

Graywolf BIU Design Specification

|  |  |
| --- | --- |
| Document Number | DSP-00094 |
| Date Issued |  |
| Status | In-progress |



**CONFIDENTIAL**

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 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 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

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

Table of Contents

[Revision History ii](#_Toc444090096)

[List of Tables vi](#_Toc444090097)

[List of Figures vii](#_Toc444090098)

[1. Overview 1](#_Toc444090099)

[1.1. Introduction 1](#_Toc444090100)

[1.2. Features 1](#_Toc444090101)

[1.3. Block Diagram 2](#_Toc444090102)

[1.4. Function Description 2](#_Toc444090103)

[1.4.1. BIU PATH 2](#_Toc444090104)

[1.4.2. BIU SYNC 2](#_Toc444090105)

[1.4.3. BUS Wrapper 2](#_Toc444090106)

[2. Signal Description 3](#_Toc444090107)

[2.1. Signal Definition 3](#_Toc444090108)

[2.2. Timing Diagram for Interface Protocol 10](#_Toc444090109)

[2.2.1. Read transactions 10](#_Toc444090110)

[2.2.2. Write transactions 12](#_Toc444090111)

[2.2.3. MISC 13](#_Toc444090112)

[3. Micro-Architecture 16](#_Toc444090113)

[3.1. BIU\_PATH 16](#_Toc444090114)

[3.2. BIU\_SYNC 17](#_Toc444090115)

[3.3. AXI bus wrapper 19](#_Toc444090116)

[3.4. Write Transfer 20](#_Toc444090117)

[3.5. Read Transfer 20](#_Toc444090118)

[3.6. Lock Transfer 21](#_Toc444090119)

[3.7. Write Acknowledge 21](#_Toc444090120)

[3.8. Conservative write order 21](#_Toc444090121)

[3.9. AHB wrapper 22](#_Toc444090122)

[3.9.1. 3-byte command handle 23](#_Toc444090123)

[3.9.2. Retry recover handler 23](#_Toc444090124)

[3.9.1. Read FIFO full handle 24](#_Toc444090125)

List of Tables

[Table 1. Signal Definition 3](#_Toc442173792)

List of Figures

[Figure 1. Block Diagram 2](#_Toc445889132)

[Figure 2. The waveform of single read data transaction without wait cycles 11](#_Toc445889133)

[Figure 3. The waveform of burst read transaction without wait cycles 11](#_Toc445889134)

[Figure 4. The waveform of burst read data transaction with wait cycles 12](#_Toc445889135)

[Figure 5. The waveform of read data return with the wait state 12](#_Toc445889136)

[Figure 6. The waveform of single write transaction without the wait cycles 13](#_Toc445889137)

[Figure 7. The waveform of burst write transaction with the wait cycles 13](#_Toc445889138)

[Figure 8. The waveform of two sequential read-write transactions with non-sequential grant and ack 14](#_Toc445889139)

[Figure 9. The waveform of write transaction with the bus error 15](#_Toc445889140)

[Figure 10. The waveform of two sequential read transaction with non-sequential ack and grant 16](#_Toc445889141)

[Figure 11. The block diagram of BIU 17](#_Toc445889142)

[Figure 12. The block diagram of the biu\_path 18](#_Toc445889143)

[Figure 13 the data path in synchronization module 19](#_Toc445889144)

[Figure 14. AXI wrapper block diagram 20](#_Toc445889145)

[Figure 15. BID write ack logic 22](#_Toc445889146)

[Figure 16 the data path of AHB wrapper 23](#_Toc445889147)

[Figure 17 flow chart of 7-byte access 24](#_Toc445889148)

[Figure 18 three cases of 3-byte access 25](#_Toc445889149)

[Figure 19. The waveform of single read with the read FIFO full 26](#_Toc445889150)

[Figure 20. The waveform of burst read with the read FIFO full 26](#_Toc445889151)

[Figure 21. The waveform of LLW & SCW exclusive access 27](#_Toc445889152)

**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 |

# Overview

## 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.

## 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

## Block Diagram



Figure 1. Block Diagram

## Function Description

### BIU PATH

Allocate the priority of requests for read and write command.

