AMBA® 3 AHB-Lite Protocol

v1.0

Specification

ARM®

ARM IHI 0033A

AMBA 3 AHB-Lite Protocol
Specification

Release Information

Change history

Date	Issue	Confidentiality	Change
06 June 2006	A	Non-Confidential	First release for v1.0

Proprietary Notice

ARM AMBA Specification Licence

1. Subject to the provisions of Clauses 2, 3 and 4, ARM hereby grants to LICENSEE a perpetual, non-exclusive, non-transferable, royalty free, worldwide licence to:

(i) use and copy the relevant AMBA Specification for the purpose of developing and having developed products that comply with the relevant AMBA Specification;

(ii) manufacture and have manufactured products which either: (a) have been created by or for LICENSEE under the licence granted in Clause 1(i); or (b) incorporate a product(s) which has been created by a third party(s) under a licence granted by ARM in Clause 1(i) of such third party's ARM AMBA Specification Licence; and

(iii) offer to sell, sell, supply or otherwise distribute products which have either been (a) created by or for LICENSEE under the licence granted in Clause 1(i); or (b) manufactured by or for LICENSEE under the licence granted in Clause 1(ii).

2. LICENSEE hereby agrees that the licence granted in Clause 1 is subject to the following restrictions:

(i) where a product created under Clause 1(i) is an integrated circuit which includes a CPU then either: (a) such CPU shall only be manufactured under licence from ARM; or (b) such CPU is neither substantially compliant with nor marketed as being compliant with the ARM instruction sets licensed by ARM from time to time;

(ii) the licences granted in Clause 1(iii) shall not extend to any portion or function of a product that is not itself compliant with part of the relevant AMBA Specification; and

(iii) no right is granted to LICENSEE to sublicense the rights granted to LICENSEE under this Agreement.

3. Except as specifically licensed in accordance with Clause 1, LICENSEE acquires no right, title or interest in any ARM technology or any intellectual property embodied therein. In no event shall the licences granted in accordance with Clause 1 be construed as granting LICENSEE, expressly or by implication, estoppel or otherwise, a licence to use any ARM technology except the relevant AMBA Specification.

4. THE RELEVANT AMBA SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NONINFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE.

5. No licence, express, implied or otherwise, is granted to LICENSEE, under the provisions of Clause 1, to use the ARM tradename, or AMBA trademark in connection with the relevant AMBA Specification or any products based thereon. Nothing in Clause 1 shall be construed as authority for LICENSEE to make any representations on behalf of ARM in respect of the relevant AMBA Specification.

6. This Licence shall remain in force until terminated by you or by ARM. Without prejudice to any of its other rights if LICENSEE is in breach of any of the terms and conditions of this Licence then ARM may terminate this Licence immediately upon giving written notice to You. You may terminate this Licence at any time. Upon expiry or termination of this Licence by You or by ARM LICENSEE shall stop using the relevant AMBA Specification and destroy all copies of the relevant AMBA Specification in your possession together with all documentation and related materials. Upon expiry or termination of this Licence, the provisions of clauses 6 and 7 shall survive.

7. The validity, construction and performance of this Agreement shall be governed by English Law.

ARM contract references: LEC-PRE-00490-V4.0 ARM AMBA Specification Licence.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Note

In this PDF, published March 2010, pages ii to iv have been replaced, by an edit to the PDF, to include an updated Proprietary Notice. The remainder of the PDF is the original published PDF. This PDF therefore retains the original document number.

Contents
AMBA 3 AHB-Lite Protocol Specification

List of Tables
AMBA 3 AHB-Lite Protocol Specification

List of Figures
AMBA 3 AHB-Lite Protocol Specification

Preface

This preface introduces the *Advanced Microcontroller Bus Architecture* (AMBA) 3 *AHB-Lite Protocol Specification*. It contains the following sections:

About this book

This is the specification for the AMBA 3 AHB-Lite protocol.

Intended audience

This book is written to help hardware and software engineers design systems and modules that are compliant with the AHB-Lite protocol.

Using this specification

This specification is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an overview of the AHB-Lite protocol.

Chapter 2 ***Signal Descriptions***

Read this chapter for descriptions of the signals.

Chapter 3 ***Transfers***

Read this chapter for information about the different types of transfer initiated by an AHB-Lite compliant master.

Chapter 4 ***Bus Interconnection***

Read this chapter for information about the additional interconnect logic required for AHB-Lite systems.

Chapter 5 ***Slave Response Signaling***

Read this chapter for information about the slave response signaling.

Chapter 6 ***Data Buses***

Read this chapter for information about the read and write data buses and how to interface to different data bus widths.

Chapter 7 ***Clock and Reset***

Read this chapter for information about the clock and reset signals.

Glossary Read the Glossary for definitions of terms used in this specification.

Conventions

This section describes the conventions that this specification uses:

- *Typographical* on page xiii

- *Timing diagrams*
- *Signals* on page xiv.

Typographical

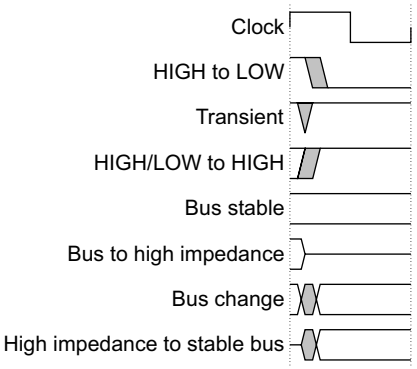
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, when appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example: • `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>` • The Opcode_2 value selects which register is accessed.

Timing diagrams

The figure named *Key to timing diagram conventions* on page xiv explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined so the bus or signal can assume any value that the shaded area represents. The actual level is unimportant and does not affect normal operation.



Key to timing diagram conventions

Note

Single-bit signals are sometimes shown as HIGH and LOW at the same time and they look similar to the bus change shown in *Key to timing diagram conventions*. If a single-bit signal is shown like this then its value does not affect the accompanying description.

Signals

The signal conventions are:

Lower-case n Denotes an active-LOW signal.

Prefix H Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix P Denotes *Advanced Peripheral Bus* (APB) signals.

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM Limited periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets, addenda, and the Frequently Asked Questions list.

ARM publications

This specification contains information that is specific to the protocol. See the following documents for other relevant information:

- *AMBA 3 APB Protocol Specification* (ARM IHI 0024)
- *AMBA AXI Protocol Specification* (ARM IHI 0022)
- *Multi-layer AHB Overview* (ARM DVI 0045B).

Feedback

ARM Limited welcomes feedback on the AHB-Lite protocol and its documentation.

Feedback on the protocol

Contact ARM Limited if you have any comments or suggestions about the AHB-Lite protocol.

Feedback on this specification

If you have any comments on this specification, send email to `errata@arm.com` giving:

- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM Limited also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter provides an overview of the AHB-Lite protocol. It contains the following sections:

- *About the protocol* on page 1-2
- *Operation* on page 1-5
- *Multi-layer AHB-Lite* on page 1-6.

Note

For illustrative purposes, a 32-bit data bus is used in this specification. Additional data bus widths are permitted, as *Data bus width* on page 6-5 shows.

1.1 About the protocol

AMBA AHB-Lite addresses the requirements of high-performance synthesizable designs. It is a bus interface that supports a single bus master and provides high-bandwidth operation.

Note

See *Multi-layer AHB-Lite* on page 1-6 for information about how to implement a multi-master system based on the AHB-Lite bus interface.

AHB-Lite implements the features required for high-performance, high clock frequency systems including:

- burst transfers
- single-clock edge operation
- non-tristate implementation
- wide data bus configurations, 64, 128, 256, 512, and 1024 bits.

The most common AHB-Lite slaves are internal memory devices, external memory interfaces, and high bandwidth peripherals. Although low-bandwidth peripherals can be included as AHB-Lite slaves, for system performance reasons they typically reside on the AMBA *Advanced Peripheral Bus* (APB). Bridging between this higher level of bus and APB is done using a AHB-Lite slave, known as an APB bridge.

Figure 1-1 shows a single master AHB-Lite system design with one AHB-Lite master and three AHB-Lite slaves. The bus interconnect logic consists of one address decoder and a slave-to-master multiplexor. The decoder monitors the address from the master so that the appropriate slave is selected and the multiplexor routes the corresponding slave output data back to the master.

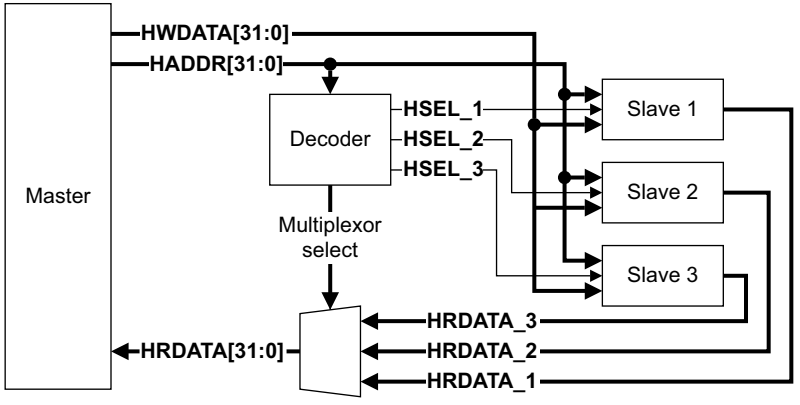


Figure 1-1 AHB-Lite block diagram

Note

Figure 1-1 on page 1-2 does not show the master control signals. This is for clarity.

The main component types of an AHB-Lite system are described in:

- *Master*
- *Slave*
- *Decoder* on page 1-4.
- *Multiplexor* on page 1-4.

1.1.1 Master

An AHB-Lite master provides address and control information to initiate read and write operations. Figure 1-2 shows an AHB-Lite master interface.

