

# AMBA™ Specification

(Rev 2.0)

**ARM**

# AMBA Specification (Rev 2.0)

© Copyright ARM Limited 1999. All rights reserved.

## Release information

## Change history

Date	Issue	Change
13th May 1999	A	First release

## Proprietary notice

ARM, the ARM Powered logo, Thumb and StrongARM are registered trademarks of ARM Limited.

The ARM logo, AMBA, PrimeCell, Angel, ARMulator, EmbeddedICE, ModelGen, Multi-ICE, ARM7TDMI, ARM7TDMI-S, ARM9TDMI, TDMI and STRONG are trademarks of ARM Limited.

All other products or services mentioned herein may be trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM Limited in good faith. However, all warranties implied or expressed, including but not limited to implied warranties or merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

## Document confidentiality status

This document is Open Access. This document has no restriction on distribution.

## Product status

The information in this document is Final (information on a developed product).

## ARM web address

<http://www.arm.com>

# Preface

This preface introduces the *Advanced Microcontroller Bus Architecture* (AMBA) specification. It contains the following sections:

- *About this document* on page iv
- *Feedback* on page vii.

## About this document

This document is the AMBA specification.

## Intended audience

This document has been written to help experienced hardware and software engineers to design modules that conform to the AMBA specification.

## Organization

This document is organized into the following chapters:

**Chapter 1** *Introduction to the AMBA Buses*

Read this chapter for an overview of the AMBA buses.

**Chapter 2** *AMBA Signals*

Read this chapter for a description of the signals used by AMBA devices.

**Chapter 3** *AMBA AHB*

Read this chapter for an introduction to the AMBA Advanced High-performance Bus.

**Chapter 4** *AMBA ASB*

Read this chapter for an introduction to the AMBA Advanced System Bus.

**Chapter 5** *AMBA APB*

Read this chapter for an introduction to the AMBA Advanced Peripheral Bus.

**Chapter 6** *AMBA Test Methodology*

Read this chapter for an introduction to the test methodology used in AMBA buses.

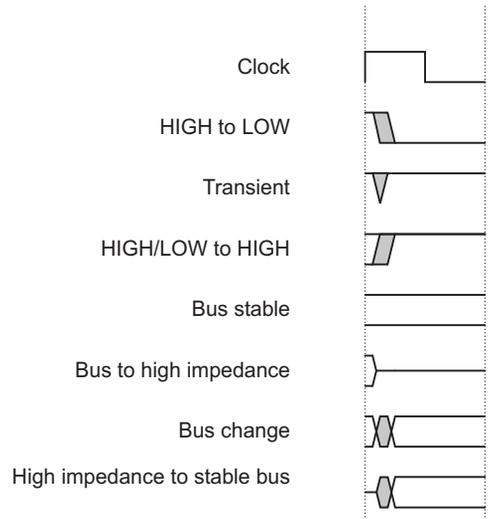
## Typographical conventions

The following typographical conventions are used in this document:

<b>bold</b>	Highlights ARM processor signal names within text, and interface elements such as menu names. May also be used for emphasis in descriptive lists where appropriate.
<i>italic</i>	Highlights special terminology, cross-references and citations.
<code>typewriter</code>	Denotes text that may be entered at the keyboard, such as commands, file names and program names, and source code.
<u>typewriter</u>	Denotes a permitted abbreviation for a command or option. The underlined text may be entered instead of the full command or option name.
<code><i>typewriter italic</i></code>	Denotes arguments to commands or functions where the argument is to be replaced by a specific value.
<b><code>typewriter bold</code></b>	Denotes language keywords when used outside example code.

## Timing diagram conventions

This manual contains one or more timing diagrams. The following key explains the components used in these diagrams. Any variations are clearly labelled when they occur. Therefore, no additional meaning should be attached unless specifically stated.



### Key to timing diagram conventions

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

## Feedback

ARM Limited welcomes feedback both on AMBA and the AMBA specification.

### Feedback on this document

If you have any comments on this document, please send email to [errata@arm.com](mailto:errata@arm.com) giving:

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

General suggestions for additions and improvements are also welcome.

### Feedback on the AMBA Specification

If you have any comments or suggestions about this product, please contact your supplier giving:

- the product name
- a concise explanation of your comments.



# Contents

## AMBA Specification

	<b>Preface</b>	
	About this document .....	iv
	Feedback .....	vii
<b>Chapter 1</b>	<b>Introduction to the AMBA Buses</b>	
	1.1 Overview of the AMBA specification .....	1-2
	1.2 Objectives of the AMBA specification .....	1-3
	1.3 A typical AMBA-based microcontroller .....	1-4
	1.4 Terminology .....	1-6
	1.5 Introducing the AMBA AHB .....	1-7
	1.6 Introducing the AMBA ASB .....	1-9
	1.7 Introducing the AMBA APB .....	1-10
	1.8 Choosing the right bus for your system.....	1-12
	1.9 Notes on the AMBA specification.....	1-14
<b>Chapter 2</b>	<b>AMBA Signals</b>	
	2.1 AMBA signal names.....	2-2
	2.2 AMBA AHB signal list.....	2-3
	2.3 AMBA ASB signal list.....	2-6
	2.4 AMBA APB signal list.....	2-8

## Chapter 3

### AMBA AHB

3.1	About the AMBA AHB.....	3-3
3.2	Bus interconnection .....	3-4
3.3	Overview of AMBA AHB operation .....	3-5
3.4	Basic transfer.....	3-6
3.5	Transfer type .....	3-9
3.6	Burst operation .....	3-11
3.7	Control signals.....	3-17
3.8	Address decoding.....	3-19
3.9	Slave transfer responses.....	3-20
3.10	Data buses .....	3-25
3.11	Arbitration .....	3-28
3.12	Split transfers.....	3-35
3.13	Reset .....	3-40
3.14	About the AHB data bus width.....	3-41
3.15	Implementing a narrow slave on a wider bus .....	3-42
3.16	Implementing a wide slave on a narrow bus .....	3-43
3.17	About the AHB AMBA components .....	3-44
3.18	AHB bus slave .....	3-45
3.19	AHB bus master .....	3-49
3.20	AHB arbiter .....	3-53
3.21	AHB decoder .....	3-57

## Chapter 4

### AMBA ASB

4.1	About the AMBA ASB.....	4-2
4.2	AMBA ASB description .....	4-4
4.3	ASB transfers .....	4-6
4.4	Address decode.....	4-14
4.5	Transfer response .....	4-16
4.6	Multi-master operation.....	4-19
4.7	Reset operation .....	4-23
4.8	Description of ASB signals .....	4-25
4.9	About the ASB AMBA components .....	4-46
4.10	ASB bus slave .....	4-47
4.11	ASB bus master.....	4-52
4.12	ASB decoder .....	4-63
4.13	ASB arbiter .....	4-71

## Chapter 5

### AMBA APB

5.1	About the AMBA APB.....	5-2
5.2	APB specification.....	5-4
5.3	About the APB AMBA components .....	5-7
5.4	APB bridge .....	5-8
5.5	APB slave .....	5-11
5.6	Interfacing APB to AHB .....	5-14
5.7	Interfacing APB to ASB .....	5-20
5.8	Interfacing rev D APB peripherals to rev 2.0 APB .....	5-22

## Chapter 6

### AMBA Test Methodology

6.1	About the AMBA test interface .....	6-2
6.2	External interface .....	6-4
6.3	Test vector types.....	6-6
6.4	Test interface controller .....	6-7
6.5	The AHB Test Interface Controller .....	6-12
6.6	Example AMBA AHB test sequences .....	6-17
6.7	The ASB test interface controller .....	6-25
6.8	Example AMBA ASB test sequences.....	6-27

### Index



# Chapter 1

## Introduction to the AMBA Buses

This chapter introduces the *Advanced Microcontroller Bus Architecture* (AMBA) specification. The following sections are included:

- *Overview of the AMBA specification* on page 1-2
- *Objectives of the AMBA specification* on page 1-3
- *A typical AMBA-based microcontroller* on page 1-4
- *Terminology* on page 1-6
- *Introducing the AMBA AHB* on page 1-7
- *Introducing the AMBA ASB* on page 1-9
- *Introducing the AMBA APB* on page 1-10
- *Choosing the right bus for your system* on page 1-12
- *Notes on the AMBA specification* on page 1-14.

## 1.1 Overview of the AMBA specification

The *Advanced Microcontroller Bus Architecture* (AMBA) specification defines an on-chip communications standard for designing high-performance embedded microcontrollers.

Three distinct buses are defined within the AMBA specification:

- the *Advanced High-performance Bus* (AHB)
- the *Advanced System Bus* (ASB)
- the *Advanced Peripheral Bus* (APB).

A test methodology is included with the AMBA specification which provides an infrastructure for modular macrocell test and diagnostic access.

### 1.1.1 Advanced High-performance Bus (AHB)

The AMBA AHB is for high-performance, high clock frequency system modules.

The AHB acts as the high-performance system *backbone* bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques.

### 1.1.2 Advanced System Bus (ASB)

The AMBA ASB is for high-performance system modules.

AMBA ASB is an alternative system bus suitable for use where the high-performance features of AHB are not required. ASB also supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions.

### 1.1.3 Advanced Peripheral Bus (APB)

The AMBA APB is for low-power peripherals.

AMBA APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. APB can be used in conjunction with either version of the system bus.

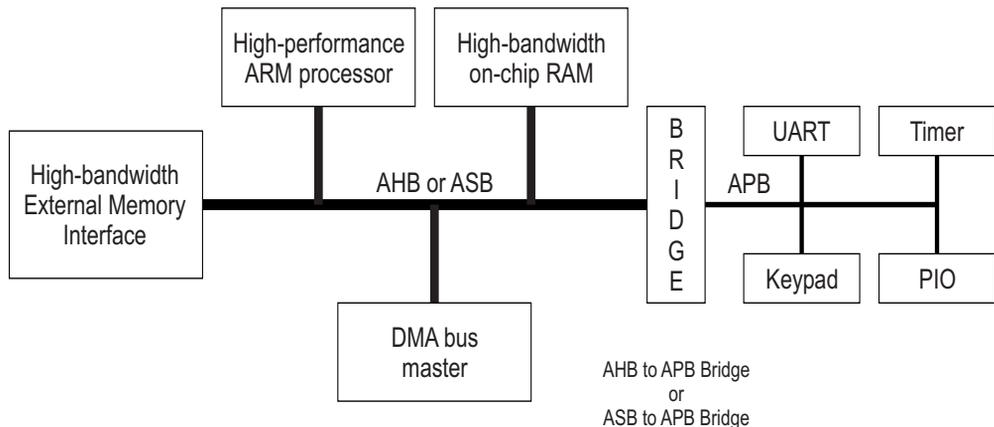
## 1.2 Objectives of the AMBA specification

The AMBA specification has been derived to satisfy four key requirements:

- to facilitate the *right-first-time* development of embedded microcontroller products with one or more CPUs or signal processors
- to be *technology-independent* and ensure that highly reusable peripheral and system macrocells can be migrated across a diverse range of IC processes and be appropriate for full-custom, standard cell and gate array technologies
- to encourage *modular system design* to improve processor independence, providing a development road-map for advanced cached CPU cores and the development of peripheral libraries
- to minimize the silicon infrastructure required to support efficient on-chip and off-chip communication for both operation and manufacturing test.

### 1.3 A typical AMBA-based microcontroller

An AMBA-based microcontroller typically consists of a high-performance system *backbone* bus (AMBA AHB or AMBA ASB), able to sustain the external memory bandwidth, on which the CPU, on-chip memory and other *Direct Memory Access* (DMA) devices reside. This bus provides a high-bandwidth interface between the elements that are involved in the majority of transfers. Also located on the high-performance bus is a bridge to the lower bandwidth APB, where most of the peripheral devices in the system are located (see Figure 1-1).



#### AMBA AHB

- \* High performance
- \* Pipelined operation
- \* Multiple bus masters
- \* Burst transfers
- \* Split transactions

#### AMBA ASB

- \* High performance
- \* Pipelined operation
- \* Multiple bus masters

#### AMBA APB

- \* Low power
- \* Latched address and control
- \* Simple interface
- \* Suitable for many peripherals

**Figure 1-1 A typical AMBA system**

AMBA APB provides the basic peripheral macrocell communications infrastructure as a secondary bus from the higher bandwidth pipelined main system bus. Such peripherals typically:

- have interfaces which are memory-mapped registers
- have no high-bandwidth interfaces
- are accessed under programmed control.

The external memory interface is application-specific and may only have a narrow data path, but may also support a test access mode which allows the internal AMBA AHB, ASB and APB modules to be tested in isolation with system-independent test sets.

## 1.4 Terminology

The following terms are used throughout this specification.

- Bus cycle** A bus cycle is a basic unit of one bus clock period and for the purpose of AMBA AHB or APB protocol descriptions is defined from rising-edge to rising-edge transitions. An ASB bus cycle is defined from falling-edge to falling-edge transitions. Bus signal timing is referenced to the bus cycle clock.
- Bus transfer** An AMBA ASB or AHB bus transfer is a read or write operation of a data object, which may take one or more bus cycles. The bus transfer is terminated by a *completion* response from the addressed slave.
- The transfer sizes supported by AMBA ASB include byte (8-bit), halfword (16-bit) and word (32-bit). AMBA AHB additionally supports wider data transfers, including 64-bit and 128-bit transfers. An AMBA APB bus transfer is a read or write operation of a data object, which always requires two bus cycles.
- Burst operation** A burst operation is defined as one or more data transactions, initiated by a bus master, which have a consistent width of transaction to an incremental region of address space. The increment step per transaction is determined by the width of transfer (byte, halfword, word). No burst operation is supported on the APB.

## 1.5 Introducing the AMBA AHB

AHB is a new generation of AMBA bus which is intended to address the requirements of high-performance synthesizable designs. It is a high-performance system bus that supports multiple bus masters and provides high-bandwidth operation.

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

- burst transfers
- split transactions
- single-cycle bus master handover
- single-clock edge operation
- non-tristate implementation
- wider data bus configurations (64/128 bits).

Bridging between this higher level of bus and the current ASB/APB can be done efficiently to ensure that any existing designs can be easily integrated.

An AMBA AHB design may contain one or more bus masters, typically a system would contain at least the processor and test interface. However, it would also be common for a *Direct Memory Access* (DMA) or *Digital Signal Processor* (DSP) to be included as bus masters.

The external memory interface, APB bridge and any internal memory are the most common AHB slaves. Any other peripheral in the system could also be included as an AHB slave. However, low-bandwidth peripherals typically reside on the APB.

A typical AMBA AHB system design contains the following components:

- |                    |  |
|--------------------|--|
| <b>AHB master</b>  | A bus master is able to initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at any one time.  |
| <b>AHB slave</b>   | A bus slave responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.  |
| <b>AHB arbiter</b> | The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as <i>highest priority</i> or <i>fair</i> access can be implemented depending on the application requirements.<br><br>An AHB would include only one arbiter, although this would be trivial in single bus master systems. |

**AHB decoder**

The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer.

A single centralized decoder is required in all AHB implementations.

## 1.6 Introducing the AMBA ASB

ASB is the first generation of AMBA system bus. ASB sits above the current APB and implements the features required for high-performance systems including:

- burst transfers
- pipelined transfer operation
- multiple bus master.

A typical AMBA ASB system may contain one or more bus masters. For example, at least the processor and test interface. However, it would also be common for a *Direct Memory Access* (DMA) or *Digital Signal Processor* (DSP) to be included as bus masters.

The external memory interface, APB bridge and any internal memory are the most common ASB slaves. Any other peripheral in the system could also be included as an ASB slave. However, low-bandwidth peripherals typically reside on the APB.

An AMBA ASB system design typically contains the following components:

<b>ASB master</b>	A bus master is able to initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at any one time.
<b>ASB slave</b>	A bus slave responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.
<b>ASB decoder</b>	The bus decoder performs the decoding of the transfer addresses and selects slaves appropriately. The bus decoder also ensures that the bus remains operational when no bus transfers are required. A single centralized decoder is required in all ASB implementations.
<b>ASB arbiter</b>	The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as <i>highest priority</i> or <i>fair</i> access can be implemented depending on the application requirements.  An ASB would include only one arbiter, although this would be trivial in single bus master systems.

## 1.7 Introducing the AMBA APB

The APB is part of the AMBA hierarchy of buses and is optimized for minimal power consumption and reduced interface complexity.

The AMBA APB appears as a local secondary bus that is encapsulated as a single AHB or ASB slave device. APB provides a low-power extension to the system bus which builds on AHB or ASB signals directly.

The APB bridge appears as a slave module which handles the bus handshake and control signal retiming on behalf of the local peripheral bus. By defining the APB interface from the starting point of the system bus, the benefits of the system diagnostics and test methodology can be exploited.

The AMBA APB should be used to interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface.

The latest revision of the APB is specified so that all signal transitions are only related to the rising edge of the clock. This improvement ensures the APB peripherals can be integrated easily into any design flow, with the following advantages:

- high-frequency operation easier to achieve
- performance is independent of the mark-space ratio of the clock
- static timing analysis is simplified by the use of a single clock edge
- no special considerations are required for automatic test insertion
- many *Application Specific Integrated Circuit* (ASIC) libraries have a better selection of rising edge registers
- easy integration with cycle-based simulators.

These changes to the APB also make it simpler to interface it to the new AHB.

An AMBA APB implementation typically contains a single APB bridge which is required to convert AHB or ASB transfers into a suitable format for the slave devices on the APB. The bridge provides latching of all address, data and control signals, as well as providing a second level of decoding to generate slave select signals for the APB peripherals.

All other modules on the APB are APB slaves. The APB slaves have the following interface specification:

- address and control valid throughout the access (unpipelined)

- zero-power interface during non-peripheral bus activity (peripheral bus is static when not in use)
- timing can be provided by decode with strobe timing (unlocked interface)
- write data valid for the whole access (allowing glitch-free transparent latch implementations).

## 1.8 Choosing the right bus for your system

Before deciding on which bus or buses you should use in your system, you should consider the following:

- *Choice of system bus*
- *System bus and peripheral bus*
- *When to use AMBA AHB/ASB or APB on page 1-13*

### 1.8.1 Choice of system bus

Both AMBA AHB and ASB are available for use as the main system bus. Typically the choice of system bus will depend on the interface provided by the system modules required.

The AHB is recommended for all new designs, not only because it provides a higher-bandwidth solution, but also because the single-clock-edge protocol results in a smoother integration with design automation tools used during a typical ASIC development.

### 1.8.2 System bus and peripheral bus

Building all peripherals as fully functional AHB or ASB modules is feasible but may not always be desirable:

- In designs with a large number of peripheral macrocells the increased bus loading may increase power dissipation and sacrifice performance.
- Where timing analysis is required, the slowest element on the bus will limit the maximum performance.
- Many simple peripheral macrocells need latched addresses and control signals as opposed to the high-bandwidth macrocells which benefit from pipelined signalling.
- Many peripheral functions simply require a selection *strobe* which conveys macrocell selection and read/write bus operation, without the requirement to broadcast the high-frequency clock signal to every peripheral.

### 1.8.3 When to use AMBA AHB/ASB or APB

A full AHB or ASB interface is used for:

- bus masters
- on-chip memory blocks
- external memory interfaces
- high-bandwidth peripherals with FIFO interfaces
- DMA slave peripherals.

A simple APB interface is recommended for:

- simple register-mapped slave devices
- very low power interfaces where clocks cannot be globally routed
- grouping narrow-bus peripherals to avoid loading the system bus.

## 1.9 Notes on the AMBA specification

The following points should be considered when reading the AMBA specification:

- *Technology independence*
- *Electrical characteristics*
- *Timing specification.*

### 1.9.1 Technology independence

AMBA is a technology-independent on-chip protocol. The specification only details the bus protocol at the clock cycle level.

### 1.9.2 Electrical characteristics

No information regarding the electrical characteristics is supplied within the AMBA specification as this will be entirely dependent on the manufacturing process technology that is selected for the design.

### 1.9.3 Timing specification

The AMBA protocol defines the behavior of various signals at the cycle level. The exact timing requirements will depend on the process technology used and the frequency of operation.

Because the exact timing requirements are not defined by the AMBA protocol, the system integrator is given maximum flexibility in allocating the signal timing budget amongst the various modules on the bus.

# Chapter 2

## **AMBA Signals**

This chapter introduces the AMBA signals. It contains the following sections:

- *AMBA signal names* on page 2-2
- *AMBA AHB signal list* on page 2-3
- *AMBA ASB signal list* on page 2-6
- *AMBA APB signal list* on page 2-8.

## 2.1 AMBA signal names

All AMBA signals are named such that the first letter of the name indicates which bus the signal is associated with. A lower case **n** in the signal name indicates that the signal is active LOW, otherwise signal names are always all upper case.

Test signals have a prefix **T** regardless of the bus type. More information on test signals can be found in Chapter 6 *AMBA Test Methodology*.

### 2.1.1 AHB signal prefixes

**H** indicates an AHB signal.

For example, **HREADY** is the signal used to indicate that the data portion of an AHB transfer can complete. It is active HIGH.

### 2.1.2 ASB signal prefixes

**A** is a unidirectional signal between ASB bus masters and the arbiter

**B** is an ASB signal

**D** is a unidirectional ASB decoder signal.

For example, **BnRES** is the ASB reset signal. It is active LOW.

### 2.1.3 APB signal prefixes

**P** indicates an APB signal.

For example, **PCLK** is the main clock used by the APB.

## 2.2 AMBA AHB signal list

This section contains an overview of the AMBA AHB signals (see Table 2-1). A full description of each of the signals can be found in later sections of this document.

All signals are prefixed with the letter **H**, ensuring that the AHB signals are differentiated from other similarly named signals in a system design.

**Table 2-1 AMBA AHB signals**

Name	Source	Description
<b>HCLK</b> Bus clock	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of <b>HCLK</b> .
<b>HRESETn</b> Reset	Reset controller	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
<b>HADDR[31:0]</b> Address bus	Master	The 32-bit system address bus.
<b>HTRANS[1:0]</b> Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
<b>HWRITE</b> Transfer direction	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.
<b>HSIZE[2:0]</b> Transfer size	Master	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
<b>HBURST[2:0]</b> Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.
<b>HPROT[3:0]</b> Protection control	Master	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.

Table 2-1 AMBA AHB signals (continued)

Name	Source	Description
<b>HWDATA[31:0]</b> Write data bus	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
<b>HSELx</b> Slave select	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.
<b>HRDATA[31:0]</b> Read data bus	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
<b>HREADY</b> Transfer done	Slave	When HIGH the <b>HREADY</b> signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require <b>HREADY</b> as both an input and an output signal.
<b>HRESP[1:0]</b> Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

AMBA AHB also has a number of signals required to support multiple bus master operation (see Table 2-2). Many of these arbitration signals are dedicated point to point links and in Table 2-2 the suffix **x** indicates the signal is from module X. For example there will be a number of **HBUSREQx** signals in a system, such as **HBUSREQarm**, **HBUSREQdma** and **HBUSREQtic**.

Table 2-2 Arbitration signals

Name	Source	Description
<b>HBUSREQx</b> Bus request	Master	A signal from bus master x to the bus arbiter which indicates that the bus master requires the bus. There is an <b>HBUSREQx</b> signal for each bus master in the system, up to a maximum of 16 bus masters.
<b>HLOCKx</b> Locked transfers	Master	When HIGH this signal indicates that the master requires locked access to the bus and no other master should be granted the bus until this signal is LOW.
<b>HGRANTx</b> Bus grant	Arbiter	This signal indicates that bus master x is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when <b>HREADY</b> is HIGH, so a master gets access to the bus when both <b>HREADY</b> and <b>HGRANTx</b> are HIGH.
<b>HMASTER[3:0]</b> Master number	Arbiter	These signals from the arbiter indicate which bus master is currently performing a transfer and is used by the slaves which support SPLIT transfers to determine which master is attempting an access. The timing of <b>HMASTER</b> is aligned with the timing of the address and control signals.
<b>HMASTLOCK</b> Locked sequence	Arbiter	Indicates that the current master is performing a locked sequence of transfers. This signal has the same timing as the <b>HMASTER</b> signal.
<b>HSPLITx[15:0]</b> Split completion request	Slave (SPLIT-capable)	This 16-bit split bus is used by a slave to indicate to the arbiter which bus masters should be allowed to re-attempt a split transaction. Each bit of this split bus corresponds to a single bus master.

## 2.3 AMBA ASB signal list

Table 2-3 lists the AMBA ASB signals.

**Table 2-3 AMBA ASB signals**

<b>Name</b>	<b>Description</b>
<b>AGNTx</b> Bus grant	A signal from the bus arbiter to a bus master <b>x</b> which indicates that the bus master will be granted the bus when <b>BWAIT</b> is LOW. There is an <b>AGNTx</b> signal for each bus master in the system, as well as an associated bus request signal, <b>AREQx</b> .
<b>AREQx</b> Bus request	A signal from bus master <b>x</b> to the bus arbiter which indicates that the bus master requires the bus. There is an <b>AREQx</b> signal for each bus master in the system, as well as an associated bus grant signal, <b>AGNTx</b> .
<b>BA[31:0]</b> Address bus	The system address bus, which is driven by the active bus master.
<b>BCLK</b> Bus clock	This clock times all bus transfers. Both the LOW phase and HIGH phase of <b>BCLK</b> are used to control transfers on the bus.
<b>BD[31:0]</b> Data bus	This is the bidirectional system data bus. The data bus is driven by the current bus master during write transfers and by the selected bus slave during read transfers.
<b>BERROR</b> Error response	A transfer error is indicated by the selected bus slave using the <b>BERROR</b> signal. When <b>BERROR</b> is HIGH a transfer error has occurred, when <b>BERROR</b> is LOW then the transfer is successful. This signal is also used in combination with the <b>BLAST</b> signal to indicate a bus retract operation. When no slave is selected this signal is driven by the bus decoder.
<b>BLAST</b> Last response	This signal is driven by the selected bus slave to indicate if the current transfer should be the last of a burst sequence. When <b>BLAST</b> is HIGH the decoder must allow sufficient time for address decoding. When <b>BLAST</b> is LOW, the next transfer may continue a burst sequence. This signal is also used in combination with the <b>BERROR</b> signal to indicate a bus retract operation. When no slave is selected this signal is driven by the bus decoder.
<b>BLOCK</b> Locked transfers	When HIGH this signal indicates that the current transfer and the next transfer are to be indivisible and no other bus master should be given access to the bus. This signal is used by the bus arbiter. This signal is driven by the active bus master.
<b>BnRES</b> Reset	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.

Table 2-3 AMBA ASB signals (continued)

Name	Description
<b>BPROT[1:0]</b> Protection control	The protection control signals provide additional information about a bus access and are primarily intended for use by a bus decoder when acting as a basic protection unit. The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. The signals are driven by the active bus master and have the same timing as the address bus.
<b>BFSIZE[1:0]</b> Transfer size	The transfer size signals indicate the size of the transfer, which may be byte, halfword or word. The signals are driven by the active bus master and have the same timing as the address bus.
<b>BTRAN[1:0]</b> Transfer type	These signals indicate the type of the next transaction, which may be ADDRESS-ONLY, NONSEQUENTIAL or SEQUENTIAL. These signals are driven by a bus master when the appropriate <b>AGNTx</b> signal is asserted.
<b>BWAIT</b> Wait response	This signal is driven by the selected bus slave to indicate if the current transfer may complete. If <b>BWAIT</b> is HIGH a further bus cycle is required, if <b>BWAIT</b> is LOW then the transfer may complete in the current bus cycle. When no slave is selected this signal is driven by the bus decoder.
<b>BWRITE</b> Transfer direction	When HIGH this signal indicates a write transfer and when LOW a read transfer. This signal is driven by the active bus master and has the same timing as the address bus.
<b>DSELx</b> Slave select	A signal from the bus decoder to a bus slave <b>x</b> which indicates that the slave device is selected and a data transfer is required. There is a <b>DSELx</b> signal for each ASB bus slave.

## 2.4 AMBA APB signal list

All AMBA APB signals use the single letter **P** prefix. Some APB signals, such as the clock, may be connected directly to the system bus equivalent signal.

Table 2-4 shows the list of AMBA APB signal names, along with a description of how each of the signals is used.

**Table 2-4 AMBA APB signals**

<b>Name</b>	<b>Description</b>
<b>PCLK</b> Bus clock	The rising edge of <b>PCLK</b> is used to time all transfers on the APB.
<b>PRESETn</b> APB reset	The APB bus reset signal is active LOW and this signal will normally be connected directly to the system bus reset signal.
<b>PADDR[31:0]</b> APB address bus	This is the APB address bus, which may be up to 32-bits wide and is driven by the peripheral bus bridge unit.
<b>PSELx</b> APB select	A signal from the secondary decoder, within the peripheral bus bridge unit, to each peripheral bus slave x. This signal indicates that the slave device is selected and a data transfer is required. There is a <b>PSELx</b> signal for each bus slave.
<b>PENABLE</b> APB strobe	This strobe signal is used to time all accesses on the peripheral bus. The enable signal is used to indicate the second cycle of an APB transfer. The rising edge of <b>PENABLE</b> occurs in the middle of the APB transfer.
<b>PWRITE</b> APB transfer direction	When HIGH this signal indicates an APB write access and when LOW a read access.
<b>PRDATA</b> APB read data bus	The read data bus is driven by the selected slave during read cycles (when <b>PWRITE</b> is LOW). The read data bus can be up to 32-bits wide.
<b>PWDATA</b> APB write data bus	The write data bus is driven by the peripheral bus bridge unit during write cycles (when <b>PWRITE</b> is HIGH). The write data bus can be up to 32-bits wide.

# Chapter 3

## AMBA AHB

This chapter describes the *Advanced High-performance Bus* (AHB) architecture. It contains the following sections:

- *About the AMBA AHB* on page 3-3
- *Bus interconnection* on page 3-4
- *Overview of AMBA AHB operation* on page 3-5
- *Basic transfer* on page 3-6
- *Transfer type* on page 3-9
- *Burst operation* on page 3-11
- *Control signals* on page 3-17
- *Address decoding* on page 3-19
- *Slave transfer responses* on page 3-20
- *Data buses* on page 3-25
- *Arbitration* on page 3-28
- *Split transfers* on page 3-35
- *Reset* on page 3-40
- *About the AHB data bus width* on page 3-41
- *Implementing a narrow slave on a wider bus* on page 3-42
- *Implementing a wide slave on a narrow bus* on page 3-43

- *About the AHB AMBA components on page 3-44*
- *AHB bus slave on page 3-45*
- *AHB bus master on page 3-49*
- *AHB decoder on page 3-57*
- *AHB arbiter on page 3-53.*

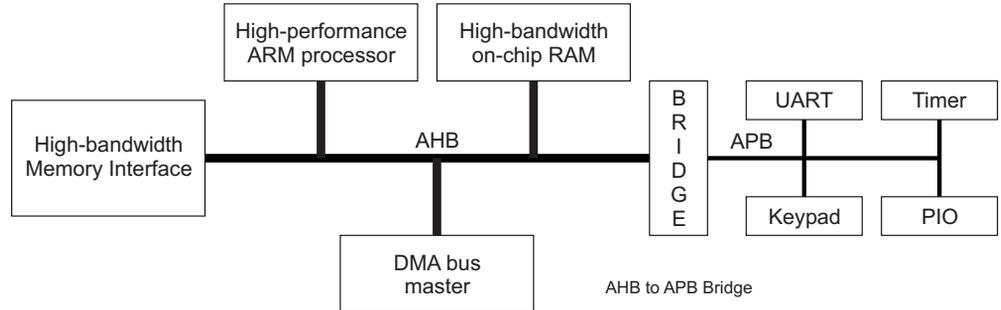
## 3.1 About the AMBA AHB

AHB is a new generation of AMBA bus which is intended to address the requirements of high-performance synthesizable designs. AMBA AHB is a new level of bus which sits above the APB and implements the features required for high-performance, high clock frequency systems including:

- burst transfers
- split transactions
- single cycle bus master handover
- single clock edge operation
- non-tristate implementation
- wider data bus configurations (64/128 bits).

### 3.1.1 A typical AMBA AHB-based microcontroller

An AMBA-based microcontroller typically consists of a high-performance system *backbone* bus, able to sustain the external memory bandwidth, on which the CPU and other *Direct Memory Access* (DMA) devices reside, plus a bridge to a narrower APB bus on which the lower bandwidth peripheral devices are located. Figure 3-1 shows both AHB and APB in a typical AMBA system.



#### AMBA Advanced High-performance Bus (AHB)

- \* High performance
- \* Pipelined operation
- \* Burst transfers
- \* Multiple bus masters
- \* Split transactions

#### AMBA Advanced Peripheral Bus (APB)

- \* Low power
- \* Latched address and control
- \* Simple interface
- \* Suitable for many peripherals

Figure 3-1 A typical AMBA AHB-based system

### 3.2 Bus interconnection

The AMBA AHB bus protocol is designed to be used with a central multiplexor interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexor, which selects the appropriate signals from the slave that is involved in the transfer.

Figure 3-2 illustrates the structure required to implement an AMBA AHB design with three masters and four slaves.

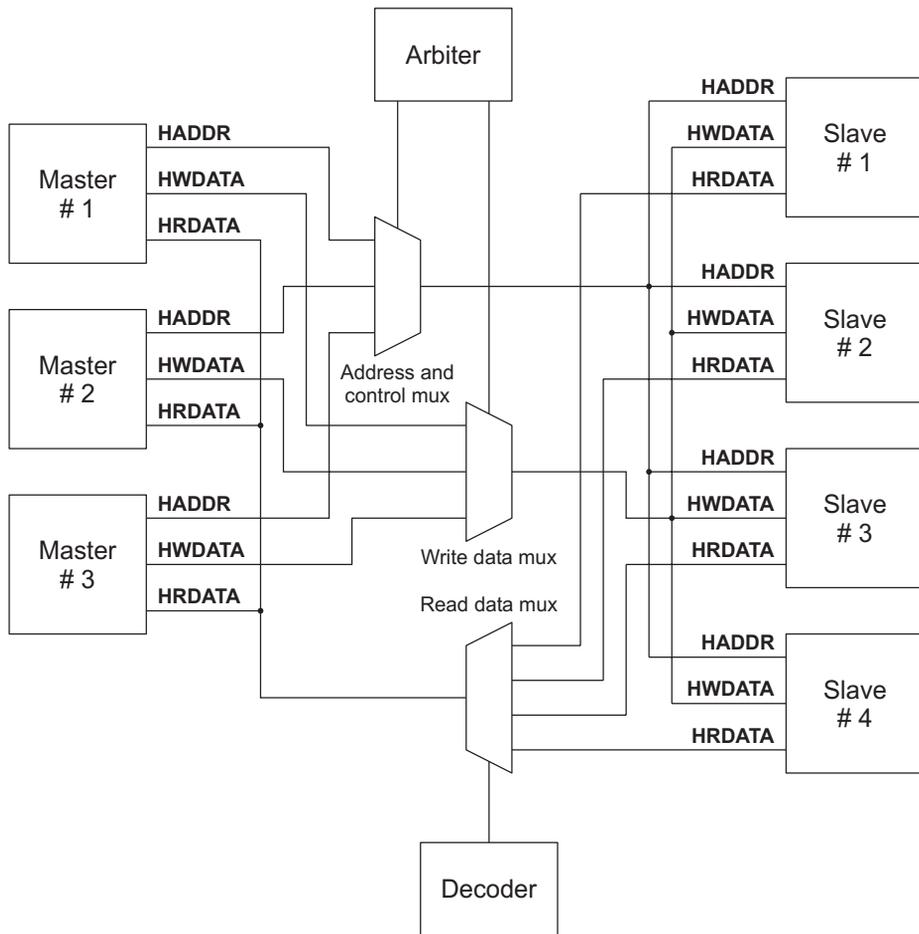


Figure 3-2 Multiplexor interconnection

### 3.3 Overview of AMBA AHB operation

Before an AMBA AHB transfer can commence the bus master must be granted access to the bus. This process is started by the master asserting a request signal to the arbiter. Then the arbiter indicates when the master will be granted use of the bus.

A granted bus master starts an AMBA AHB transfer by driving the address and control signals. These signals provide information on the address, direction and width of the transfer, as well as an indication if the transfer forms part of a burst. Two different forms of burst transfers are allowed:

- incrementing bursts, which do not wrap at address boundaries
- wrapping bursts, which wrap at particular address boundaries.

A write data bus is used to move data from the master to a slave, while a read data bus is used to move data from a slave to the master.

Every transfer consists of:

- an address and control cycle
- one or more cycles for the data.

The address cannot be extended and therefore all slaves must sample the address during this time. The data, however, can be extended using the **HREADY** signal. When LOW this signal causes wait states to be inserted into the transfer and allows extra time for the slave to provide or sample data.

During a transfer the slave shows the status using the response signals, **HRESP[1:0]**:

OKAY	The OKAY response is used to indicate that the transfer is progressing normally and when <b>HREADY</b> goes HIGH this shows the transfer has completed successfully.
ERROR	The ERROR response indicates that a transfer error has occurred and the transfer has been unsuccessful.
RETRY and SPLIT	Both the RETRY and SPLIT transfer responses indicate that the transfer cannot complete immediately, but the bus master should continue to attempt the transfer.

