus response is RETRY or SPLIT. The

格式化: 字型: (中文) 新細明體

格式化: 縮排: 第一行: 0.5 字元

register 1 of Figure 16 stores the HADDR, HTRANS, HWRITE, HSIZE, HBURST, HLOCK and HPROT. The signals can completely recover the address phase of the retry request. The enable signal of register 1 in Figure 16 is asserted when ahb wrapper completes an address phase.

3.9.4. Read FIFO full handler

When the read FIFO is full, the next read data is not received. The ABH wrapper will inject the busy cycles to the bus for burst read transactions or inject the idle cycles for single read transaction until the read FIFO not full.

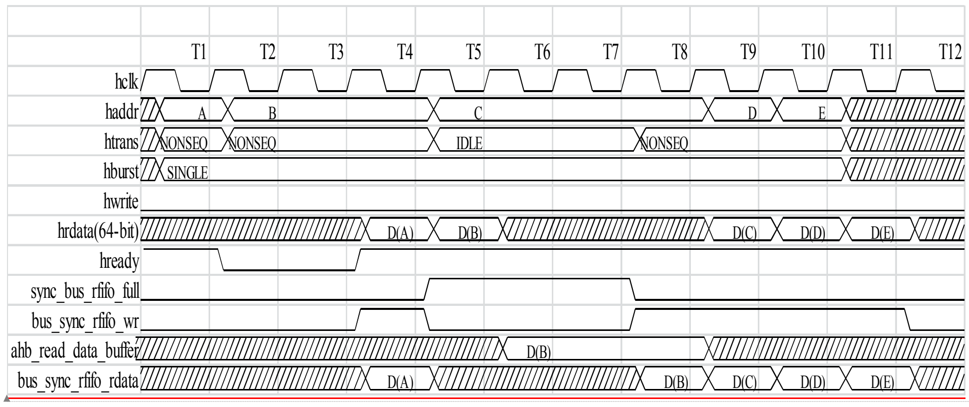
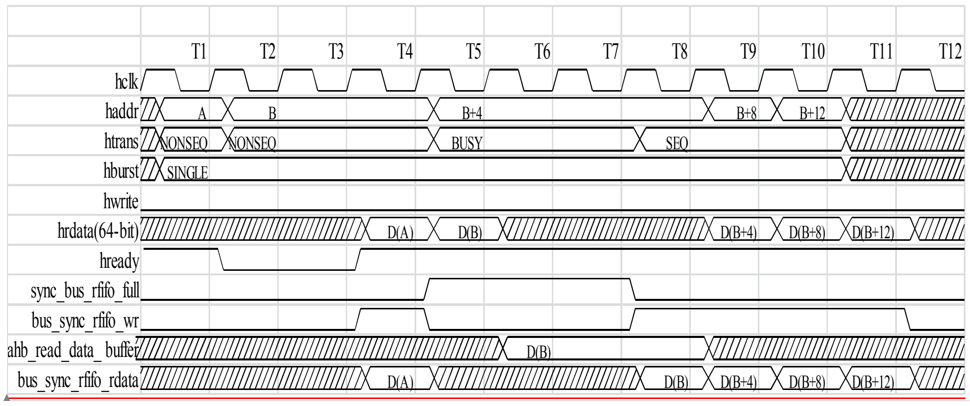


Figure 19. The waveform of single read with the read FIFO full

At T5, the read data FIFO is full, the data B will be received into read data buffer. Meanwhile, the htrans will be driven to IDLE. Then, at T8, the read data FIFO is not full, the data B will be popped into read data FIFO and the htrans will be driven to NONSEQ.



功能變數代碼變更

格式化: 行距: 單行間距, 段落遺留字串控制, 貼齊格線

功能變數代碼變更

Figure 20. The waveform of burst read with the read FIFO full

At T5, the read data FIFO is full, the data B will be received into read data buffer. Meanwhile, the htrans will be driven to BUSY. Then, at T8, the read data FIFO is not full, the data B will be popped into read data FIFO and the htrans will be restored to SEQ.

格式化: 標號, 縮排: 左: 2.55 公分, 第一行: 0.85 公分

格式化: 字型: (英文)Georgia, 12 點, 非粗體

格式化: 行距: 1.5 倍行高, 沒有段落遺留字串控制, 不要貼齊格線

3.9.5. LLW & SCW exclusive access

In AHB wrapper, the sideband signals hllsc_req & hllsc_error are used to support exclusive access as the following Figure 21 .

格式化: 行距: 單行間距, 段落遺留字串控制, 貼齊格線

功能變數代碼變更

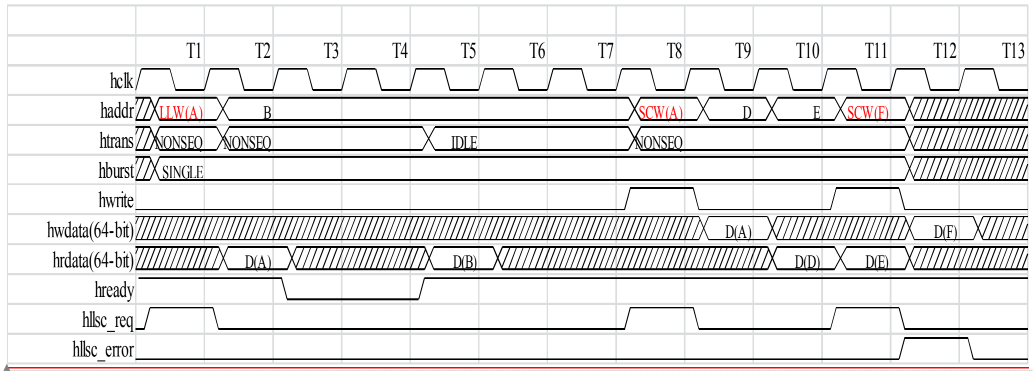


Figure 21. The waveform of LLW & SCW exclusive access

格式化: 行距: 單行間距, 段落遺留字串控制, 貼齊格線

第 vi 頁: [1] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，字型: (中文) Georgia，使用拼字與文法檢查

第 vi 頁: [1] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，字型: (中文) Georgia，使用拼字與文法檢查

第 vi 頁: [2] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [2] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [3] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [3] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [4] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [4] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [5] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [5] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，使用拼字與文法檢查

第 vi 頁: [6] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，字型: (中文) Georgia，使用拼字與文法檢查

第 vi 頁: [6] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，字型: (中文) Georgia，使用拼字與文法檢查

第 vi 頁: [7] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型，字型: (中文) Georgia，使用拼字與文法檢查

第 vi 頁: [7] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [8] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [8] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [9] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [9] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [10] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [10] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [11] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [11] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [12] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [12] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [13] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [13] 格式化　　ANDESTECH\ryjou　　2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [14] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [14] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [15] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [15] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [16] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [16] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [17] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [17] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [18] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [18] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [19] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [19] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [20] 格式化 ANDESTECH\ryjou 2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [20] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [21] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [21] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型, 字型: (中文) Georgia, 使用拼字與文法檢查

第 vi 頁: [22] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [22] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [23] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查

第 vi 頁: [23] 格式化	ANDESTECH\ryjou	2016/2/24 3:12:00 PM

預設段落字型, 使用拼字與文法檢查