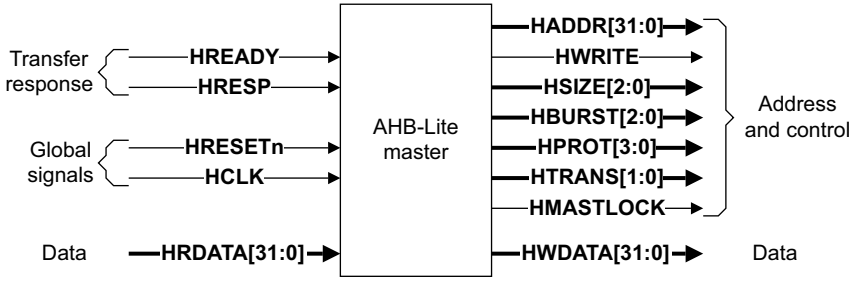


Figure 1-2 Master interface

1.1.2 Slave

An AHB-Lite slave responds to transfers initiated by masters in the system. The slave uses the **HSELx** select signal from the decoder to control when it responds to a bus transfer. The slave signals back to the master:

- the success
- failure
- or waiting of the data transfer.

Figure 1-3 on page 1-4 shows an AHB-Lite slave interface.

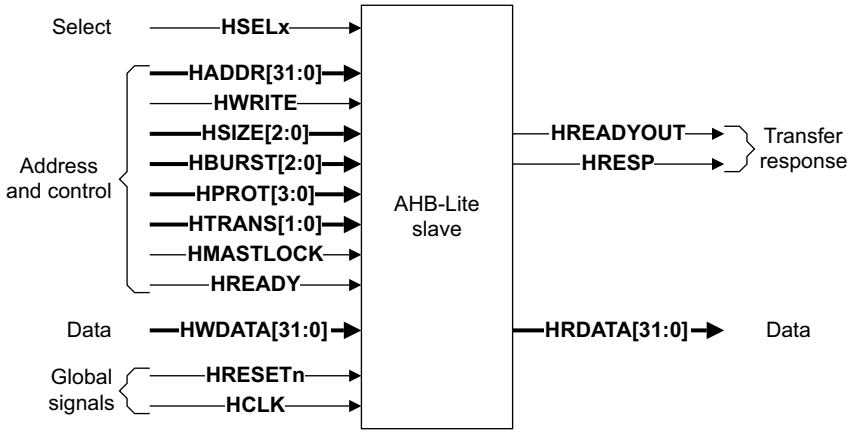


Figure 1-3 Slave interface

1.1.3 Decoder

This component decodes the address of each transfer and provides a select signal for the slave that is involved in the transfer. It also provides a control signal to the multiplexor.

A single centralized decoder is required in all AHB-Lite implementations that use two or more slaves. See *Address decoding* on page 4-2 for more information.

Note

In multi-layer AHB-Lite implementations, the decoder function is usually included in the multi-layer interconnect component.

1.1.4 Multiplexor

A slave-to-master multiplexor is required to multiplex the read data bus and response signals from the slaves to the master. The decoder provides control for the multiplexor.

A single centralized multiplexor is required in all AHB-Lite implementations that use two or more slaves. See *Bus interconnection* on page 4-3 for more information.

Note

In multi-layer AHB-Lite implementations, the multiplexor function is typically included in the multi-layer interconnect component.

1.2 Operation

The master starts a transfer by driving the address and control signals. These signals provide information about the address, direction, width of the transfer, and indicate if the transfer forms part of a burst. Transfers can be:

- single
- incrementing bursts that do not wrap at address boundaries
- wrapping bursts that wrap at particular address boundaries.

The write data bus moves data from the master to a slave, and the read data bus moves data from a slave to the master.

Every transfer consists of:

Address phase one address and control cycle

Data phase one or more cycles for the data.

A slave cannot request that the address phase is extended and therefore all slaves must be capable of sampling the address during this time. However, a slave can request that the master extends the data phase by using **HREADY**. This signal, when LOW, causes wait states to be inserted into the transfer and enables the slave to have extra time to provide or sample data.

The slave uses **HRESP** to indicate the success or failure of a transfer.

1.3 Multi-layer AHB-Lite

Because AHB-Lite is a single master bus interface then if a multi-master system is required, the system designer must include a component that isolates all masters from each other. To achieve this isolation function, each master can be considered to be on its own layer, therefore the component must create a multi-layer interconnect where all masters are isolated from each other, but can share access to the slaves. Slave arbitration must be performed by the multi-layer interconnect component.

Figure 1-4 shows an example multi-layer AHB-Lite system.

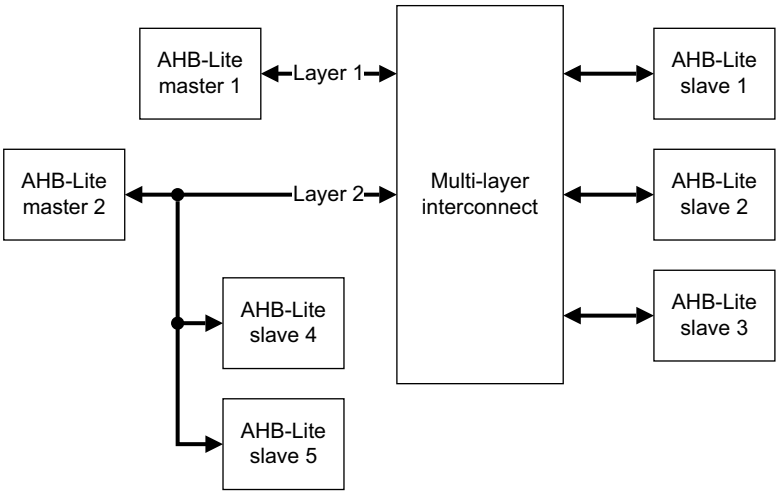


Figure 1-4 Example multi-layer AHB-Lite block diagram

In Figure 1-4, master 1 and master 2 each have access to slaves 1, 2, and 3. The multi-layer interconnect must prevent simultaneous access to a single slave by implementing an arbitration scheme for the three shared slaves. Master 1 does not require access to slaves 4 and 5, so these two slaves are kept local to master 2. This reduces the complexity of the multi-layer interconnect component.

The design of a multi-layer interconnect component is outside the scope of this specification. See the *Multi-layer AHB Overview* for more information about implementing a multi-layer AHB-Lite interconnect.

Chapter 2
Signal Descriptions

This chapter describes the protocol signals. It contains the following sections:

- *Global signals* on page 2-2
- *Master signals* on page 2-3
- *Slave signals* on page 2-5
- *Decoder signals* on page 2-6
- *Multiplexor signals* on page 2-7.

Note

All AHB-Lite signals are prefixed with the letter **H** to differentiate them from other similarly named signals in a system design.

2.1 Global signals

Table 2-1 lists the protocol global signals.

Table 2-1 Global signals

Name	Source	Description
HCLK	Clock source	The bus clock times all bus transfers. All signal timings are related to the rising edge of **HCLK**. See *Clock* on page 7-2 for more information.
HRESETn	Reset controller	The bus reset signal is active LOW and resets the system and the bus. This is the only active LOW AHB-Lite signal. See *Reset* on page 7-2 for more information.

2.2 Master signals

Table 2-2 lists the protocol signals generated by a master.

Table 2-2 Master signals

Name	Destination	Description
HADDR[31:0]	Slave and decoder	The 32-bit system address bus.
HBURST[2:0]	Slave	The burst type indicates if the transfer is a single transfer or forms part of a burst. Fixed length bursts of 4, 8, and 16 beats are supported. The burst can be incrementing or wrapping. Incrementing bursts of undefined length are also supported. See *Burst operation* on page 3-9 for more information.
HMASTLOCK	Slave	When HIGH, this signal indicates that the current transfer is part of a locked sequence. It has the same timing as the address and control signals. See *Locked transfers* on page 3-7 for more information.
HPROT[3:0]	Slave	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wants to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, and if the transfer is a privileged mode access or user mode access. For masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable. See *Protection control* on page 3-22 for more information.
HSIZE[2:0]	Slave	Indicates the size of the transfer, that is typically byte, halfword, or word. The protocol allows for larger transfer sizes up to a maximum of 1024 bits. See *Transfer size* on page 3-8 for more information.

Table 2-2 Master signals (continued)

Name	Destination	Description
HTRANS[1:0]	Slave	Indicates the transfer type of the current transfer. This can be: • IDLE • BUSY • NONSEQUENTIAL • SEQUENTIAL. See *Transfer types* on page 3-5 for more information.
HWDATA[31:0][a]	Slave	The write data bus transfers data from the master to the slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this can be extended to enable higher bandwidth operation. See *Data buses* on page 6-2 for more information.
HWRITE	Slave	Indicates the transfer direction. When HIGH this signal indicates a write transfer and when LOW a read transfer. It has the same timing as the address signals, however, it must remain constant throughout a burst transfer. See *Basic transfers* on page 3-2 for more information.

a. The write data bus width is not restricted to 32bits. *Data bus width* on page 6-5 lists the other permitted data widths.

2.3 Slave signals

Table 2-3 lists the protocol signals generated by a slave.

Table 2-3 Slave signals

Name	Destination	Description
HRDATA[31:0][a]	Multiplexor	During read operations, the read data bus transfers data from the selected slave to the multiplexor. The multiplexor then transfers the data to the master. A minimum data bus width of 32 bits is recommended. However, this can be extended to enable higher bandwidth operation. See *Data buses* on page 6-2 for more information.
HREADYOUT	Multiplexor	When HIGH, the **HREADYOUT** signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer. See *Bus interconnection* on page 4-3 for more information.
HRESP	Multiplexor	The transfer response, after passing through the multiplexor, provides the master with additional information on the status of a transfer. When LOW, the **HRESP** signal indicates that the transfer status is OKAY. When HIGH, the **HRESP** signal indicates that the transfer status is ERROR. See *Slave transfer responses* on page 5-2 for more information.

a. The read data bus width is not restricted to 32bits. *Data bus width* on page 6-5 lists the other permitted data widths.

2.4 Decoder signals

Table 2-4 lists the protocol signals generated by the decoder.