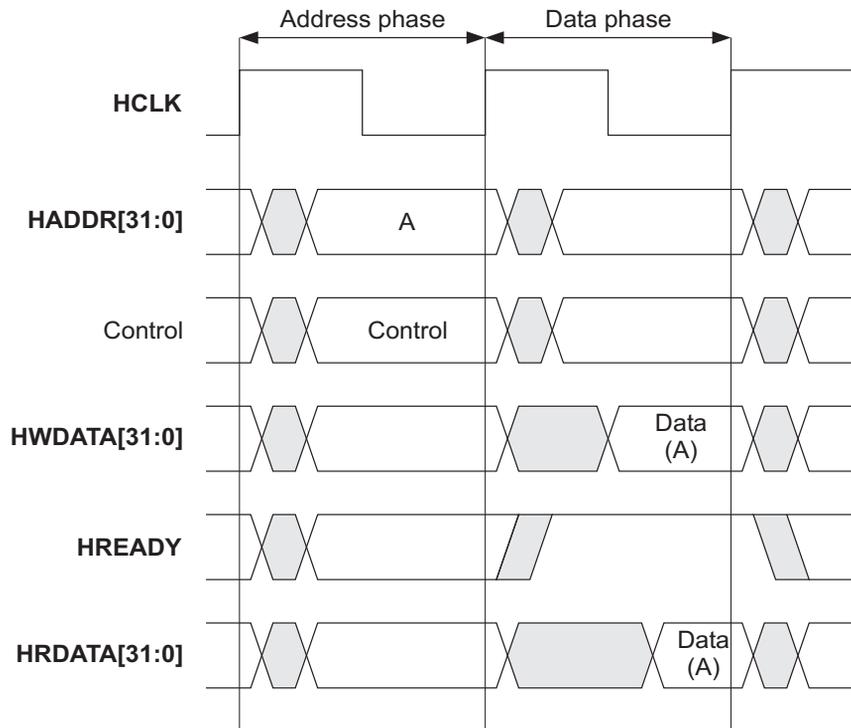
In normal operation a master is allowed to complete all the transfers in a particular burst before the arbiter grants another master access to the bus. However, in order to avoid excessive arbitration latencies it is possible for the arbiter to break up a burst and in such cases the master must re-arbitrate for the bus in order to complete the remaining transfers in the burst.

### 3.4 Basic transfer

An AHB transfer consists of two distinct sections:

- The address phase, which lasts only a single cycle.
- The data phase, which may require several cycles. This is achieved using the **HREADY** signal.

Figure 3-3 shows the simplest transfer, one with no wait states.



**Figure 3-3 Simple transfer**

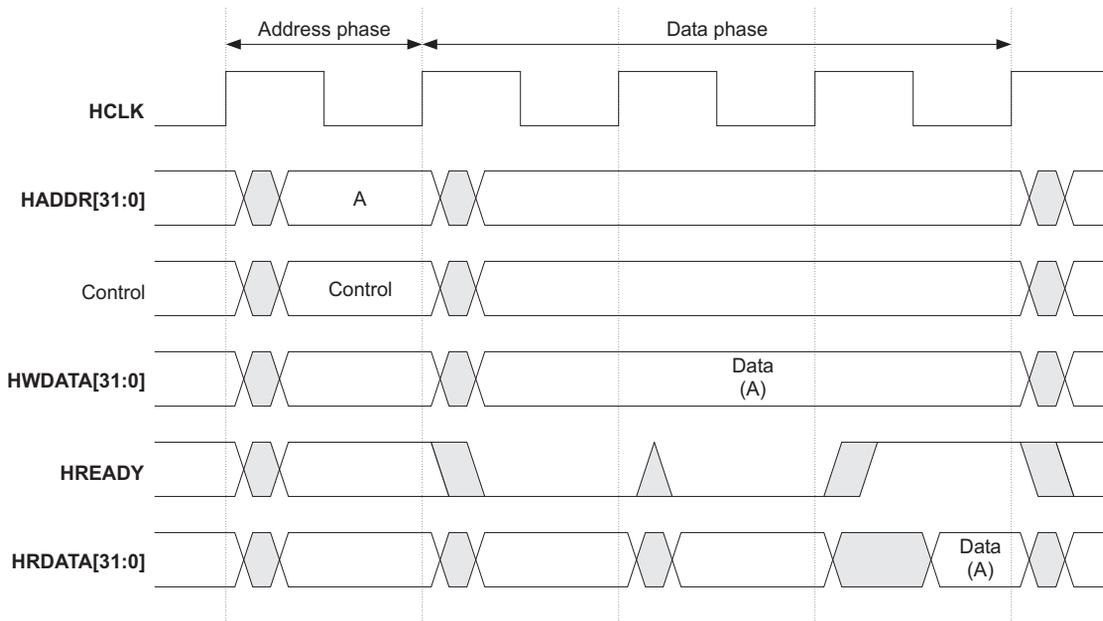
In a simple transfer with no wait states:

- The master drives the address and control signals onto the bus after the rising edge of **HCLK**.
- The slave then samples the address and control information on the next rising edge of the clock.

- After the slave has sampled the address and control it can start to drive the appropriate response and this is sampled by the bus master on the third rising edge of the clock.

This simple example demonstrates how the address and data phases of the transfer occur during different clock periods. In fact, 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 allows for high performance operation, while still providing adequate time for a slave to provide the response to a transfer.

A slave may insert wait states into any transfer, as shown in Figure 3-4, which extends the transfer allowing additional time for completion.



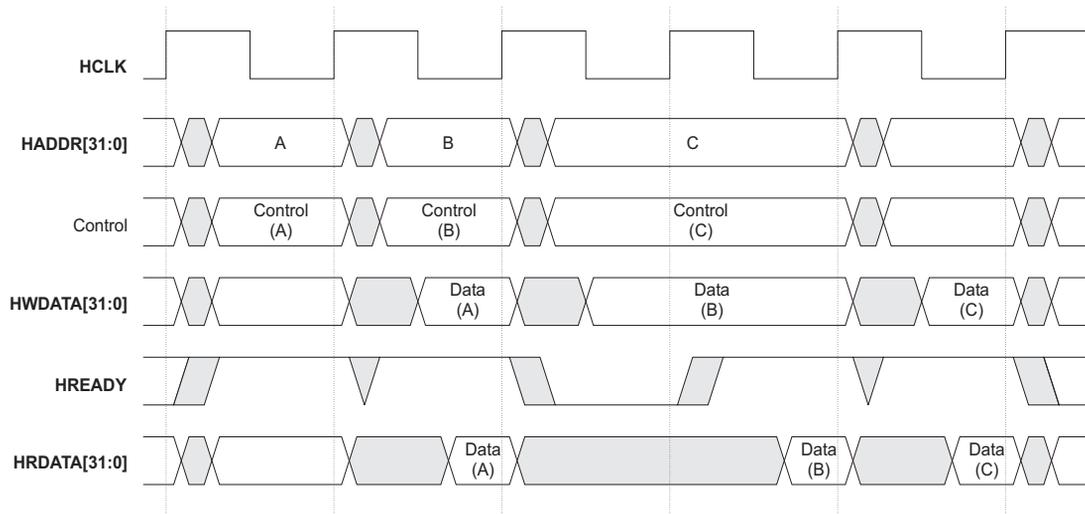
**Figure 3-4 Transfer with wait states**

**Note**

For write operations the bus master will hold 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 will have the side-effect of extending the address phase of the following transfer. This is illustrated in Figure 3-5 which shows three transfers to unrelated addresses, A, B & C.



**Figure 3-5 Multiple transfers**

In Figure 3-5:

- the transfers to addresses A and C are both 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.

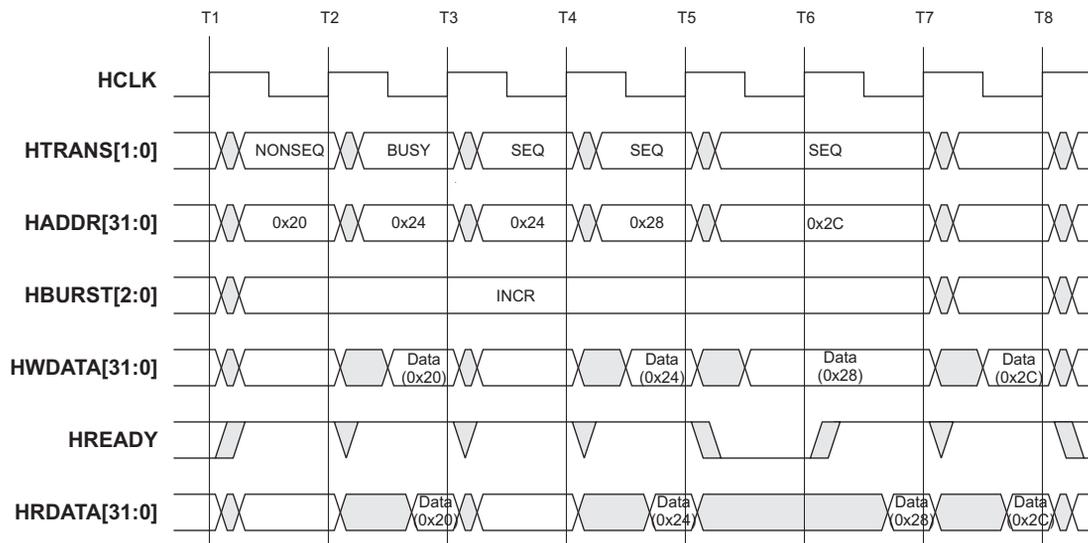
### 3.5 Transfer type

Every transfer can be classified into one of four different types, as indicated by the **HTRANS[1:0]** signals as shown in Table 3-1.

**Table 3-1 Transfer type encoding**

HTRANS[1:0]	Type	Description
00	IDLE	Indicates that no data transfer is required. The IDLE transfer type is used when a bus master is granted the bus, but does not wish to perform a data transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer should be ignored by the slave.
01	BUSY	The BUSY transfer type allows bus masters to insert IDLE cycles in the middle of bursts of transfers. This transfer type indicates that the bus master is continuing with a burst of transfers, 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. The transfer should be ignored by the slave. Slaves must always provide a zero wait state OKAY response, in the same way that they respond to IDLE transfers.
10	NONSEQ	Indicates the first transfer of a burst or a single transfer. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of one and therefore the transfer type is NONSEQUENTIAL.
11	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 size (in bytes). In the case of a wrapping burst the address of the transfer wraps at the address boundary equal to the size (in bytes) multiplied by the number of beats in the transfer (either 4, 8 or 16).

Figure 3-6 shows a number of different transfer types being used.



**Figure 3-6 Transfer type examples**

In Figure 3-6:

- The first transfer is the start of a burst and therefore is **NONSEQUENTIAL**.
- The master is unable to perform the second transfer of the burst immediately and therefore the master uses a **BUSY** transfer to delay the start of the next transfer. In this example the master only requires one cycle before it is ready to start the next transfer in the burst, which completes with no wait states.
- The master performs the third transfer of the burst immediately, but this time the slave is unable to complete and uses **HREADY** to insert a single wait state.
- The final transfer of the burst completes with zero wait states.

### 3.6 Burst operation

Four, eight and sixteen-beat bursts are defined in the AMBA AHB protocol, as well as undefined-length bursts and single transfers. Both incrementing and wrapping bursts are supported in the protocol:

- Incrementing bursts access sequential locations and the address of each transfer in the burst is just an increment of the previous address.
- For wrapping bursts, if the start address of the transfer is not aligned to the total number of bytes in the burst (size x beats) then the address of the transfers in the burst will wrap when the boundary is reached. For example, a four-beat wrapping burst of word (4-byte) accesses will wrap 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.

Burst information is provided using **HBURST[2:0]** and the eight possible types are defined in Table 3-2.

**Table 3-2 Burst signal encoding**

<b>HBURST[2:0]</b>	<b>Type</b>	<b>Description</b>
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

Bursts must not cross a 1kB address boundary. Therefore it is important that masters do not attempt to start a fixed-length incrementing burst which would cause this boundary to be crossed.

It is acceptable to perform single transfers using an unspecified-length incrementing burst which only has a burst of length one.

An incrementing burst can be of any length, but the upper limit is set by the fact that the address must not cross a 1kB boundary

---

**Note**

---

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

---

All transfers within a burst must be aligned to the address boundary equal to the size of the transfer. For example, word transfers must be aligned to word address boundaries (that is  $A[1:0] = 00$ ), halfword transfers must be aligned to halfword address boundaries (that is  $A[0] = 0$ ).

### 3.6.1 Early burst termination

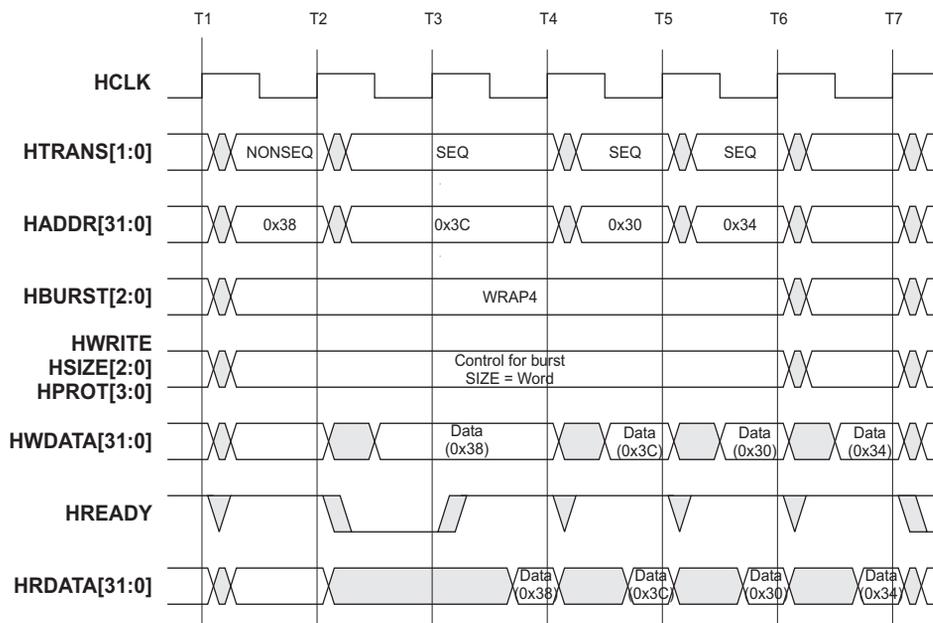
There are certain circumstances when a burst will not be allowed to complete and therefore it is important that any slave design which makes use of the burst information can take the correct course of action if the burst is terminated early. The slave can determine when a burst has terminated early by monitoring the **HTRANS** signals and ensuring that after the start of the burst every transfer is labelled as **SEQUENTIAL** or **BUSY**. If a **NONSEQUENTIAL** or **IDLE** transfer occurs then this indicates that a new burst has started and therefore the previous one must have been terminated.

If a bus master cannot complete a burst because it loses ownership of the bus then it must rebuild the burst appropriately when it next gains access to the bus. For example, if a master has only completed one beat of a four-beat burst then it must use an undefined-length burst to perform the remaining three transfers.

Examples are shown on the following pages:

- Figure 3-7 shows a *Four-beat wrapping burst* on page 3-13
- Figure 3-8 shows a *Four-beat incrementing burst* on page 3-14
- Figure 3-9 shows an *Eight-beat wrapping burst* on page 3-15
- Figure 3-10 shows an *Eight-beat incrementing burst* on page 3-15
- Figure 3-11 shows *Undefined-length bursts* on page 3-16.

The example in Figure 3-7 shows a four-beat wrapping burst with a wait state added for the first transfer.



**Figure 3-7 Four-beat wrapping burst**

As the burst is a four-beat burst of word transfers the address will wrap at 16-byte boundaries, hence the transfer to address 0x3C is followed by a transfer to address 0x30. The only difference with the incrementing burst, shown in Figure 3-8 on page 3-14, is that the addresses continue past the 16-byte boundary.

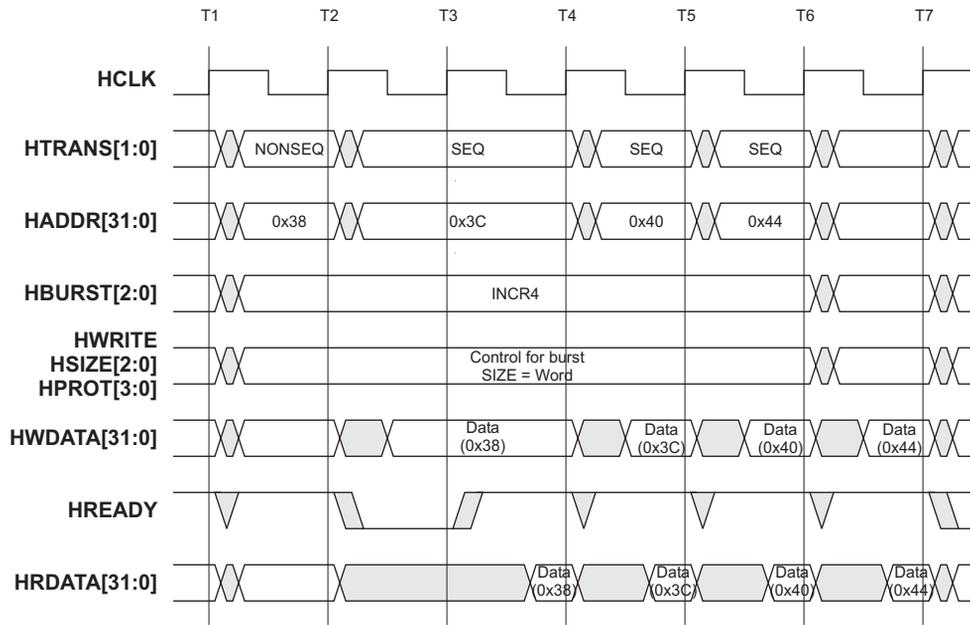
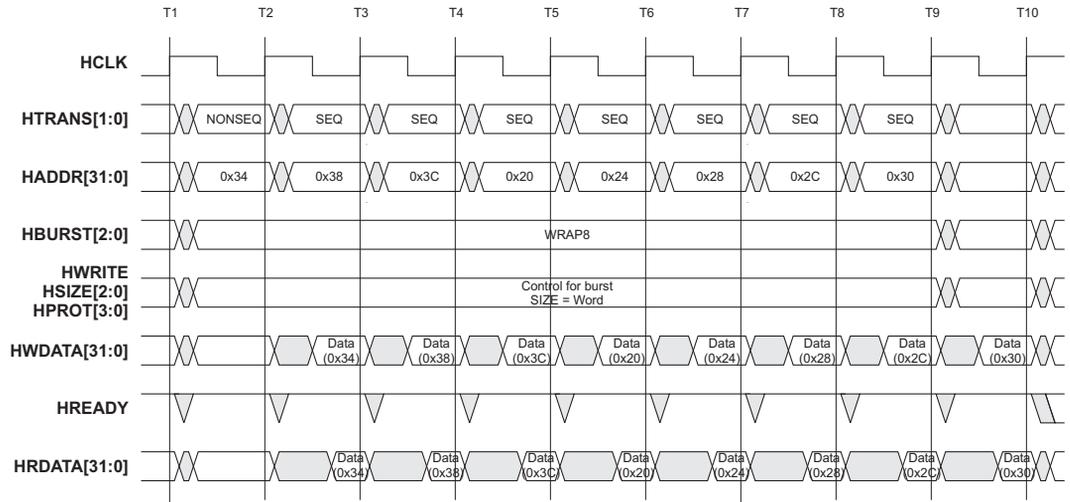


Figure 3-8 Four-beat incrementing burst

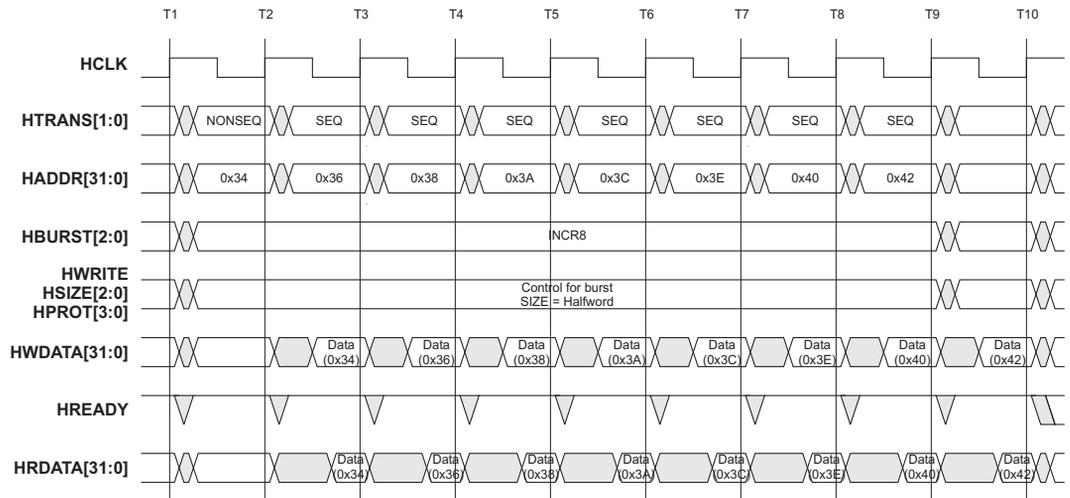
The example in Figure 3-9 is an eight-beat burst of word transfers.



**Figure 3-9 Eight-beat wrapping burst**

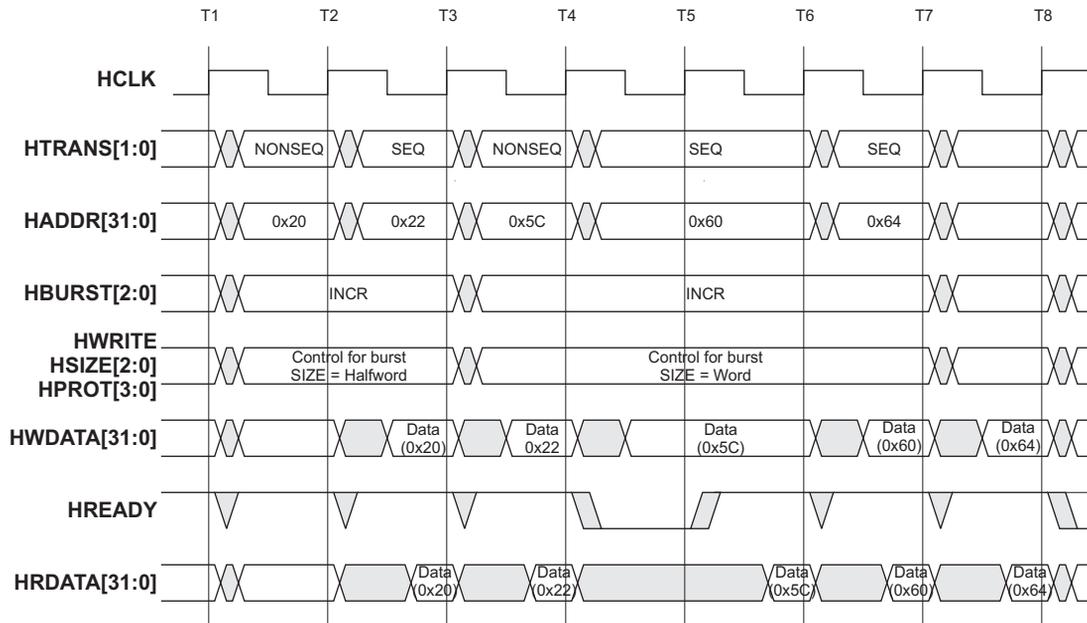
The address will wrap at 32-byte boundaries and therefore address 0x3C is followed by 0x20.

The burst in Figure 3-10 uses halfword transfers, so the addresses increase by 2 and the burst is incrementing so the addresses continue to increment past the 16-byte boundary.



**Figure 3-10 Eight-beat incrementing burst**

The final example in Figure 3-11 shows incrementing bursts of undefined length.



**Figure 3-11 Undefined-length bursts**

Figure 3-11 shows two bursts:

- Two halfword transfers starting at address 0x20. The halfword transfer addresses increment by 2.
- Three word transfers starting at address 0x5C. The word transfer addresses increment by 4.

## 3.7 Control signals

As well as the transfer type and burst type each transfer will have a number of control signals that provide additional information about the transfer. These control signals have exactly the same timing as the address bus. However, they must remain constant throughout a burst of transfers.

### 3.7.1 Transfer direction

When **HWRITE** is HIGH, this signal indicates a write transfer and the master will broadcast data on the write data bus, **HWDATA[31:0]**. When LOW a read transfer will be performed and the slave must generate the data on the read data bus **HRDATA[31:0]**.

### 3.7.2 Transfer size

**HSIZE[2:0]** indicates the size of the transfer, as shown in Table 3-3.

**Table 3-3 Size encoding**

<b>HSIZE[2]</b>	<b>HSIZE[1]</b>	<b>HSIZE[0]</b>	<b>Size</b>	<b>Description</b>
0	0	0	8 bits	Byte
0	0	1	16 bits	Halfword
0	1	0	32 bits	Word
0	1	1	64 bits	-
1	0	0	128 bits	4-word line
1	0	1	256 bits	8-word line
1	1	0	512 bits	-
1	1	1	1024 bits	-

The size is used in conjunction with the **HBURST[2:0]** signals to determine the address boundary for wrapping bursts.

### 3.7.3 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 wishes to implement some level of protection (see Table 3-4).

The signals indicate if the transfer is:

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

For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.

**Table 3-4 Protection signal encodings**

<b>HPROT[3] cacheable</b>	<b>HPROT[2] bufferable</b>	<b>HPROT[1] privileged</b>	<b>HPROT[0] data/opcode</b>	<b>Description</b>
-	-	-	0	Opcode fetch
-	-	-	1	Data access
-	-	0	-	User access
-	-	1	-	Privileged access
-	0	-	-	Not bufferable
-	1	-	-	Bufferable
0	-	-	-	Not cacheable
1	-	-	-	Cacheable

Not all bus masters will be capable of generating accurate protection information, therefore it is recommended that slaves do not use the **HPROT** signals unless strictly necessary.

### 3.8 Address decoding

A central address decoder is used to provide a select signal, **HSEL<sub>x</sub>**, for each slave on the bus. The select signal is a combinatorial decode of the high-order address signals, and simple address decoding schemes are encouraged to avoid complex decode logic and to ensure high-speed operation.

A slave must only sample the address and control signals and **HSEL<sub>x</sub>** when **HREADY** is HIGH, indicating that the current transfer is completing. Under certain circumstances it is possible that **HSEL<sub>x</sub>** will be asserted when **HREADY** is LOW, but the selected slave will have changed by the time the current transfer completes.

The minimum address space that can be allocated to a single slave is 1kB. All bus masters are designed such that they will not perform incrementing transfers over a 1kB boundary, thus ensuring that a burst never crosses an address decode boundary.

In the case where a system design does not contain a completely filled memory map an additional default slave should be implemented 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 should provide an ERROR response. IDLE or BUSY transfers to nonexistent locations should result in a zero wait state OKAY response. Typically the default slave functionality will be implemented as part of the central address decoder.

Figure 3-12 shows a typical address decoding system and the slave select signals.

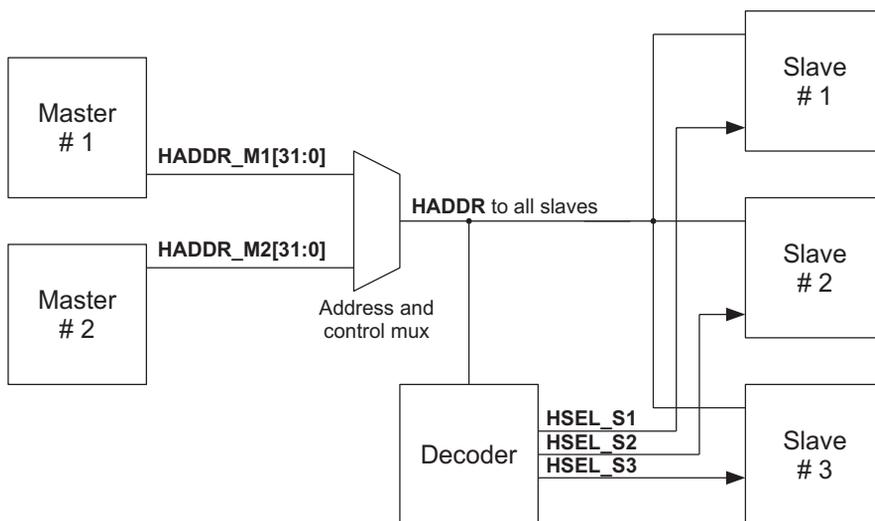


Figure 3-12 Slave select signals

## 3.9 Slave transfer responses

After a master has started a transfer, the slave then determines how the transfer should progress. No provision is made within the AHB specification for a bus master to cancel a transfer once it has commenced.

Whenever a slave is accessed it must provide a response which indicates the status of the transfer. The **HREADY** signal is used to extend the transfer and this works in combination with the response signals, **HRESP[1:0]**, which provide the status of the transfer.

The slave can complete the transfer in a number of ways. It can:

- complete the transfer immediately
- insert one or more wait states to allow time to complete the transfer
- signal an error to indicate that the transfer has failed
- delay the completion of the transfer, but allow the master and slave to back off the bus, leaving it available for other transfers.

### 3.9.1 Transfer done

The **HREADY** signal is used to extend the data portion of an AHB transfer. When LOW the **HREADY** signal indicates the transfer is to be extended and when HIGH indicates that the transfer can complete.

———— **Note** —————

Every slave must have a predetermined maximum number of wait states that it will insert before it backs off the bus, in order to allow the calculation of the latency of accessing the bus. It is recommended, but not mandatory, 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.

### 3.9.2 Transfer response

A typical slave will use the **HREADY** signal to insert the appropriate number of wait states into the transfer and then the transfer will complete with **HREADY** HIGH and an OKAY response, which indicates the successful completion of the transfer.

The ERROR response is used by a slave to indicate some form of error condition with the associated transfer. Typically this is used for a protection error, such as an attempt to write to a read-only memory location.

The SPLIT and RETRY response combinations allow slaves to delay the completion of a transfer, but free up the bus for use by other masters. These response combinations are usually only required by slaves that have a high access latency and can make use of these response codes to ensure that other masters are not prevented from accessing the bus for long periods of time.

A full description of the SPLIT and RETRY operations can be found in *Split and retry* on page 3-24.

The encoding of **HRESP[1:0]**, the transfer response signals, and a description of each response are shown in Table 3-5.

**Table 3-5 Response encoding**

HRESP[1]	HRESP[0]	Response	Description
0	0	OKAY	When <b>HREADY</b> is HIGH this shows the transfer has completed successfully. The OKAY response is also used for any additional cycles that are inserted, with <b>HREADY</b> LOW, prior to giving one of the three other responses.
0	1	ERROR	This response shows an error has occurred. The error condition should be signalled to the bus master so that it is aware the transfer has been unsuccessful. A two-cycle response is required for an error condition.
1	0	RETRY	The RETRY response shows the transfer has not yet completed, so the bus master should retry the transfer. The master should continue to retry the transfer until it completes. A two-cycle RETRY response is required.
1	1	SPLIT	The transfer has not yet completed successfully. The bus master must retry the transfer when it is next granted access to the bus. The slave will request access to the bus on behalf of the master when the transfer can complete. A two-cycle SPLIT response is required.

When it is necessary for a slave to insert a number of wait states prior to deciding what response will be given then it must drive the response to OKAY.

### 3.9.3 Two-cycle response

Only an OKAY response can be given in a single cycle. The ERROR, SPLIT and RETRY responses require at least two cycles. To complete with any of these responses then in the penultimate (one before last) cycle the slave drives **HRESP[1:0]** to indicate ERROR, RETRY or SPLIT while driving **HREADY** LOW to extend the transfer for an extra cycle. In the final cycle **HREADY** is driven HIGH to end the transfer, while **HRESP[1:0]** remains driven to indicate ERROR, RETRY or SPLIT.

If the slave needs more than two cycles to provide the ERROR, SPLIT or RETRY response then additional wait states may be inserted at the start of the transfer. During this time the **HREADY** signal will be LOW and the response must be set to OKAY.

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

For the SPLIT and RETRY response the following transfer must be cancelled because it must not take place before the current transfer has completed. However, for the ERROR response, where the current transfer is not repeated, completion of the following transfer is optional.

Figure 3-13 shows an example of a RETRY operation.

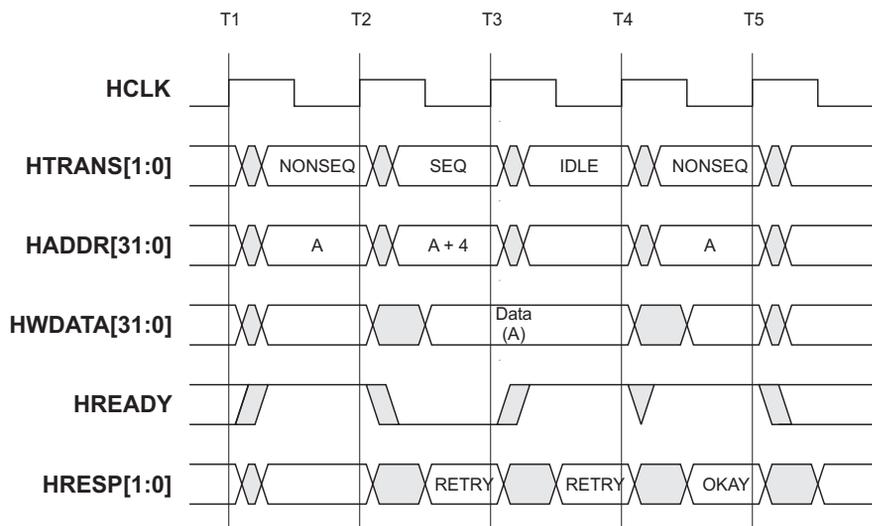
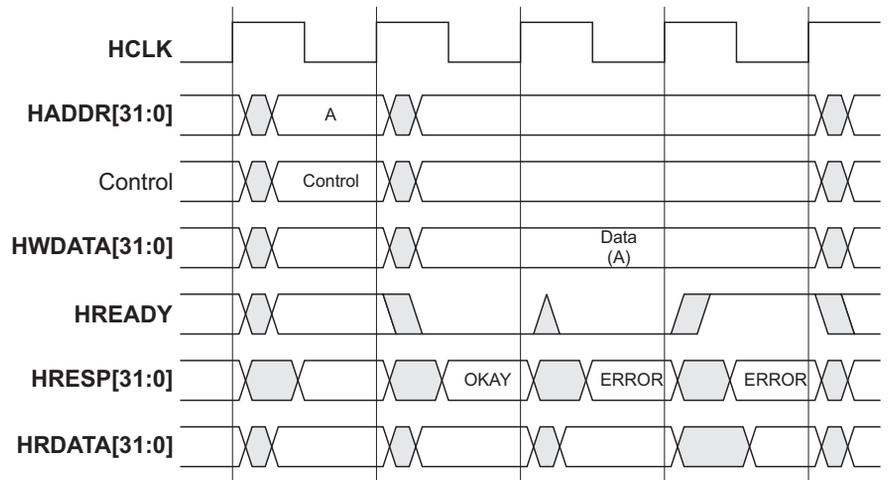


Figure 3-13 Transfer with retry response

The following events are illustrated:

- The master starts with a transfer to address A.
- Before the response is received for this transfer the master moves the address on to A + 4.
- The slave at address A is unable to complete the transfer immediately and therefore it issues a **RETRY** response. This response indicates to the master that the transfer at address A is unable to complete and so the transfer at address A + 4 is cancelled and replaced by an **IDLE** transfer.

Figure 3-14 shows a transfer where the slave requires one cycle to decide on the response it is going to give (during which time **HRESP** indicates **OKAY**) and then the slave ends the transfer with a two-cycle **ERROR** response.



**Figure 3-14** Error response

### 3.9.4 Error response

If a slave provides an **ERROR** response then the master may choose to 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.

### 3.9.5 Split and retry

The SPLIT and RETRY responses provide a mechanism for slaves to release the bus when they are unable to supply data for a transfer immediately. Both mechanisms allow the transfer to finish on the bus and therefore allow a higher-priority master to get access to the bus.

The difference between SPLIT and RETRY is the way the arbiter allocates the bus after a SPLIT or a RETRY has occurred:

- For RETRY the arbiter will continue to use the normal priority scheme and therefore only masters having a higher priority will gain access to the bus.
- For a SPLIT transfer the arbiter will adjust the priority scheme so that any other master requesting the bus will get access, even if it is a lower priority. In order for a SPLIT transfer to complete the arbiter must be informed when the slave has the data available.

The SPLIT transfer requires extra complexity in both the slave and the arbiter, but has the advantage that it completely frees the bus for use by other masters, whereas the RETRY case will only allow higher priority masters onto the bus.

A bus master should treat SPLIT and RETRY in the same manner. It should continue to request the bus and attempt the transfer until it has either completed successfully or been terminated with an ERROR response.

## 3.10 Data buses

In order to allow implementation of an AHB system without the use of tristate drivers separate read and write data buses are required. The minimum data bus width is specified as 32 bits, but the bus width can be increased as described in *About the AHB data bus width* on page 3-41.

### 3.10.1 HWDATA[31:0]

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

All transfers must be aligned to the address boundary equal to the size of the transfer. For example, word transfers must be aligned to word address boundaries (that is  $A[1:0] = 00$ ), halfword transfers must be aligned to halfword address boundaries (that is  $A[0] = 0$ ).

For transfers that are narrower than the width of the bus, for example a 16-bit transfer on a 32-bit bus, then the bus master only has to drive the appropriate byte lanes. The slave is responsible for selecting the write data from the correct byte lanes. Table 3-6 on page 3-26 and Table 3-7 on page 3-26 show which byte lanes are active for a little-endian and big-endian system respectively. If required, this information can be extended for wider data bus implementations. Burst transfers which have a transfer size less than the width of the data bus will have different active byte lanes for each beat of the burst.

The active byte lane is dependent on the endianness of the system, but AHB does not specify the required endianness. Therefore, it is important that all masters and slaves on the bus are of the same endianness.