Sends or receives the command or data from/to BIU\_SYNC.

### BIU SYNC

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

### BUS Wrapper

Transfer the data from/to the bus.

# Signal Description

## 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** | | |
| lsu\_biu\_addr[`NDS\_ADDR\_MSB:0] | 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\_ADDR\_MSB: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 |
| 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] | 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** | | |
| ldma\_biu\_addr[`NDS\_ADDR\_MSB:0] | 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 |
| 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 |
| 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 |
| 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\_request | 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 |
| 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 |

## Timing Diagram for Interface Protocol

Assumption:

Two bubbles are always encountered for the data access cycles. The bubbles are introduced by the command FIFO and the read data FIFO.

### Read transactions



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 shows the read data with the one bus wait cycle.



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

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.

### Write transactions



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

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 shows the write data with the one bus wait cycle.

### 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 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

# Micro-Architecture



Figure 11. The block diagram of BIU

## 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.

Request\_ID

Cmd\_request\_IN

Biu\_sync

Return\_ID

RData

acks

Requests

Addr\_grant

Grants

Ack logic

Wdata

Wdatas

ADDRs

ADDR&Control

CMDs

Cacheablity&BWEs

Arbiter & grant gen

Data\_ack

Hold\_arbitration

Rdata\_wait

Read\_data\_wait

Figure 12. The block diagram of the biu\_path

## 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.

biu\_sync

Cmd\_sync: Depth 4-entry

Wdata\_sync: Depth 4-entry

Rdata\_sync: Depth 2-entry

Cmd\_request\_IN

ADDR&controls\_IN

Path\_cmd\_grant

RID&Rdata\_OUT

Rdata\_OUT\_ack/

Precise\_wdata\_OUT\_ack

Wdata\_IN

ADDR&controls\_OUT

Wdata\_OUT

Cmd\_request\_OUT

Bus\_cmd\_grant

Bus\_wdata\_grant

RID&Rdata\_IN

Rdata\_IN\_ack/

Precise\_wdata\_IN\_ack

Path\_wdata\_grant

Read\_data\_wait

sync\_bus\_rfifo\_full

Figure 13 the data path in synchronization module

## AXI bus wrapper



Figure 14. AXI wrapper block diagram

## 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.

## 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.

## 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.

## 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.

Valid0

ID0

Write ack

wca\_grant

req\_id

write ack

Valid1

ID1

Write ack

Precise1

Valid2

ID2

Write ack

ecise2

Valid3

ID3

Write ack

ack

cise3

1’b0

Bid\_write\_ack

Figure 15. BID write ack logic

## 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.

## 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.

Biu\_sync

AHB

&controls

7-byte handler

EN

Retry recover handler

ADDR & controls

HWDATA

HRDATA

Hready

HADDR,HTRANS, HSIZE, HWRITE…

ADDR\_grant

Data\_ack

Cmd\_request\_out

Hbus\_control\_signal

Hgrant

Hresp

1

Sync\_bus\_rfifo\_full

Figure 16 the data path of AHB wrapper

### 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.

7-byte\_access

Un-align access

Done

Low 4-bwe

is 3yte access?

3-byte\_access

handler

Align access

7-byte\_access as following example

EX: bwe =8’b01100110

Yes

High 4-bwe

is 3yte access?

3-byte\_access

handler

Yes

Align access

High 4-bwe is 4’b0000 ?

Yes

Figure 17 flow chart of 7-byte access

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-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.

ADDR= 0

bwe =4’b0111

0

B

C

D

ADDR= 0

hsize = halfword

0

0

C

D

ADDR= 2

hsize = byte

0

B

0

0

ADDR= 0

bwe =4’b1110

A

B

C

0

ADDR= 1

hsize = byte

0

0

C

0

ADDR= 2

hsize = halfword

A

B

0

0

ADDR= 0

bwe =4’b0110

0

B

C

0

ADDR= 1

hsize = byte

ADDR= 2

hsize = byte

0

0

C

0

0

B

0

0

Figure 18 three cases of 3-byte access

### Retry recover handler

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

### 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.



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.

### 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 .



Figure 21. The waveform of LLW & SCW exclusive access