Table 2-4 Decoder signals

Name	Destination	Description
HSELx[a]	Slave	Each AHB-Lite slave has its own slave select signal **HSELx** and this signal indicates that the current transfer is intended for the selected slave. When the slave is initially selected, it must also monitor the status of **HREADY** to ensure that the previous bus transfer has completed, before it responds to the current transfer. The **HSELx** signal is a combinatorial decode of the address bus. See *Address decoding* on page 4-2 for more information.

a. The letter x used in **HSELx** must be changed to a unique identifier for each AHB-Lite slave in a system. For example, **HSEL_S1**, **HSEL_S2**, and **HSEL_Memory**.

Note

Usually the decoder also provides the multiplexor with the **HSELx** signals, or a signal/bus derived from the **HSELx** signals, to enable the multiplexor to route the appropriate signals, from the selected slave to the master. It is important that these additional multiplexor control signals are retimed to the data phase.

2.5 Multiplexor signals

Table 2-5 lists the protocol signals generated by the multiplexor.

Table 2-5 Multiplexor signals

Name	Destination	Description
HRDATA[31:0]	Master	Read data bus, selected by the decoder.[a]
HREADY	Master and slave	When HIGH, the **HREADY** signal indicates to the master and all slaves, that the previous transfer is complete. See *Bus interconnection* on page 4-3 for more information.
HRESP	Master	Transfer response, selected by the decoder.[a]

a. Because the **HRDATA[31:0]** and **HRESP** signals pass through the multiplexor and retain the same signal naming, the full signal description for these two signals are provided in Table 2-3 on page 2-5.

Chapter 3
Transfers

This chapter describes AHB-Lite read and write transfers. It contains the following sections:

3.1 Basic transfers

An AHB-Lite transfer consists of two phases:

Address Lasts for a single **HCLK** cycle unless its extended by the previous bus transfer.

Data That might require several **HCLK** cycles. Use the **HREADY** signal to control the number of clock cycles required to complete the transfer.

HWRITE controls the direction of data transfer to or from the master. Therefore, when:

- **HWRITE** is HIGH, it indicates a write transfer and the master broadcasts data on the write data bus, **HWDATA[31:0]**
- **HWRITE** is LOW, a read transfer is performed and the slave must generate the data on the read data bus, **HRDATA[31:0]**.

The simplest transfer is one with no wait states, so the transfer consists of one address cycle and one data cycle. Figure 3-1 shows a simple read transfer and Figure 3-2 shows a simple write transfer.

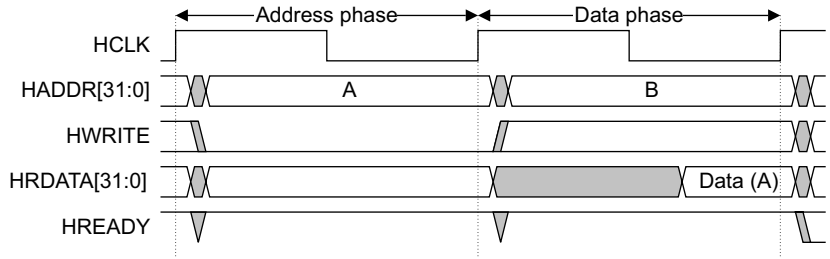


Figure 3-1 Read transfer

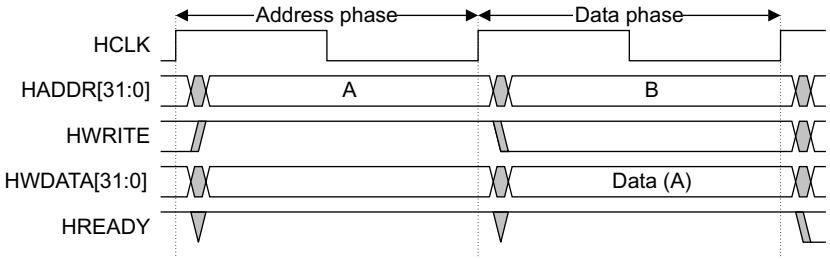


Figure 3-2 Write transfer

In a simple transfer with no wait states:

1. The master drives the address and control signals onto the bus after the rising edge of **HCLK**.
2. The slave then samples the address and control information on the next rising edge of **HCLK**.
3. After the slave has sampled the address and control it can start to drive the appropriate **HREADY** response. This response is sampled by the master on the third rising edge of **HCLK**.

This simple example demonstrates how the address and data phases of the transfer occur during different clock cycles. The address phase of any transfer occurs during the data phase of the previous transfer. This overlapping of address and data is fundamental to the pipelined nature of the bus and enables high performance operation while still providing adequate time for a slave to provide the response to a transfer.

A slave can insert wait states into any transfer to enable additional time for completion.

Figure 3-3 shows a read transfer with two wait states.

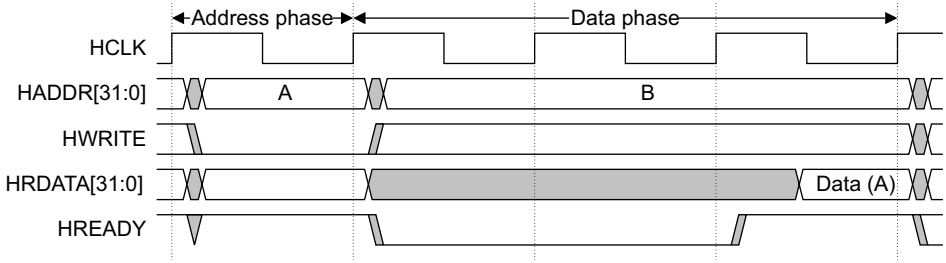


Figure 3-3 Read transfer with wait states

Figure 3-4 shows a write transfer with one wait state.

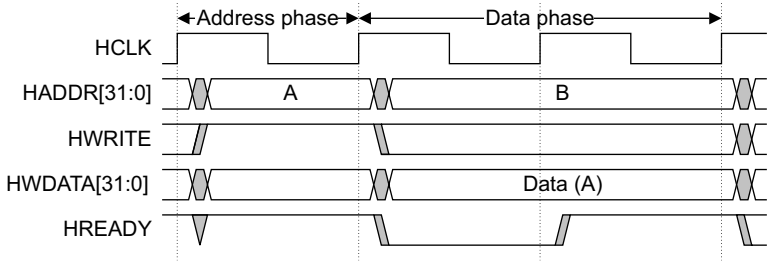


Figure 3-4 Write transfer with wait state

Note

For write operations the master holds the data stable throughout the extended cycles. For read transfers the slave does not have to provide valid data until the transfer is about to complete.

When a transfer is extended in this way it has the side-effect of extending the address phase of the next transfer. Figure 3-5 shows three transfers to unrelated addresses, A, B, and C with an extended address phase for address C.

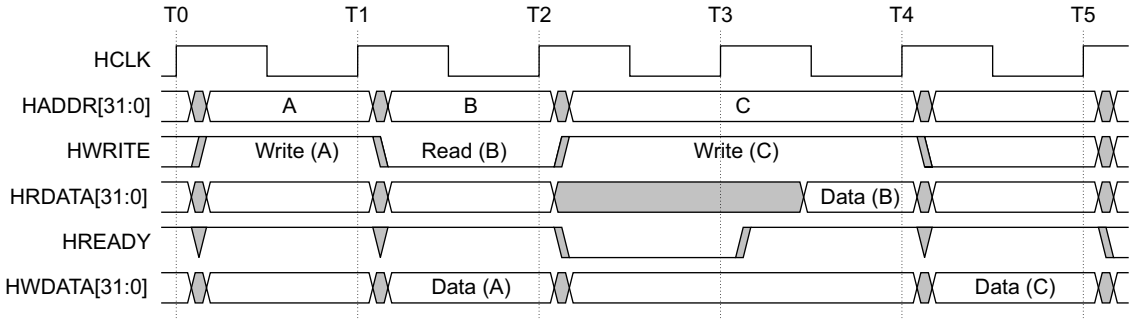


Figure 3-5 Multiple transfers

In Figure 3-5:

- the transfers to addresses A and C are zero wait state
- the transfer to address B is one wait state
- extending the data phase of the transfer to address B has the effect of extending the address phase of the transfer to address C.

3.2 Transfer types

Transfers can be classified into one of four types, as controlled by **HTRANS[1:0]**. Table 3-1 lists these.

Table 3-1 Transfer type encoding

HTRANS[1:0]	Type	Description
b00	IDLE	Indicates that no data transfer is required. A master uses an IDLE transfer when it does not want to perform a data transfer. It is recommended that the master terminates a locked transfer with an IDLE transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer must be ignored by the slave.
b01	BUSY	The BUSY transfer type enables masters to insert idle cycles in the middle of a burst. This transfer type indicates that the master is continuing with a burst but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst. Only undefined length bursts can have a BUSY transfer as the last cycle of a burst. See *Burst termination after a BUSY transfer* on page 3-10 for more information. Slaves must always provide a zero wait state OKAY response to BUSY transfers and the transfer must be ignored by the slave.
b10	NONSEQ	Indicates a single transfer or the first transfer of a burst. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of length one and therefore the transfer type is NONSEQUENTIAL.
b11	SEQ	The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. The address is equal to the address of the previous transfer plus the transfer size, in bytes, with the transfer size being signaled by the **HSIZE[2:0]** signals. In the case of a wrapping burst the address of the transfer wraps at the address boundary.

Figure 3-6 on page 3-6 shows the use of the NONSEQ, BUSY, and SEQ transfer types.

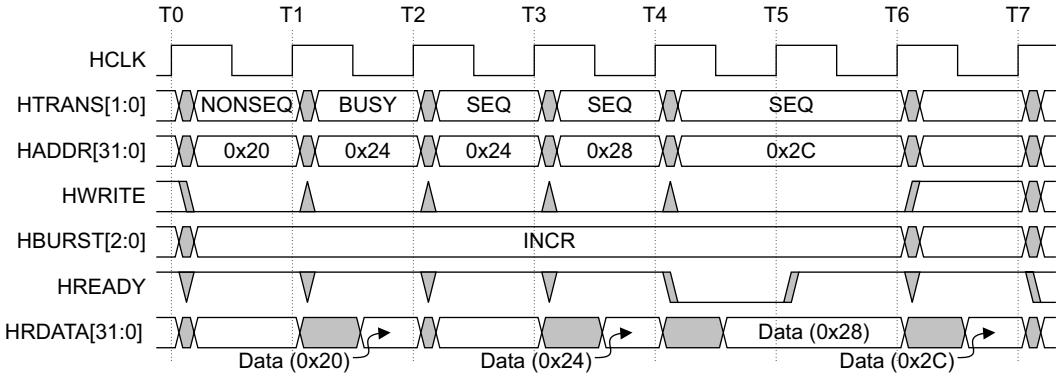


Figure 3-6 Transfer type examples

In Figure 3-6:

T0-T1 The 4-beat read starts with a NONSEQ transfer.