### 3.10.2 HRDATA[31:0]

The read data bus is driven by the appropriate slave during read transfers. If the slave extends the read transfer by holding **HREADY LOW** then the slave only needs to provide valid data at the end of 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 needs to provide valid data on the active byte lanes, as indicated in Table 3-6 and Table 3-7. The bus master is responsible for selecting the data from the correct byte lanes.

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

**Table 3-6 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	✓	✓	✓	✓
Halfword	0	-	-	✓	✓
Halfword	2	✓	✓	-	-
Byte	0	-	-	-	✓
Byte	1	-	-	✓	-
Byte	2	-	✓	-	-
Byte	3	✓	-	-	-

**Table 3-7 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	✓	✓	✓	✓
Halfword	0	✓	✓	-	-
Halfword	2	-	-	✓	✓
Byte	0	✓	-	-	-
Byte	1	-	✓	-	-
Byte	2	-	-	✓	-
Byte	3	-	-	-	✓

### 3.10.3 Endianness

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

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

For module designers it is recommended that only modules which will be used in a wide variety of applications should be 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 will result in a smaller, lower power, higher performance interface.

## 3.11 Arbitration

The arbitration mechanism is used to ensure that only one master has access to the bus at any one time. The arbiter performs this function by observing a number of different requests to use the bus and deciding which is currently the highest priority master requesting the bus. The arbiter also receives requests from slaves that wish to complete SPLIT transfers.

Any slaves which are not capable of performing SPLIT transfers do not need to be aware of the arbitration process, except that they need to observe the fact that a burst of transfers may not complete if the ownership of the bus is changed.

### 3.11.1 Signal description

A brief description of each of the arbitration signals is given below:

<b>HBUSREQx</b>	The bus request signal is used by a bus master to request access to the bus. Each bus master has its own <b>HBUSREQx</b> signal to the arbiter and there can be up to 16 separate bus masters in any system.
<b>HLOCKx</b>	The lock signal is asserted by a master at the same time as the bus request signal. This indicates to the arbiter that the master is performing a number of indivisible transfers and the arbiter must not grant any other bus master access to the bus once the first transfer of the locked transfers has commenced. <b>HLOCKx</b> must be asserted at least a cycle before the address to which it refers, in order to prevent the arbiter from changing the grant signals.
<b>HGRANTx</b>	The grant signal is generated by the arbiter and indicates that the appropriate master is currently the highest priority master requesting the bus, taking into account locked transfers and SPLIT transfers. A master gains ownership of the address bus when <b>HGRANTx</b> is HIGH and <b>HREADY</b> is HIGH at the rising edge of <b>HCLK</b> .
<b>HMASTER[3:0]</b>	The arbiter indicates which master is currently granted the bus using the <b>HMASTER[3:0]</b> signals and this can be used to control the central address and control multiplexor. The master number is also required by SPLIT-capable slaves so that they can indicate to the arbiter which master is able to complete a SPLIT transaction.
<b>HMASTLOCK</b>	The arbiter indicates that the current transfer is part of a locked sequence by asserting the <b>HMASTLOCK</b> signal, which has the same timing as the address and control signals.

**HSPLIT[15:0]** The 16-bit *Split Complete* bus is used by a SPLIT-capable slave to indicate which bus master can complete a SPLIT transaction. This information is needed by the arbiter so that it can grant the master access to the bus to complete the transfer.

Further information is provided in:

- *Requesting bus access*
- *Granting bus access* on page 3-30
- *Early burst termination* on page 3-33
- *Locked transfers* on page 3-34.

### 3.11.2 Requesting bus access

A bus master uses the **HBUSREQx** signal to request access to the bus and may request the bus during any cycle. The arbiter will sample the request on the rising of the clock and then use an internal priority algorithm to decide which master will be the next to gain access to the bus.

Normally the arbiter will only grant a different bus master when a burst is completing. However, if required, the arbiter can terminate a burst early to allow a higher priority master access to the bus.

If the master requires locked accesses then it must also assert the **HLOCKx** signal to indicate to the arbiter that no other masters should be granted the bus.

When a master is granted the bus and is performing a fixed length burst it is not necessary to continue to request the bus in order to complete the burst. The arbiter observes the progress of the burst and uses the **HBURST[2:0]** signals to determine how many transfers are required by the master. If the master wishes to perform a second burst after the one that is currently in progress then it should re-assert the request signal during the burst.

If a master loses access to the bus in the middle of a burst then it must re-assert the **HBUSREQx** request line to regain access to the bus.

For undefined length bursts the master should continue to assert the request until it has started the last transfer. The arbiter cannot predict when to change the arbitration at the end of an undefined length burst.

It is possible that a master can be granted the bus when it is not requesting it. This may occur when no masters are requesting the bus and the arbiter grants access to a default master. Therefore, it is important that if a master does not require access to the bus it drives the transfer type **HTRANS** to indicate an IDLE transfer.

### 3.11.3 Granting bus access

The arbiter indicates which bus master is currently the highest priority requesting the bus by asserting the appropriate **HGRANTx** signal. When the current transfer completes, as indicated by **HREADY** HIGH, then the master will become granted and the arbiter will change the **HMASTER[3:0]** signals to indicate the bus master number.

Figure 3-15 shows the process when all transfers are zero wait state and the **HREADY** signal is HIGH.

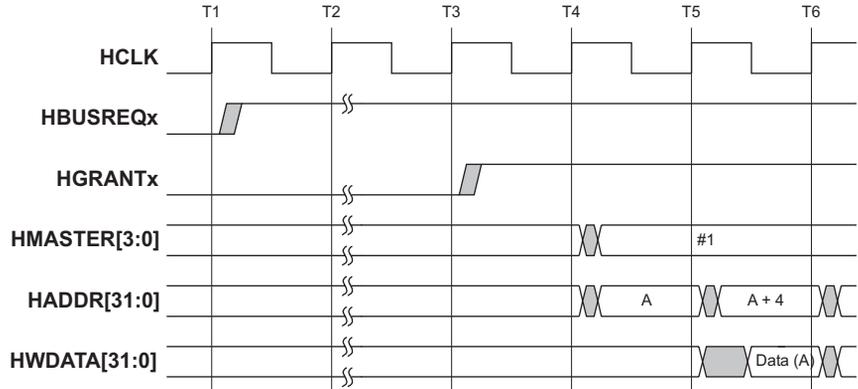


Figure 3-15 Granting access with no wait states

Figure 3-16 shows the effect of wait states on the bus handover.

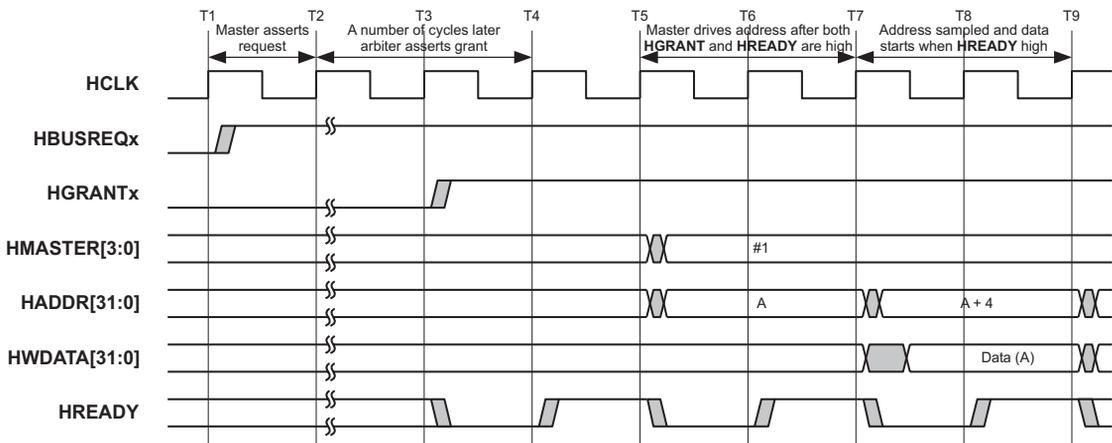


Figure 3-16 Granting access with wait states

The ownership of the data bus is delayed from the ownership of the address bus. Whenever a transfer completes, as indicated by **HREADY** HIGH, then the master that owns the address bus will be able to use the data bus and will continue to own the data bus until the transfer completes. Figure 3-17 shows how the ownership of the data bus is transferred when handover occurs between two bus masters.

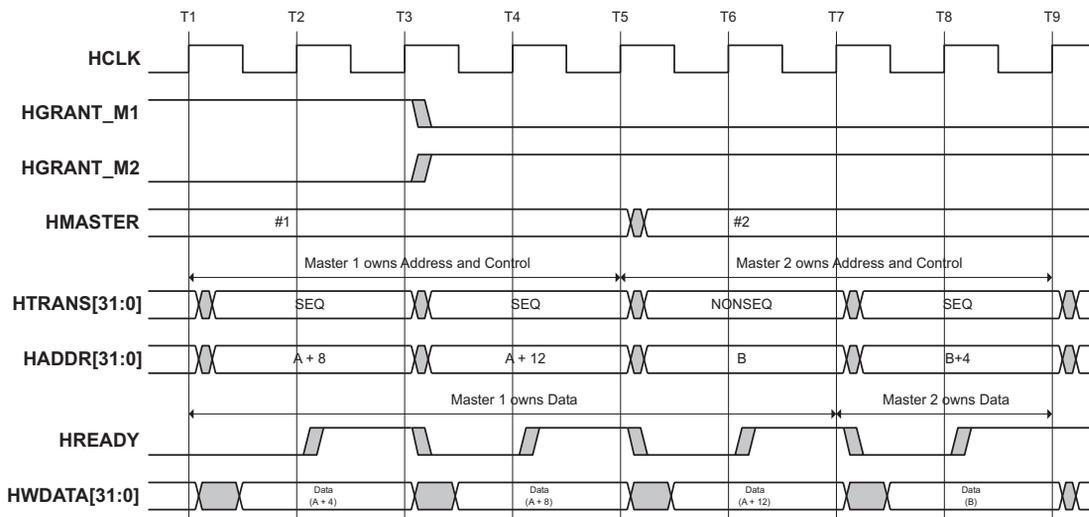
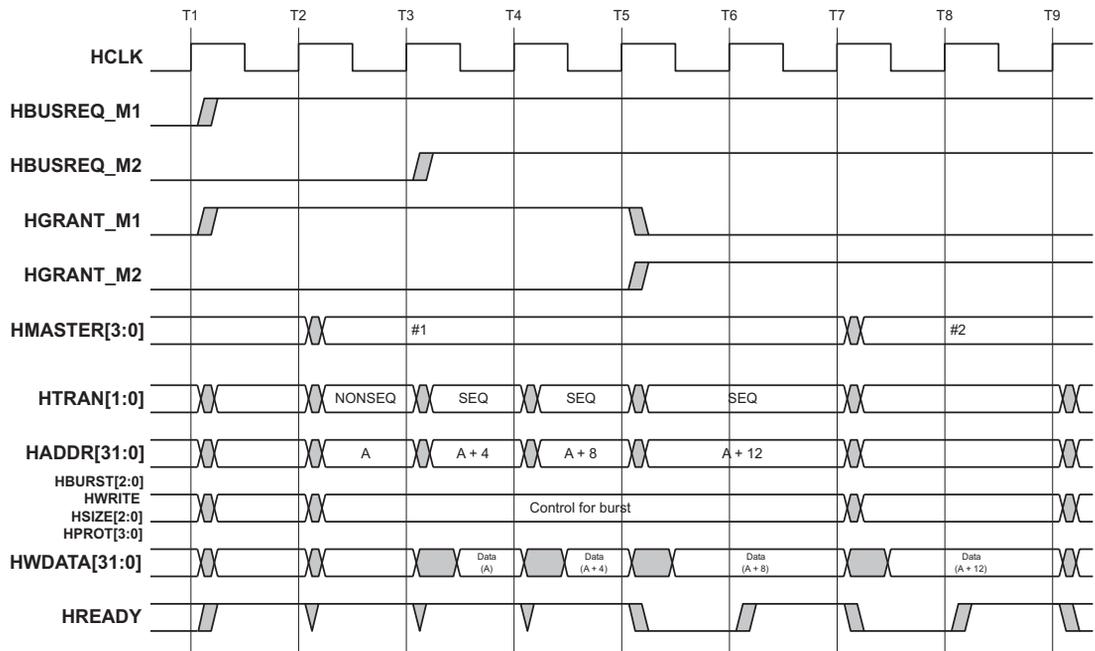


Figure 3-17 Data bus ownership

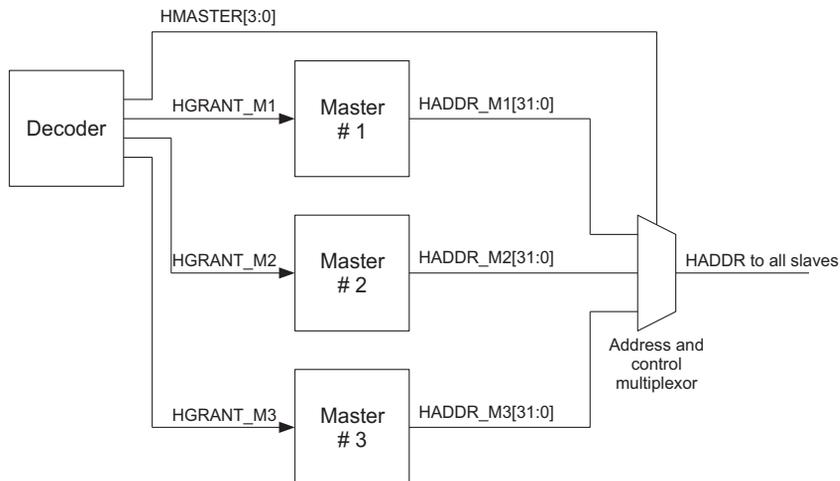
Figure 3-18 shows an example of how the arbiter can hand over the bus at the end of a burst of transfers.



**Figure 3-18 Handover after burst**

The arbiter changes the **HGRANTx** signals when the penultimate (one before last) address has been sampled. The new **HGRANTx** information will then be sampled at the same point as the last address of the burst is sampled.

Figure 3-19 shows how **HGRANTx** and **HMASTER** signals are used in a system.



**Figure 3-19 Bus master grant signals**

———— **Note** ————

Because a central multiplexor is used, each master can drive out the address of the transfer it wishes to perform immediately and it does not need to wait until it is granted the bus. The **HGRANTx** signal is only used by the master to determine when it owns the bus and hence when it should consider that the address has been sampled by the appropriate slave.

A delayed version of the **HMASTER** bus is used to control the write data multiplexor.

### 3.11.4 Early burst termination

Normally the arbiter will not hand over the bus to a new master until the end of a burst of transfers. However, if the arbiter determines that the burst must be terminated early in order to prevent excessive access time to the bus then it may transfer the grant to another bus master before a burst has completed.

If a master loses ownership of the bus in the middle of a burst it must re-arbitrate for the bus in order to complete the burst. The master must ensure that the **HBURST** and **HTRANS** signals are adapted to reflect the fact that it no longer has to perform a complete 4, 8 or 16-beat burst.

For example, if a master is only able to complete 3 transfers of an 8-beat burst, then when it regains the bus it must use a legal burst encoding to complete the remaining 5 transfers. Any legal combination can be used, so either a 5-beat undefined length burst or a 4-beat fixed length burst followed by a single-beat undefined length burst would be acceptable.

### 3.11.5 Locked transfers

The arbiter must observe the **HLOCKx** signal from each master to determine when the master wishes to perform a locked sequence of transfers. The arbiter is then responsible for ensuring that no other bus masters are granted the bus until the locked sequence has completed.

After a sequence of locked transfers the arbiter will always keep the bus master granted for an additional transfer to ensure that the last transfer in the locked sequence has completed successfully and has not received either a **SPLIT** or **RETRY** response. Therefore it is recommended, but not mandatory, that the master inserts an **IDLE** transfer after any locked sequence to provide an opportunity for the arbitration to change before commencing another burst of transfers.

The arbiter is also responsible for asserting the **HMASTLOCK** signal, which has the same timing as the address and control signals. This signal indicates to any slave that the current transfer is locked and therefore must be processed before any other masters are granted the bus.

### 3.11.6 Default bus master

Every system must include a default bus master which is granted the bus if all other masters are unable to use the bus. When granted, the default bus master must only perform **IDLE** transfers.

If no masters are requesting the bus then the arbiter may either grant the default master or alternatively it may grant the master that would benefit the most from having low access latency to the bus.

Granting the default master access to the bus also provides a useful mechanism for ensuring that no new transfers are started on the bus and is a useful step to perform prior to entering a low-power mode of operation.

The default master must be granted if all other masters are waiting for **SPLIT** transfers to complete.

## 3.12 Split transfers

SPLIT transfers improve the overall utilization of the bus by separating (or splitting) the operation of the master providing the address to a slave from the operation of the slave responding with the appropriate data.

When a transfer occurs the slave can decide to issue a SPLIT response if it believes the transfer will take a large number of cycles to perform. This signals to the arbiter that the master which is attempting the transfer should not be granted access to the bus until the slave indicates it is ready to complete the transfer. Therefore the arbiter is responsible for observing the response signals and internally masking any requests from masters which have been SPLIT.

During the address phase of a transfer the arbiter generates a tag, or bus master number, on **HMASTER[3:0]** which identifies the master that is performing the transfer. Any slave issuing a SPLIT response must be capable of indicating that it can complete the transfer, and it does this by making a note of the master number on the **HMASTER[3:0]** signals.

Later, when the slave can complete the transfer, it asserts the appropriate bit, according to the master number, on the **HSPLITx[15:0]** signals from the slave to the arbiter. The arbiter then uses this information to unmask the request signal from the master and in due course the master will be granted access to the bus to retry the transfer. The arbiter samples the **HSPLITx** bus every cycle and therefore the slave only needs to assert the appropriate bit for a single cycle in order for the arbiter to recognize it.

In a system with multiple SPLIT-capable slaves the **HSPLITx** buses from each slave can be ORed together to provide a single resultant **HSPLIT** bus to the arbiter.

In the majority of systems the maximum capacity of 16 bus masters will not be used and therefore the arbiter only requires an **HSPLIT** bus which has the same number of bits as there are bus masters. However, it is recommended that all SPLIT-capable slaves are designed to support up to 16 masters.

### 3.12.1 Split transfer sequence

The basic stages of a SPLIT transaction are:

1. The master starts the transfer in an identical way to any other transfer and issues address and control information
2. If the slave is able to provide data immediately it may do so. If the slave decides that it may take a number of cycles to obtain the data it gives a SPLIT transfer response.

During every transfer the arbiter broadcasts a number, or tag, showing which master is using the bus. The slave must record this number, to use it to restart the transfer at a later time.

3. The arbiter grants other masters use of the bus and the action of the SPLIT response allows bus master handover to occur. If all other masters have also received a SPLIT response then the default master is granted.
4. When the slave is ready to complete the transfer it asserts the appropriate bit of the **HSPLITx** bus to the arbiter to indicate which master should be regranted access to the bus.
5. The arbiter observes the **HSPLITx** signals on every cycle, and when any bit of **HSPLITx** is asserted the arbiter restores the priority of the appropriate master.
6. Eventually the arbiter will grant the master so it can re-attempt the transfer. This may not occur immediately if a higher priority master is using the bus.
7. When the transfer eventually takes place the slave finishes with an OKAY transfer response.

### 3.12.2 Multiple split transfers

The bus protocol only allows a single outstanding transaction per bus master. If any master module is able to deal with more than one outstanding transaction it requires an additional set of request and grant signals for each outstanding transaction that it can handle. At the protocol level a single module may appear as a number of different bus masters, each of which can only have one outstanding transaction.

It is, however, possible that a SPLIT-capable slave could receive more transfer requests than it is able to process concurrently. If this happens then it is acceptable for the slave to issue a SPLIT response without recording the appropriate address and control information for the transfer and it is only necessary for the slave to record the bus master number. The slave can then indicate that it can process another transfer by asserting the appropriate bits on the **HSPLITx** bus for all masters that the slave has previously SPLIT, but that the slave has not recorded the address and control information.

The arbiter is then able to regrant the masters access to the bus and they will retry the transfer, giving the address and control information required by the slave. This means that a master may be granted the bus a number of times before it is finally allowed to complete the transfer it requires.

### 3.12.3 Preventing deadlock

Both the SPLIT and RETRY transfer responses must be used with care to prevent bus deadlock. A single transfer can never lock the AHB as every slave must be designed to finish a transfer within a predetermined number of cycles. However, it is possible for deadlock to occur if a number of different masters attempt to access a slave which issues SPLIT or RETRY responses in a manner which the slave is unable to deal with.

#### Split transfers

For slaves that can issue a SPLIT transfer response, bus deadlock is prevented by ensuring that the slave can withstand a request from every master in the system, up to a maximum of 16. The slave does not need to store the address and control information for every transfer, it simply needs to record the fact that a transfer request has been made and a SPLIT response issued. Eventually all masters will be at a low priority and the slave can then work through the requests in an orderly manner, indicating to the arbiter which request it is servicing, thus ensuring that all requests are eventually serviced.

When a slave has a number of outstanding requests it may choose to process them in any order, although the slave must be aware that a locked transfer will have to be completed before any other transfers can continue.

It is perfectly legal for the slave to use a SPLIT response without latching the address and control information. The slave only needs to record that a transfer attempt has been made by that particular master and then at a later point the slave can obtain the address and control information by indicating that it is ready to complete the transfer. The master will be granted the bus and will rebroadcast the transfer, allowing the slave to latch the address and control information and either respond with the data immediately, or issue another SPLIT response if a number of additional cycles are required.

Ideally the slave should never have more outstanding transfers than it can support, but the mechanism to support this is required to prevent bus deadlock.

## Retry transfers

A slave which issues RETRY responses must only be accessed by one master at a time. This is not enforced by the protocol of the bus and should be ensured by the system architecture. In most cases slaves that issue RETRY responses will be peripherals which need to be accessed by just one master at a time, so this will be ensured by some higher level protocol.

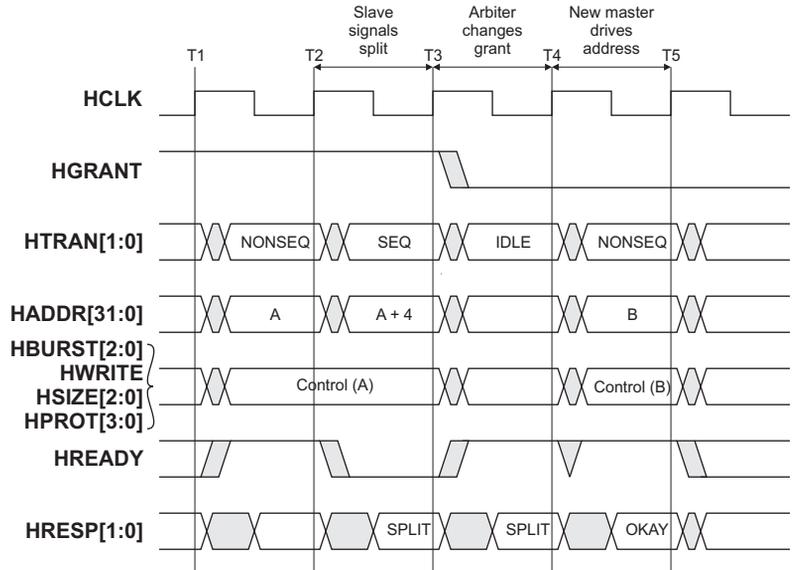
Hardware protection against multiple masters accessing RETRY slaves is not a requirement of the protocol, but may be implemented as described in the following paragraph. The only bus-level requirement is that the slave must drive **HREADY** HIGH within a predetermined number of clock cycles.

If hardware protection is required then this may be implemented within the RETRY slave itself. When a slave issues a RETRY it can sample the master number. Between that point and the time when the transfer is finally completed the RETRY slave can check every transfer attempt that is made to ensure the master number is the same. If it ever detects that the master number is different then it can take an alternative course of action, such as:

- an ERROR response
- a signal to the arbiter
- a system level interrupt
- a complete system reset.

### 3.12.4 Bus handover with split transfers

The protocol requires that a master performs an IDLE transfer immediately after receiving a SPLIT or RETRY response allowing the bus to be transferred to another master. Figure 3-20 shows the sequence of events that occur for a split transfer.



**Figure 3-20 Handover after split transfer**

The following points should be noted:

- The address for the transfer is on the bus after time T1. The slave returns the two-cycle SPLIT response after the clock edges at T2 and T3.
- At the end of the first response cycle, T3, the master can detect that the transfer will be SPLIT and so it changes the control signals for the following transfer to show an IDLE transfer.
- Also at time T3 the arbiter samples the response signals and determines that the transfer has been SPLIT. The arbiter can then adjust the arbitration priorities and the grant signals change during the following cycle, such that the new master can be granted the address bus after time T4.
- The new master is guaranteed immediate access because the IDLE transfer always completes in a single cycle.

### 3.13 Reset

The reset, **HRESETn**, is the only active LOW signal in the AMBA AHB specification and is the primary reset for all bus elements. The reset may 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.

### 3.14 About the AHB data bus width

One way to improve bus bandwidth without increasing the frequency of operation is to make the data path of the on-chip bus wider. Both the increased layers of metal and the use of large on-chip memory blocks (such as Embedded DRAM) are driving factors which encourage the use of wider on-chip buses.

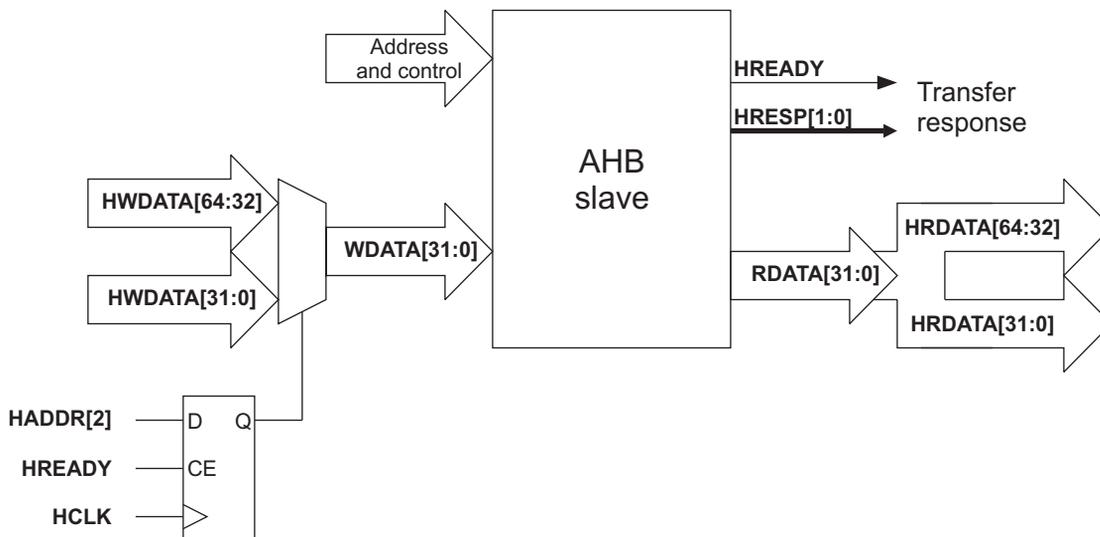
Specifying a fixed width of bus will mean that in many cases the width of the bus is not optimal for the application. Therefore an approach has been adopted which allows flexibility of the width of bus, but still ensures that modules are highly portable between designs.

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

For both 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.

### 3.15 Implementing a narrow slave on a wider bus

Figure 3-21 shows how a slave module, which has been originally designed to operate with a 32-bit data bus, can be easily converted to operate on a wider 64-bit bus. This only requires the addition of external logic, rather than any internal design changes, and therefore the technique is applicable to hard macrocells.



**Figure 3-21** Narrow slave on a wide bus

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

- Replicate the data onto both halves of the wide bus (as shown in the diagram above)
- Use an additional level of logic to ensure that only the appropriate half of the bus is changed. This will lead to a reduction in 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.

### 3.16 Implementing a wide slave on a narrow bus

The example in Figure 3-22 shows a wide slave being implemented on a narrow bus. Again only external logic is required and hence predesigned or imported blocks can be easily modified to work with a different width of data bus.

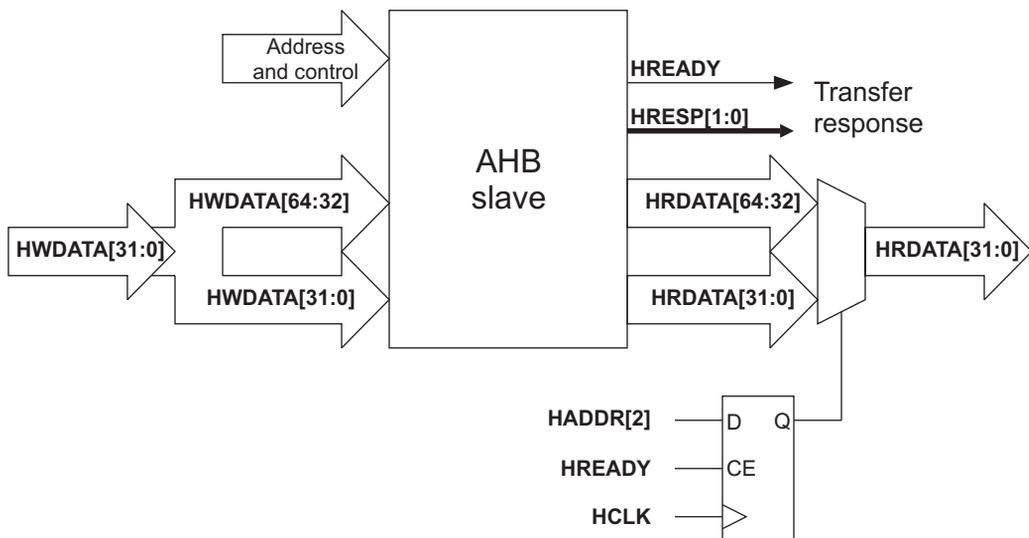


Figure 3-22 Wide slave on a narrow bus

#### 3.16.1 Masters

Bus masters can easily be modified to work on a wider bus than originally intended, in the same way that the slave is modified to work on a wider bus, by:

- multiplexing the input bus
- replication of the output bus.

However, bus masters cannot be made to work on a narrower bus than originally intended, unless there is some mechanism included within the master to limit the width of transfers that the bus master attempts. The master must never attempt a transfer where the width (as indicated by **HSIZE**) is wider than the data bus to which it is connected.

### 3.17 About the AHB AMBA components

This section describes each of the elements in an AHB-based AMBA system and provides the generic timing parameters that are required to analyze an AMBA design.

The following notation is used for the timing parameters:

- $T_{is}$  - input setup time
- $T_{ih}$  - input hold time
- $T_{ov}$  - output valid time
- $T_{oh}$  - output hold time.

### 3.18 AHB bus slave

An AHB bus slave responds to transfers initiated by bus masters within the system. The slave uses a **HSELx** select signal from the decoder to determine when it should respond to a bus transfer. All other signals required for the transfer, such as the address and control information, will be generated by the bus master.

#### 3.18.1 Interface diagram

Figure 3-23 shows an AHB bus slave interface.

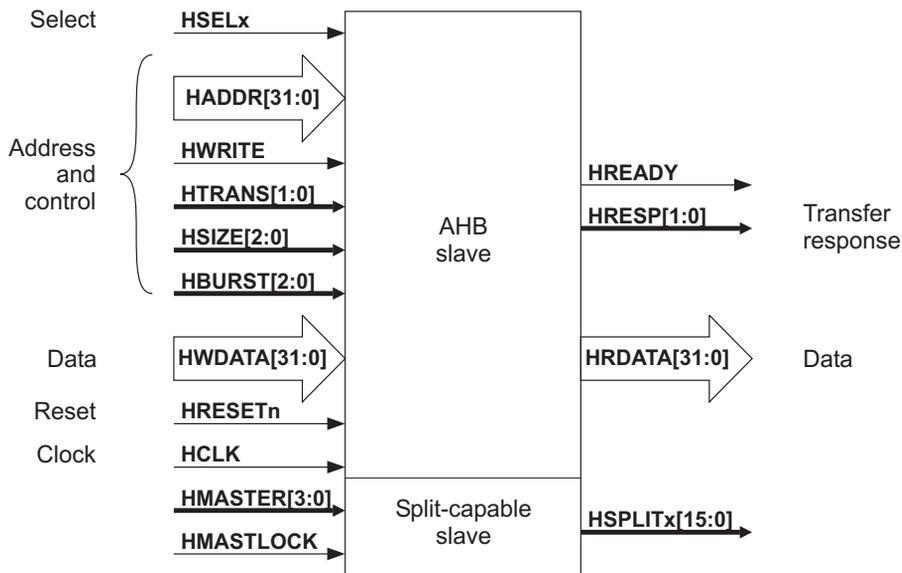


Figure 3-23 AHB bus slave interface

### 3.18.2 Timing diagrams

The following diagrams show the timing parameters related to an access to an AHB bus slave operating in an AMBA system:

- Figure 3-24 shows the AHB slave reset timing parameters
- Figure 3-25 shows the main AHB slave timing parameters
- Figure 3-26 shows the additional timing parameters for split-capable AHB slaves.

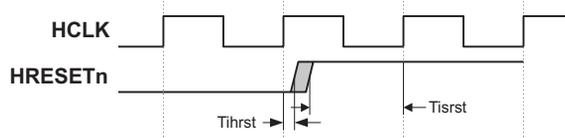


Figure 3-24 AHB slave reset timing

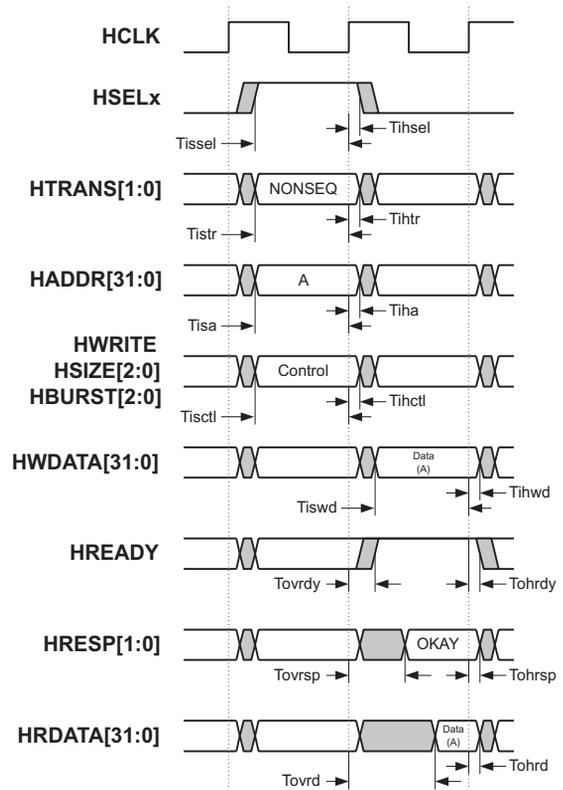


Figure 3-25 AHB timing parameters

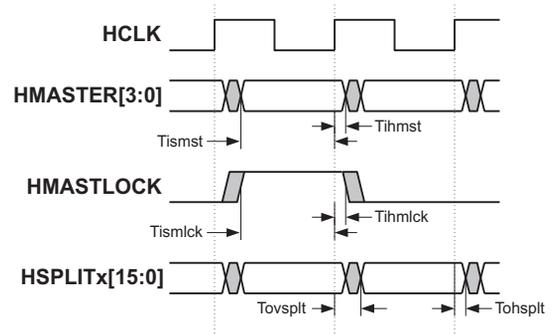


Figure 3-26 Additional split-capable slave parameters

### 3.18.3 Timing parameters

The timing parameters related to an AHB bus slave are given for input signals in Table 3-8 and for output signals in Table 3-9.

Table 3-8 AHB slave input parameters

Parameter	Description
$T_{clk}$	<b>HCLK</b> minimum clock period
$T_{isrst}$	<b>HRESETn</b> deasserted setup time before <b>HCLK</b>
$T_{ihrst}$	<b>HRESETn</b> deasserted hold time after <b>HCLK</b>
$T_{issel}$	<b>HSELx</b> setup time before <b>HCLK</b>
$T_{ihsel}$	<b>HSELx</b> hold time after <b>HCLK</b>
$T_{istr}$	Transfer type setup time before <b>HCLK</b>
$T_{ihtr}$	Transfer type hold time after <b>HCLK</b>
$T_{isa}$	<b>HADDR[31:0]</b> setup time before <b>HCLK</b>
$T_{iha}$	<b>HADDR[31:0]</b> hold time after <b>HCLK</b>
$T_{isctl}$	<b>HWRITE</b> , <b>HSIZE[2:0]</b> and <b>HBURST[2:0]</b> control signal setup time before <b>HCLK</b>

**Table 3-8 AHB slave input parameters (continued)**

<b>Parameter</b>	<b>Description</b>
$T_{ihctl}$	<b>HWRITE</b> , <b>HSIZE[2:0]</b> and <b>HBURST[2:0]</b> control signal hold time after <b>HCLK</b>
$T_{iswd}$	Write data setup time before <b>HCLK</b>
$T_{ihwd}$	Write data hold time after <b>HCLK</b>
$T_{isrdy}$	Ready setup time before <b>HCLK</b>
$T_{ihrdy}$	Ready hold time after <b>HCLK</b>
$T_{ismst}$	Master number setup time before <b>HCLK</b> (SPLIT-capable only)
$T_{ihmst}$	Master number hold time after <b>HCLK</b> (SPLIT-capable only)
$T_{ismlck}$	Master locked setup time before <b>HCLK</b> (SPLIT-capable only)
$T_{ihmlck}$	Master locked hold time after <b>HCLK</b> (SPLIT-capable only)

**Table 3-9 AHB slave output parameters**

<b>Parameter</b>	<b>Description</b>
$T_{ovrsp}$	Response valid time after <b>HCLK</b>
$T_{ohrsp}$	Response hold time after <b>HCLK</b>
$T_{ovrdy}$	Ready valid time after <b>HCLK</b>
$T_{ohrdy}$	Ready hold time after <b>HCLK</b>
$T_{ovsplt}$	Split valid time after <b>HCLK</b> (SPLIT-capable only)
$T_{ohsplt}$	Split hold time after <b>HCLK</b> (SPLIT-capable only)

### 3.19 AHB bus master

An AHB bus master has the most complex bus interface in an AMBA system. Typically an AMBA system designer would use predesigned bus masters and therefore would not need to be concerned with the detail of the bus master interface.