T1-T2 The master is unable to perform the second beat and inserts a BUSY transfer to delay the start of the second beat.

The slave provides the read data for the first beat.

T2-T3 The master is now ready to start the second beat, so a SEQ transfer is signaled. The master ignores any data that the slave provides on the read data bus.

T3-T4 The master performs the third beat.

The slave provides the read data for the second beat.

T4-T5 The master performs the last beat.

The slave is unable to complete the transfer and uses **HREADY** to insert a single wait state.

T5-T6 The slave provides the read data for the third beat.

T6-T7 The slave provides the read data for the last beat.

3.3 Locked transfers

If the master requires locked accesses then it must also assert the **HMASTLOCK** signal. This signal indicates to any slave that the current transfer sequence is indivisible and must therefore be processed before any other transactions are processed.

Typically the locked transfer is used to maintain the integrity of a semaphore, by ensuring that the slave does not perform other operations between the read and write phases of a microprocessor SWP instruction.

Figure 3-7 shows the **HMASTLOCK** signal with a microprocessor SWP instruction.

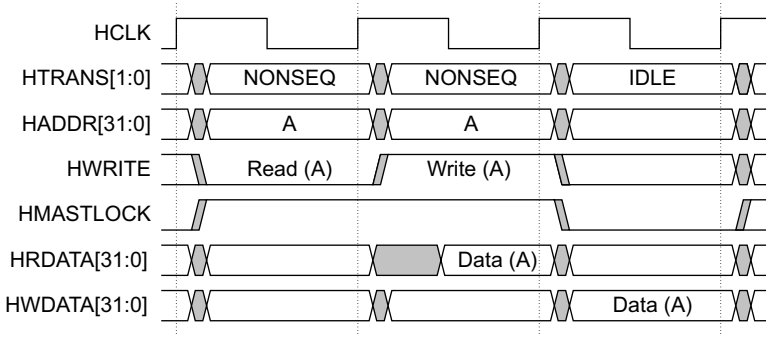


Figure 3-7 Locked transfer

Note

After a locked transfer, it is recommended that the master inserts an IDLE transfer.

Most slaves have no requirement to implement **HMASTLOCK** because they are only capable of performing transfers in the order they are received. Slaves that can be accessed by more than one master, for example, a *Multi-Port Memory Controller* (MPMC) must implement the **HMASTLOCK** signal.

3.4 Transfer size

HSIZE[2:0] indicates the size of a data transfer. Table 3-2 lists the possible transfer sizes.

Table 3-2 Transfer size encoding

HSIZE[2]	HSIZE[1]	HSIZE[0]	Size (bits)	Description
0	0	0	8	Byte
0	0	1	16	Halfword
0	1	0	32	Word
0	1	1	64	Doubleword
1	0	0	128	4-word line
1	0	1	256	8-word line
1	1	0	512	-
1	1	1	1024	-

Note

The transfer size set by **HSIZE** must be less than or equal to the width of the data bus. For example, with a 32-bit data bus, **HSIZE** must only use the values b000, b001, or b010.

Use **HSIZE** in conjunction with **HBURST**, to determine the address boundary for wrapping bursts.

The **HSIZE** signals have exactly the same timing as the address bus. However, they must remain constant throughout a burst transfer.

3.5 Burst operation

Bursts of 4, 8, and 16-beats, undefined length bursts, and single transfers are defined in this protocol. It supports incrementing and wrapping bursts:

- Incrementing bursts access sequential locations and the address of each transfer in the burst is an increment of the previous address.
- Wrapping bursts wrap when they cross an address boundary. The address boundary is calculated as the product of the number of beats in a burst and the size of the transfer. The number of beats are controlled by **HBURST** and the transfer size is controlled by **HSIZE**.

  For example, a four-beat wrapping burst of word (4-byte) accesses wraps at 16-byte boundaries. Therefore, if the start address of the transfer is `0x34`, then it consists of four transfers to addresses `0x34`, `0x38`, `0x3C`, and `0x30`.

HBURST[2:0] controls the burst type. Table 3-3 lists the possible burst types.

Table 3-3 Burst signal encoding

HBURST[2:0]	Type	Description
b000	SINGLE	Single burst
b001	INCR	Incrementing burst of undefined length
b010	WRAP4	4-beat wrapping burst
b011	INCR4	4-beat incrementing burst
b100	WRAP8	8-beat wrapping burst
b101	INCR8	8-beat incrementing burst
b110	WRAP16	16-beat wrapping burst
b111	INCR16	16-beat incrementing burst

Masters must not attempt to start an incrementing burst that crosses a 1KB address boundary.

Masters can perform single transfers using either:

- SINGLE burst
- undefined length burst that has a burst of length one.

Note

The burst size indicates the number of beats in the burst and not the number of bytes transferred. Calculate the total amount of data transferred in a burst by multiplying the number of beats by the amount of data in each beat, as indicated by **HSIZE[2:0]**.

All transfers in a burst must be aligned to the address boundary equal to the size of the transfer. For example, you must align word transfers to word address boundaries (**HADDR[1:0]** = b00), and halfword transfers to halfword address boundaries (**HADDR[0]** = 0). The address for IDLE transfers must also be aligned, otherwise during simulation it is likely that bus monitors could report spurious warnings.

3.5.1 Burst termination after a BUSY transfer

After a burst has started, the master uses BUSY transfers if it requires more time before continuing with the next transfer in the burst.

During an undefined length burst, INCR, the master might insert BUSY transfers and then decide that no more data transfers are required. Under these circumstances, it is acceptable for the master to then perform a NONSEQ or IDLE transfer that then effectively terminates the undefined length burst.

The protocol does not permit a master to end a burst with a BUSY transfer for fixed length bursts of type:

- incrementing INCR4, INCR8, and INCR16
- or wrapping WRAP4, WRAP8, and WRAP16.

These fixed length burst types must terminate with a SEQ transfer.

The master is not permitted to perform a BUSY transfer immediately after a SINGLE burst. SINGLE bursts must be followed by an IDLE transfer or a NONSEQ transfer.

3.5.2 Early burst termination

Bursts can be terminated by either:

- *Slave error response*
- *Multi-layer interconnect termination* on page 3-11.

Slave error response

If a slave provides an ERROR response then the master can cancel the remaining transfers in the burst. However, this is not a strict requirement and it is also acceptable for the master to continue the remaining transfers in the burst.

If the master does not complete that burst then there is no requirement for it to rebuild the burst when it next accesses that slave. For example, if a master only completes three beats of an eight-beat burst then it does not have to complete the remaining five transfers when it next accesses that slave.

Multi-layer interconnect termination

Although masters are not permitted to terminate a burst request early, slaves must be designed to work correctly if the burst is not completed.

When a multi-layer interconnect component is used in a multi-master system then it can terminate a burst so that another master can gain access to the slave. The slave must terminate the burst from the original master and then respond appropriately to the new master if this occurs.

3.5.3 Burst examples

Examples of various bursts are shown in the following sections:

- *Four-beat wrapping burst, WRAP4*
- *Four-beat incrementing burst, INCR4* on page 3-12
- *Eight-beat wrapping burst, WRAP8* on page 3-13
- *Eight-beat incrementing burst, INCR8* on page 3-14
- *Undefined length bursts, INCR* on page 3-14.

Four-beat wrapping burst, WRAP4

Figure 3-8 on page 3-12 shows a write transfer using a four-beat wrapping burst, with a wait state added for the first transfer.

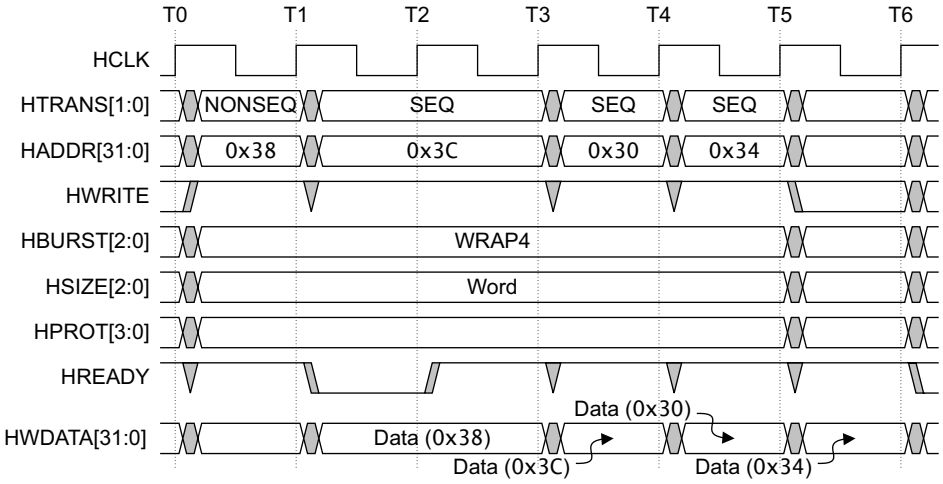


Figure 3-8 Four-beat wrapping burst

Because the burst is a four-beat burst of word transfers, the address wraps at 16-byte boundaries, and the transfer to address `0x3C` is followed by a transfer to address `0x30`.

Four-beat incrementing burst, INCR4

Figure 3-9 on page 3-13 shows a read transfer using a four-beat incrementing burst, with a wait state added for the first transfer. In this case, the address does not wrap at a 16-byte boundary and the address `0x3C` is followed by a transfer to address `0x40`.

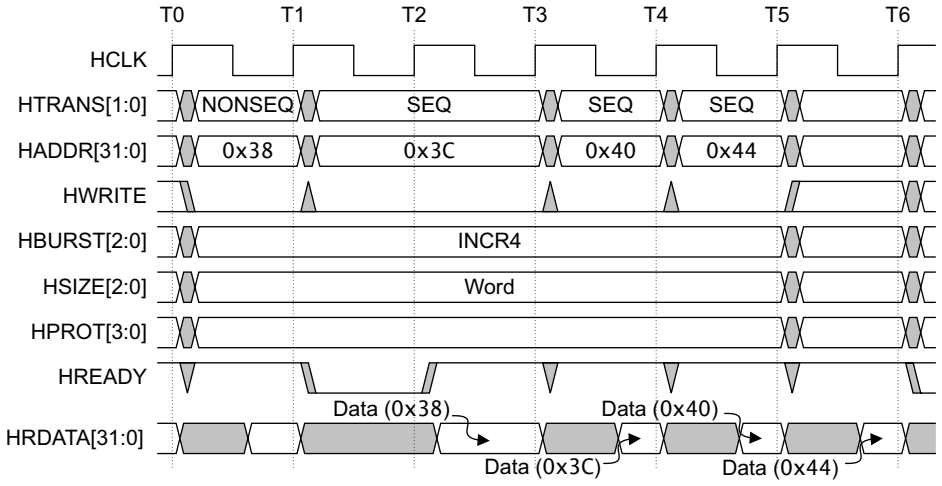


Figure 3-9 Four-beat incrementing burst

Eight-beat wrapping burst, WRAP8

Figure 3-10 shows a read transfer using an eight-beat wrapping burst.

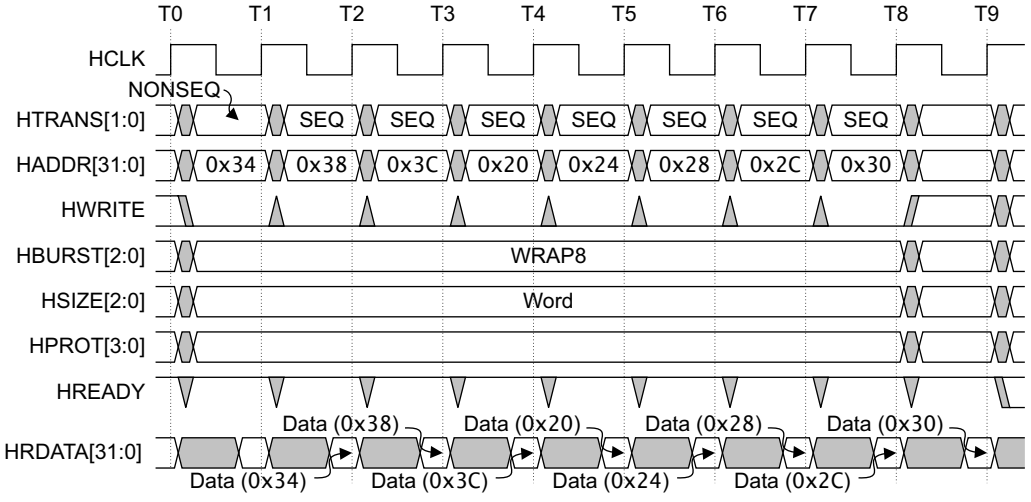


Figure 3-10 Eight-beat wrapping burst

Because the burst is an eight-beat burst of word transfers, the address wraps at 32-byte boundaries, and the transfer to address `0x3C` is followed by a transfer to address `0x20`.

Eight-beat incrementing burst, INCR8

Figure 3-11 shows a write transfer using an eight-beat incrementing burst.

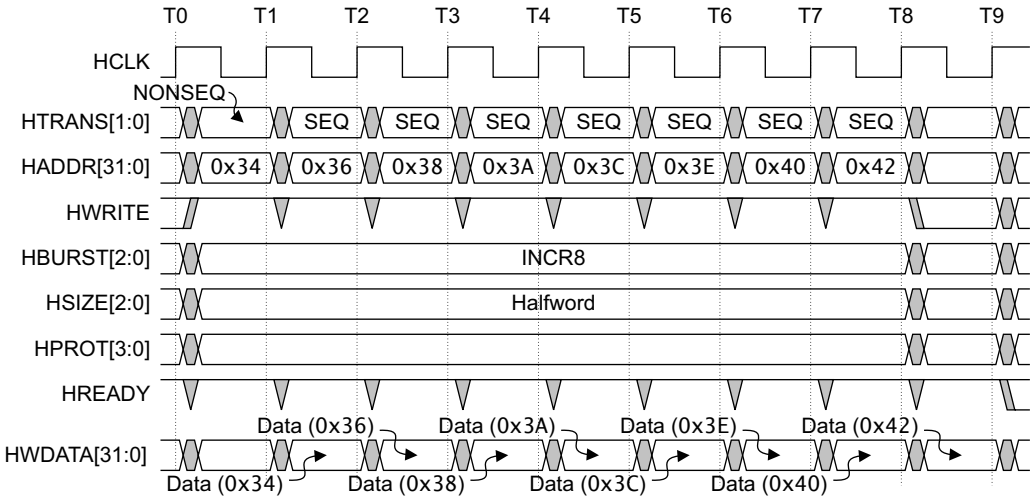


Figure 3-11 Eight-beat incrementing burst

This burst uses halfword transfers, therefore the addresses increase by two. Because the burst is incrementing, the addresses continue to increment beyond the 16-byte address boundary.

Undefined length bursts, INCR

Figure 3-12 on page 3-15 shows incrementing bursts of undefined length.

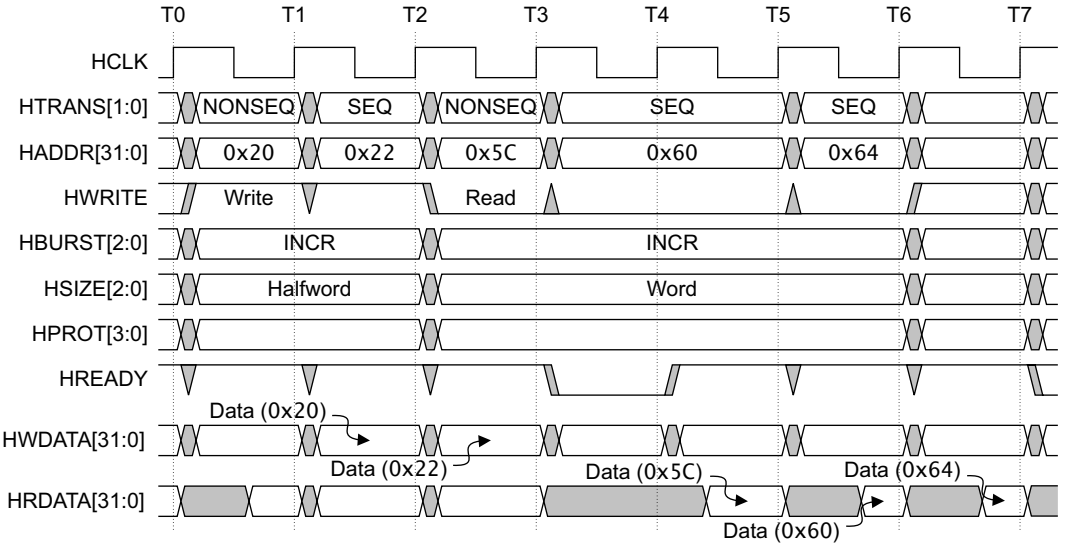


Figure 3-12 Undefined length bursts

Figure 3-12 shows two bursts:

- The first burst is a write consisting of two halfword transfers starting at address `0x20`. These transfer addresses increment by two.
- The second burst is a read consisting of three word transfers starting at address `0x5C`. These transfer addresses increment by four.

3.6 Waited transfers

Slaves use **HREADY** to insert wait states if they require more time to provide or sample the data. During a waited transfer, the master is restricted to what changes it can make to the transfer type and address. These restrictions are described in the following sections:

- *Transfer type changes during wait states*
- *Address changes during wait states* on page 3-19.

3.6.1 Transfer type changes during wait states

When the slave is requesting wait states, the master must not change the transfer type, except as described in:

- *IDLE transfer*
- *BUSY transfer, fixed length burst* on page 3-17
- *BUSY transfer, undefined length burst* on page 3-18.

IDLE transfer

During a waited transfer, the master is permitted to change the transfer type from IDLE to NONSEQ. When the **HTRANS** transfer type changes to NONSEQ the master must keep **HTRANS** constant, until **HREADY** is HIGH.

Figure 3-13 shows a waited transfer for a SINGLE burst, with a transfer type change from IDLE to NONSEQ.

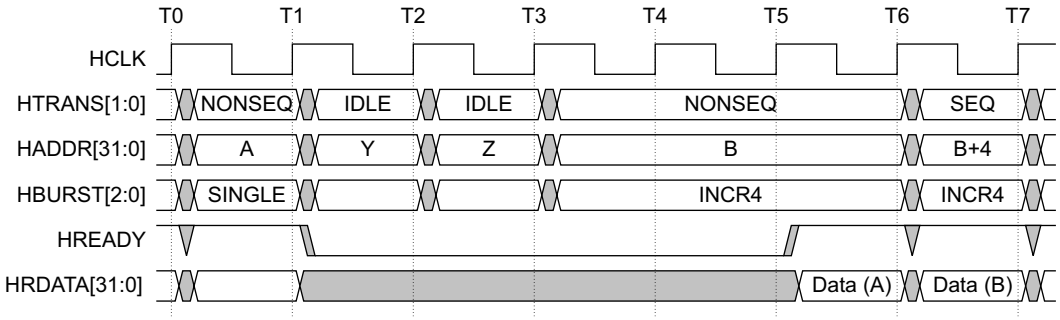


Figure 3-13 Waited transfer, IDLE to NONSEQ

In Figure 3-13:

T0-T1 The master initiates a SINGLE burst to address A.

T1-T2 The master inserts one IDLE transfer to address Y.

The slave inserts a wait state with **HREADY** = LOW.

T2-T3 The master inserts one IDLE transfer to address Z.

T3-T4 The master changes the transfer type to NONSEQ and initiates an INCR4 transfer to address B.