#### 3.19.1 Interface diagram

The interface diagram of an AHB bus master shows the main signal groups.

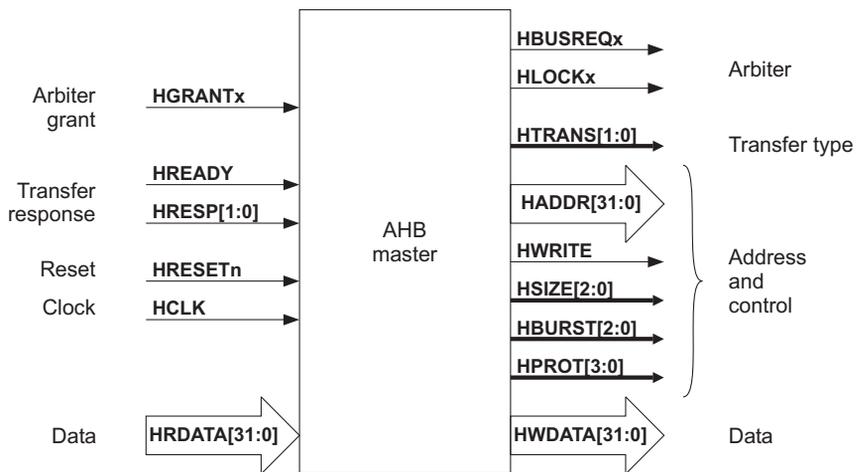


Figure 3-27 AHB bus master interface diagram

#### 3.19.2 Bus master timing diagrams

The following diagrams show the timing parameters related to an AHB bus master operating in an AMBA system:

- Figure 3-28 shows the AHB master reset timing parameters
- Figure 3-29 shows the AHB master transfer timing parameters
- Figure 3-30 shows the AHB master arbitration timing parameters.

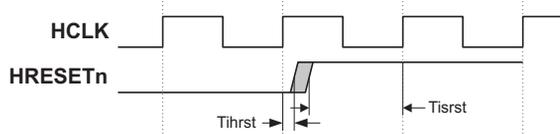


Figure 3-28 AHB master reset timing parameters

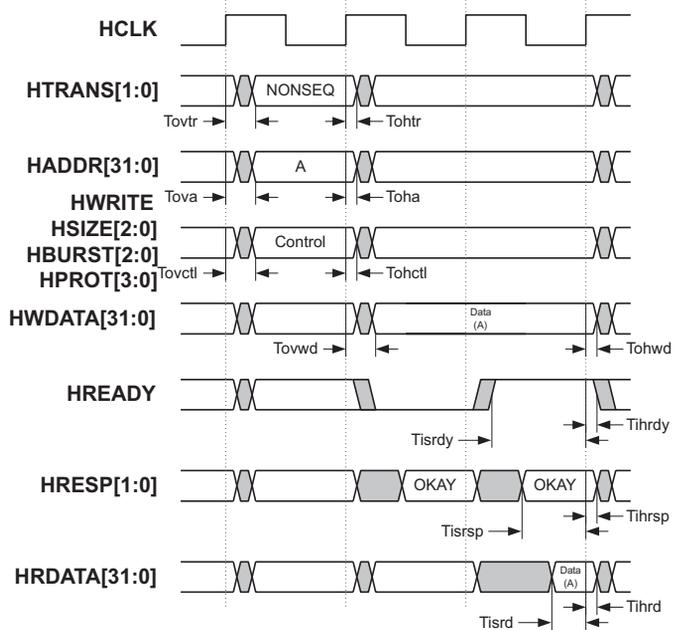


Figure 3-29 AHB master transfer timing parameters

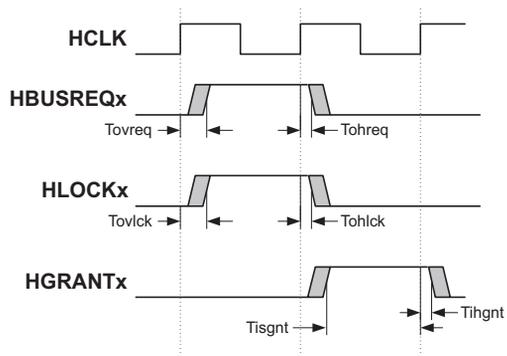


Figure 3-30 AHB master arbitration timing parameters

### 3.19.3 Timing parameters

The timing parameters related to an AHB bus master operating in an AMBA system are also shown in textual form in the following two tables. Table 3-10 details the input signals. Table 3-11 details output signals.

**Table 3-10 Bus master input timing parameters**

Parameter	Description
$T_{clk}$	<b>HCLK</b> minimum clock period time
$T_{isrst}$	Reset deasserted setup time before <b>HCLK</b>
$T_{ihrst}$	Reset deasserted hold time after <b>HCLK</b>
$T_{isgnt}$	<b>HGRANTx</b> setup time before <b>HCLK</b>
$T_{ihgnt}$	<b>HGRANTx</b> hold time after <b>HCLK</b>
$T_{isrdy}$	Ready setup time before <b>HCLK</b>
$T_{ihrdy}$	Ready hold time after <b>HCLK</b>
$T_{isrsp}$	Response setup time before <b>HCLK</b>
$T_{ihrsp}$	Response hold time after <b>HCLK</b>
$T_{isrd}$	Read data setup time before <b>HCLK</b>
$T_{ihrd}$	Read data hold time after <b>HCLK</b>

**Table 3-11 Bus master output timing parameters**

Parameter	Description
$T_{ovtr}$	Transfer type valid time after <b>HCLK</b>
$T_{ohtr}$	Transfer type hold time after <b>HCLK</b>
$T_{ova}$	Address valid time after <b>HCLK</b>
$T_{oha}$	Address hold time after <b>HCLK</b>
$T_{ovctl}$	Control signal valid time after <b>HCLK</b>

**Table 3-11 Bus master output timing parameters (continued)**

<b>Parameter</b>	<b>Description</b>
$T_{ohctl}$	Control signal hold time after <b>HCLK</b>
$T_{ovwd}$	Write data valid time after <b>HCLK</b>
$T_{ohwd}$	Write data hold time after <b>HCLK</b>
$T_{ovreq}$	Request valid time after <b>HCLK</b>
$T_{ohreq}$	Request hold time after <b>HCLK</b>
$T_{ovlck}$	Lock valid time after <b>HCLK</b>
$T_{ohlck}$	Lock hold time after <b>HCLK</b>

## 3.20 AHB arbiter

The role of the arbiter in an AMBA system is to control which master has access to the bus. Every bus master has a REQUEST/GRANT interface to the arbiter and the arbiter uses a prioritization scheme to decide which bus master is currently the highest priority master requesting the bus.

Each master also generates an HLOCKx signal which is used to indicate that the master requires exclusive access to the bus.

The detail of the priority scheme is not specified and is defined for each application. It is acceptable for the arbiter to use other signals, either AMBA or non-AMBA, to influence the priority scheme that is in use.

### 3.20.1 Interface diagram

Figure 3-31 shows the signal interface of an AHB arbiter.

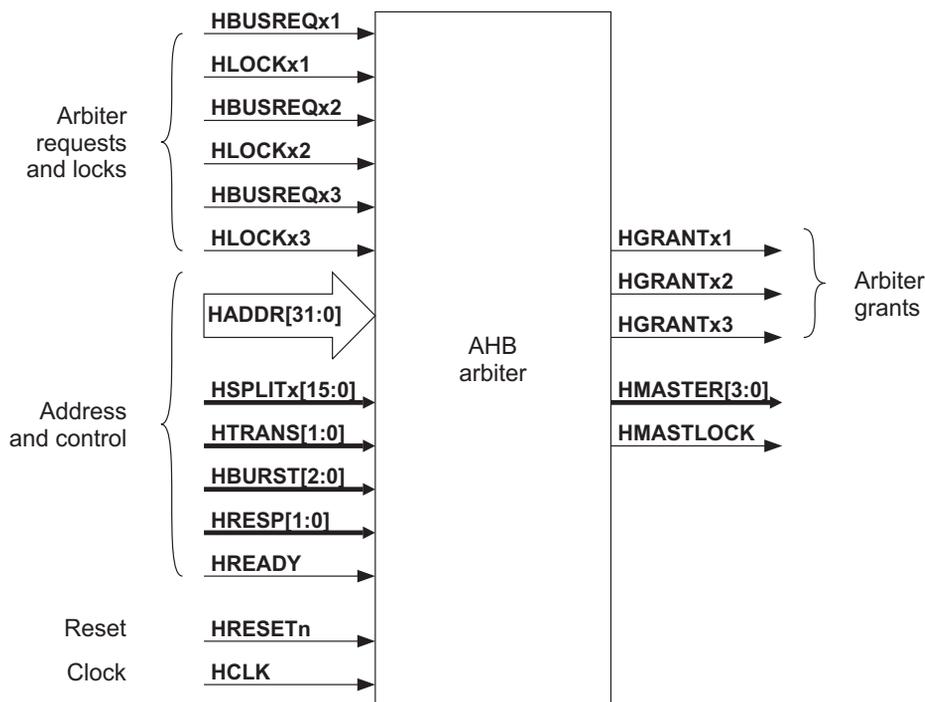


Figure 3-31 AHB arbiter interface diagram

### 3.20.2 Timing diagrams

The following diagrams show the timing parameters related to an AHB bus arbiter operating in an AMBA system:

- Figure 3-32 shows the AHB arbiter reset timing parameters
- Figure 3-33 shows the AHB arbiter transfer timing parameters
- Figure 3-34 shows the AHB arbiter split timing parameters.

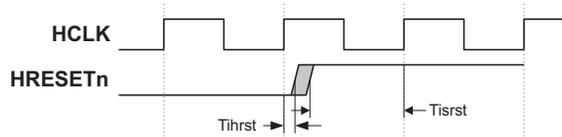


Figure 3-32 AHB arbiter reset timing parameters

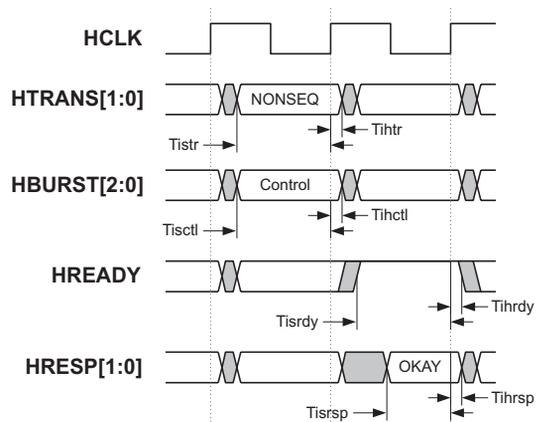


Figure 3-33 AHB arbiter transfer timing parameters

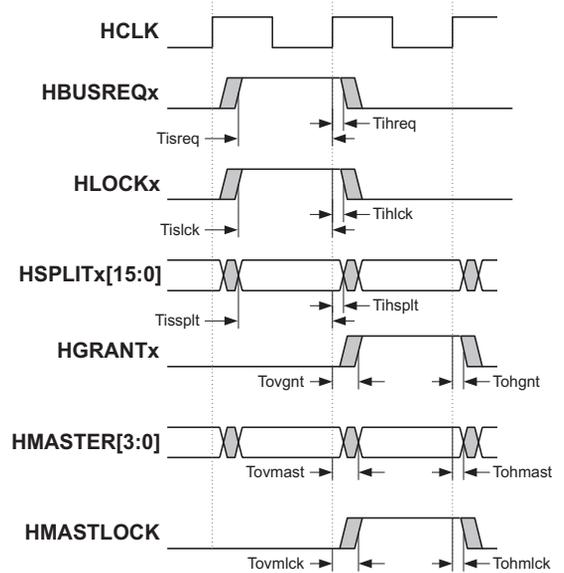


Figure 3-34 AHB arbiter split timing parameters

### 3.20.3 Timing parameters

The timing parameters related to an AHB arbiter are given in the following tables:

- Table 3-12 is for input signals
- Table 3-13 is for output signals.

Table 3-12 AHB arbiter input parameters

Parameter	Description
$T_{clk}$	<b>HCLK</b> minimum clock period
$T_{isrst}$	Reset deasserted setup time before <b>HCLK</b>
$T_{ihrst}$	Reset deasserted hold time after <b>HCLK</b>
$T_{isrdy}$	Ready setup time before <b>HCLK</b>
$T_{ihrdy}$	Ready hold time after <b>HCLK</b>
$T_{isrsp}$	Response setup time before <b>HCLK</b>

Table 3-12 AHB arbiter input parameters (continued)

Parameter	Description
$T_{ihrsp}$	Response hold time after <b>HCLK</b>
$T_{isreq}$	Request setup time before <b>HCLK</b>
$T_{ihreq}$	Request hold time after <b>HCLK</b>
$T_{islck}$	Lock setup time before <b>HCLK</b>
$T_{ihlck}$	Lock hold time after <b>HCLK</b>
$T_{issplt}$	Split setup time before <b>HCLK</b>
$T_{ihspit}$	Split hold time after <b>HCLK</b>
$T_{istr}$	Transfer type setup time before <b>HCLK</b>
$T_{ihtr}$	Transfer type hold time after <b>HCLK</b>
$T_{isctl}$	Control signal setup time before <b>HCLK</b>
$T_{ihctl}$	Control signal hold time after <b>HCLK</b>

Table 3-13 AHB arbiter output parameters

Parameter	Description
$T_{ovgnt}$	Grant valid time after <b>HCLK</b>
$T_{ohgnt}$	Grant hold time after <b>HCLK</b>
$T_{ovmst}$	Master number valid time after <b>HCLK</b>
$T_{ohmst}$	Master number hold time after <b>HCLK</b>
$T_{ovmlck}$	Master locked valid time after <b>HCLK</b>
$T_{ohmlck}$	Master locked hold time after <b>HCLK</b>

### 3.21 AHB decoder

The decoder in an AMBA system is used to perform a centralized address decoding function, which improves the portability of peripherals, by making them independent of the system memory map.

#### 3.21.1 Interface diagram

Figure 3-35 shows an AHB decoder.

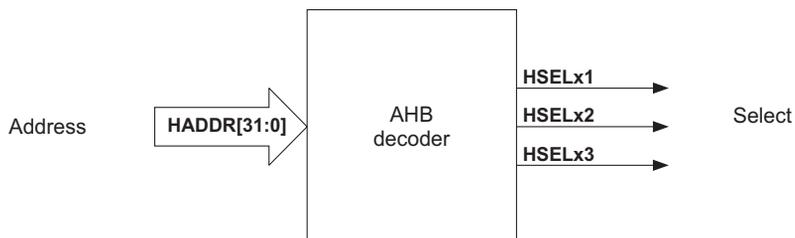


Figure 3-35 AHB decoder interface diagram

#### 3.21.2 Timing diagram

The timing parameters for an AHB decoder are shown in Figure 3-36.

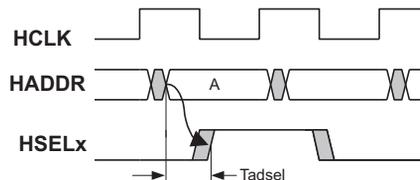


Figure 3-36 AHB decoder timing parameter

#### 3.21.3 Timing parameter

The timing parameter related to an AHB decoder is given in Table 3-14.

Table 3-14 AHB decoder output parameter

Parameter	Description
$T_{adssel}$	Delay from Address to Select valid



# Chapter 4

## AMBA ASB

This chapter introduces the *Advanced Microcontroller Bus Architecture* (AMBA) Advanced System Bus specification. It contains the following sections:

- *About the AMBA ASB* on page 4-2
- *AMBA ASB description* on page 4-4
- *ASB transfers* on page 4-6
- *Address decode* on page 4-14
- *Transfer response* on page 4-16
- *Multi-master operation* on page 4-19
- *Reset operation* on page 4-23
- *Description of ASB signals* on page 4-25
- *About the ASB AMBA components* on page 4-46
- *ASB bus slave* on page 4-47
- *ASB bus master* on page 4-52
- *ASB decoder* on page 4-63
- *ASB arbiter* on page 4-71.

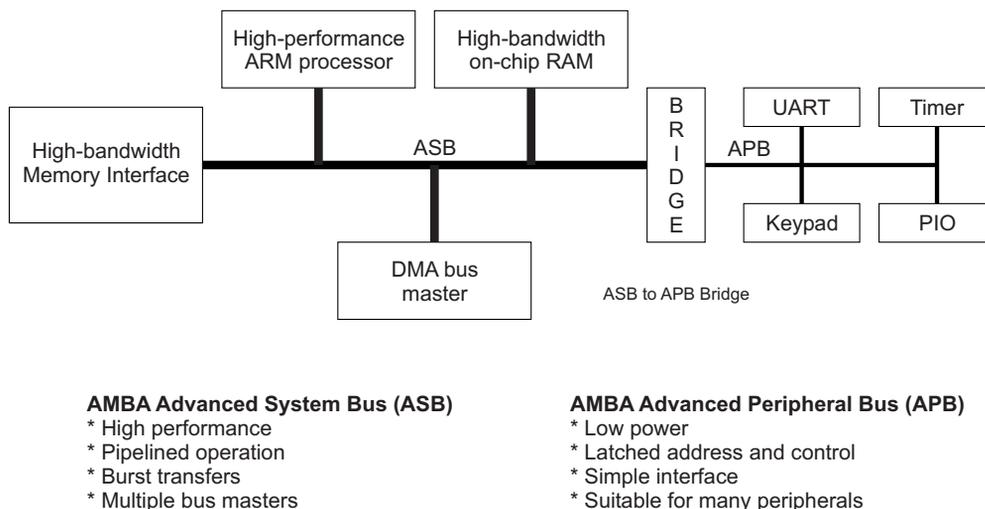
## 4.1 About the AMBA ASB

The *Advanced System Bus (ASB)* specification defines a high-performance bus that can be used in the design of high performance 16 and 32-bit embedded microcontrollers.

AMBA ASB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions. The bus also provides the test infrastructure for modular macrocell test and diagnostic access.

### 4.1.1 A typical AMBA ASB-based microcontroller

An AMBA-based microcontroller typically consists of a high-performance system *backbone* bus, able to sustain the external memory bandwidth, on which the CPU and other *Direct Memory Access (DMA)* devices reside, plus a bridge to a narrower APB bus on which the lower bandwidth peripheral devices are located. Figure 4-1 shows both ASB and APB in a typical AMBA system.



**Figure 4-1 A typical AMBA system**

The external memory interface is application-specific and may only have a narrow data path, but it supports a test access mode which allows the internal ASB and APB modules to be tested in isolation with system-independent test sets.

### 4.1.2 AMBA ASB and APB

The APB appears as a local secondary bus that is encapsulated as a single ASB slave device. APB provides a low-power extension to the system bus which builds on ASB signals directly.

The APB bridge appears as a slave module which handles the bus handshake and control signal retiming on behalf of the local peripheral bus. By defining the APB interface from the starting point of the system bus, the benefits of the system diagnostics and test methodology can be exploited.

## 4.2 AMBA ASB description

The ASB is a high-performance pipelined bus, which supports multiple bus masters.

The basic flow of the bus operation is:

1. The arbiter determines which master is granted access to the bus.
2. When granted, a master initiates transfers on the bus.
3. The decoder uses the high order address lines to select a bus slave.
4. The slave provides a transfer response back to the bus master and data is transferred between the master and slave.

There are three types of transfer that can occur on the ASB:

### NONSEQUENTIAL

Used for single transfers or for the first transfer of a burst.

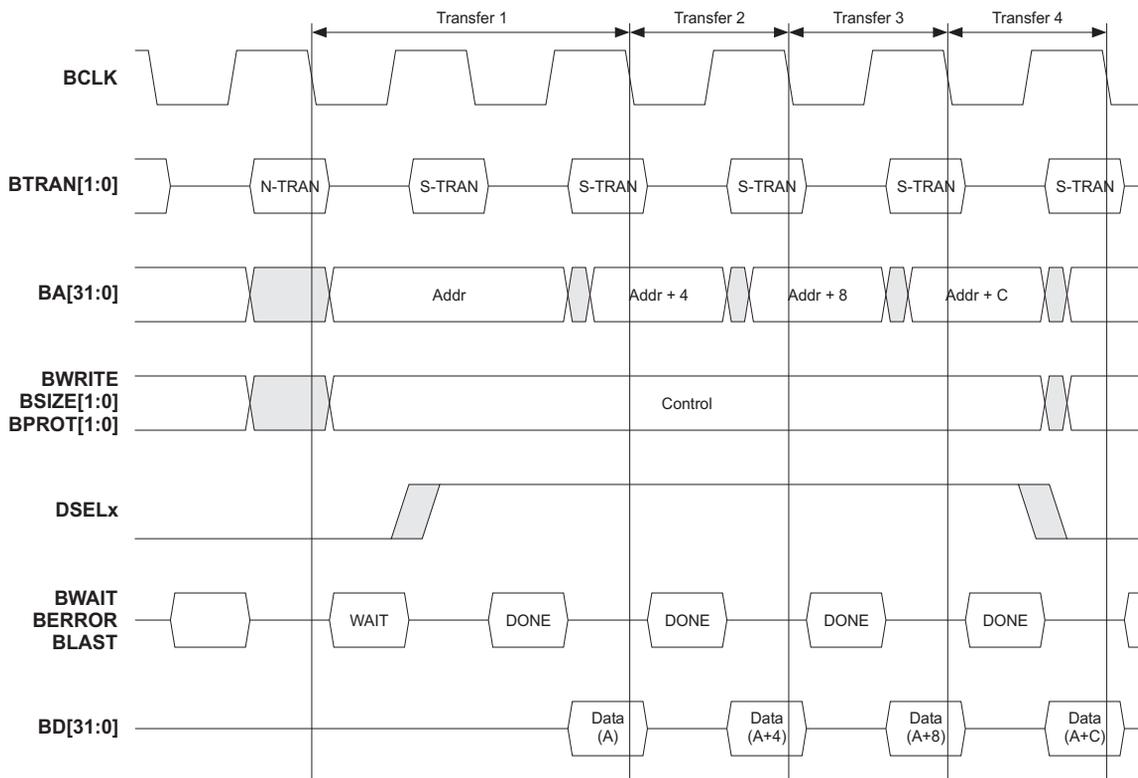
### SEQUENTIAL

Used for transfers in a burst. The address of a SEQUENTIAL transfer is always related to the previous transfer.

### ADDRESS-ONLY

Used when no data movement is required. The three main uses for ADDRESS-ONLY transfers are for IDLE cycles, for bus master HANDOVER cycles, and for speculative address decoding without committing to a data transfer.

Figure 4-2 shows the use of NONSEQUENTIAL and SEQUENTIAL transfers to perform a burst transaction.



**Figure 4-2 ASB transfers**

The burst starts with a NONSEQUENTIAL transfer to address A. The following SEQUENTIAL transfers are to successive addresses A+4, A+8 and A+12.

### 4.3 ASB transfers

When a master has been granted the bus it can perform the following transfers:

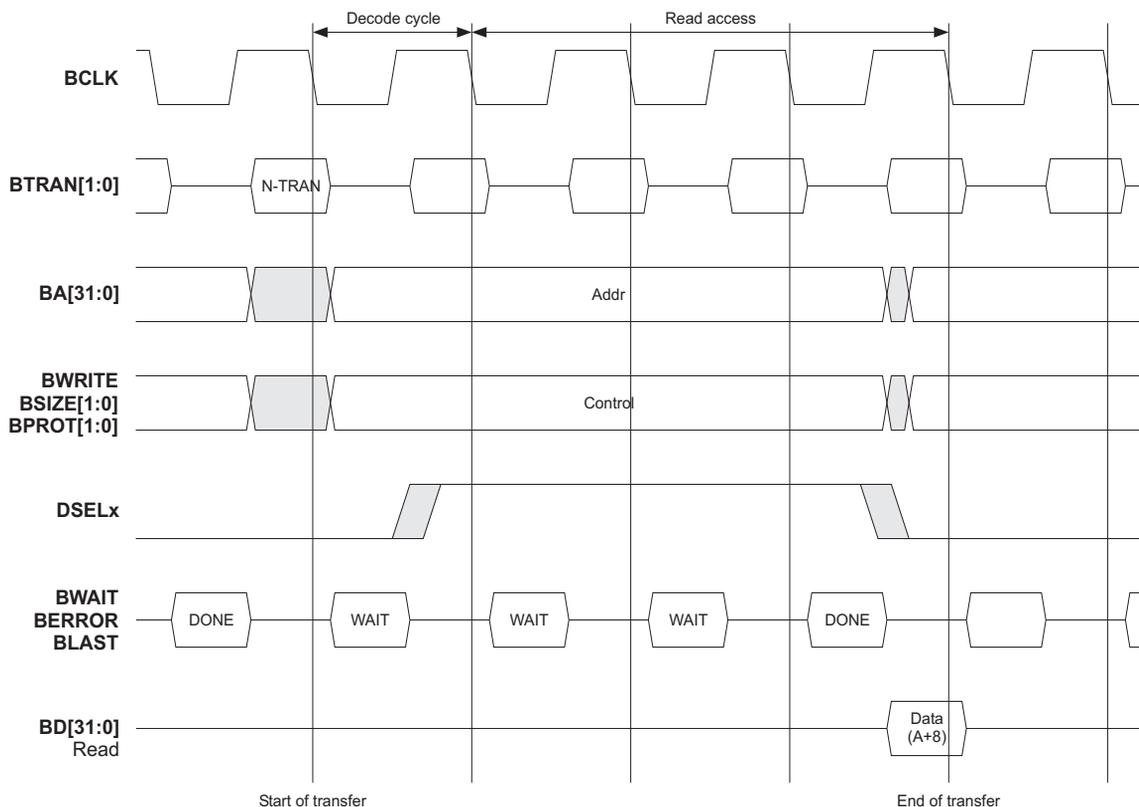
- NONSEQUENTIAL data transfer
- SEQUENTIAL data transfer
- ADDRESS-ONLY transfer.

A transfer is defined as starting at the falling edge of **BCLK** after the previous transfer has completed, as indicated by **BWAIT** being LOW, and running until the falling edge of **BCLK** after a complete transfer response is received, again indicated by **BWAIT** being LOW.

The type of transfer that a bus master will perform can be determined by the value on the **BTRAN** signals at the start of the transfer. During the transfer the **BTRAN** signals will change to indicate the type of the following transfer.

### 4.3.1 Nonsequential transfer

A NONSEQUENTIAL transfer occurs for either a single transfer or at the start of a burst of transfers. Figure 4-3 shows a typical NONSEQUENTIAL read transfer including wait states.



**Figure 4-3 Nonsequential transfer**

The following points should be noted:

- The address and control signals start to change in the **BCLK HIGH** phase before the transfer starts.
- For a NONSEQUENTIAL transfer a valid address may not be available until very late in the **BCLK HIGH** phase, or even until the start of the clock **LOW** phase at the beginning of the transfer.

- The decoder, which requires a stable address in order to select the correct slave, will automatically insert a wait state in the first cycle of **NONSEQUENTIAL** transfers. This is referred to as a **DECODE** cycle and provides an adequate time for the decoder to examine the high order address lines and assert the appropriate **DSELx** during the **HIGH** phase of the **DECODE** cycle.
- For the remaining cycles of the transfer, the slave will provide a transfer response and the data exchange will occur between the master and slave.

---

**Note**

---

In certain system designs, which are typically those with a low-frequency system clock, the address is valid early enough in the **BCLK HIGH** phase before the start of the transfer, allowing the decoder to generate a valid **DSELx** signal before the falling edge of **BCLK**. Such systems do not require the addition of a **DECODE** cycle at the start of the **NONSEQUENTIAL** transfers and the operation of such a system is described in more detail in *Address decode* on page 4-14.

---

- The data bus, **BD[31:0]**, must be valid by the falling edge of **BCLK** at the end of the transfer. During a write cycle, the bus master is responsible for driving the data bus, which it will do from the start of the clock **HIGH** phase, in order that the slave may accept valid data by the falling edge of the clock. During a read cycle the appropriate slave must drive the data bus, such that it is valid by the end of the **HIGH** phase.
- Because a number of different bus slaves may drive data on to the ASB it is necessary to ensure that different slaves do not overlap when driving data onto the bus. An entire phase of non-overlap is provided as slaves and masters may not drive data during the clock **LOW** phase at the start of a **NONSEQUENTIAL** transfer.
- As many of the bus signals are shared and have turnaround periods when there is no active driver, it is necessary to ensure that bus hold cells are provided to prevent floating levels being present on the bus.

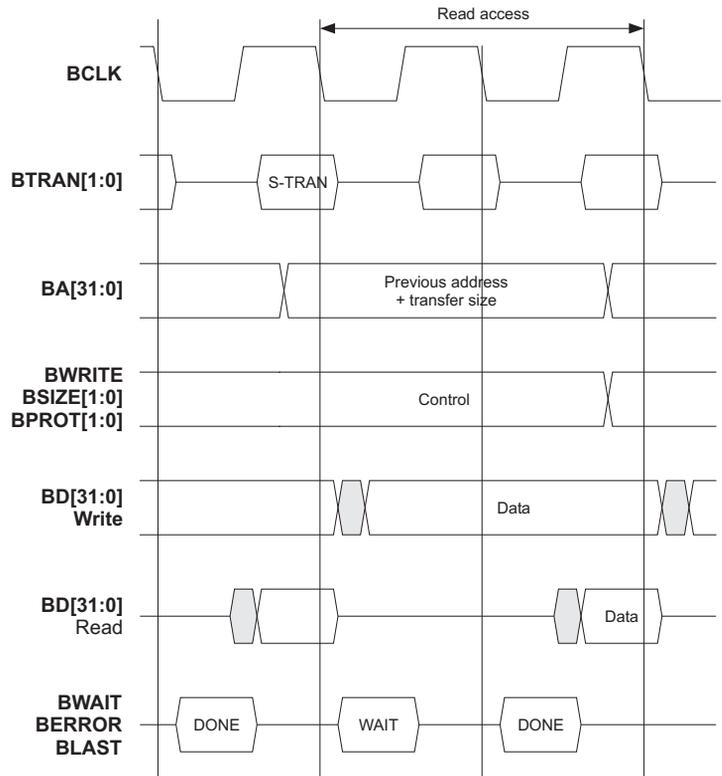
### 4.3.2 Sequential transfer

A **SEQUENTIAL** transfer occurs when the address is related to that of the previous transfer. The control information, as indicated by **BWRITE**, **BPROT** and **BSIZE**, will be the same as the previous transfer.

If the **SEQUENTIAL** transfer follows a **NONSEQUENTIAL** or another **SEQUENTIAL** transfer, the address can be calculated by using the previous size and address. For example a burst of word accesses would be to addresses A, A+4, A+8, whereas a burst of halfword accesses would be to addresses A, A+2, A+4.

If a **SEQUENTIAL** transfer follows an **ADDRESS-ONLY** cycle then the address will be the same as that of the **ADDRESS-ONLY** cycle. This combination of an **ADDRESS-ONLY** followed by **SEQUENTIAL** allows both a single access using a **SEQUENTIAL** transfer and also allows a burst of transfers to start with a **SEQUENTIAL** transfer. An example of the use of an **ADDRESS-ONLY** followed by **SEQUENTIAL** is shown later in Figure 4-6.

Figure 4-4 shows a **SEQUENTIAL** transfer with one wait state. This closely resembles a **NONSEQUENTIAL** transfer.



**Figure 4-4 Sequential transfer**

The main differences are:

- **BTRAN** signals indicate a **SEQUENTIAL** transfer
- address is always valid in the **BCLK HIGH** phase at the start of the transfer
- address is related to the preceding transfer

- control information remains the same as the preceding transfer
- for a write the data bus is driven throughout the entire transfer.

The data bus, **BD[31:0]**, can be driven throughout the entire transfer because, unlike the **NONSEQUENTIAL** case, there is no requirement to provide a period of time to allow for bus turnaround.

### 4.3.3 Address-only transfer

An **ADDRESS-ONLY** transfer indicates that no data transaction is required. During an **ADDRESS-ONLY** transfer it is possible that the address and control information may also be invalid. The only signals that must be driven to valid levels are:

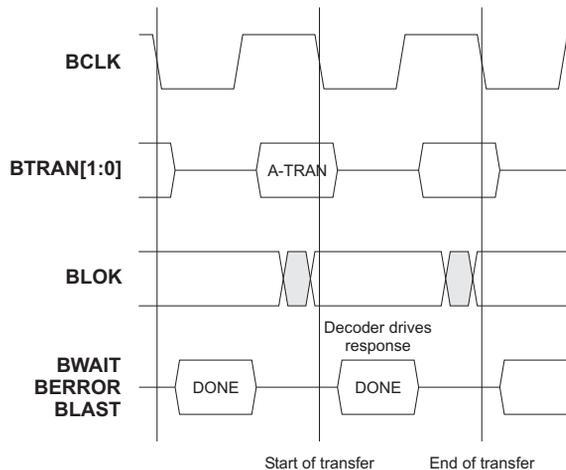
- **BTRAN** - to indicate the type of the next transfer
- **BLOK** - to allow the arbitration process to continue.

As **ADDRESS-ONLY** transactions do not access slaves on the bus, they only require a single cycle and therefore the **BWAIT** signal will be **LOW**. This signal is driven by the bus decoder, as no slave will be selected during the **ADDRESS-ONLY** cycle. A bus master may perform a number of **ADDRESS-ONLY** transfers in succession if it does not require the bus for data transfer.

The **ADDRESS-ONLY** transfer can be used in three different ways:

- as a true **IDLE** cycle (when the bus master does not require the bus)
- to speculatively broadcast an address for the next transfer, without committing to the transfer
- to provide a turnaround cycle during bus master handover.

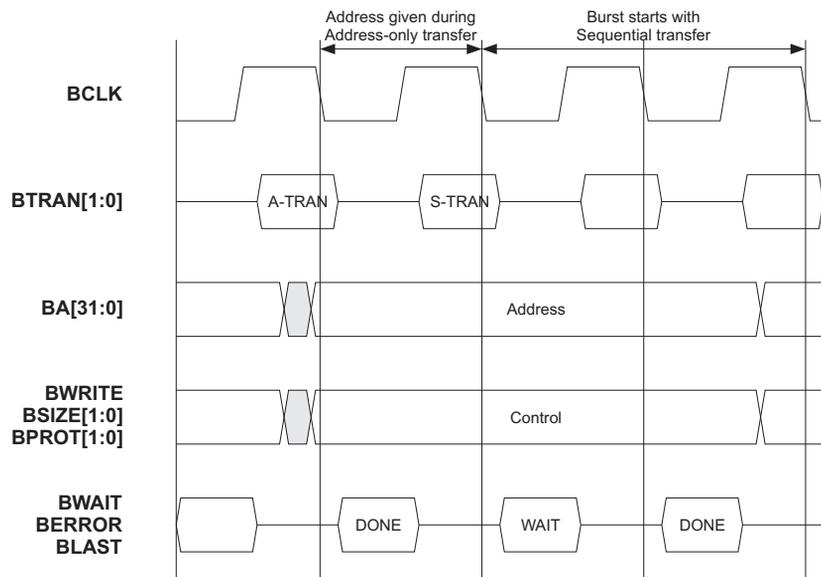
If the ADDRESS-ONLY transfer is used as a true IDLE cycle then the address and control signals are not required to be valid at any point during the transfer (see Figure 4-5).



**Figure 4-5 Address-only transfer**

The **BLOK** signal is the only exception and this must be driven to a valid level during all ADDRESS-ONLY transfers to allow the arbitration process to continue.

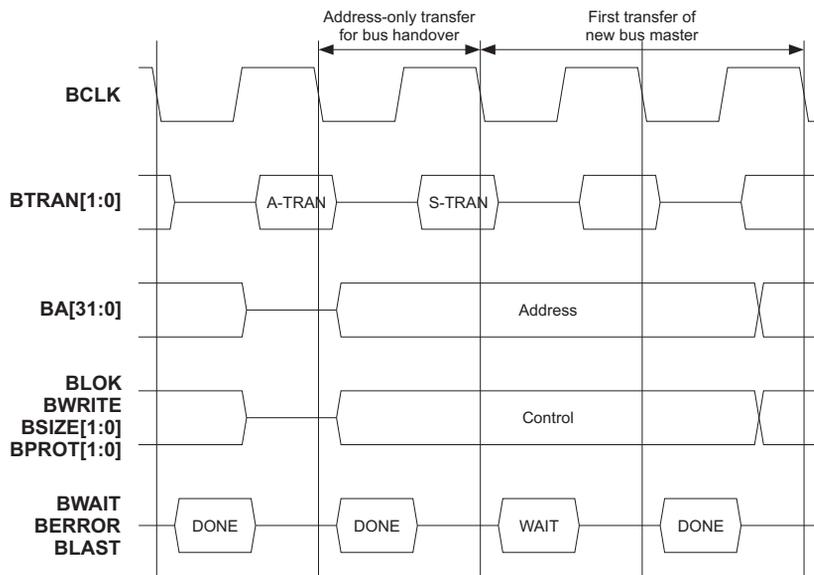
The second use of the ADDRESS-ONLY transfer is to speculatively broadcast the address for a transfer, without actually committing to the transfer (see Figure 4-6).



**Figure 4-6 Address-only transfer to start burst**

Using an ADDRESS-ONLY transfer to speculatively broadcast the address allows address decoding to be performed by the decoder during the ADDRESS-ONLY cycle. If the bus master then commits to the burst it is possible to start the burst with a SEQUENTIAL transfer, thus removing the need for an extra DECODE cycle before the transfer starts.

The final use of an ADDRESS-ONLY transfer is to provide a turnaround period during bus master handover (see Figure 4-7).



**Figure 4-7 Address-only transfer for bus master handover**

A bus master which becomes granted on the bus must start with an ADDRESS-ONLY transfer and, in this case, the new bus master does not drive the address and control signals immediately, but provides a phase of turnaround before driving the signals in the LOW phase of the transfer.

**Note**

In this case, the address and control information will not become valid until the LOW phase of **BCLK**.

## 4.4 Address decode

In an ASB-based AMBA system the address decoding is performed by a centralized decoder.

The decoder uses the type of each transfer to determine which of the following functions should be performed:

- For an ADDRESS-ONLY transfer the decoder will respond with a DONE transfer response and no slaves will be selected. During ADDRESS-ONLY transfers the decoder performs an address decode speculatively in case the ADDRESS-ONLY transfer is followed immediately by a SEQUENTIAL transfer.
- For NONSEQUENTIAL transfers (or when the previous transfer was terminated with a LAST transaction response) the decoder will insert a single wait state at the start of the transfer to allow sufficient time for address decoding (although the additional wait state may not be required in all systems).

The additional wait state inserted by the decoder is referred to as a *DECODE cycle* and during the DECODE cycle no select signals, **DSELx**, are asserted.

In the second cycle of the transfer the decoder will either select the appropriate slave or provide an ERROR transfer response.

An ERROR response is provided in the following circumstances:

- there are no slaves present at the address of the transfer
- the transfer is to a protected region of memory
- the alignment of the transfer is not supported by the memory system.

In the more usual case of a valid transfer, the decoder will assert the appropriate slave **DSELx** signal and allow the selected slave to provide the transfer response for the remaining cycles of the transfer.

- For SEQUENTIAL transfers the decoder asserts the appropriate **DSELx** signal and the selected slave provides the transfer response. It is not necessary for the decoder to decode the address as this will have been performed in a previous NONSEQUENTIAL or ADDRESS-ONLY transfer.

As the decoder does not perform an address decode on SEQUENTIAL transfers it is necessary for the slave to provide a LAST transfer response if a transfer is about to cross a memory boundary. The decoder is also responsible for generating an internal version of the **LAST** signal when the decoder detects that a SEQUENTIAL transfer will cross a memory boundary.

The insertion of a DECODE cycle on NONSEQUENTIAL transfers can be used to improve the performance of the system. In a typical design the time required for address decoding will increase the critical path of an access to a slave and often result in the

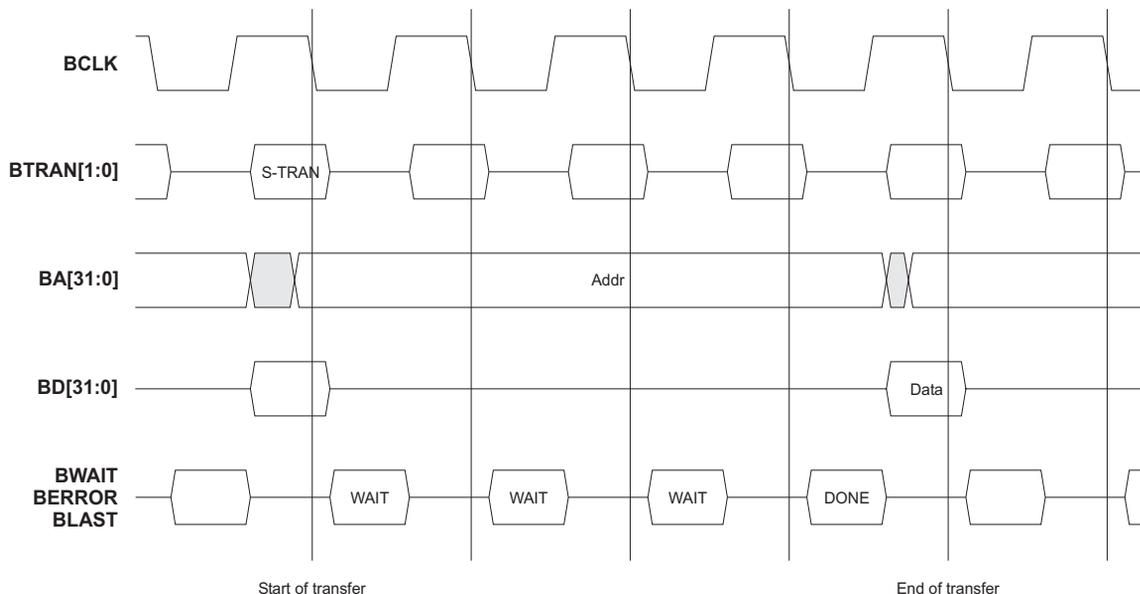
need for additional wait states. The decoder can be used to reduce this overhead by automatically inserting a DECODE cycle on NONSEQUENTIAL transfers only, but allowing SEQUENTIAL transfers to complete without additional wait states.

In some systems, typically those with a low clock frequency, additional wait states are not required for address decoding and in such systems the decoder may be simplified, such that both SEQUENTIAL and NONSEQUENTIAL transfers occur without the addition of a DECODE cycle.

## 4.5 Transfer response

For every transfer that is initiated by a bus master a response must be generated and this is provided either by the decoder or by the selected bus slave. The transfer response is provided using the **BWAIT**, **BERROR** and **BLAST** signals, which are driven during the LOW phase of the clock.

Figure 4-8 shows an example of how the transfer response is used to insert three wait states in order to extend a transfer.



**Figure 4-8 Transfer response**

The following transfer responses are available:

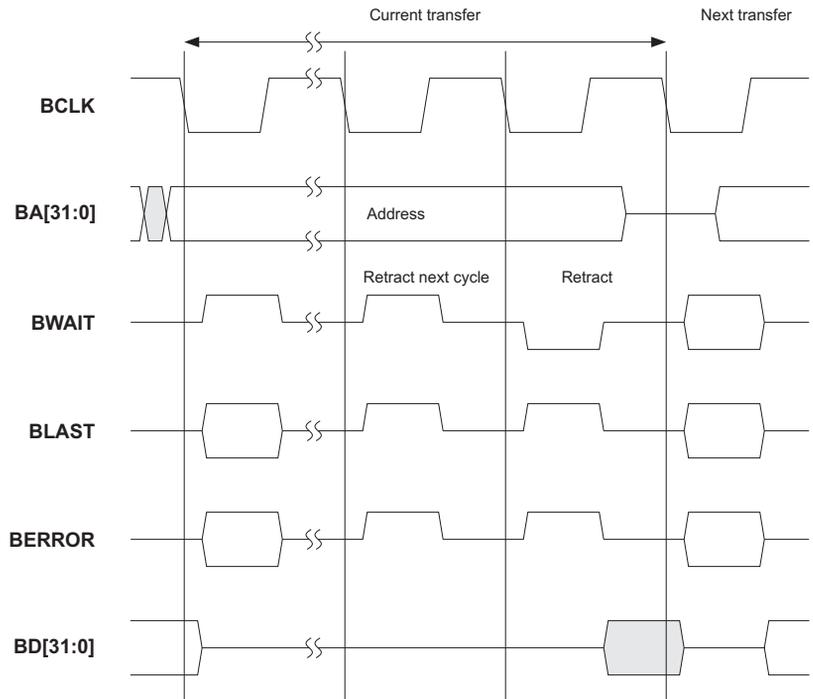
- |              |  |
|--------------|--|
| <b>WAIT</b>  | The transfer must be extended before it can complete.  |
| <b>DONE</b>  | The transfer has completed successfully.   |
| <b>ERROR</b> | The transfer has completed, but an error has occurred. The error condition should be signalled to the bus master so it is aware that the transfer has been unsuccessful. |

**LAST** The transfer has completed successfully, but the slave is unable to accept further burst transfers or a memory boundary has been reached. This response is identical to **DONE** for the bus master, but indicates to the decoder that it must insert a **DECODE** cycle at the start of the next transfer.

**RETRACT** The transfer has not yet completed, so the bus master should retry the transfer. The **RETRACT** response can be used by a slave to signal to a bus master that the transfer can complete, but this may take a number of bus cycles.

Using the **RETRACT** response prevents the bus from being locked up by a transfer which may take a long time to complete and frees the bus for use by a higher priority bus master.

Unlike the other response codes, which take a single cycle, the **RETRACT** response is a two-stage response, as shown in Figure 4-9.



**Figure 4-9 Retract operation**

The following points should be noted:

- In the first stage the slave signals to the bus master that a RETRACT is going to take place, using the RETNEXT response (**BWAIT**, **BLAST** and **BERROR** all HIGH).
- In the second stage the transfer is completed when the slave provides the RETRACT response (**BWAIT** LOW, **BLAST** and **BERROR** both HIGH).

This two-stage response provides the bus master with adequate warning that it should not consider the transfer to have completed when the **BWAIT** signal goes LOW.

All bus masters must support the RETRACT mechanism, however not all slaves are required to implement the RETRACT response. Typically, a RETRACT response would only be provided by a slave which does not have a short guaranteed completion time and hence could deadlock the bus for a significant period of time.

For most transfers the response will be provided by the selected bus slave, however the decoder provides the response when:

- the transfer is ADDRESS-ONLY
- the transfer is to an area of memory where there are no bus slaves
- an access violation occurs to a protected region of memory.

## 4.6 Multi-master operation

The AMBA bus specification supports multiple bus masters on the high-performance ASB. A simple two-wire request and grant mechanism is implemented between the arbiter and each bus master. The arbiter ensures that only one bus master is active on the bus and also ensures that when no masters are requesting the bus a default master is granted.

The specification also supports a shared lock signal. This allows bus masters to indicate that the current transfer should not be separated from the following transfer and will prevent other bus masters from gaining access to the bus until the locked transfers have completed.

Efficient arbitration is important to reduce *dead-time* between successive masters being active on the bus. The bus protocol supports pipelined arbitration, such that arbitration for the next transfer is performed during the current transfer.

The arbitration protocol is defined, but the prioritization is flexible and left to the application. Typically, however, the *test interface* would be given the highest priority to ensure test access under all conditions. Every system must also include a default bus master, which is granted the bus when no bus masters are requesting it.

The request signal, **AREQx**, from each bus master to the arbiter indicates that the bus master requires the bus. The grant signal from the arbiter to the bus master, **AGNTx**, indicates that the bus master is currently the highest priority master requesting the bus.

The bus master:

- must drive the **BTRAN** signals during **BCLK** HIGH when **AGNTx** is HIGH
- will become granted when **AGNTx** is HIGH and **BWAIT** is LOW on a rising edge of **BCLK**.

The shared bus lock signal, **BLOK**, indicates to the arbiter that the following transfer is indivisible from the current transfer and that no other bus master should be given access to the bus.

A bus master must always drive a valid level on the **BLOK** signal when granted the bus to ensure that the arbitration process can continue, even if the bus master is not performing any transfers.

### 4.6.1 Arbiter

The arbiter functions as follows:

1. Bus masters assert **AREQx** during the HIGH phase of **BCLK**.
2. The arbiter samples all **AREQx** signals on the falling edge of **BCLK**.
3. During the LOW phase of **BCLK** the arbiter also samples the **BLOK** signal and then asserts the appropriate **AGNTx** signal:
  - if **BLOK** is LOW, then the arbiter will grant the highest priority bus master
  - if **BLOK** is HIGH, then the arbiter will keep the same bus master granted.

The arbiter can update the grant signals every bus cycle. However, a new bus master can only become granted and start driving the bus when the current transfer completes, as indicated by **BWAIT** being LOW. Therefore, it is possible for the potential next bus master to change during waited transfers.

The **BLOK** signal is ignored by the arbiter during the single cycle of handover between two different bus masters.

If no bus masters are requesting the bus then the arbiter must grant the default bus master.

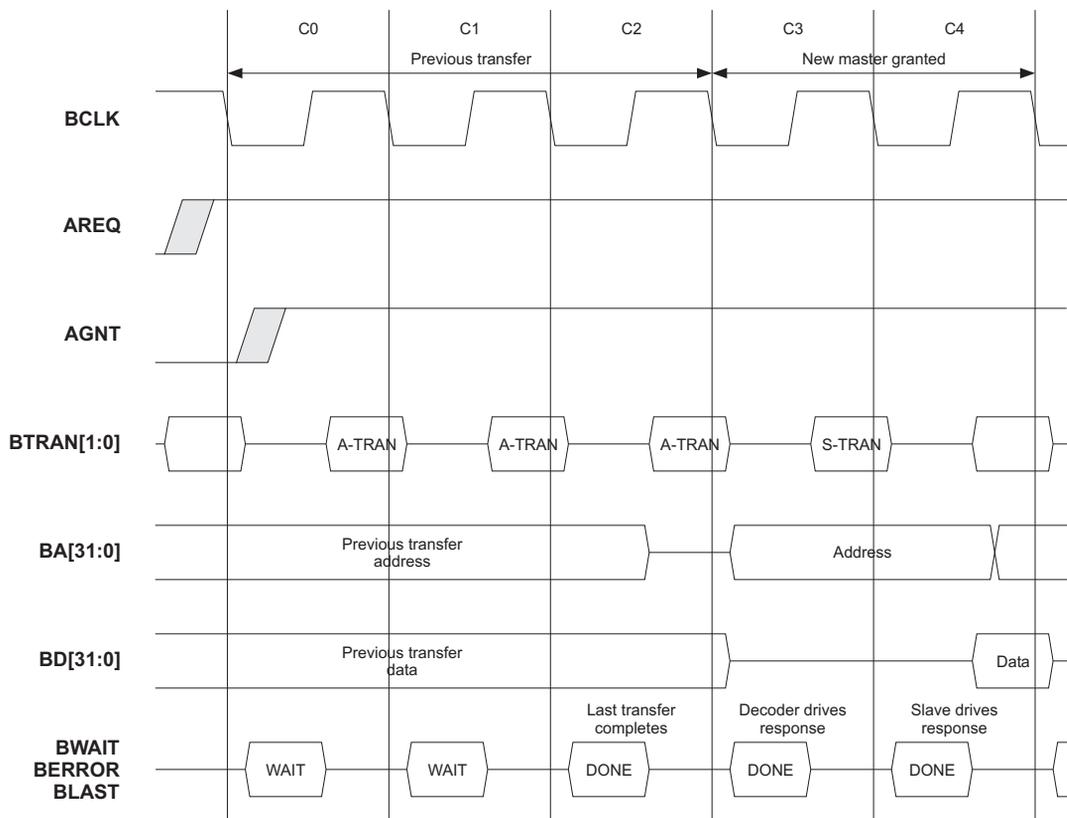
The arbitration protocol is defined, but the prioritization is flexible and left to the application. A simple fixed-priority scheme may be used. Alternatively, a more complex scheme can be implemented if required by the application.

### 4.6.2 Bus master handover

Bus master handover occurs when a bus master, which is not currently granted the bus, becomes the new granted bus master.

A bus master becomes granted when **AGNTx** is HIGH and **BWAIT** is LOW. **AGNT** HIGH indicates that the bus master is currently the highest priority master requesting the bus and **BWAIT** LOW indicates the previous transfer has completed.

Figure 4-10 shows the bus master handover process.



**Figure 4-10 Bus master handover**

The following points should be noted:

- When **AGNT<sub>x</sub>** is asserted a bus master must drive the **BTRAN** signals during **BCLK** HIGH. This may continue for many cycles if the previous transfer is waited.
- Prior to handover **BTRAN** must indicate an ADDRESS-ONLY cycle because the new bus master must commence with an ADDRESS-ONLY cycle to allow for bus turnaround.
- When the previous transfer completes the new bus master will become granted.
- In the last clock HIGH phase of the previous transfer the address bus will stop being driven by the previous bus master.

- The new bus master starts to drive the address bus and control signals during the clock LOW phase. The first transfer may then commence in the following bus cycle.

During a waited transfer, bus master handover may be delayed and it is possible that the **AGNTx** to a particular bus master may be asserted and then negated, if another higher priority bus master then requests the bus, before the current transfer has completed.

#### 4.6.3 Default bus master

Every system must be designed with a single default bus master, which will be granted when no other bus master is requesting the bus. The default bus master is responsible for driving the following signals to ensure the bus remains active:

- **BTRAN** must be driven to indicate ADDRESS-ONLY transfer
- **BLOK** must be driven LOW.

#### 4.6.4 Locked transfers

It is important that bus masters do not attempt to perform locked transfers to slaves which can give a RETRACT response. There are two reasons for this:

- The bus could remain locked for a large number of cycles.
- If the RETRACT occurs on the last transfer of the locked sequence, then the arbiter could have changed ownership of the bus to the next master before it completes and therefore the final transfer will not have been locked to the sequence.

## 4.7 Reset operation

The reset signal, **BnRES**, is active LOW and may be asserted asynchronously to guarantee the bus is in a safe state. During reset the following actions occur on the bus:

- the arbiter grants the default bus master
- the default bus master must:
  - drive **BTRAN** to indicate ADDRESS-ONLY transfer
  - drive **BLOK** LOW to allow arbitration.
- all other bus masters tristate shared bus signals
- the decoder must:
  - de-assert all slave select signals, **DSELx**
  - provide the appropriate transfer response.
- all slaves tristate shared bus signals.

### 4.7.1 Exit from reset

Figure 4-11 shows an example of the exit from reset sequence.

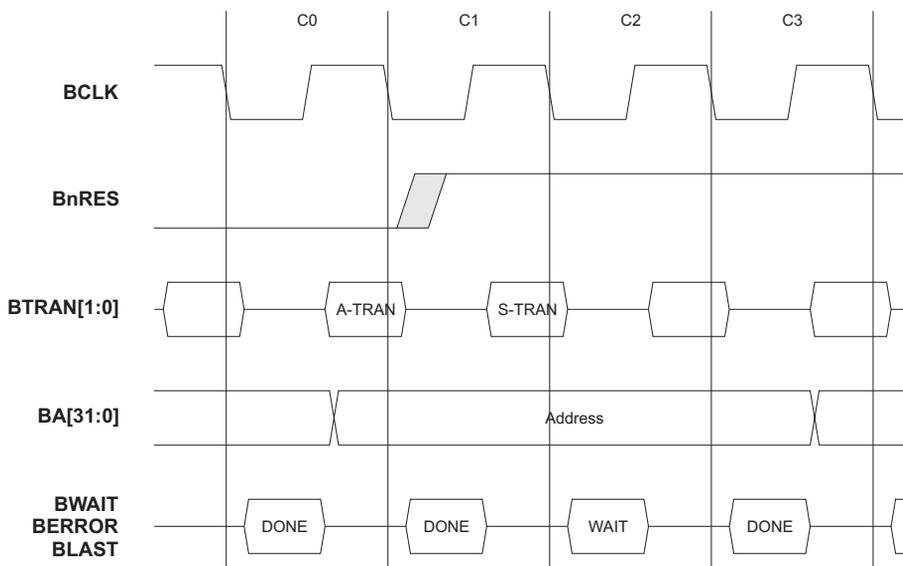


Figure 4-11 Exiting from reset

The following points should be noted:

- During cycle C1 **BnRES** is de-asserted during the clock LOW phase.

- During the clock **HIGH** phase of cycle C1 the default bus master may drive the **BTRAN** signal to indicate that it wishes to start a transfer.
- The transfer will start during cycle C2 and, in the example shown, the transfer is waited and continues into cycle C3.

## 4.8 Description of ASB signals

This section provides more detailed information about all the AMBA ASB signals, including their intended use and phase-accurate timing requirements.

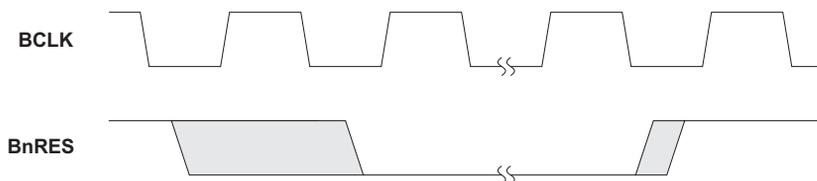
It is necessary to ensure that bus hold cells are provided to prevent floating levels being present on the bus. This is possible because many of the bus signals are shared, and have turnaround periods when there is no active driver.

### 4.8.1 Clock

**BCLK** is the primary clock, which is used to time all bus transfers. Both edges of the clock are used.

### 4.8.2 Reset

A single active LOW reset signal, **BnRES**, is supported which is used to reset the bus. The reset signal may be asserted LOW asynchronously during either clock phase, but is always de-asserted during the LOW phase of **BCLK**.



**Figure 4-12 Reset signal**

During reset the following actions occur on the bus:

- the arbiter grants the default bus master
- the default bus master must:
  - drive **BTRAN** to indicate ADDRESS-ONLY transfer
  - drive **BLOK** LOW to allow arbitration.
- all other bus masters tristate shared bus signals
- the decoder must:
  - de-assert all slave select signals, **DSELx**
  - provide the appropriate transfer response.
- all slaves tristate shared bus signals.

The **BnRES** signal may be used to reset the bus during time-out conditions.

In the majority of bus masters and slaves the **BnRES** signal will be used to reset both the bus interface and the main core of the component. However, it is acceptable for some system elements, such as a real-time clock, to use **BnRES** to only reset the bus interface. Such system elements would typically have a second reset input to allow the component core to be reset at initial power-up and for testing purposes.

### 4.8.3 Transfer type

Before a transfer starts the bus master indicates which type of the transfer it is, using **BTRAN[1:0]**. The following transfer types can be set:

- ADDRESS-ONLY
- NONSEQUENTIAL
- SEQUENTIAL.

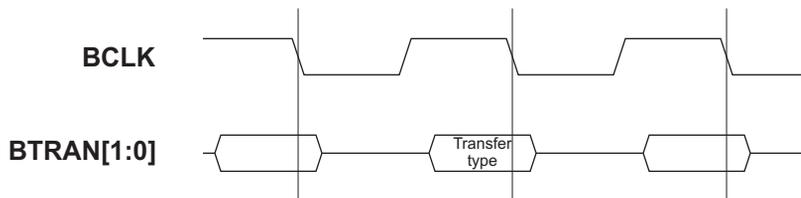
Table 4-1 shows the encoding of the **BTRAN[1:0]** signals:

**Table 4-1 BTRAN encoding**

<b>BTRAN</b>		<b>Transfer type</b>	<b>Description</b>
<b>[1]</b>	<b>[0]</b>		
0	0	ADDRESS-ONLY	Used when no data movement is required. The three main uses for ADDRESS-ONLY transfers are: <ul style="list-style-type: none"> <li>• for IDLE cycles</li> <li>• for bus master handover cycles</li> <li>• for speculative address decoding without committing to a data transfer.</li> </ul>
0	1	-	Reserved
1	0	NONSEQUENTIAL	Used for single transfers or for the first transfer of a burst. The address of the transfer is unrelated to the previous bus access.
1	1	SEQUENTIAL	Used for successive transfers in a burst. The address of a SEQUENTIAL transfer is always related to the previous transfer.

From the table it can be deduced that **BTRAN[1]** can be used to determine that a data transfer is required next cycle.

The **BTRAN** signals are driven by a bus master during the HIGH phase of **BCLK** when the **AGNTx** input is HIGH (see Figure 4-13).



**Figure 4-13 BTRAN timing**

In a multi-master system, the bus master that drives **BTRAN** may change during an extended transfer. Therefore, **BTRAN** must only be considered valid when the previous transfer has completed, as indicated by **BWAIT** LOW.

#### 4.8.4 Address and control information

The address and control signals are:

- address bus - **BA[31:0]**
- transfer direction - **BWRITE**
- transfer size - **BSIZE[1:0]**
- protection information - **BPROT[1:0]**.

#### 4.8.5 Address bus

The 32-bit address bus, **BA[31:0]**, provides the address of the transfer. All transfers are memory-mapped and therefore all memory and peripherals within the system must have an address range within which they are accessed. The decoder uses the address bus (usually the higher order bits) to determine which bus slave is to be accessed.

#### 4.8.6 Transfer direction

The **BWRITE** signal is used to indicate the direction of the transfer (see Table 4-2). When **BWRITE** is LOW the transfer is a read access and when **BWRITE** is HIGH the transfer is a write access.

**Table 4-2 BWRITE encoding**

<b>BWRITE</b>	<b>Transfer direction</b>
0	Read transfer
1	Write transfer

### 4.8.7 Transfer size

**BSIZE[1:0]** encodes the size of a transfer (see Table 4-3). Byte, halfword and word are all defined, with the final encoding being reserved for future use.

**Table 4-3 BSIZE encoding**

BSIZE		Transfer width
[1]	[0]	
0	0	Byte (8 bits)
0	1	Halfword (16 bits)
1	0	Word (32 bits)
1	1	Reserved

When performing transfers that are narrower than the data bus, such as a byte or halfword transfer, the bus master may replicate the data across the bus, making the bus master effectively bi-endian. When responding to read cycles, a typical slave will not replicate the data on the bus and therefore it is important that the master is expecting data on the same byte lane as that which the slave is driving.

### 4.8.8 Protection information

The bus master may use the **BPROT** signals to provide additional information about the transfer it is performing (see Table 4-4). This information is primarily intended for use by the decoder when it is acting as a bus protection unit and the majority of bus slaves will not use these signals.

**Table 4-4 BPROT encoding**

BPROT		Transfer privilege
[1]	[0]	
-	0	Opcode fetch
-	1	Data access
0	-	User access
1	-	Privileged access

### 4.8.9 Address and control signal timing

The address and control information is generated by the bus master from the rising edge of **BCLK**. However, the timing of the address and control information is considered separately for **NONSEQUENTIAL** and **SEQUENTIAL** transfer types. This is because a bus master will typically have significantly different timing parameters in each case.

It is a common characteristic that bus masters will have fast address and control output valid timings for **SEQUENTIAL** transfers, as shown in Figure 4-14. This is because a bus master is usually able to generate a **SEQUENTIAL** address well before the start of the transfer and therefore the output valid time from the bus master is mainly dependent on the time required to drive the new value onto the bus.

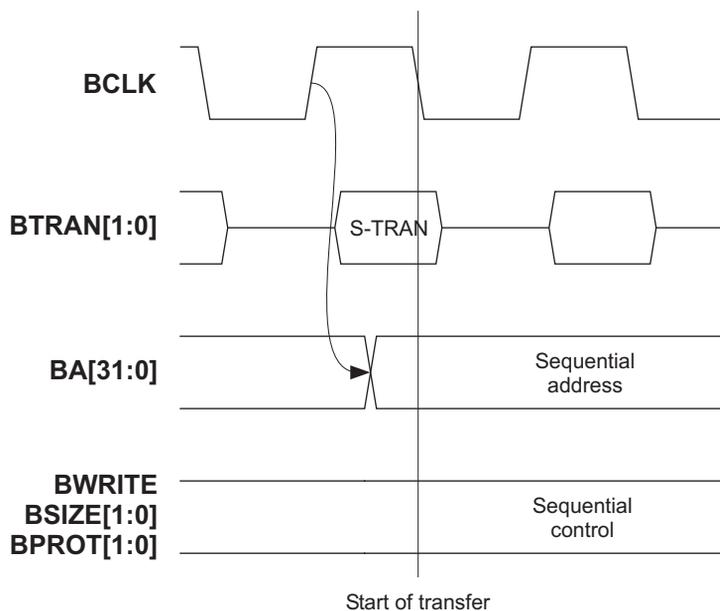
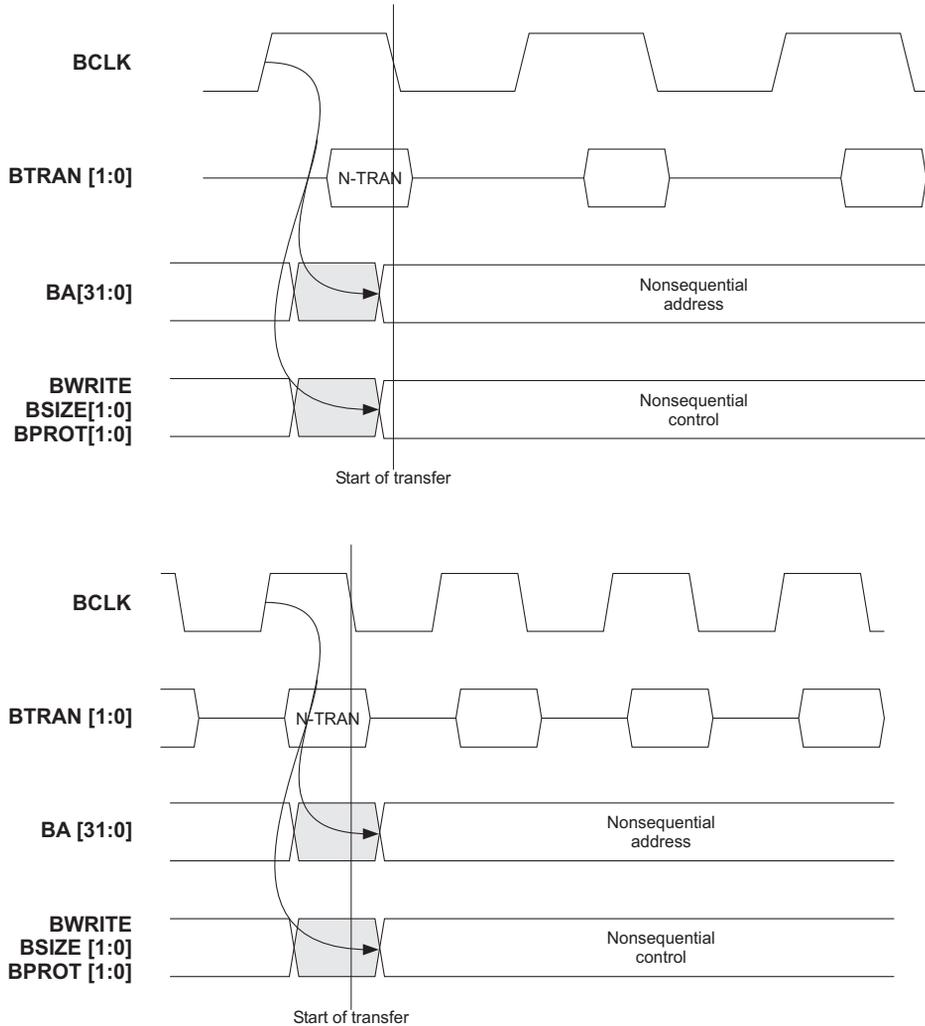


Figure 4-14 Sequential address and control timing

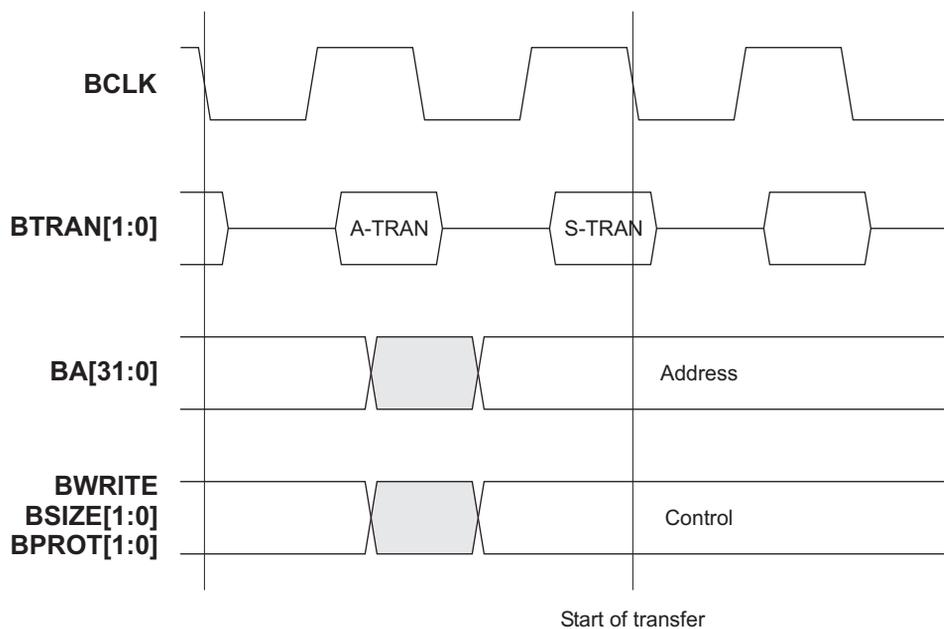
For NONSEQUENTIAL transfers the bus master will often have significantly slower output valid times for address and control signals, compared to those for SEQUENTIAL transfers and this is shown in Figure 4-15.



**Figure 4-15 Nonsequential address and control timing with low-frequency and high-frequency clocks**

In systems where the clock frequency is approaching the maximum possible, it is common for the address and control output valid time to be greater than a clock phase, thus resulting in the address not becoming valid until the **BCLK** LOW phase at the start of the transfer, as shown in Figure 4-15.

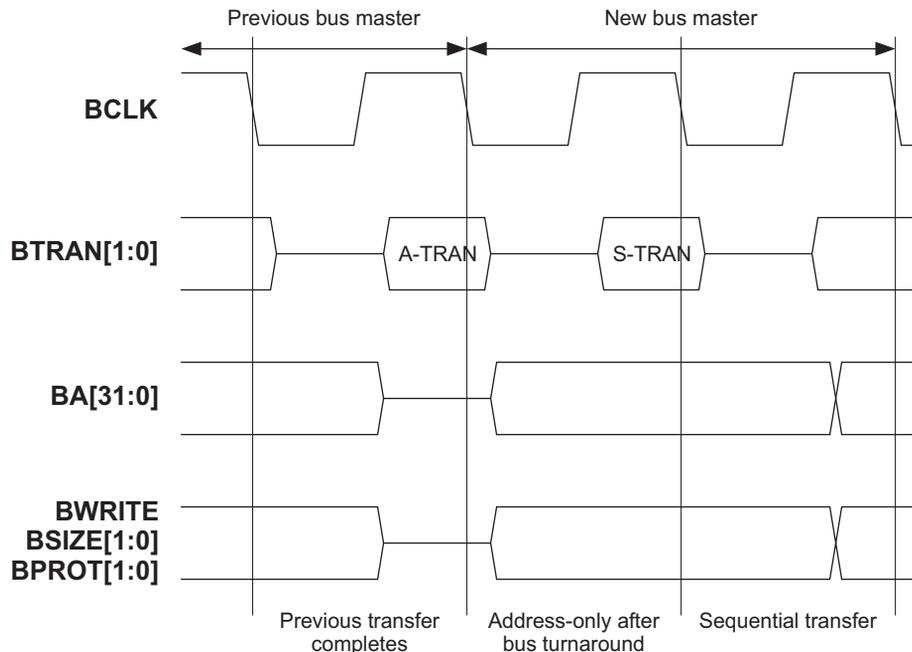
For ADDRESS-ONLY transfers the address and control information is not valid. In the special case of the ADDRESS-ONLY followed immediately by a SEQUENTIAL transfer, as shown in Figure 4-16, the bus master generates the address and control information during the ADDRESS-ONLY transfer, such that it is valid throughout the **BCLK** HIGH phase before the start of the SEQUENTIAL transfer.



**Figure 4-16 Address-only followed by sequential transfer address and control timing**

#### 4.8.10 Tristate enable of address and control signals

A bus master may only drive the address and control signals when the bus master is granted the bus. To allow for a period of bus turnaround, when a bus master is first granted the bus it will not drive in the **BCLK HIGH** phase before the first transfer. Instead, the bus master must always start a period of bus ownership with an ADDRESS-ONLY transfer and the address and control signals are not driven until the **BCLK LOW** phase of the ADDRESS-ONLY transfer (see Figure 4-17).



**Figure 4-17 Address and control signals during bus master handover**

For multi-master systems:

- When the master is first granted, the signals are not driven until the first LOW phase of **BCLK** when the master becomes active on the bus.
- When a bus master is granted the bus it drives the address and control signals during both phases of **BCLK**.
- The bus master stops driving the signals in the last **BCLK HIGH** phase, when the master loses mastership of the bus.

#### 4.8.11 Slave select signals

Each ASB slave in the system has a **DSEL** select input signal. This signal indicates that the slave is responsible for supplying a transfer response and that a data transfer is required. The signal name **DSEL<sub>x</sub>** is used to indicate the **DSEL** signal to slave **x**.

There is one **DSEL<sub>x</sub>** signal for each slave on the ASB and these signals are generated by the decoder. Only one **DSEL<sub>x</sub>** signal will be active during a transfer and there may be cycles when no **DSEL<sub>x</sub>** signal is active, such as during ADDRESS-ONLY transfers.

**DSEL<sub>x</sub>** changes during the **BCLK HIGH** phase before the start of a transfer and remains valid during the transfer. It will change for the next transfer in the **BCLK HIGH** phase following a transfer response with **BWAIT LOW**.