T4-T6 With **HREADY** LOW, the master must keep **HTRANS** constant.

T5-T6 SINGLE burst to address A completes with **HREADY** HIGH and the master starts the first beat to address B.

T6-T7 First beat of the INCR4 transfer to address B completes and the master starts the next beat to address B+4.

BUSY transfer, fixed length burst

During a waited transfer for a fixed length burst, the master is permitted to change the transfer type from BUSY to SEQ. When the **HTRANS** transfer type changes to SEQ the master must keep **HTRANS** constant, until **HREADY** is HIGH.

Note

Because BUSY transfers must only be inserted between successive beats of a burst, this does not apply to SINGLE bursts. Therefore this situation applies to the following burst types:

- INCR4, INCR8, and INCR16
- WRAP4, WRAP8, and WRAP16.

Figure 3-14 shows a waited transfer in a fixed length burst, with a transfer type change from BUSY to SEQ.

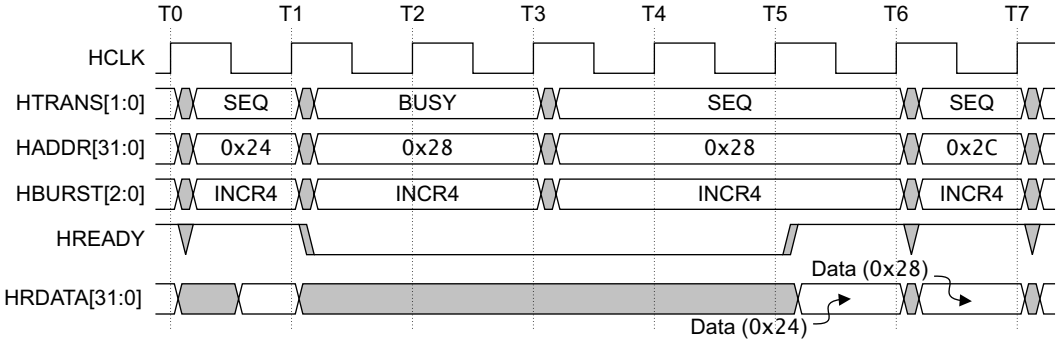


Figure 3-14 Waited transfer, BUSY to SEQ for a fixed length burst

In Figure 3-14 on page 3-17:

T0-T1 The master initiates the next beat of the INCR4 burst to address `0x24`.

T1-T3 The master inserts a BUSY transfer to address `0x28`.

The slave inserts wait states with **HREADY** = LOW.

T3-T4 The master changes the transfer type to SEQ and initiates the next beat of the burst to address `0x28`.

T4-T6 With **HREADY** LOW, the master must keep **HTRANS** constant.

T5-T6 Beat to address `0x24` completes with **HREADY** HIGH.

T6-T7 Third beat of the INCR4 transfer to address `0x28` completes and the master starts the final beat to address `0x2C`.

BUSY transfer, undefined length burst

During a waited transfer for an undefined length burst, INCR, the master is permitted to change from BUSY to any other transfer type, when **HREADY** is LOW. The burst continues if a SEQ transfer is performed but terminates if an IDLE or NONSEQ transfer is performed.

Figure 3-15 shows a waited transfer during an undefined length burst, with a transfer type change from BUSY to NONSEQ.

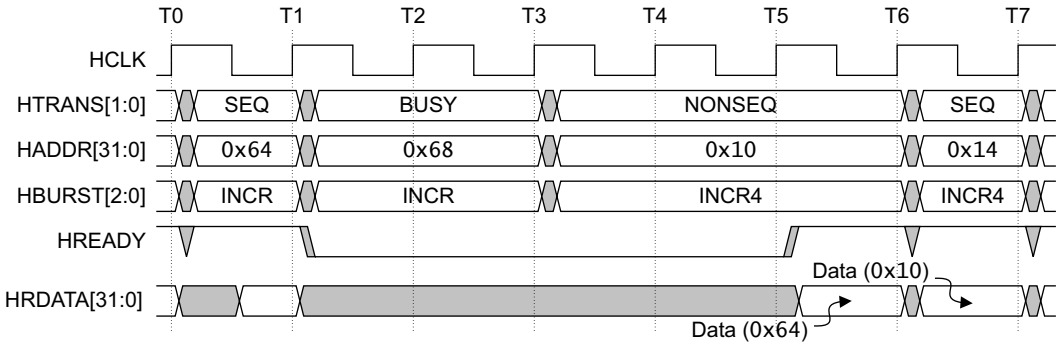


Figure 3-15 Waited transfer, BUSY to NONSEQ for an undefined length burst

In Figure 3-15:

T0-T1 The master initiates the next beat of the INCR burst to address `0x64`.

T1-T3 The master inserts a BUSY transfer to address `0x68`.

The slave inserts wait states with **HREADY** = LOW.

T3-T4 The master changes the transfer type to NONSEQ and initiates a new burst to address `0x10`.

T4-T6 With **HREADY** LOW, the master must keep **HTRANS** constant.

T5-T6 Undefined length burst completes with **HREADY** HIGH and the master starts the first beat to address `0x10`.

T6-T7 First beat of the INCR4 transfer to address `0x10` completes and the master starts the next beat to address `0x14`.

3.6.2 Address changes during wait states

When the slave is requesting wait states, the master can only change the address once, except as described in:

- *During an IDLE transfer*
- *After an ERROR response* on page 3-20.

During an IDLE transfer

During a waited transfer, the master is permitted to change the address for IDLE transfers. When the **HTRANS** transfer type changes to NONSEQ the master must keep the address constant, until **HREADY** is HIGH.

Figure 3-16 shows a waited transfer for a SINGLE burst, with the address changing during the IDLE transfers.

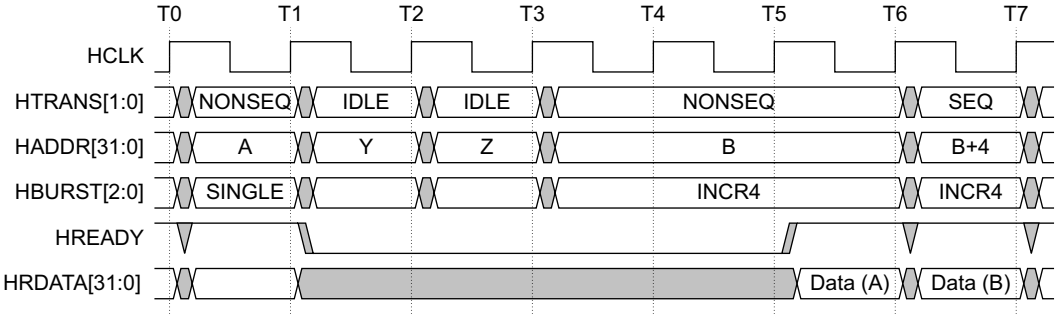


Figure 3-16 Address changes during a waited transfer, with an IDLE transfer

In Figure 3-16:

T0-T1 The master initiates a SINGLE burst to address A.

T1-T2 The master inserts one IDLE transfer to address Y.

The slave inserts a wait state with **HREADY** = LOW.

T2-T3 The master inserts one IDLE transfer to address Z.

T3-T4 The master changes the transfer type to NONSEQ and initiates an INCR4 transfer to address B. Until **HREADY** goes HIGH, no more address changes are permitted.

T5-T6 SINGLE burst to address A completes with **HREADY** HIGH and the master starts the first beat to address B.

T6-T7 First beat of the INCR4 transfer to address B completes and the master starts the next beat to address B+4.

After an ERROR response

During a waited transfer, if the slave responds with an ERROR response then the master is permitted to change the address when **HREADY** is LOW. See *ERROR response* on page 5-3 for more information about the ERROR response.

Figure 3-17 shows a waited transfer, with the address changing following an ERROR response from the slave.

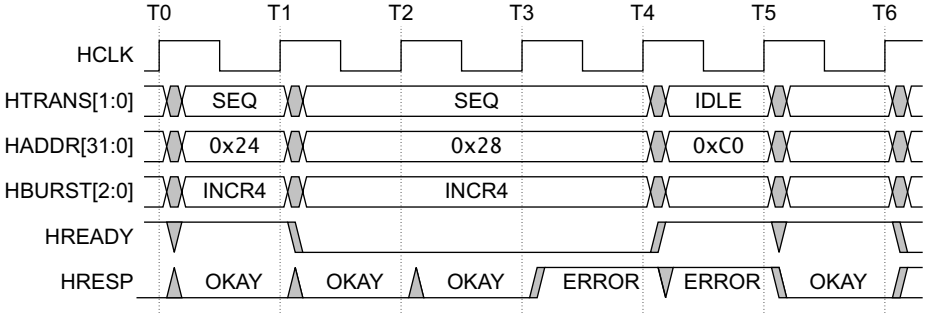


Figure 3-17 Address changes during a waited transfer, after an ERROR

In Figure 3-17:

T0-T1 The master initiates the next beat of the burst to address `0x24`.

T1-T3 The master initiates the next beat of the burst to address `0x28`.

The slave responds with OKAY.

T3-T4 The slave responds with ERROR.

T4-T5 The master changes the transfer type to IDLE and is permitted to change the address while **HREADY** is LOW.

The slave completes the ERROR response.

T5-T6 The slave at address `0xC0` responds with OKAY.

3.7 Protection control

The protection control signals, **HPROT[3:0]**, provide additional information about a bus access and are primarily intended for use by any module that implements some level of protection.

The signals indicate if the transfer is:

- an opcode fetch or data access
- a privileged mode access or user mode access.

For masters with a memory management unit these signals also indicate if the current access is cacheable or bufferable. Table 3-4 lists the **HPROT** signal encoding.

Table 3-4 Protection signal encoding

HPROT[3] Cacheable	HPROT[2] Bufferable	HPROT[1] Privileged	HPROT[0] Data/Opcode	Description
-	-	-	0	Opcode fetch
-	-	-	1	Data access
-	-	0	-	User access
-	-	1	-	Privileged access
-	0	-	-	Non-bufferable
-	1	-	-	Bufferable
0	-	-	-	Non-cacheable
1	-	-	-	Cacheable

Note

Many masters are not capable of generating accurate protection information. If a master is not capable of generating accurate protection information, ARM Limited recommends that:

- the master sets **HPROT** to b0011 to correspond to a non-cacheable, non-bufferable, privileged, data access
- slaves do not use **HPROT** unless absolutely necessary.