When designing a system there are two options for the implementation of an ASB decoder:

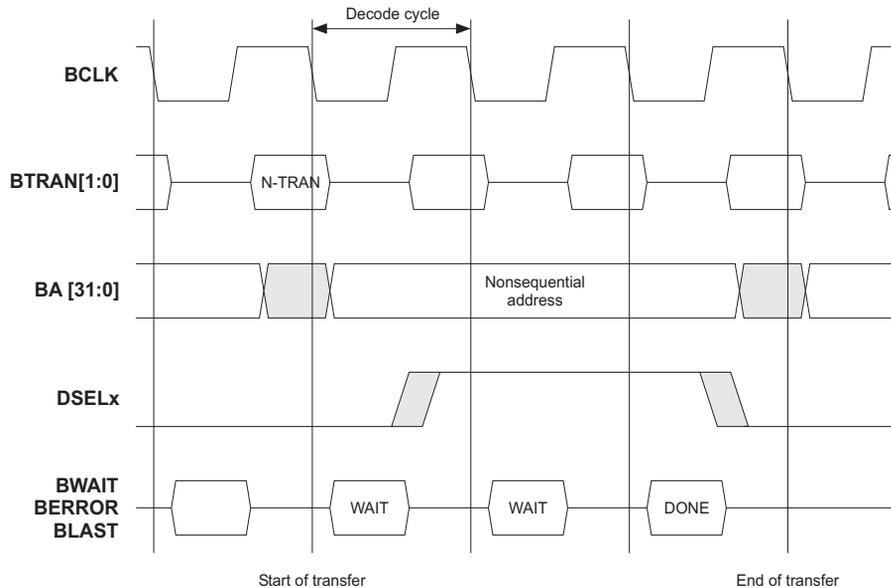
- *Decoder with decode cycles*
- *Decoder without decode cycles* on page 4-34.

This choice is fixed at the design stage and will be based on a timing analysis of the system. In general, a system that is operating up to the maximum speed of the processor will require DECODE cycles. It is only those systems which operate at a frequency significantly lower than the possible maximum that do not require DECODE cycles.

##### Decoder with decode cycles

In systems with a high clock frequency the critical path to decode the address and select a slave within a single clock phase tends to limit the maximum bus clock speed. In such systems the decoder can be used to automatically insert a wait state, or DECODE cycle, at the start of every NONSEQUENTIAL transfer. This implementation allows SEQUENTIAL transfers to continue to operate without the addition of a wait state, as it is known that the address decoding critical path can be avoided on SEQUENTIAL transfers, thus resulting in an overall improvement in bus bandwidth.

For NONSEQUENTIAL transfers **DSELx** is asserted in the **BCLK** HIGH phase during the DECODE cycle, as shown in Figure 4-18 below.

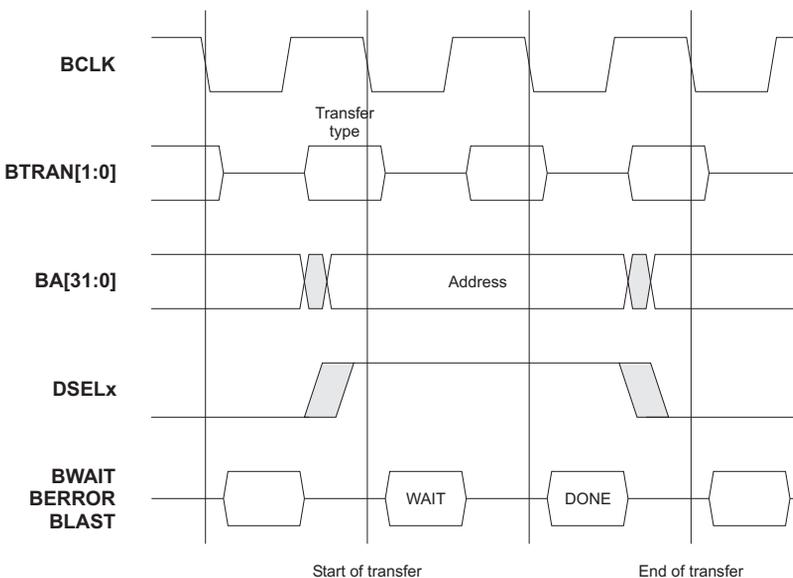


**Figure 4-18 Select signal timing with decode cycle**

- When DECODE cycles are implemented the timing of **DSELx** is dependent only on **BTRAN[1:0]** and there is no timing dependency on the address and control signals. This is because no **DSELx** signals are asserted for ADDRESS-ONLY transfers.
- For a NONSEQUENTIAL transfer the DECODE cycle is inserted to provide an entire phase for the address and control information to become valid.
- For SEQUENTIAL transfers the address and control information from the previous cycle is used.

### Decoder without decode cycles

In systems with a low clock frequency the address and control information will be valid in time to decode the address and select a slave within a single clock phase. In such systems a DECODE cycle is not required (see Figure 4-19).



**Figure 4-19 Select signal timing without decode cycle**

The select signal becomes valid during the **HIGH** phase of **BCLK** before the transfer commences and remains valid throughout the transfer, until the **HIGH** phase of the last cycle.

When the decoder does not insert **DECODE** cycles, the timing of **DSELx** becomes dependent on the timing of the address and control signals generated by the currently granted bus master.

#### 4.8.12 Transfer response

The transfer response signals are used by slave devices to indicate the status of a transfer.

A valid transfer response must be provided during the **LOW** phase of **BCLK**. Whenever a slave is selected (as indicated by **DSELx** being asserted) the slave must provide the response. When no slave is selected, for example during an **ADDRESS-ONLY** transfer, the response is provided by the decoder.

### Wait response

**BWAIT** is used to indicate when a transfer may complete. **BWAIT** is asserted HIGH when the slave requires extra bus cycles to complete the current transfer. **BWAIT LOW** indicates that the transfer may finish. Whether or not the transfer has completed successfully can only be determined by examining the other transfer response signals.

### Error response

An error condition is signalled by the **BERROR** signal. This may be used to indicate a failed transfer, a transfer to an address where there is no slave device or a protection error.

Many simple bus slaves will not implement error logic and will therefore have a fixed response of **BERROR LOW**.

**BERROR** is also used in conjunction with **BLAST** to indicate a RETRACT operation. When both these signals are HIGH this indicates that a bus RETRACT is required.

### Last response

**BLAST** is used to signal if the current transfer must be the last of a burst. This would typically be used to prevent a burst from continuing over a page boundary or other burst length limit.

**BLAST** is used by the decoder to make sure that the following transfer has the same characteristics as a NONSEQUENTIAL type transfer, rather than a burst transfer. Typically this involves ensuring there is adequate time to perform a new address decode.

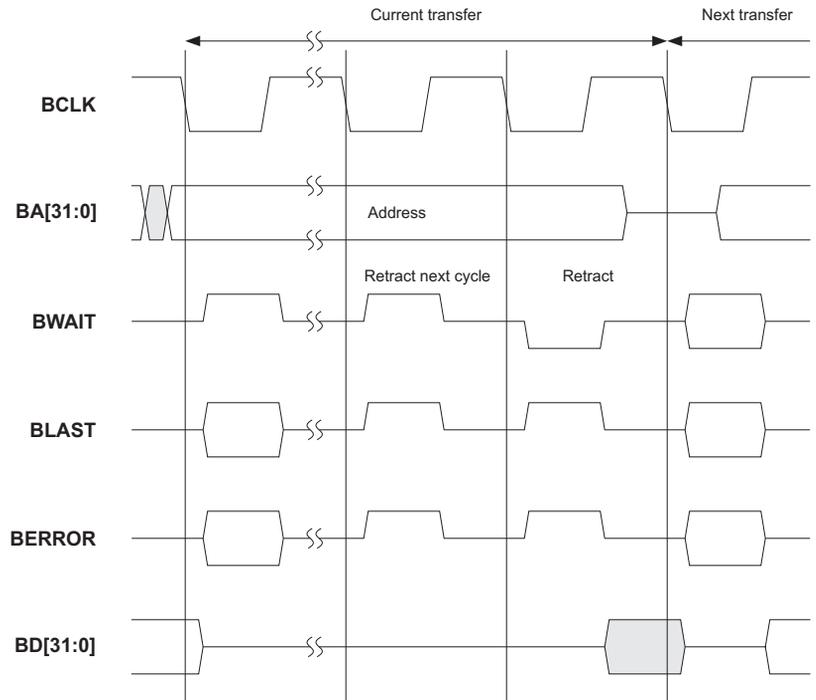
Many bus slave devices will be able to accept any number of burst accesses and these slaves will have a fixed response of **BLAST LOW**.

**BLAST** is also used in conjunction with **BERROR** to indicate a RETRACT operation. When both these signals are HIGH this indicates that a bus RETRACT is required.

### Bus retract

Slaves that cannot guarantee to complete transfers in a small number of wait states can potentially block the bus and stop higher priority transfers occurring. To prevent such slaves impacting the overall system latency a RETRACT mechanism is provided which allows a slave to indicate that a transfer is unable to complete at present, but the operation should be retried until it is able to complete successfully.

A RETRACT is performed in a two stage process, as shown in Figure 4-20.



**Figure 4-20 Retract operation**

- First the slave responds with **BWAIT**, **BLAST** and **BERROR** all HIGH, which indicates that a RETRACT is to occur and the transfer will finish in the next bus cycle.
- In the second cycle the transfer response is **BWAIT** LOW, **BLAST** and **BERROR** both HIGH. This indicates that the transfer has RETRACTed and that the bus is free to be used.

Basic slaves, which have a guaranteed completion time, do not need to support the bus RETRACT mechanism.

## Response combinations

Table 4-5 shows the combinations of the three slave transfer response signals.

**Table 4-5 Transfer response combinations**

BWAIT	BLAST	BERROR	Status	Description
0	0	0	DONE	Complete, transfer successful
0	0	1	ERROR	Complete, transfer error
0	1	0	LAST	Complete, cannot continue with burst
0	1	1	RETRACT	Complete, bus RETRACT
1	0	0	WAIT	Incomplete, insert wait cycle
1	0	1	-	Reserved
1	1	0	-	Reserved
1	1	1	RETNEXT	Bus RETRACT next cycle

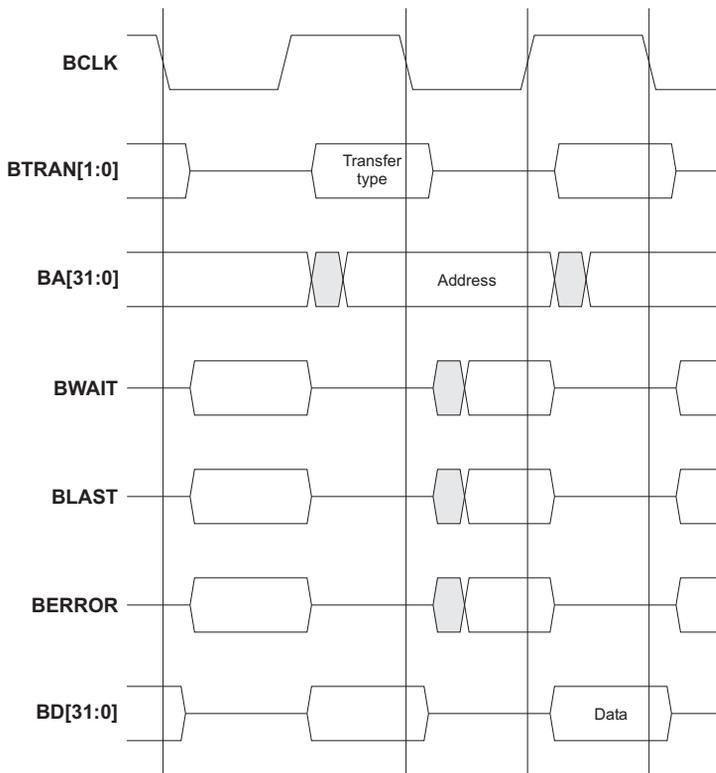
To ensure that the bus remains synchronized, a transfer response must be driven every cycle. During bus transfers, when a slave is selected and its appropriate **DSELx** signal is asserted, the slave is responsible for driving the transfer response signals.

The bus decoder is responsible for driving the transfer response signals during:

- ADDRESS-ONLY transfers
- DECODE cycles
- transfers to an address space where no slave is defined
- transfers to protected areas, when the access permissions are not met
- unaligned transfers which are not supported by the memory system.

### Transfer response timing

The transfer response signals must be set up valid before the rising edge of **BCLK** (see Figure 4-20).



**Figure 4-21 Transfer response signal timing**

The signals are not driven during the HIGH phase of **BCLK** to allow an entire phase of turnaround between signal drivers.

### 4.8.13 Data bus

The bidirectional data bus, **BD[31:0]**, is used to transfer data between bus masters and slaves. The size and direction of the transfer is given by the control signals, as described in *Address and control information* on page 4-27.

The data bus must not be driven during the first **BCLK LOW** phase of a NONSEQUENTIAL transfer. It may be driven, by the appropriate master or slave, at all other times except reset.

During a write transfer:

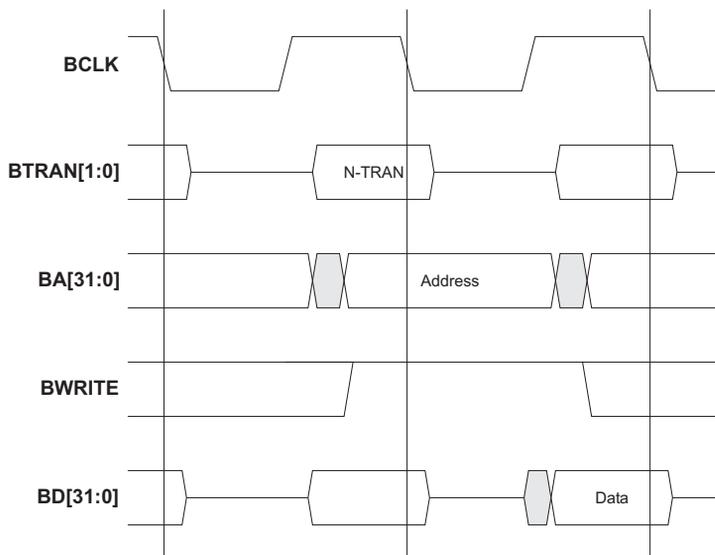
- the master drives the data bus during all phases of the transfer, except the first **BCLK LOW** phase of a NONSEQUENTIAL transfer
- the slave does not drive the bus.

During a read transfer:

- The master does not drive the data bus.
- The slave must drive the data bus during the last **BCLK HIGH** phase of the transfer. For the rest of the transfer, the slave may drive the data bus or leave it tristate, with the provision that it is not driven during the first **BCLK LOW** phase of a NONSEQUENTIAL transfer.

The following diagrams show some examples of how the data bus is driven.

Figure 4-22 shows an example of a NONSEQUENTIAL write transfer.



**Figure 4-22 Nonsequential write transfer**

The data bus is driven by the bus master, except for the **BCLK LOW** phase of the first cycle. Not driving the data bus at the start of NONSEQUENTIAL transfers provides a full phase of turnaround between different data bus drivers.

When a write transfer is extended using **BWAIT**, the data remains valid through the **BCLK** LOW phase of the extra cycles that are required to complete the transfer, as shown in Figure 4-23.

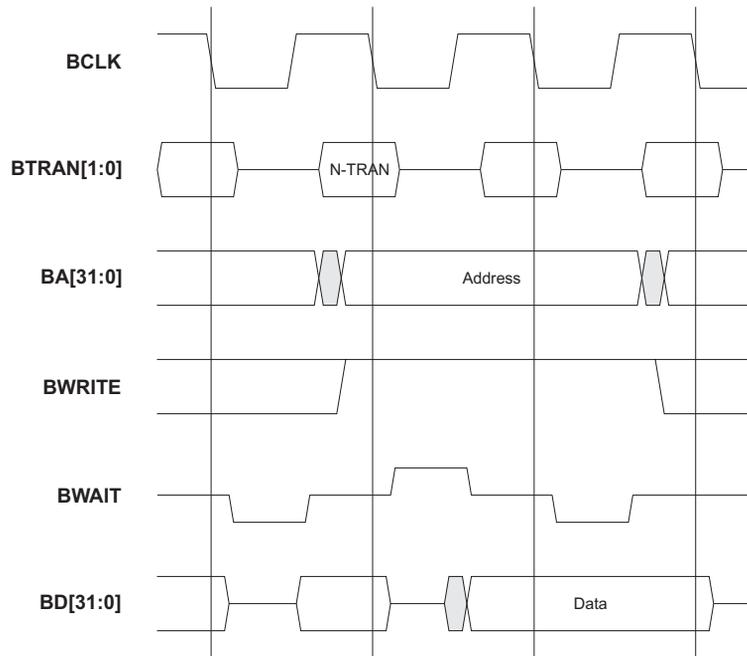
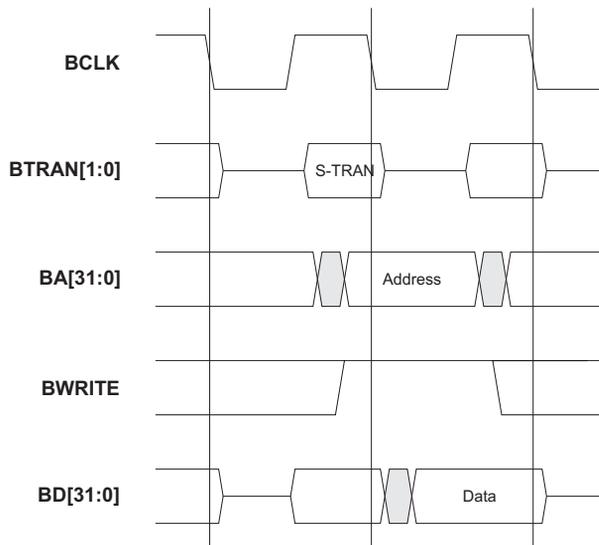


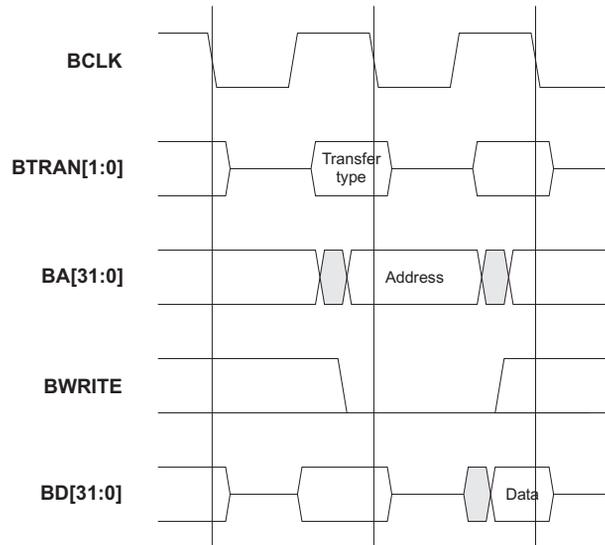
Figure 4-23 Extended write transfer

For SEQUENTIAL transfers the bus master may drive data during the LOW phase of **BCLK** at the start of the transfer, as shown in Figure 4-24. This is permitted as a phase of turnaround is not required for SEQUENTIAL transfers.



**Figure 4-24 Sequential write transfer**

During read cycles the slave drives the data bus and, as in the write cycle case, for NONSEQUENTIAL transfers the data bus is not driven in the **BCLK LOW** phase of the first cycle (see Figure 4-25). The bus slave may then drive the bus throughout the rest of the transfer.



**Figure 4-25 Read transfer**

There is no requirement for the slave to drive the data bus throughout the transfer. The only requirement is that the data is driven such that it is valid by the end of the last **BCLK HIGH** phase of the transfer.

#### 4.8.14 Arbitration signals

##### **AREQx - Bus request**

**AREQx** is the request signal from a master to the arbiter which indicates that the master requires the bus. Each master has an **AREQx** signal, which changes during the HIGH phase of **BCLK**.

##### **AGNTx - Bus grant**

The grant signal from the arbiter to a bus master indicates that the bus master is currently the highest priority master requesting the bus. There is an **AGNTx** signal for each bus master in the system.

It is important to note that **AGNTx** does not indicate which master is currently granted the bus. Instead, it shows which master is currently the highest priority and at the completion of a transfer, as indicated by **BWAIT LOW**, the master which has **AGNTx** asserted is granted the bus.

**AGNTx** is changed by the arbiter during the LOW phase of **BCLK** and remains valid through the HIGH phase.

When **AGNTx** is HIGH, the master must:

- drive the **BTRAN** signals during **BCLK HIGH**
- become granted when **BWAIT** is LOW.

### **BLOK - Bus lock**

**BLOK** is the shared bus lock signal. This signal indicates the following transfer is indivisible from the current transfers and no other bus master should be given access to the bus.

A master must always drive a valid level on the **BLOK** signal when granted the bus, even if the master is not performing any transfers. This is necessary to ensure the arbitration process can continue.

If **BLOK** is LOW the arbiter will grant the highest priority master requesting the bus.

If **BLOK** is HIGH the arbiter will keep the same master granted.

**BLOK** is sampled by the arbiter during the LOW phase of **BCLK** and it must be valid such that the arbiter can generate valid **AGNTx** outputs before the rising edge of **BCLK**. **BLOK** is ignored by the arbiter during the bus master handover cycle.

## 4.9 About the ASB AMBA components

This section describes each of the elements in an AMBA system and provides the generic timing parameters that are required to analyze an ASB-based AMBA design.

The following notation is used for the timing parameters:

- $T_{is}$  - input setup time
- $T_{ih}$  - input hold time
- $T_{ov}$  - output valid time
- $T_{oh}$  - output hold time.

Unless otherwise stated, the timing parameters apply to both the rising and falling edges of the signal. Tristate enable and disable times are not explicitly specified. All tristate disable times must be less than a phase of **BCLK** to prevent a bus clash occurring. In certain cases the tristate enable time may need to be factored in to the output valid time if the enabling of the tristate driver is the dominant factor.

## 4.10 ASB bus slave

An ASB bus slave responds to transfers initiated by bus masters within the system. The slave uses a **DSEL** select signal from the decoder to determine when it should respond to a bus transfer. All other signals required for the transfer, such as the address and control information, will be generated by the bus master.

The decoder greatly simplifies the slave interface and removes the need for the slave to understand the different types of transfer that may occur on the bus.

### 4.10.1 Interface diagram

Figure 4-26 shows an ASB bus slave interface.

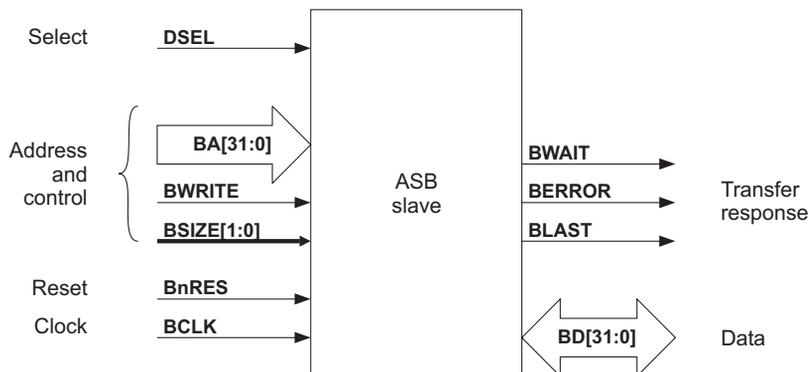


Figure 4-26 ASB bus slave interface

### 4.10.2 Bus slave interface description

The bus slave interface is described in terms of:

- *Transfer response*
- *Data* on page 4-48.

#### Transfer response

A slave must provide a transfer response in the LOW phase of **BCLK** when **DSEL** is asserted. Using the **BWAIT**, **BERROR** and **BLAST** signals one of the following responses must be generated.

**WAIT**                      The transfer must be extended before it can complete.

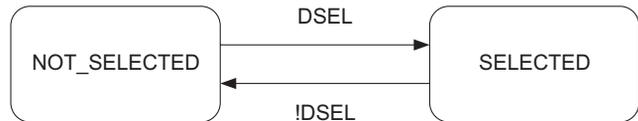
DONE	The transfer has completed successfully.
LAST	The transfer has completed successfully, but the slave is unable to accept further burst transfers or a memory boundary has been reached.
ERROR	The transfer has not completed successfully. The error condition will be signalled to the bus master so that it is aware the transfer has not completed correctly.
RETRACT	The transfer has not yet completed, so the bus master should retry the transfer. The RETRACT response is used by a slave to prevent the bus from being locked up by a transfer which may take many cycles to complete.

Many slaves will only use the WAIT and DONE responses and in this case, when a transfer response is supplied, both **BERROR** and **BLAST** will be LOW.

When the slave is not selected, as indicated by **DSEL** LOW, the transfer response signals must be tristate. The response signals must also be tristate during reset.

**Data**

The slave interface is implemented as a simple state machine, operating from the falling edge of the clock to determine when data transfer can occur. During reset the state machine enters the NOT\_SELECTED state (see Figure 4-27).



**Figure 4-27 ASB slave bus interface state machine**

For write transfers the slave samples the data on the falling edge of the clock when in the SELECTED state. If required, the slave may extend the transfer using the transfer response signals described above.

For read transfers the slave must drive the data bus during the last clock HIGH phase of the transfer.

If the transfer is extended, by the insertion of wait cycles, then the slave may either drive the data bus during the additional cycles of the transfer, or alternatively it may leave the data bus tristate until the last phase of the transfer.

To avoid the slave having to determine whether the transfer is SEQUENTIAL or NONSEQUENTIAL it is usually simpler to design a slave which does not drive the data bus during the first phase of any transfer.

During reset or when the slave is NOT\_SELECTED the data bus must be tristate.

### 4.10.3 Timing diagrams

The timing parameters related to an access to an ASB bus slave are shown in Figure 4-28.

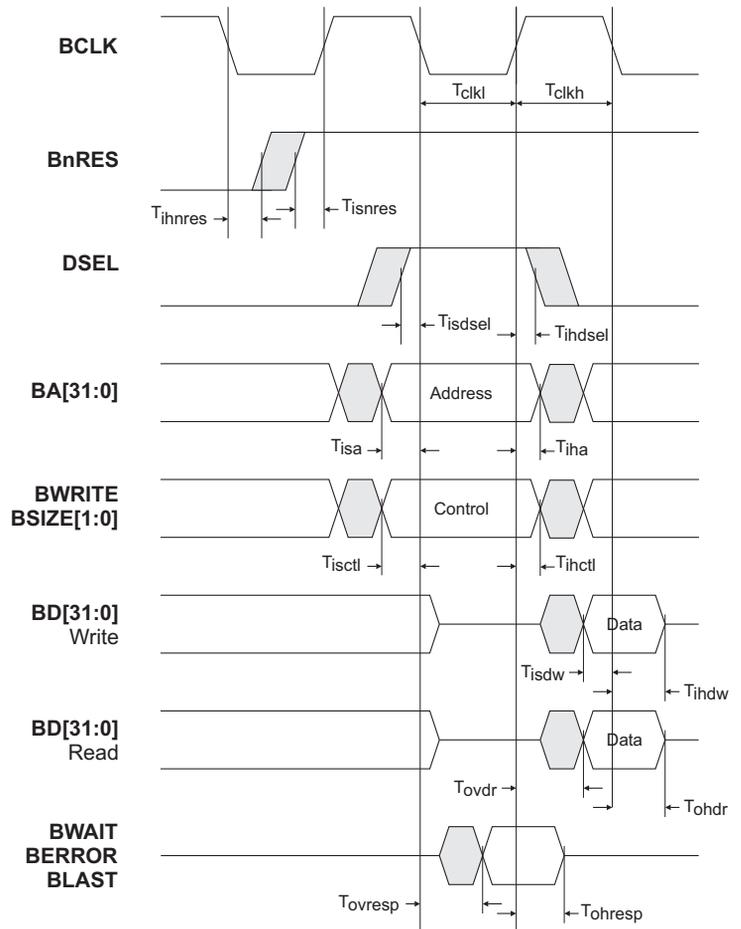


Figure 4-28 ASB slave transfer

#### 4.10.4 Timing parameters

The timing parameters related to an ASB bus slave are given for input signals in Table 4-6 and for output signals in Table 4-7. Bidirectional signals can be found in both tables.

**Table 4-6 ASB slave input parameters**

<b>Parameter</b>	<b>Description</b>
$T_{\text{clk}l}$	<b>BCLK</b> LOW time
$T_{\text{clk}h}$	<b>BCLK</b> HIGH time
$T_{\text{isnres}}$	<b>BnRES</b> de-asserted setup to rising <b>BCLK</b>
$T_{\text{ihnres}}$	<b>BnRES</b> de-asserted hold after falling <b>BCLK</b>
$T_{\text{isd}sel}$	<b>DSEL</b> setup to falling <b>BCLK</b>
$T_{\text{ihd}sel}$	<b>DSEL</b> hold after rising <b>BCLK</b>
$T_{\text{isa}}$	<b>BA[31:0]</b> setup to falling <b>BCLK</b>
$T_{\text{iha}}$	<b>BA[31:0]</b> hold after rising <b>BCLK</b>
$T_{\text{isct}l}$	<b>BWRITE</b> and <b>BSIZE[1:0]</b> setup to falling <b>BCLK</b>
$T_{\text{ihct}l}$	<b>BWRITE</b> and <b>BSIZE[1:0]</b> hold after rising <b>BCLK</b>
$T_{\text{isd}w}$	For write transfers, <b>BD[31:0]</b> setup to falling <b>BCLK</b>
$T_{\text{ihd}w}$	For write transfers, <b>BD[31:0]</b> hold after falling <b>BCLK</b>

**Table 4-7 ASB slave output parameters**

<b>Parameter</b>	<b>Description</b>
$T_{\text{ovresp}}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> valid after falling <b>BCLK</b>
$T_{\text{ohresp}}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> hold after rising <b>BCLK</b>
$T_{\text{ovdr}}$	For read transfers, <b>BD[31:0]</b> valid after rising <b>BCLK</b>
$T_{\text{ohdr}}$	For read transfers, <b>BD[31:0]</b> hold after falling <b>BCLK</b>

———— **Note** —————

If the bus slave is designed such that the decoder, address and control signals are all sampled on the falling edge of **BCLK** then an entire phase of input hold time is guaranteed by the bus protocol.

You can ensure that an entire phase of hold time is provided on the data bus by inserting an extra wait state into the transfer.

## 4.11 ASB bus master

An ASB bus master has the most complex bus interface in an AMBA system. Typically an AMBA system designer would use predesigned bus masters and therefore would not need to be concerned with the detail of the bus master interface.

A bus master interface may also include a slave interface, either for test or for programming the operation of the bus master. In such cases a number of the interface signals will become shared between the master interface and slave interface.

### 4.11.1 Interface diagram

The interface diagram of an ASB bus master shows the main signal groups.

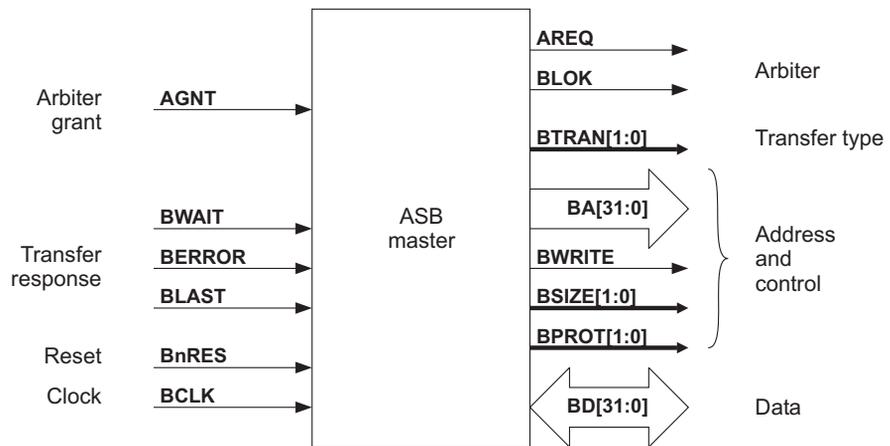


Figure 4-29 ASB bus master interface diagram

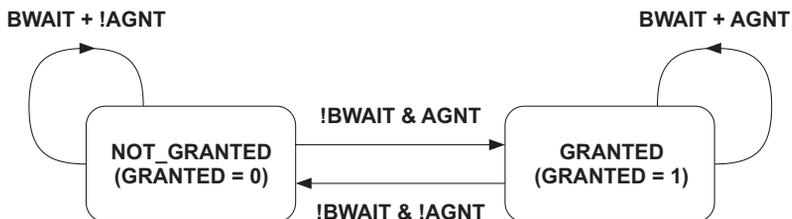
### 4.11.2 Bus master interface description

The bus master interface consists of two state machines:

- the first state machine determines if the master is currently granted the bus
- the second, more complex, state machine is used to control the bus interface of the master.

## GRANTED state machine

The GRANTED state machine is used to determine whether or not the bus master has been granted the bus. It is synchronized to the rising edge of **BCLK** and has only two states, GRANTED and NOT\_GRANTED. The state diagram is shown in Figure 4-30.



**Figure 4-30** Bus master granted state machine

The output from the state machine is the GRANTED signal, which is used in the main bus master state machine.

### Note

The **AGNT** signal may be asserted for a number of clock cycles, but it is only when **AGNT** is asserted and **BWAIT** is LOW that the bus master actually becomes GRANTED.

An important design consideration is that the state machine may be asynchronously reset into either state depending on the value of the **AGNT** signal. During reset one bus master in the system is set as the default bus master, as indicated by **AGNT** being asserted during reset, and will be reset into the GRANTED state. All other bus masters will be reset into the NOT\_GRANTED state.

### 4.11.3 Bus interface state machine

The main bus interface state machine is falling edge triggered and contains six states. The entire state diagram, shown in Figure 4-32, is quite complex but can be considered in four quadrants as shown in Figure 4-31:

NO TRANSFER REQUEST NOT GRANTED	TRANSFER REQUEST NOT GRANTED
NO TRANSFER REQUEST GRANTED	TRANSFER REQUEST GRANTED

**Figure 4-31 Bus interface state machine quadrants**

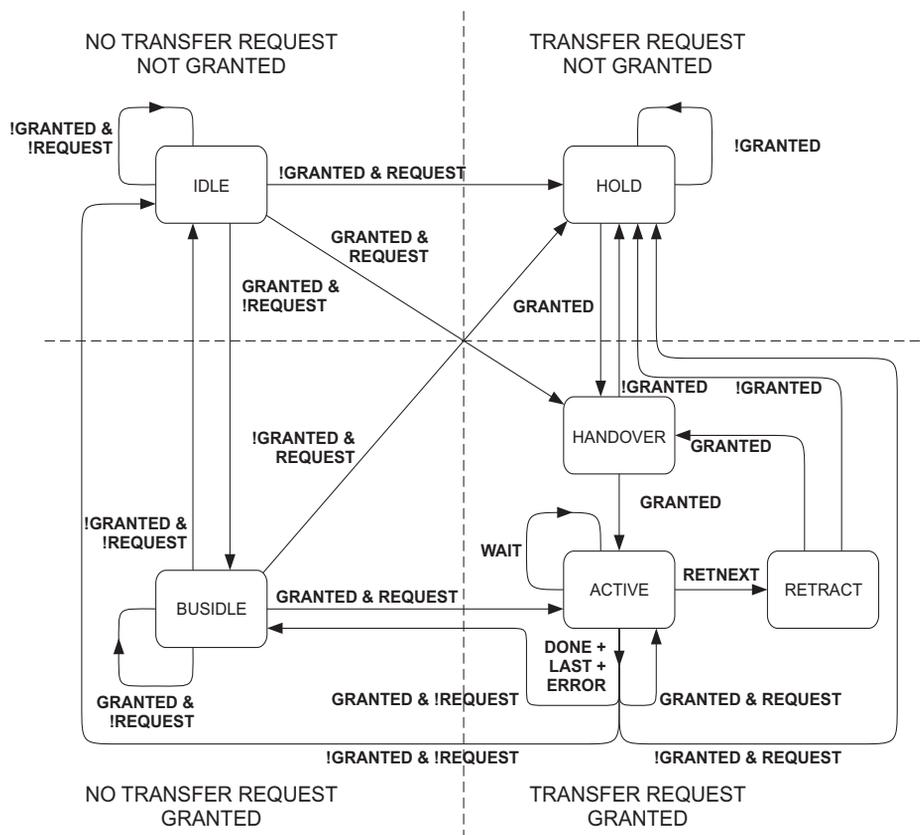
The TRANSFER REQUEST GRANTED quadrant contains three states, which handle bus turn around and the RETRACT operation.

The two internal bus master signals, GRANTED and REQUEST, control the majority of the transitions around the state diagram (see Figure 4-32):

- GRANTED is generated from the simpler state machine described above
- REQUEST is generated directly by the bus master.

REQUEST is asserted HIGH when the bus master requires a transfer on the bus and is LOW when the bus master does not need access to the bus.

The only time when a transition around the state diagram is not controlled by GRANTED and REQUEST is when the bus master is in the ACTIVE state. In this state the transition to the next state is determined by the transfer response that is received. WAIT, DONE, LAST, ERROR and RETNEXT shown in the diagram correspond to the encodings of the transfer response signals.



**Figure 4-32** Bus master main state machine

Note that the state diagram assumes that once the bus master has made a request for a transfer, as indicated by **REQUEST**, then **REQUEST** will remain asserted until the bus master has performed a transfer.

As the main bus master state machine is operating from the falling edge of the clock it is necessary to use latched versions of the transfer response signals **BWAIT**, **BERROR** and **BLAST** to control the exit from the **ACTIVE** state.

The reset conditions are not shown on the state diagram and, in a similar manner to the granted state machine, the main bus master state machine has a complex reset term. If **AGNT** is asserted during reset, when **BnRES** is **LOW**, the bus master is the default bus master and enters the **BUSIDLE** state. However, if **AGNT** is not asserted during reset then the bus master enters the **IDLE** state.

Table 4-8 indicates the actions that must occur in each state.

**Table 4-8 Actions that must occur in each state**

<b>Name</b>	<b>Description</b>	<b>Actions</b>
IDLE	The master does not require the bus and is not granted.	Internal <b>BTRAN</b> is ADDRESS-ONLY . Master clock is enabled. Master address bus is tristate. Master data bus is tristate.
BUSIDLE	The master does not require the bus, but has been granted anyway.	Internal <b>BTRAN</b> as indicated by master. Master clock is enabled. Master address bus enable is generated from GRANTED signal. Master data bus is tristate.
HOLD	The master requires the bus, but has not been granted.	Internal <b>BTRAN</b> is ADDRESS-ONLY . Master clock is disabled. Master address bus is tristate. Master data bus is tristate.
HANDOVER	This state provides bus turnaround when changing between different bus masters.	Internal <b>BTRAN</b> is SEQUENTIAL . Master clock is disabled. Master address bus enable is generated from GRANTED signal. Master data bus is tristate.
ACTIVE	Active state when data transfers occur. Exiting this state is dependent on the transfer response.	Internal <b>BTRAN</b> as indicated by master. Master clock enable is derived from <b>BWAIT</b> Master address bus enable is generated from GRANTED signal. Master data bus enable is enabled if a write transaction.
RETRACT	Retract state, where the rest of the elements in the system see the transfer finish, but the bus master is not advanced.	Internal <b>BTRAN</b> is ADDRESS-ONLY . Master clock is disabled. Master address bus enable is generated from GRANTED signal. Master data bus enable is enabled if a write transaction.

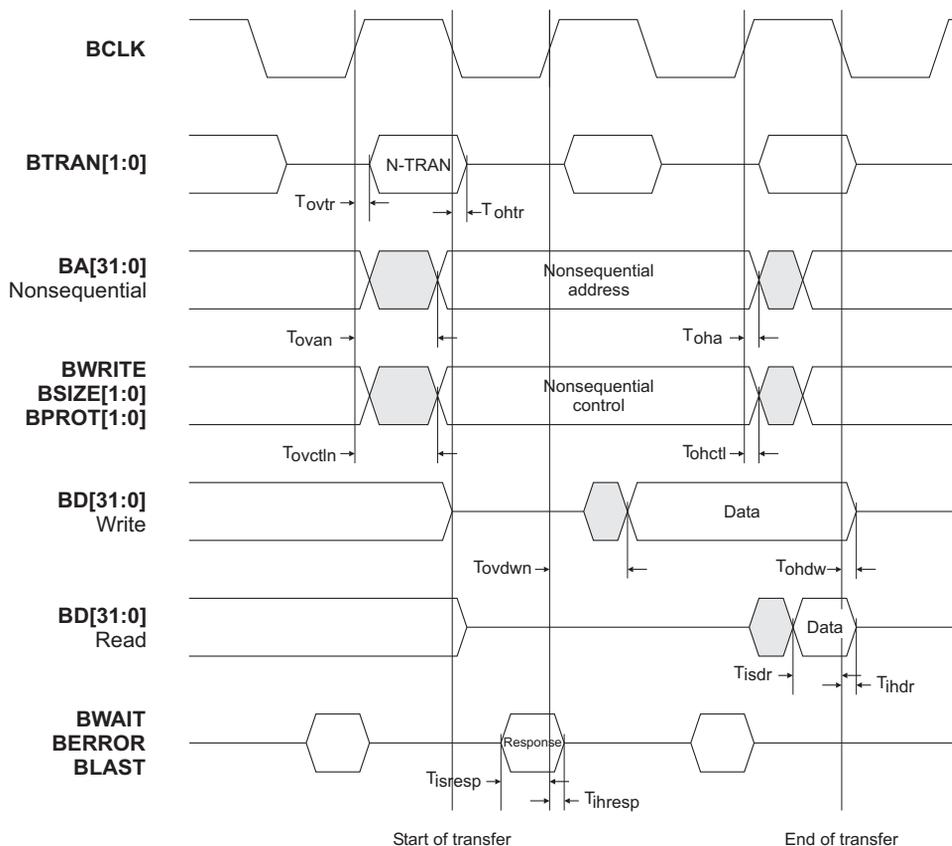
**BTRAN[1:0]** tristate drivers are enabled when **AGNT** and **BCLK** are both HIGH.

Master address bus enable is used to control the tristate enable of **BA[31:0]**, **BWRITE**, **BSIZE[1:0]**, **BPROT[1:0]** and **BLOK**. Master data bus enable is used to control the tristate enable of **BD[31:0]**.

#### 4.11.4 Bus master timing diagrams

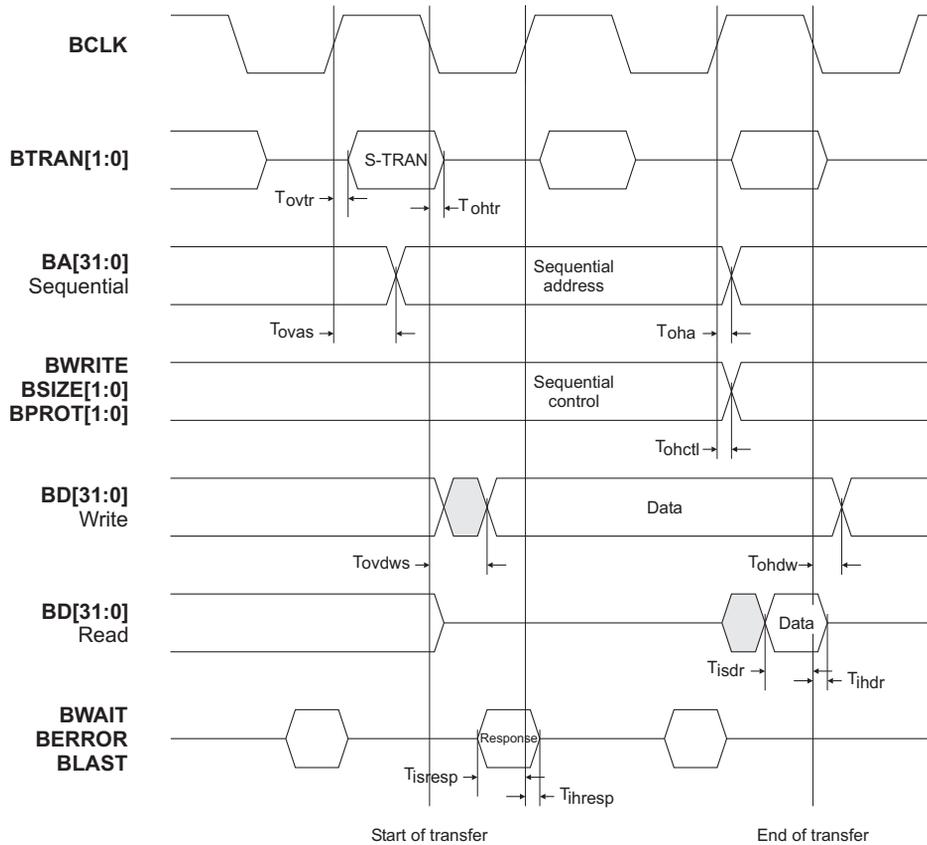
The following diagrams show the timing parameters related to an ASB bus master operating in an AMBA system:

- Figure 4-33 shows an *ASB bus master nonsequential transfer*
- Figure 4-34 shows an *ASB bus master sequential transfer* on page 4-58
- Figure 4-35 shows an *ASB master address-only transfer* on page 4-59
- Figure 4-36 shows *ASB bus master arbitration and reset signals* on page 4-60.



**Figure 4-33 ASB bus master nonsequential transfer**

For the NONSEQUENTIAL transfer, shown in Figure 4-33, the address and control signals become valid in the **BCLK** HIGH phase before the start of the transfer. An important feature of the AMBA protocol is to allow for poor output valid times on NONSEQUENTIAL transfers, which is provided through the automatic insertion of a wait state at the start of every NONSEQUENTIAL transfer by the decoder.

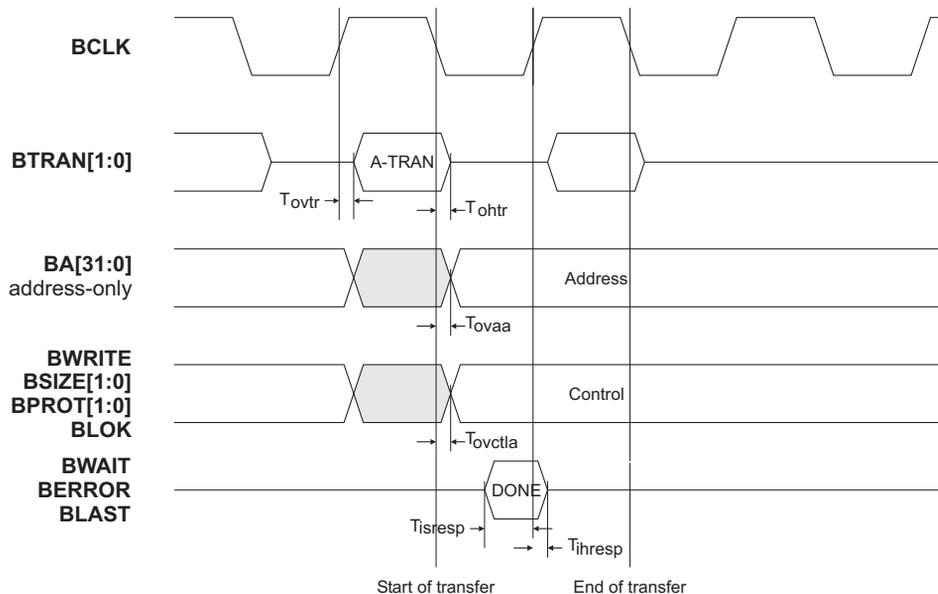


**Figure 4-34 ASB bus master sequential transfer**

A SEQUENTIAL transfer has different timing parameters for the address and control signal valid times (see Figure 4-34). In a typical bus master, the output valid times for SEQUENTIAL transfers will be far better than for NONSEQUENTIAL transfers. The output hold times for address, control and data are identical and independent of the transfer type.

The other difference between the SEQUENTIAL and NONSEQUENTIAL transfers is that during a SEQUENTIAL transfer the data may be driven during the first phase of the transfer and hence the data valid parameter is specified from the falling edge of BCLK.

For an ADDRESS-ONLY transfer the address and control signals may be driven in the clock HIGH phase before the start of the transfer, or in the case of bus master handover may only be driven during the clock LOW phase of the transfer itself (see Figure 4-35). The address and control valid timing parameters are only relevant when the ADDRESS-ONLY transfer is followed immediately by a SEQUENTIAL transfer and in this case the address and control signals must be driven such that they are valid during the LOW phase of the ADDRESS-ONLY transfer, which in turn means they are valid throughout the clock HIGH phase that precedes the SEQUENTIAL transfer.



**Figure 4-35 ASB master address-only transfer**

Figure 4-36 shows ASB bus master arbitration and reset signals.

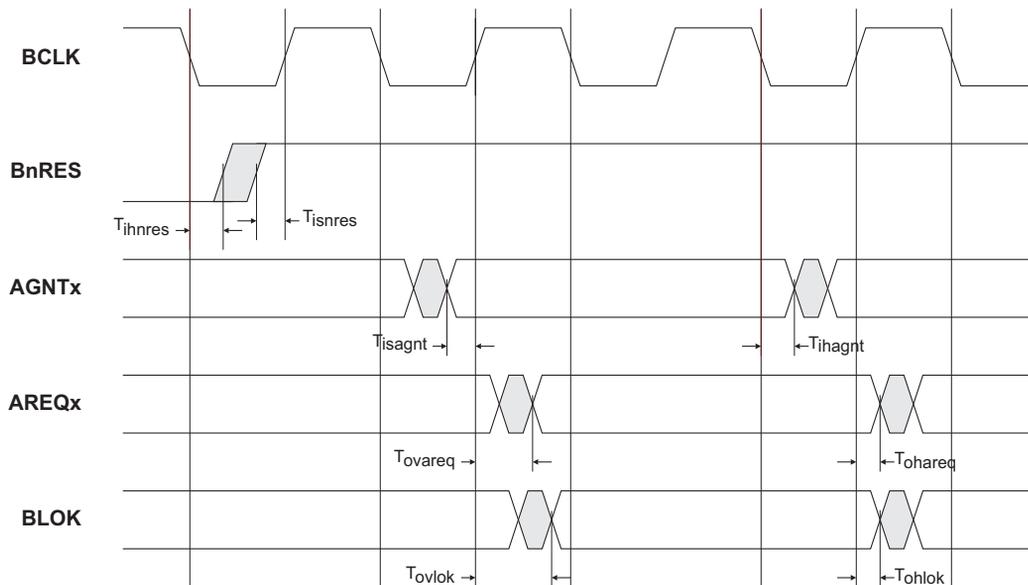


Figure 4-36 ASB bus master arbitration and reset signals

The **BnRES** signal may be asserted asynchronously, so there is no setup and hold parameter relating to the assertion of the signal. The **AREQ** signal, which is an output from the bus master, changes during the HIGH clock phase and the **AGNT** signal, which is returned from the arbiter changes during the LOW clock phase.

#### 4.11.5 Timing parameters

The timing parameters related to an ASB bus master operating in an AMBA system are also shown in textual form in the following two tables. Table 4-9 details the input signals. Table 4-10 details output signals. Bidirectional signals can be found in both tables.

Table 4-9 Bus master input timing parameters

Parameter	Description
$T_{clkL}$	<b>BCLK</b> LOW time
$T_{clkH}$	<b>BCLK</b> HIGH time
$T_{isnres}$	<b>BnRES</b> de-asserted setup to rising <b>BCLK</b>

Table 4-9 Bus master input timing parameters (continued)

Parameter	Description
$T_{ihres}$	<b>BnRES</b> de-asserted hold after falling <b>BCLK</b>
$T_{isresp}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> setup to rising <b>BCLK</b>
$T_{ihresp}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> hold after rising <b>BCLK</b>
$T_{isdr}$	For read transfers, <b>BD[31:0]</b> setup to falling <b>BCLK</b>
$T_{ihdr}$	For read transfers, <b>BD[31:0]</b> hold after falling <b>BCLK</b>
$T_{isagnt}$	<b>AGNT</b> setup to rising <b>BCLK</b>
$T_{ihagnt}$	<b>AGNT</b> hold after falling <b>BCLK</b>

Table 4-10 Bus master output timing parameters

Parameter	Description
$T_{ovtr}$	<b>BTRAN</b> valid after rising <b>BCLK</b>
$T_{ohtr}$	<b>BTRAN</b> hold after falling <b>BCLK</b>
$T_{ovan}$	For NONSEQUENTIAL transfers, <b>BA[31:0]</b> valid after rising <b>BCLK</b>
$T_{ovas}$	For SEQUENTIAL transfers, <b>BA[31:0]</b> valid after rising <b>BCLK</b>
$T_{ovaa}$	For ADDRESS-ONLY transfers, <b>BA[31:0]</b> valid after falling <b>BCLK</b>
$T_{oha}$	<b>BA[31:0]</b> hold after rising <b>BCLK</b>
$T_{ovctl}$	For NONSEQUENTIAL transfers, <b>BWRITE</b> , <b>BSIZE[1:0]</b> and <b>BPROT[1:0]</b> valid after rising <b>BCLK</b>
$T_{ovctla}$	For ADDRESS-ONLY transfers, <b>BWRITE</b> , <b>BSIZE[1:0]</b> and <b>BPROT[1:0]</b> valid after falling <b>BCLK</b>
$T_{ohctl}$	<b>BWRITE</b> , <b>BSIZE[1:0]</b> and <b>BPROT[1:0]</b> hold after rising <b>BCLK</b>
$T_{ovdwn}$	For NONSEQUENTIAL write transfers, <b>BD[31:0]</b> valid after rising <b>BCLK</b>
$T_{ovdws}$	For SEQUENTIAL write transfers, <b>BD[31:0]</b> valid after falling <b>BCLK</b>
$T_{ohdw}$	For write transfers, <b>BD[31:0]</b> hold after falling <b>BCLK</b>

Table 4-10 Bus master output timing parameters (continued)

Parameter	Description
$T_{\text{ovlok}}$	<b>BLOK</b> valid after rising <b>BCLK</b>
$T_{\text{ohlok}}$	<b>BLOK</b> hold after rising <b>BCLK</b>
$T_{\text{ovareq}}$	<b>AREQ</b> valid after rising <b>BCLK</b>
$T_{\text{ohareq}}$	<b>AREQ</b> hold after rising <b>BCLK</b>

## 4.12 ASB decoder

The decoder in an AMBA system is used to perform a centralized address decoding function, which gives two main advantages:

- It improves the portability of peripherals, by making them independent of the system memory map.
- It simplifies the design of bus slaves, by centralizing the address decoding and bus control functions.

The three main tasks of the decoder are:

- address decoder
- default transfer response
- protection unit.

An ASB decoder generates a select signal for each slave on the ASB bus and, under certain circumstances, will not select any slaves and provide the transaction response itself.

The decoder greatly simplifies the slave interface and removes the need for the slave to understand the different types of transfer that may occur on the bus.

An important feature of the decoder is that it is able to improve the performance of a system by providing DECODE cycles for address decoding. As the decoder is able to recognize if the transfer is SEQUENTIAL or NONSEQUENTIAL it is a simple task for the decoder to only add a DECODE cycle when required.

The decoder actually helps to significantly improve the system performance. In a non-AMBA system the critical path of, for example, a read transfer would be as follows:

1. Address out from master.
2. Address decode to select slave.
3. Data out and response from slave back to bus master.

However, in an AMBA system it is possible to remove the middle stage whenever the bus master is performing a SEQUENTIAL transfer, because it is known that the slave that is selected will be the same as the previous transfer. The decoder can use this fact to improve the system performance by only inserting a wait state for address decoding when needed, which is for NONSEQUENTIAL transfers. This is known as inserting a DECODE cycle.

In designs where the clock frequency is low enough that an additional wait state is not required for address decoding, then the role of the decoder is simplified.

The decoder is also used to provide a number of bus maintenance functions. Firstly, the decoder can act as a simple protection unit, which can issue an **ERROR** response to a bus master which attempts to access an illegal or protected area of the memory map. The decoder also provides a transfer response during **ADDRESS-ONLY** transfers, when no slave is selected.

#### 4.12.1 Interface diagram

Figure 4-37 shows an ASB decoder.

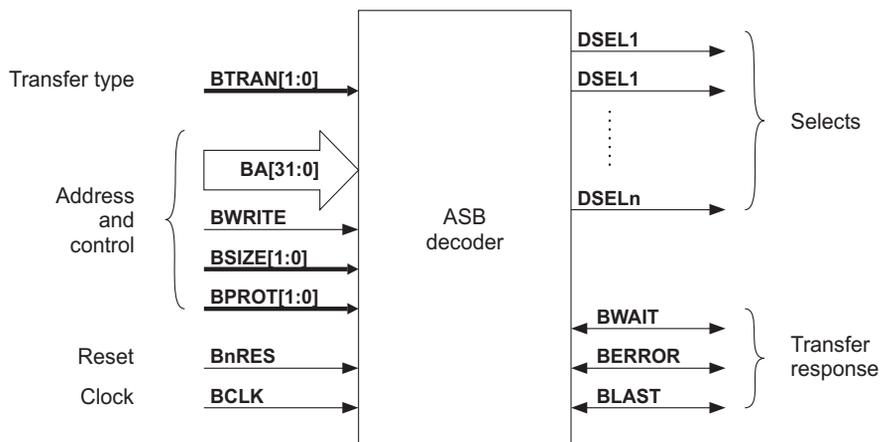


Figure 4-37 ASB decoder interface diagram

#### 4.12.2 Decoder description

There are two possible implementations of the decoder, depending on the performance requirements of the system design:

- The normal implementation of a decoder will include the insertion of **DECODE** cycles on **NONSEQUENTIAL** transfers and to break up burst transfers over memory boundaries.
- In some system designs, typically with a low clock frequency, the **DECODE** cycle will not be required and hence a simpler decoder may be implemented.

### With decode cycles

The decoder is implemented as a state machine which operates from the falling edge of the clock and has four states (see Figure 4-38). During reset the state machine should enter the ADDRONLY state.

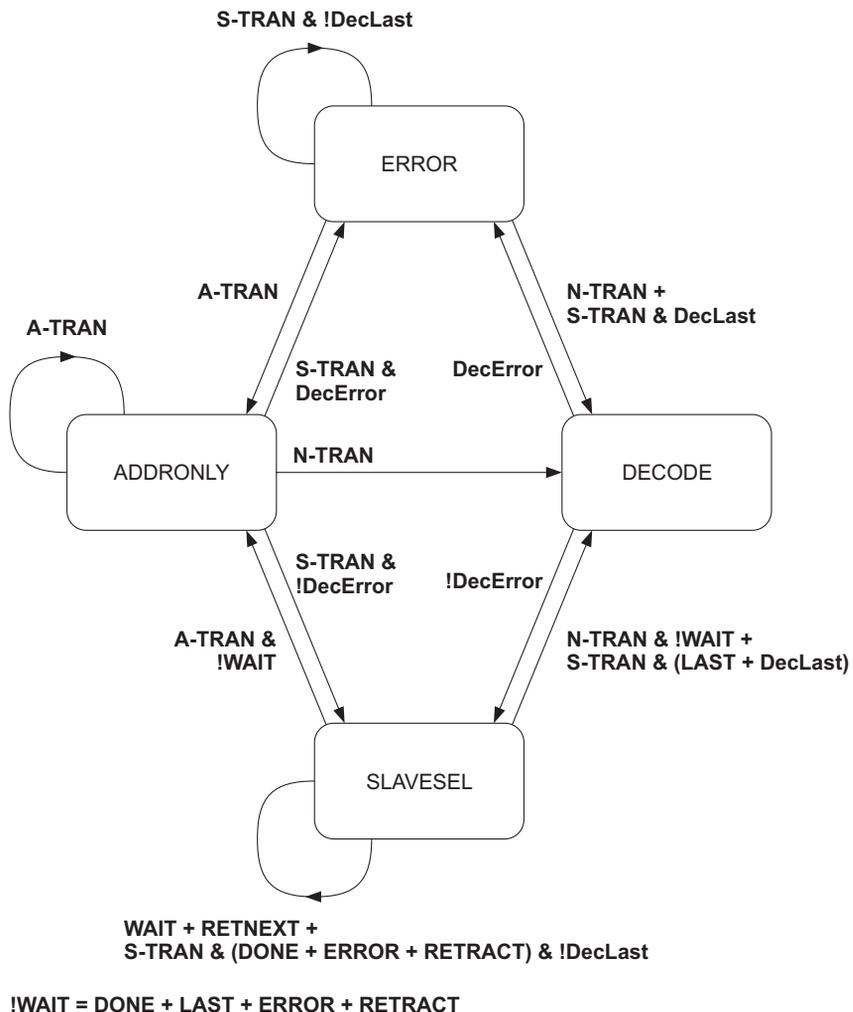


Figure 4-38 Decoder state machine with decode

Transitions around the state machine are controlled by the transfer type for the next transfer, the transfer response from the current transfer and two internal decoder signals, **DecLast** and **DecError**. The **WAIT**, **DONE**, **LAST**, **ERROR**, **RETNEXT** and **RETRACT** shown on the state diagram correspond to the encodings of the transfer response signals.

**DecLast** is generated by the decoder when it detects that a **SEQUENTIAL** transfer is about to cross a memory boundary and is used in combination with the external **BLAST** signal to force the address to be decoded, even on **SEQUENTIAL** transfers.

**DecError** is another decoder internal signal and is generated when the decoder detects that:

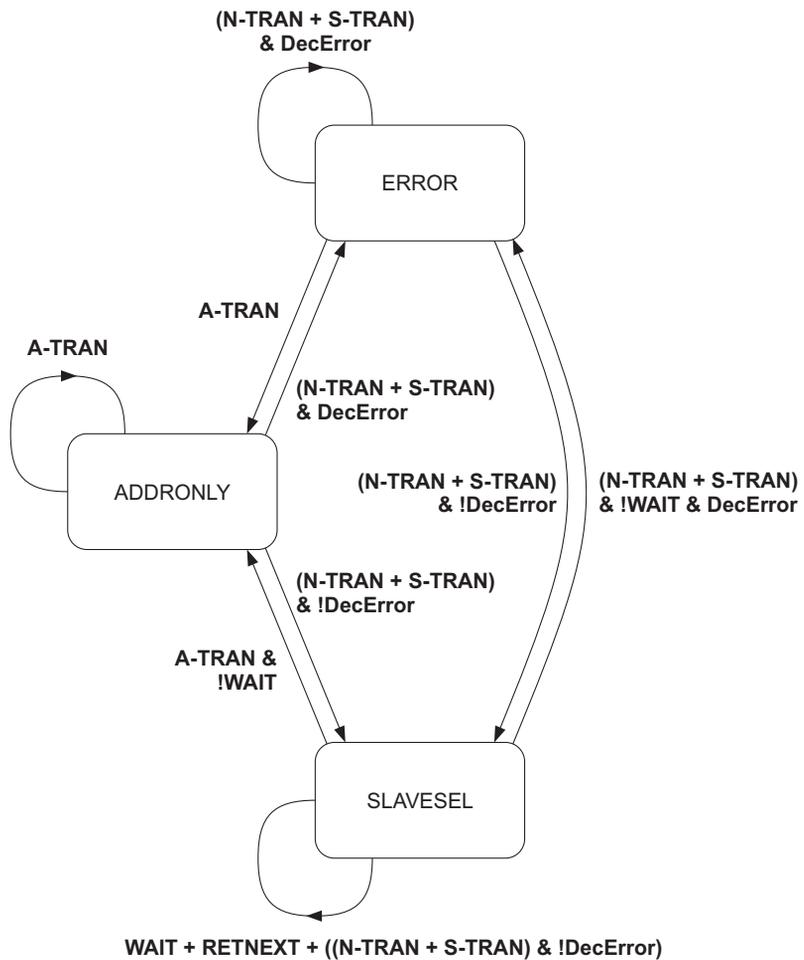
- there are no slaves present at the address of the transfer
- the transfer is to a protected region of memory
- the alignment of the transfer is not supported by the memory system.

The decoder performs the following functions:

- In the **ADDRONLY** state:
  - speculatively decodes the address
  - provides a **DONE** transfer response during the **BCLK LOW** phase
  - asserts **DSELx** during the **BCLK HIGH** phase if the transfer type for the next transfer is **S-TRAN** and the address is valid.
- In the **DECODE** state:
  - decodes the address
  - provides a **WAIT** transfer response during the **BCLK LOW** phase
  - asserts **DSELx** during the **BCLK HIGH** phase if the address is valid.
- In the **SLAVESEL** state:
  - the transfer response is driven by the selected slave
  - keeps **DSELx** asserted while the transfer is waited, or if the next transfer is **SEQUENTIAL** and no **LAST** condition is detected.
- In the **ERROR** state:
  - provides an **ERROR** transfer response during the **BCLK LOW** phase.

### Without decode cycles

A decoder which does not implement decode cycles has the DECODE state removed. This simplifies the state diagram, as shown in Figure 4-39.



!WAIT = DONE + LAST + ERROR + RETRACT

Figure 4-39 Decoder state machine without decode

### 4.12.3 Timing diagrams

The timing parameters for an ASB decoder with DECODE cycles are shown in Figure 4-40. The parameters for a decoder without DECODE cycles are shown in Figure 4-41. The main difference between the two diagrams is that when DECODE cycles are not inserted then the timing of the **DSEL** signal becomes dependent on the address and control signal timing.

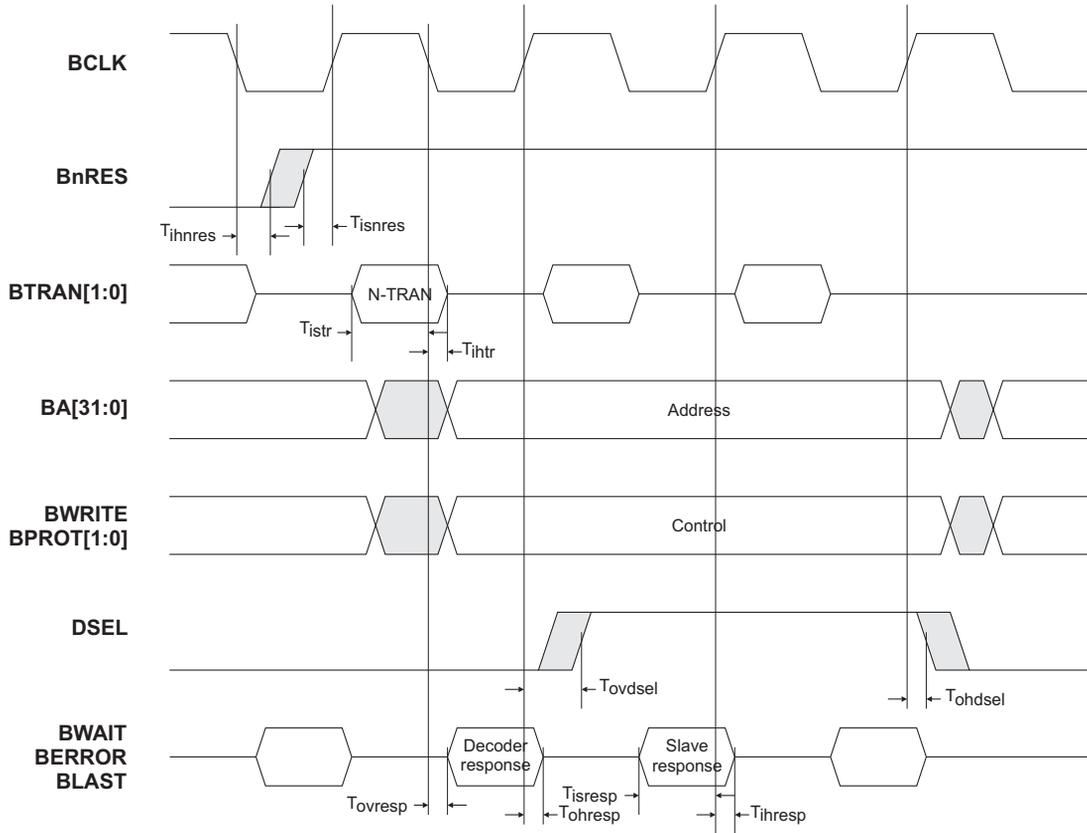


Figure 4-40 ASB decoder with decode cycles

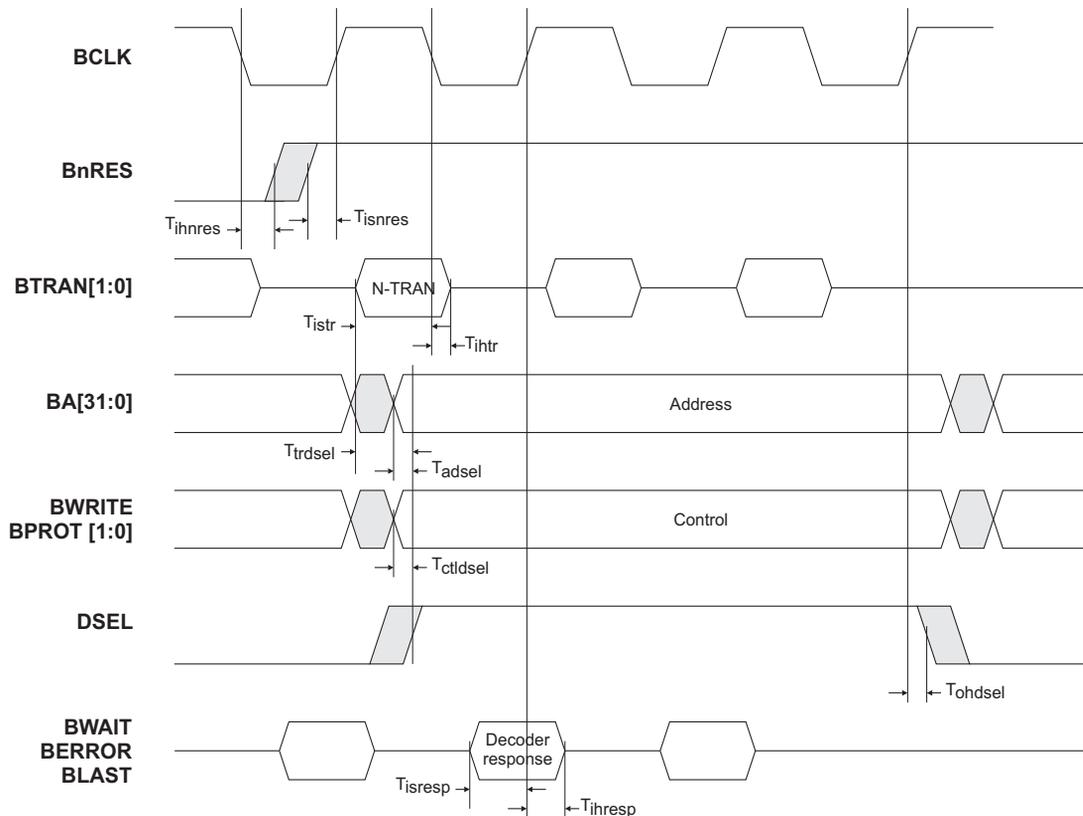


Figure 4-41 ASB decoder without decode cycles

#### 4.12.4 Timing parameters

The timing parameters related to an ASB decoder are given in the following tables:

- Table 4-11 is for input signals
- Table 4-12 is for output signals
- Table 4-13 is for combinatorially generated outputs.

Table 4-11 ASB decoder input parameters

Parameter	Description
$T_{\text{ckl}}$	<b>BCLK</b> LOW time
$T_{\text{clkh}}$	<b>BCLK</b> HIGH time
$T_{\text{isnres}}$	<b>BnRES</b> de-asserted setup to rising <b>BCLK</b>
$T_{\text{ihnres}}$	<b>BnRES</b> de-asserted hold after falling <b>BCLK</b>
$T_{\text{istr}}$	<b>BTRAN</b> setup to falling <b>BCLK</b>
$T_{\text{ihtr}}$	<b>BTRAN</b> hold after falling <b>BCLK</b>
$T_{\text{isresp}}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> setup to rising <b>BCLK</b>
$T_{\text{ihresp}}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> hold after rising <b>BCLK</b>

Table 4-12 ASB decoder output parameters

Parameter	Description
$T_{\text{ovresp}}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> valid after falling <b>BCLK</b>
$T_{\text{ohresp}}$	<b>BWAIT</b> , <b>BERROR</b> and <b>BLAST</b> hold after rising <b>BCLK</b>
$T_{\text{ovdtsel}}$	<b>DSEL</b> valid after rising <b>BCLK</b>
$T_{\text{ohdtsel}}$	<b>DSEL</b> hold after rising <b>BCLK</b>

Table 4-13 ASB decoder combinatorial parameters

Parameter	Description
$T_{\text{trdtsel}}$	Delay from valid <b>BTRAN</b> to valid <b>DSEL</b>
$T_{\text{adtsel}}$	Delay from valid <b>BA</b> to valid <b>DSEL</b>
$T_{\text{ctldtsel}}$	Delay from valid <b>BWRITE</b> and <b>BPROT[1:0]</b> to valid <b>DSEL</b>

## 4.13 ASB arbiter

The role of the arbiter in an AMBA system is to control which master has access to the bus. Every bus master has a two wire REQUEST and GRANT interface to the arbiter and on every cycle the arbiter uses a prioritization scheme to decide which bus master is currently the highest priority master requesting the bus.

A shared bus lock signal, **BLOK**, driven by the currently granted bus master is used to indicate that the current transfer is indivisible from the following transfer and no other master should be granted the bus.

The detail of the priority scheme is not specified and is defined for each application. It is acceptable for the arbiter to use other signals, either AMBA or non-AMBA, to influence the priority scheme that is in use.

### 4.13.1 Interface diagram

Figure 4-42 shows the signal interface of an ASB arbiter.

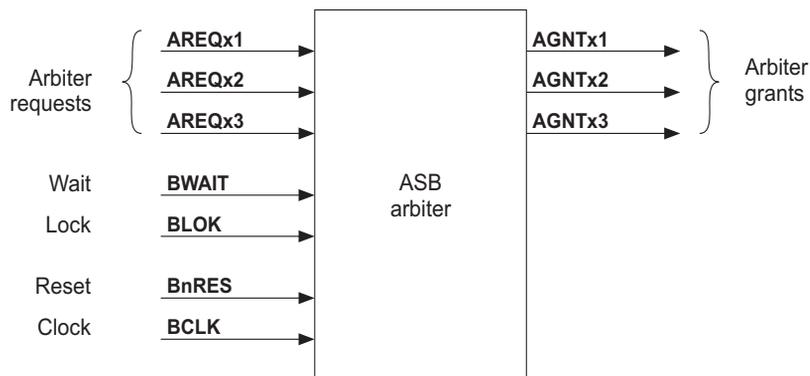


Figure 4-42 ASB arbiter interface diagram

### 4.13.2 Arbiter description

The bus can be re-arbitrated on every clock cycle. The arbiter samples all the request signals, **AREQ<sub>x</sub>**, on the falling edge of **BCLK** and during the LOW phase of **BCLK** the arbiter asserts the appropriate **AGNT<sub>x</sub>** signal using the internal priority scheme and the value of **BLOK**.

As the arbitration can change every cycle, it is possible that during an extended transfer, the highest priority bus master may change several times before the transfer eventually completes. The bus master that has **AGNT** asserted when the transfer completes will become the next active bus master.

During bus master handover the **BLOK** signal is not driven and hence the arbiter must assume that this signal is LOW.

The arbiter must retain a copy of which master is currently granted so it can:

- keep the current bus master granted if **BLOK** is asserted
- determine when the bus master changes, and so determine when there is a cycle of bus master handover.