The **HPROT** control signals have exactly the same timing as the address bus. However, they must remain constant throughout a burst transfer.

Chapter 4
Bus Interconnection

This chapter describes the additional interconnect logic required for AHB-Lite systems. It contains the following sections:

4.1 Address decoding

A central address decoder provides a select signal, **HSELx**, for each slave on the bus. The select signal is a combinatorial decode of the high-order address signals. Simple address decoding schemes are encouraged to avoid complex decode logic and to ensure high-speed operation.

A slave must only sample the **HSELx**, address, and control signals when **HREADY** is HIGH, indicating that the current transfer is completing. Under certain circumstances it is possible that **HSELx** is asserted when **HREADY** is LOW, but the selected slave has changed by the time the current transfer completes.

The minimum address space that can be allocated to a single slave is 1KB. All masters are designed so that they do not perform incrementing transfers over a 1KB address boundary. This ensures that a burst never crosses an address decode boundary.

Figure 4-1 shows the **HSELx** slave select signals generated by the decoder.

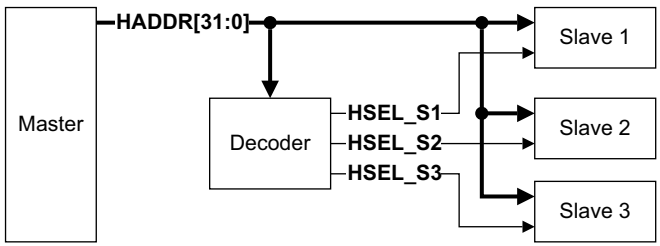


Figure 4-1 Slave select signals

4.1.1 Default slave

If a system design does not contain a completely filled memory map then you must implement an additional default slave to provide a response when any of the nonexistent address locations are accessed.

If a NONSEQUENTIAL or SEQUENTIAL transfer is attempted to a nonexistent address location then the default slave provides an ERROR response.

IDLE or BUSY transfers to nonexistent locations result in a zero wait state OKAY response.

4.2 Bus interconnection

The AHB-Lite protocol is used with a central read data multiplexor interconnection scheme. The master drives out the address and control signals to all the slaves, with the decoder selecting the appropriate slave. Any response data from the selected slave, passes through the read data multiplexor to the master.

Figure 4-2 shows the multiplexor interconnection structure required to implement an AHB-Lite design with three slaves.

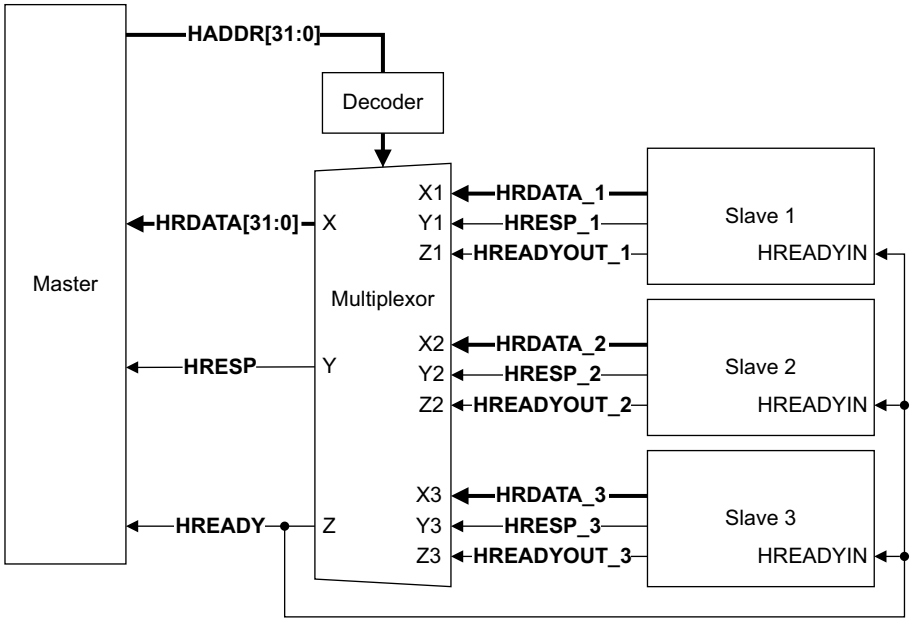


Figure 4-2 Multiplexor interconnection

Chapter 5
Slave Response Signaling

This chapter describes the slave response signaling. It contains the following section:

- *Slave transfer responses* on page 5-2.

5.1 Slave transfer responses

After a master has started a transfer, the slave controls how the transfer progresses. A master cannot cancel a transfer after it has commenced.

A slave must provide a response that indicates the status of the transfer when it is accessed. The transfer status is provided by the **HRESP** signal. Table 5-1 lists the **HRESP** states.

Table 5-1 HRESP signal

HRESP	Response	Description
0	OKAY	The transfer has either completed successfully or additional cycles are required for the slave to complete the request. The **HREADY** signal indicates whether the transfer is pending or complete.
1	ERROR	An error has occurred during the transfer. The error condition must be signaled to the master so that it is aware the transfer has been unsuccessful. A two-cycle response is required for an error condition with **HREADY** being asserted in the second cycle.

Table 5-1 shows that the complete transfer response is a combination of the **HRESP** and **HREADY** signals. Table 5-2 lists the complete transfer response based on the status of these two signals.

Table 5-2 Transfer response

	HREADY	
HRESP	**0**	**1**
0	Transfer pending	Successful transfer completed
1	ERROR response, first cycle	ERROR response, second cycle

This means the slave can complete the transfer in the following three ways:

- immediately complete the transfer
- insert one or more wait states to enable time to complete the transfer
- signal an error to indicate that the transfer has failed.

These three slave transfer responses are described in:

- *Transfer done* on page 5-3
- *Transfer pending* on page 5-3
- *ERROR response* on page 5-3.

5.1.1 Transfer done

A successful completed transfer is signaled when **HREADY** is HIGH and **HRESP** is OKAY.

5.1.2 Transfer pending

A typical slave uses **HREADY** to insert the appropriate number of wait states into the data phase of the transfer. The transfer then completes with **HREADY** HIGH and an OKAY response to indicate the successful completion of the transfer.

When a slave inserts a number of wait states prior to completing the response, it must drive **HRESP** to OKAY.

Note

In general, every slave must have a predetermined maximum number of wait states that it inserts before it backs off the bus. This enables you to calculate the latency for accessing the bus.

It is recommended that slaves do not insert more than 16 wait states, to prevent any single access locking the bus for a large number of clock cycles. However, this recommendation is not applicable to some devices, for example, a serial boot ROM. This type of device is usually only accessed during system startup and the impact on system performance is negligible if greater than 16 wait states are used.

5.1.3 ERROR response

A slave uses the ERROR response to indicate some form of error condition with the associated transfer. Usually this denotes a protection error such as an attempt to write to a read-only memory location.

Although an OKAY response can be given in a single cycle, the ERROR response requires two cycles. To start the ERROR response, the slave drives **HRESP** HIGH to indicate ERROR while driving **HREADY** LOW to extend the transfer for one extra cycle. In the next cycle **HREADY** is driven HIGH to end the transfer and **HRESP** remains driven HIGH to indicate ERROR.

The two-cycle response is required because of the pipelined nature of the bus. By the time a slave starts to issue an ERROR response then the address for the following transfer has already been broadcast onto the bus. The two-cycle response provides sufficient time for the master to cancel this next access and drive **HTRANS[1:0]** to IDLE before the start of the next transfer.

If the slave requires more than two cycles to provide the ERROR response then additional wait states can be inserted at the start of the transfer. During this time **HREADY** is LOW and the response must be set to OKAY.

Figure 5-1 shows a transfer with an ERROR response.

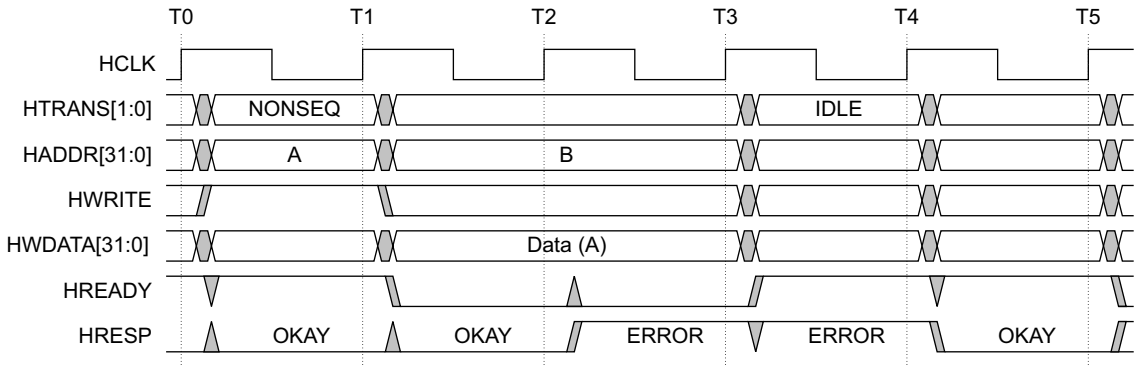


Figure 5-1 ERROR response

In Figure 5-1:

T1-T2 The slave inserts a wait state and provides an OKAY response.

T2-T3 The slave issues an ERROR response. This is the first cycle of the ERROR response because **HREADY** is LOW.

T3-T4 The slave issues an ERROR response. This is the last cycle of the ERROR response because **HREADY** is now HIGH.

The master changes the transfer type to IDLE. This cancels the intended transaction to address B, that was registered by a slave at time T2.

T4-T5 Slave responds with an OKAY response.

If a slave provides an ERROR response then the master can cancel the remaining transfers in the burst. However, this is not a strict requirement and it is also acceptable for the master to continue the remaining transfers in the burst.

Chapter 6
Data Buses

This chapter describes the AHB-Lite data buses. It contains the following sections:

6.1 Data buses

Separate read and write data buses are required to implement an AHB-Lite system without using tristate drivers. Although the recommended minimum data bus width is specified as 32 bits, you can change this as described in *Data bus width* on page 6-5. The data buses are described in:

- *HWDATA*
- *HRDATA*
- *Endianness* on page 6-3.

6.1.1 HWDATA

The master drives the write data bus during write transfers. If the transfer is extended then the master must hold the data valid until the transfer completes, as indicated by **HREADY** HIGH.

For transfers that are narrower than the width of the bus, for example a 16-bit transfer on a 32-bit bus, the master only has to drive the appropriate byte lanes. The slave selects the write data from the correct byte lanes.

Table 6-1 on page 6-3 and Table 6-2 on page 6-3 list which byte lanes on a 32-bit bus, are active for a little-endian and big-endian system respectively.

The active byte lane is dependent on the endianness of the system. Because AHB-Lite does not specify the required endianness, it is important that all masters and slaves on the bus use the same endianness.

6.1.2 HRDATA

The appropriate slave drives the read data bus during read transfers. If the slave extends the read transfer by holding **HREADY** LOW then the slave only has to provide valid data in the final cycle of the transfer, as indicated by **HREADY** HIGH.

For transfers that are narrower than the width of the bus, the slave only requires to provide valid data on the active byte lanes. The master selects the data from the correct byte lanes.

A slave only has to provide valid data when a transfer completes with an OKAY response. ERROR responses do not require valid read data.

Table 6-1 lists the byte lanes on a 32-bit bus that are active in a little-endian system.

Table 6-1 Active byte lanes for a 32-bit little-endian data bus

Transfer size	Address offset	DATA[31:24]	DATA[23:16]	DATA[15:8]	DATA[7:0]
Word	0	Active	Active	Active	Active
Halfword	0	-	-	Active	Active
Halfword	2	Active	Active	-	-
Byte	0	-	-	-	Active
Byte	1	-	-	Active	-
Byte	2	-	Active	-	-
Byte	3	Active	-	-	-

Table 6-2 lists the byte lanes on a 32-bit bus that are active in a big-endian system.

Table 6-2 Active byte lanes for a 32-bit big-endian data bus

Transfer size	Address offset	DATA[31:24]	DATA[23:16]	DATA[15:8]	DATA[7:0]
Word	0	Active	Active	Active	Active
Halfword	0	Active	Active	-	-
Halfword	2	-	-	Active	Active
Byte	0	Active	-	-	-
Byte	1	-	Active	-	-
Byte	2	-	-	Active	-
Byte	3	-	-	-	Active

If required, you can extend the 32-bit data bus listed in Table 6-1 and Table 6-2 for wider data bus implementations. Burst transfers that have a transfer size less than the width of the data bus have different active byte lanes for each beat of the burst.

6.1.3 Endianness

It is essential that all modules are of the same endianness and also that any data routing or bridges are of the same endianness for the system to function correctly.

Dynamic endianness is not supported, because in the majority of embedded systems, this leads to a significant redundant silicon overhead.

It is recommended that only modules designed for use in a wide variety of applications are made bi-endian, with either a configuration pin or internal control bit to select the endianness. For more application-specific blocks, fixing the endianness to either little-endian or big-endian results in a smaller, lower power, higher performance interface.

6.2 Data bus width

One method to improve bus bandwidth without increasing the frequency of operation is to make the data path of the on-chip bus wider. The increased layers of metal and the use of large on-chip memory blocks such as embedded DRAM are driving factors that encourage the use of wider on-chip buses.

Specifying a fixed width of bus means that, in many cases, the width of the bus is not optimal for the application. Therefore an approach has been adopted that enables flexibility of the width of bus but still ensures that modules are highly portable between designs.

The protocol allows the AHB-Lite data bus to be 8, 16, 32, 64, 128, 256, 512, or 1024-bits wide. However, it is recommended that you use a minimum bus width of 32 bits. A maximum bus width of 256 bits is adequate for almost all applications.

For read and write transfers, the receiving module must select the data from the correct byte lane on the bus. Replication of data across all byte lanes is not required.

The following sections describe:

- *Implementing a narrow slave on a wide bus*
- *Implementing a wide slave on a narrow bus* on page 6-6
- *Implementing a master on a wide bus* on page 6-6.

6.2.1 Implementing a narrow slave on a wide bus

Figure 6-1 shows how a slave module that has been originally designed to operate with a 32-bit data bus can be converted to operate on a 64-bit bus. This only requires the addition of external logic, rather than any internal design changes, the technique is therefore applicable to hard macrocells.

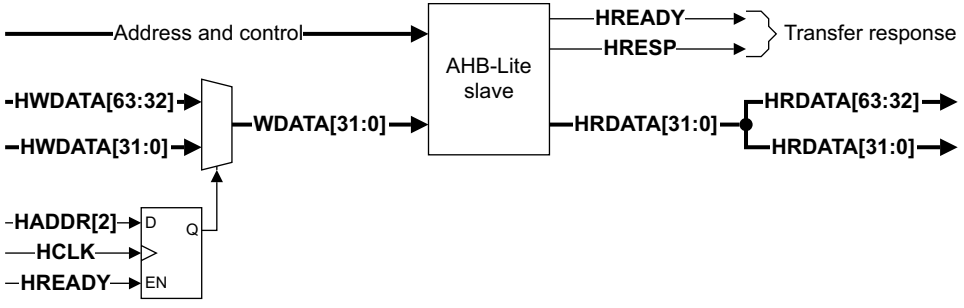


Figure 6-1 Narrow slave on a wide bus

For the output, when converting a narrow bus to a wider bus, do one of the following:

- replicate the data on both halves of the wide bus as Figure 6-1 on page 6-5 shows
- use additional logic to ensure that only the appropriate half of the bus is changed. This results in a reduction of power consumption.

A slave can only accept transfers that are as wide as its natural interface. If a master attempts a transfer that is wider than the slave can support then the slave can use the ERROR transfer response.

6.2.2 Implementing a wide slave on a narrow bus

You can adapt predesigned or imported slaves to work with a narrower data bus by using external logic. Figure 6-2 shows a wide slave being implemented on a narrow bus.

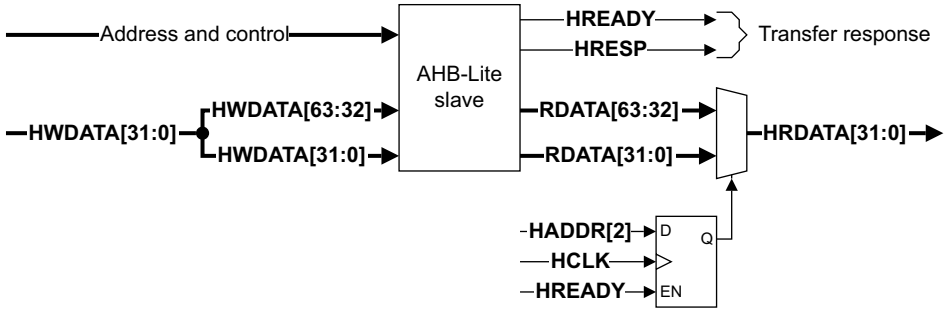


Figure 6-2 Wide slave on a narrow bus

6.2.3 Implementing a master on a wide bus

You can modify masters to work on a wider bus than originally intended in the same way that the slave is modified to work on a wider bus. Do this by:

- multiplexing the input bus
- replicating the output bus.

Note

You cannot make masters work on a narrower bus than originally intended unless you include some mechanism in the master to limit the width of transfers that the master attempts. The master must never attempt a transfer where the width, as indicated by **HSIZE**, is wider than the data bus that it connects to.

Chapter 7
Clock and Reset

This chapter describes the timing of the protocol clock and reset signals. It contains the following section:

7.1 Clock and reset requirements

This section describes the requirements for implementing the **HCLK** and **HRESETn** signals.

7.1.1 Clock

Each AHB-Lite component uses a single clock signal, **HCLK**. All input signals are sampled on the rising edge of **HCLK**. All output signal changes must occur after the rising edge of **HCLK**.

7.1.2 Reset

The reset signal, **HRESETn**, is the only active LOW signal in the AHB-Lite protocol and is the primary reset for all bus elements. The reset can be asserted asynchronously, but is deasserted synchronously after the rising edge of **HCLK**.

During reset all masters must ensure the address and control signals are at valid levels and that **HTRANS[1:0]** indicates IDLE.

During reset all slaves must ensure that **HREADYOUT** is HIGH.

Glossary

This glossary describes some of the terms used in technical documents from ARM Limited.

Advanced eXtensible Interface (AXI)
A bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced High-performance Bus (AHB)
A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and it is recommended that only a subset of the protocol is used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)
: A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules.

Advanced Peripheral Bus (APB)
: A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

AHB
: *See* Advanced High-performance Bus.

AHB-Lite
: A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently by using an AMBA AXI protocol interface.

Aligned
: A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.

AMBA
: *See* Advanced Microcontroller Bus Architecture.

APB
: *See* Advanced Peripheral Bus.

AXI
: *See* Advanced eXtensible Interface.

Beat
: Alternative word for an individual transfer in a burst. For example, an INCR4 burst comprises four beats.

  See also Burst.

Big-endian
: Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory.

  See also Little-endian and Endianness.

Burst
: A group of transfers to consecutive addresses. Bursts over AMBA are controlled using signals to indicate the length of the burst and how the addresses are incremented.

  See also Beat.

Byte An 8-bit data item.

Direct Memory Access (DMA)
An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.

DMA *See* Direct Memory Access.

Doubleword A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.

Endianness Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping.

See also Little-endian and Big-endian.

Halfword A 16-bit data item.

Little-endian Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.

See also Big-endian and Endianness.

Multi-master AHB Typically a shared, not multi-layer, AHB interconnect scheme. More than one master connects to a single AMBA AHB link. In this case, the bus is implemented with a set of full AMBA AHB master interfaces. Masters that use the AMBA AHB-Lite protocol must connect through a wrapper to supply full AMBA AHB master signals to support multi-master operation.

Processor A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

Unaligned A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

Word A 32-bit data item.