### 4.13.3 Timing diagrams

Figure 4-43 shows the arbiter timing parameters.

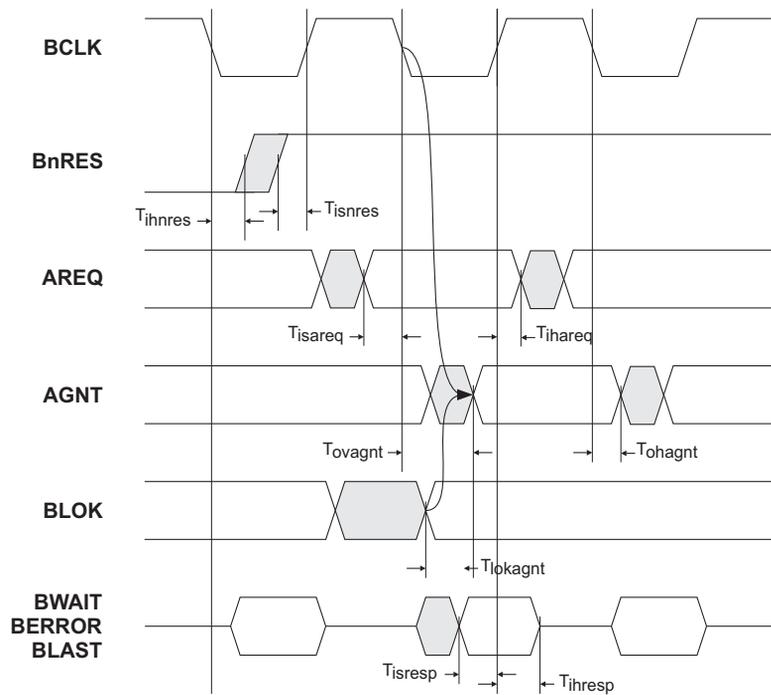


Figure 4-43 ASB arbiter parameters

#### 4.13.4 Timing parameters

The timing parameters related to an ASB arbiter are given in the following tables:

- Table 4-14 is for input signals
- Table 4-15 is for output signals
- Table 4-16 is for combinatorially generated outputs.

**Table 4-14 ASB arbiter input parameters**

Parameter	Description
$T_{\text{ckl}}$	<b>BCLK</b> LOW time
$T_{\text{ckh}}$	<b>BCLK</b> HIGH time
$T_{\text{isnres}}$	<b>BnRES</b> de-asserted setup to rising <b>BCLK</b>
$T_{\text{ihnres}}$	<b>BnRES</b> de-asserted hold after falling <b>BCLK</b>
$T_{\text{isareq}}$	<b>AREQ</b> setup to falling <b>BCLK</b>
$T_{\text{ihareq}}$	<b>AREQ</b> hold after rising <b>BCLK</b>
$T_{\text{isresp}}$	<b>BWAIT</b> setup to rising <b>BCLK</b>
$T_{\text{ihresp}}$	<b>BWAIT</b> hold after rising <b>BCLK</b>

**Table 4-15 ASB arbiter output parameters**

Parameter	Description
$T_{\text{ovagnt}}$	<b>AGNT</b> valid after falling <b>BCLK</b>
$T_{\text{ohagnt}}$	<b>AGNT</b> hold after falling <b>BCLK</b>

**Table 4-16 ASB arbiter combinatorial parameters**

Parameter	Description
$T_{\text{lokagnt}}$	Delay from valid <b>BLOK</b> to valid <b>AGNT</b>



# Chapter 5

## AMBA APB

This chapter introduces the *Advanced Microcontroller Bus Architecture* (AMBA) *Advanced Peripheral Bus* (APB) specification in the following sections:

- *About the AMBA APB* on page 5-2
- *APB specification* on page 5-4
- *About the APB AMBA components* on page 5-7
- *APB bridge* on page 5-8
- *APB slave* on page 5-11
- *Interfacing APB to AHB* on page 5-14
- *Interfacing APB to ASB* on page 5-20
- *Interfacing rev D APB peripherals to rev 2.0 APB* on page 5-22.

## 5.1 About the AMBA APB

The *Advanced Peripheral Bus* (APB) is part of the *Advanced Microcontroller Bus Architecture* (AMBA) hierarchy of buses and is optimized for minimal power consumption and reduced interface complexity.

The AMBA APB should be used to interface to any peripherals which are low-bandwidth and do not require the high performance of a pipelined bus interface.

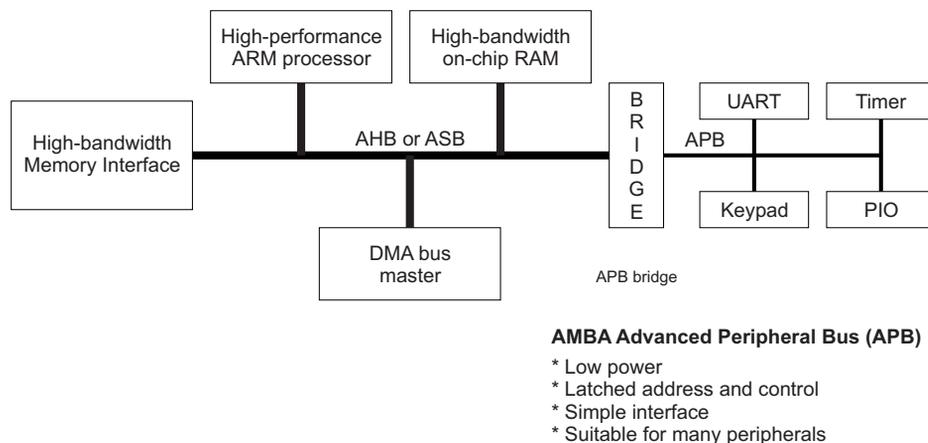
The latest revision of the APB ensures that all signal transitions are only related to the rising edge of the clock. This improvement means the APB peripherals can be integrated easily into any design flow, with the following advantages:

- performance is improved at high-frequency operation
- performance is independent of the mark-space ratio of the clock
- static timing analysis is simplified by the use of a single clock edge
- no special considerations are required for automatic test insertion
- many *Application-Specific Integrated Circuit* (ASIC) libraries have a better selection of rising edge registers
- easy integration with cycle based simulators.

These changes to the APB also make it simpler to interface it to the new *Advanced High-performance Bus* (AHB).

### 5.1.1 A typical AMBA-based microcontroller

An AMBA-based microcontroller typically consists of a high-performance system *backbone* bus, able to sustain the external memory bandwidth, on which the CPU and other *Direct Memory Access* (DMA) devices reside, plus a bridge to a narrower APB bus on which the lower bandwidth peripheral devices are located. Figure 5-1 shows the APB in a typical AMBA system.



**Figure 5-1 The APB in a typical AMBA system**

A system bus that includes a *Test Interface Controller* (TIC) allows modular testing of both system bus and APB modules.

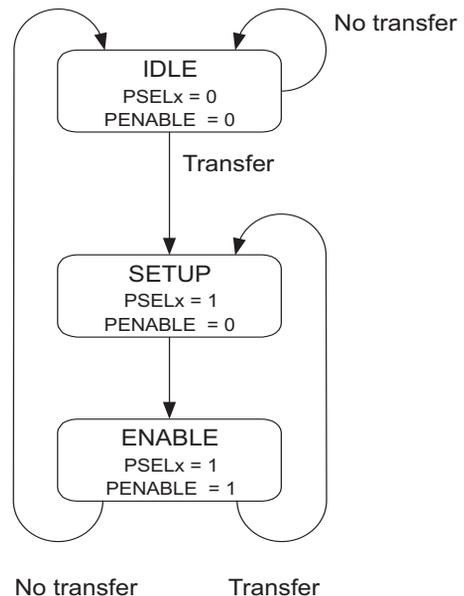
## 5.2 APB specification

The APB specification is described under the following headings:

- *State diagram*
- *Write transfer on page 5-5*
- *Read transfer on page 5-6.*

### 5.2.1 State diagram

The state diagram, shown in Figure 5-2, can be used to represent the activity of the peripheral bus.



**Figure 5-2 State diagram**

Operation of the state machine is through the three states described below:

IDLE	The default state for the peripheral bus.
SETUP	When a transfer is required the bus moves into the SETUP state, where the appropriate select signal, <b>PSELx</b> , is asserted. The bus only remains in the SETUP state for one clock cycle and will always move to the ENABLE state on the next rising edge of the clock.

**ENABLE**

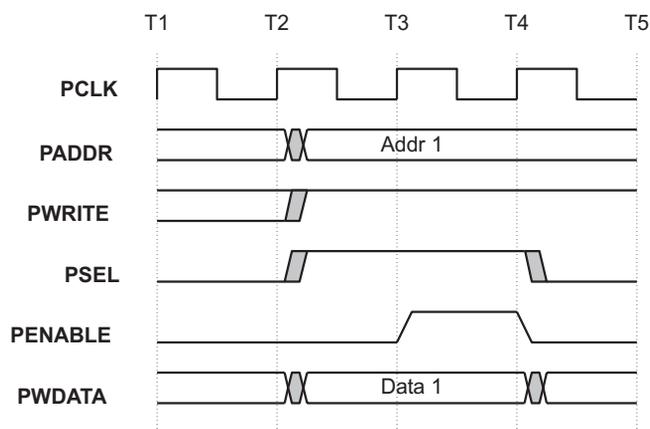
In the ENABLE state the enable signal, **PENABLE** is asserted. The address, write and select signals all remain stable during the transition from the SETUP to ENABLE state.

The ENABLE state also only lasts for a single clock cycle and after this state the bus will return to the IDLE state if no further transfers are required. Alternatively, if another transfer is to follow then the bus will move directly to the SETUP state.

It is acceptable for the address, write and select signals to glitch during a transition from the ENABLE to SETUP states.

## 5.2.2 Write transfer

The basic write transfer is shown in Figure 5-3.



**Figure 5-3 Write transfer**

The write transfer starts with the address, write data, write signal and select signal all changing after the rising edge of the clock. The first clock cycle of the transfer is called the SETUP cycle. After the following clock edge the enable signal **PENABLE** is asserted, and this indicates that the ENABLE cycle is taking place. The address, data and control signals all remain valid throughout the ENABLE cycle. The transfer completes at the end of this cycle.

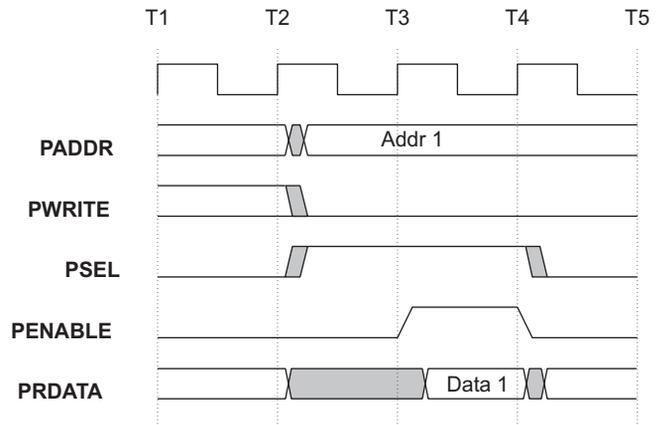
The enable signal, **PENABLE**, will be deasserted at the end of the transfer. The select signal will also go LOW, unless the transfer is to be immediately followed by another transfer to the same peripheral.

In order to reduce power consumption the address signal and the write signal will not change after a transfer until the next access occurs.

The protocol only requires a clean transition on the enable signal. It is possible that in the case of back to back transfers the select and write signals may glitch.

### 5.2.3 Read transfer

Figure 5-4 shows a read transfer.



**Figure 5-4 Read transfer**

The timing of the address, write, select and strobe signals are all the same as for the write transfer. In the case of a read, the slave must provide the data during the ENABLE cycle. The data is sampled on the rising edge of clock at the end of the ENABLE cycle.

### 5.3 About the APB AMBA components

The following notation is used for the timing parameters:

- $T_{is}$  - input setup time
- $T_{ih}$  - input hold time
- $T_{ov}$  - output valid time
- $T_{oh}$  - output hold time.

## 5.4 APB bridge

The APB bridge is the only bus master on the AMBA APB. In addition, the APB bridge is also a slave on the higher-level system bus.

### 5.4.1 Interface diagram

Figure 5-5 shows the APB signal interface of an APB bridge.

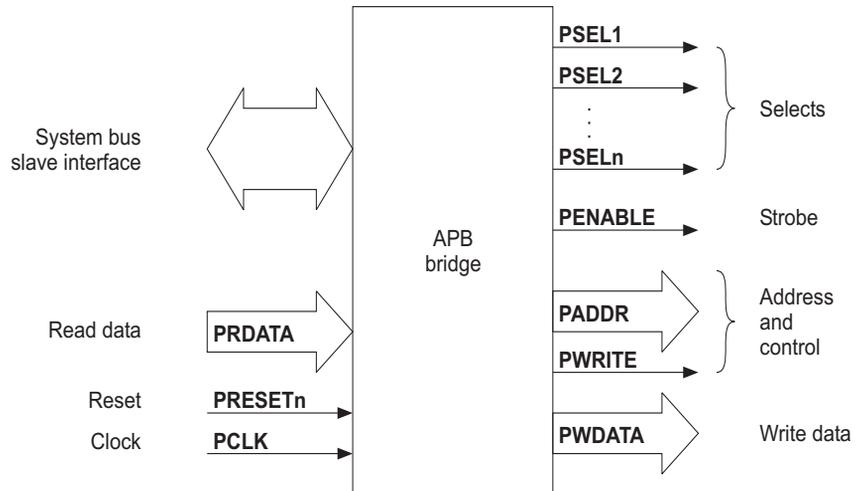


Figure 5-5 APB bridge interface diagram

### 5.4.2 APB bridge description

The bridge unit converts system bus transfers into APB transfers and performs the following functions:

- Latches the address and holds it valid throughout the transfer.
- Decodes the address and generates a peripheral select, **PSELx**. Only one select signal can be active during a transfer.
- Drives the data onto the APB for a write transfer.
- Drives the APB data onto the system bus for a read transfer.
- Generates a timing strobe, **PENABLE**, for the transfer.

### 5.4.3 Timing diagrams

The timing parameters for an APB bridge are shown in Figure 5-6.

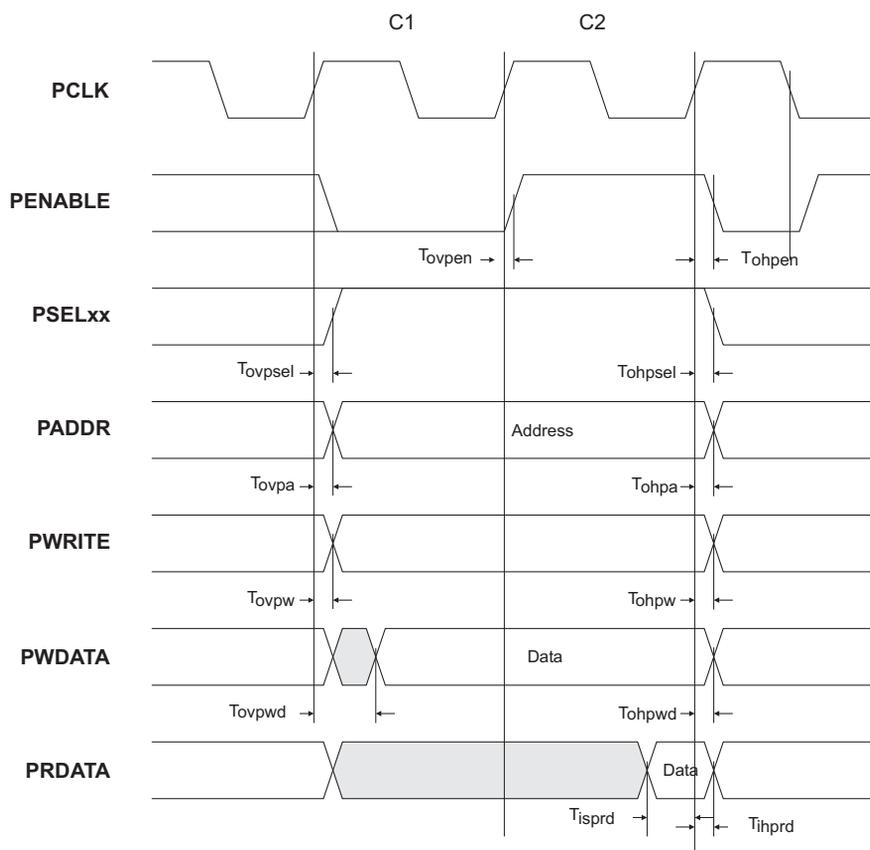


Figure 5-6 APB bridge transfer

#### 5.4.4 Timing parameters

The timing parameters related to an APB bridge are given in Table 5-1 for input signals and Table 5-2 for output signals.

**Table 5-1 APB bridge input parameters**

Parameter	Description
$T_{\text{clk}l}$	<b>PCLK</b> LOW time
$T_{\text{clk}h}$	<b>PCLK</b> HIGH time
$T_{\text{isnres}}$	<b>PRESETn</b> de-asserted setup to rising <b>PCLK</b>
$T_{\text{ihnres}}$	<b>PRESETn</b> de-asserted hold after rising <b>PCLK</b>
$T_{\text{isprd}}$	For read transfers, <b>PRDATA</b> setup to rising <b>PCLK</b>
$T_{\text{ihprd}}$	For read transfers, <b>PRDATA</b> hold after rising <b>PCLK</b>

**Table 5-2 APB bridge output parameters**

Parameter	Description
$T_{\text{open}}$	<b>PENABLE</b> valid after rising <b>PCLK</b>
$T_{\text{ohpen}}$	<b>PENABLE</b> hold after rising <b>PCLK</b>
$T_{\text{ovpsel}}$	<b>PSEL</b> valid after rising <b>PCLK</b>
$T_{\text{ohpsel}}$	<b>PSEL</b> hold after rising <b>PCLK</b>
$T_{\text{ovpa}}$	<b>PADDR</b> valid after rising <b>PCLK</b>
$T_{\text{ohpa}}$	<b>PADDR</b> hold after rising <b>PCLK</b>
$T_{\text{ovpw}}$	<b>PWRITE</b> valid after rising <b>PCLK</b>
$T_{\text{ohpw}}$	<b>PWRITE</b> hold after rising <b>PCLK</b>
$T_{\text{ovpwd}}$	For write transfers, <b>PWDATA</b> valid after rising <b>PCLK</b>
$T_{\text{ohpwd}}$	For write transfers, <b>PWDATA</b> hold after rising <b>PCLK</b>

## 5.5 APB slave

APB slaves have a simple, yet flexible, interface. The exact implementation of the interface will be dependent on the design style employed and many different options are possible.

### 5.5.1 Interface diagram

Figure 5-7 shows the signal interface of an APB slave.

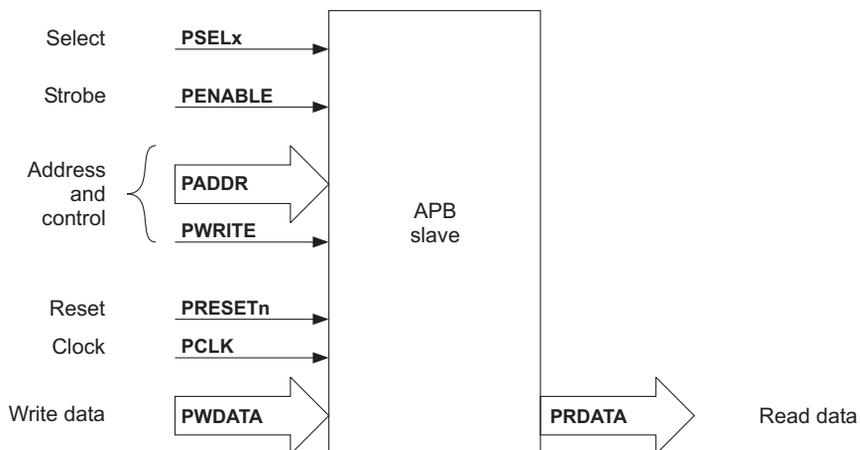


Figure 5-7 APB slave interface description

### 5.5.2 APB slave description

The APB slave interface is very flexible.

For a write transfer the data can be latched at the following points:

- on either rising edge of **PCLK**, when **PSEL** is HIGH
- on the rising edge of **PENABLE**, when **PSEL** is HIGH.

The select signal **PSELx**, the address **PADDR** and the write signal **PWRITE** can be combined to determine which register should be updated by the write operation.

For read transfers the data can be driven on to the data bus when **PWRITE** is LOW and both **PSELx** and **PENABLE** are HIGH. While **PADDR** is used to determine which register should be read.

### 5.5.3 Timing diagrams

The timing parameters related to an access to an APB bus slave are shown in Figure 5-8.

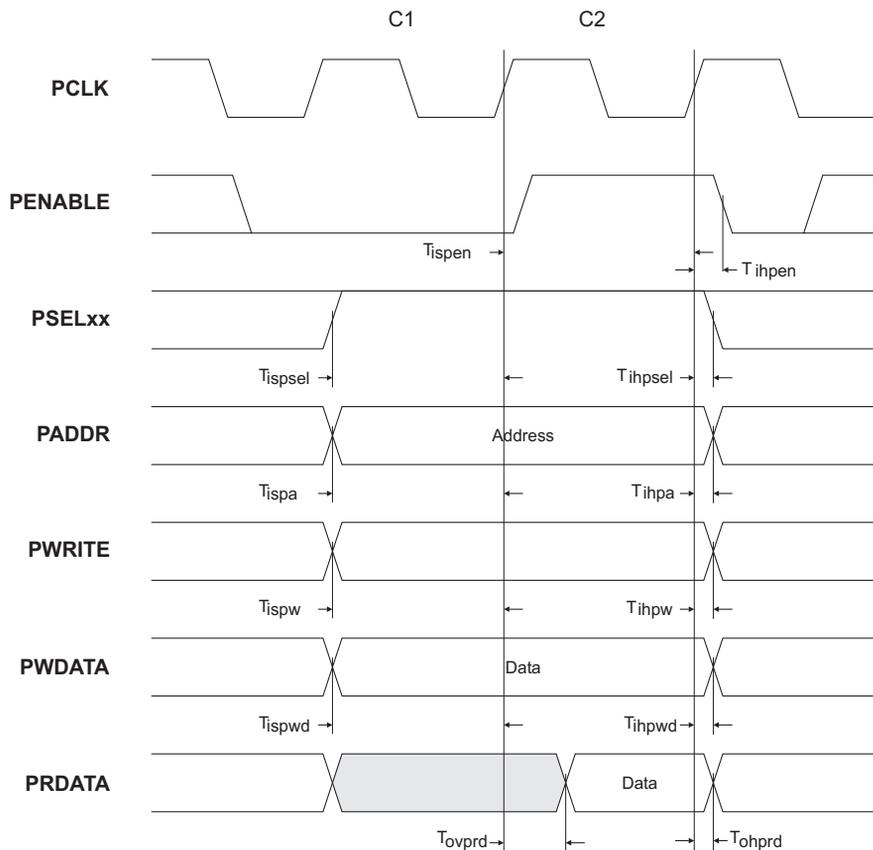


Figure 5-8 APB slave transfer

### 5.5.4 Timing parameters

The timing parameters related to an APB slave are given in Table 5-3 for input signals and Table 5-4 for output signals.

**Table 5-3 APB slave input parameters**

<b>Parameter</b>	<b>Description</b>
$T_{\text{ckl}}$	<b>PCLK</b> LOW time
$T_{\text{cklh}}$	<b>PCLK</b> HIGH time
$T_{\text{isnres}}$	<b>PRESETn</b> de-asserted setup to rising <b>PCLK</b>
$T_{\text{ihnres}}$	<b>PRESETn</b> de-asserted hold after falling <b>PCLK</b>
$T_{\text{ispen}}$	<b>PENABLE</b> setup to rising <b>PCLK</b>
$T_{\text{ihpen}}$	<b>PENABLE</b> hold after rising <b>PCLK</b>
$T_{\text{ispsel}}$	<b>PSEL</b> setup to rising <b>PCLK</b>
$T_{\text{ihpsel}}$	<b>PSEL</b> hold after rising <b>PCLK</b>
$T_{\text{ispa}}$	<b>PADDR</b> setup to rising <b>PCLK</b>
$T_{\text{ihpa}}$	<b>PADDR</b> hold after rising <b>PCLK</b>
$T_{\text{ispw}}$	<b>PWRITE</b> setup to rising <b>PCLK</b>
$T_{\text{ihpw}}$	<b>PWRITE</b> hold after rising <b>PCLK</b>
$T_{\text{ispwd}}$	For write transfers, <b>PWDATA</b> setup to rising <b>PCLK</b>
$T_{\text{ihpwd}}$	For write transfers, <b>PWDATA</b> hold after rising <b>PCLK</b>

**Table 5-4 APB slave output parameters**

<b>Parameter</b>	<b>Description</b>
$T_{\text{ovprd}}$	For read transfers, <b>PRDATA</b> valid after rising <b>PCLK</b>
$T_{\text{ohprd}}$	For read transfers, <b>PRDATA</b> hold after rising <b>PCLK</b>

## 5.6 Interfacing APB to AHB

Interfacing the AMBA APB to the AHB is described in:

- *Read transfers*
- *Write transfers* on page 5-16
- *Back to back transfers* on page 5-18
- *Tristate data bus implementations* on page 5-19.

### 5.6.1 Read transfers

Figure 5-9 illustrates a read transfer.

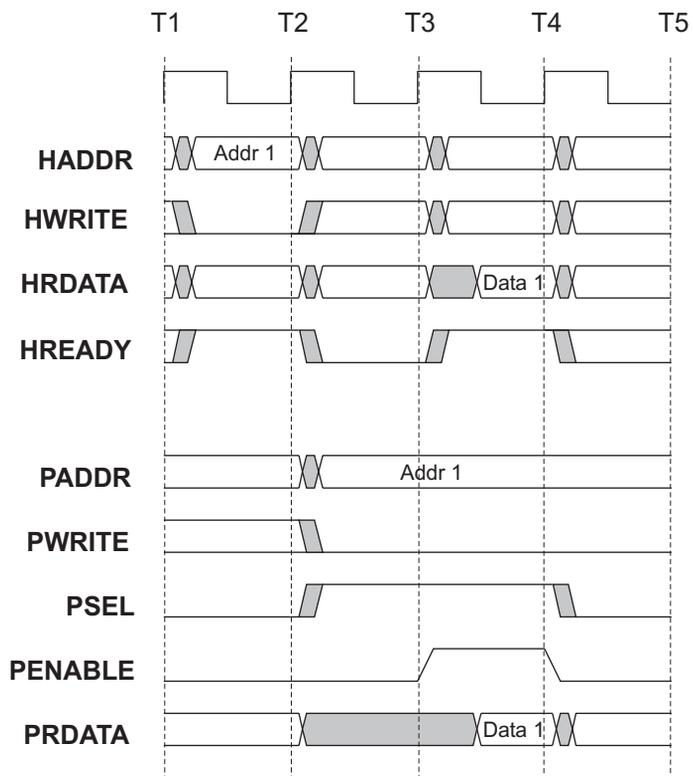


Figure 5-9 Read transfer to AHB

The transfer starts on the AHB at time T1 and the address is sampled by the APB bridge at T2. If the transfer is for the peripheral bus then this address is broadcast and the appropriate peripheral select signal is generated. This first cycle on the peripheral bus is called the SETUP cycle, this is followed by the ENABLE cycle, when the **PENABLE** signal is asserted.

During the ENABLE cycle the peripheral must provide the read data. Normally it will be possible to route this read data directly back to the AHB, where the bus master can sample it on the rising edge of the clock at the end of the ENABLE cycle, which is at time T4 in Figure 5-9.

In very high clock frequency systems it may become necessary for the bridge to register the read data at the end of the ENABLE cycle and then for the bridge to drive this back to the AHB bus master in the following cycle. Although this will require an extra wait state for peripheral bus read transfers, it allows the AHB to run at a higher clock frequency, thus resulting in an overall improvement in system performance. A burst of read transfers is shown in Figure 5-10. All read transfers require a single wait state.

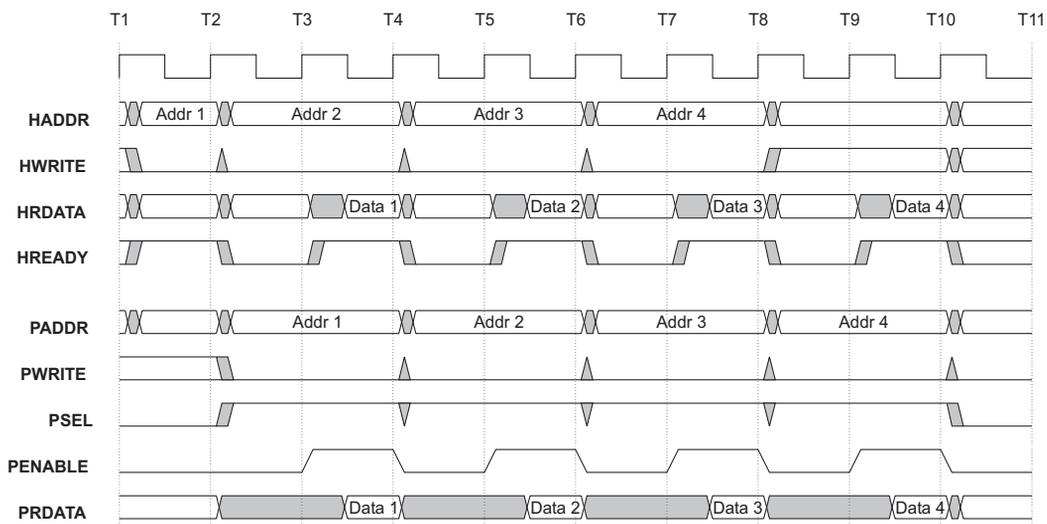
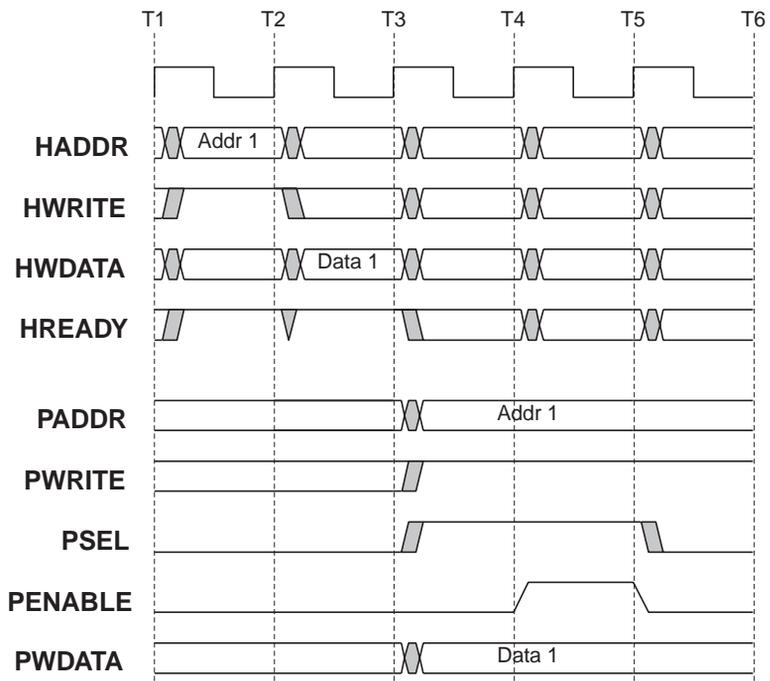


Figure 5-10 Burst of read transfers

### 5.6.2 Write transfers

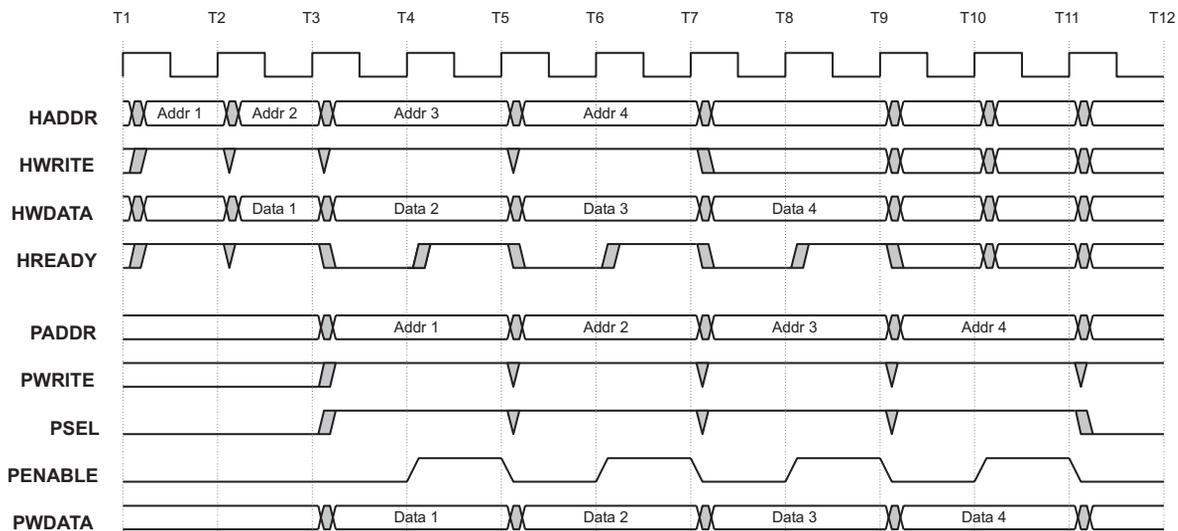
Figure 5-11 shows a write transfer.



**Figure 5-11 Write transfer from AHB**

Single write transfers to the APB can occur with zero wait states. The bridge is responsible for sampling the address and data of the transfer and then holding these values for the duration of the write transfer on the APB.

A burst of write transfers is shown in Figure 5-12.



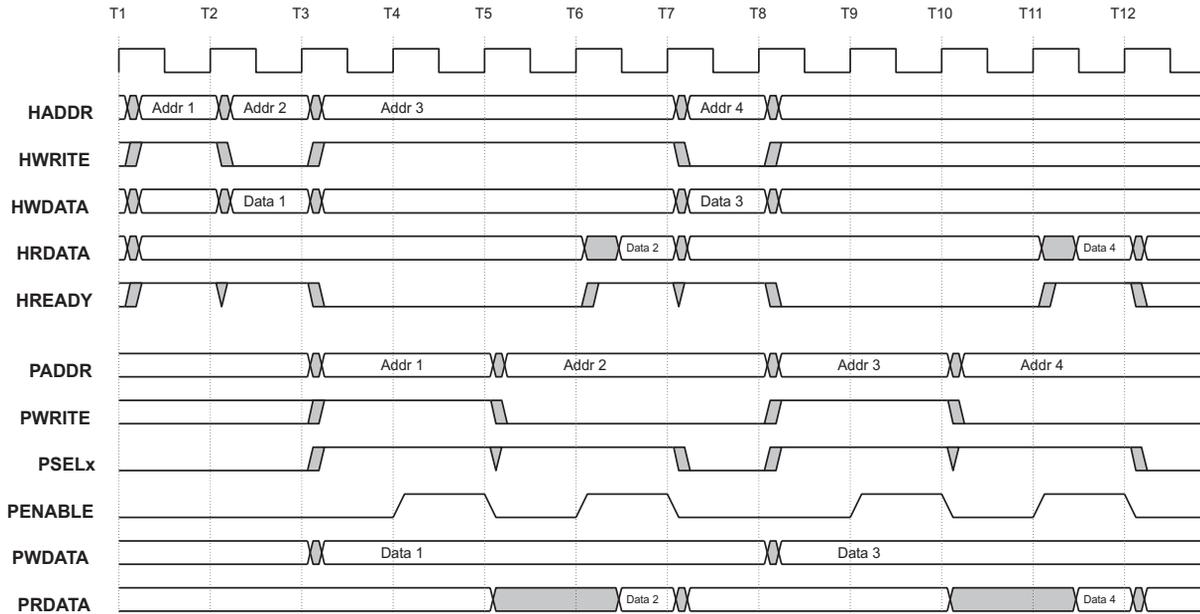
**Figure 5-12 Burst of write transfers**

While the first transfer can complete with zero wait states, subsequent transfers to the peripheral bus will require a single wait state for each transfer performed.

It is necessary for the bridge to contain two address registers, in order that the bridge can sample the address of the next transfer while the current transfer continues on the peripheral bus.

### 5.6.3 Back to back transfers

Figure 5-13 shows a number of back to back transfers. The sequence starts with a write, which is then followed by a read, then a write, then a read.



**Figure 5-13 Back to back transfers**

Figure 5-13 shows that if a read transfer immediately follows a write, then 3 wait states are required to complete the read. In fact, in a processor-based design a write followed by a read does not occur frequently as the processor will perform an instruction fetch between the two transfers and it is unlikely that the instruction memory would reside on the APB.

## 5.6.4 Tristate data bus implementations

It is recommended that the AMBA APB is implemented with separate read and write data buses, which allows the use of either a multiplexed bus or OR-bus scheme to interconnect the various slaves on the APB. If a tristate bus is used then the read and write data buses may be combined into a single bus, as read data and write data never occur simultaneously.

Figure 5-14 illustrates that no special consideration is required if the data bus is implemented using tristate buffers. If the data bus is tristate in the SETUP cycle of a read transfer and whenever the bus is in the Idle state then an entire clock cycle of turnaround always occurs between different drivers of the data. For bursts of write transfers there is no turnaround as the bridge will drive data in the SETUP cycle of every transfer, however this is perfectly acceptable as the bridge is the only driver of the data bus for write transfers and therefore no turnaround period is required.

Figure 5-14 shows how the read and write data buses can be successfully combined into a single tristate data bus.

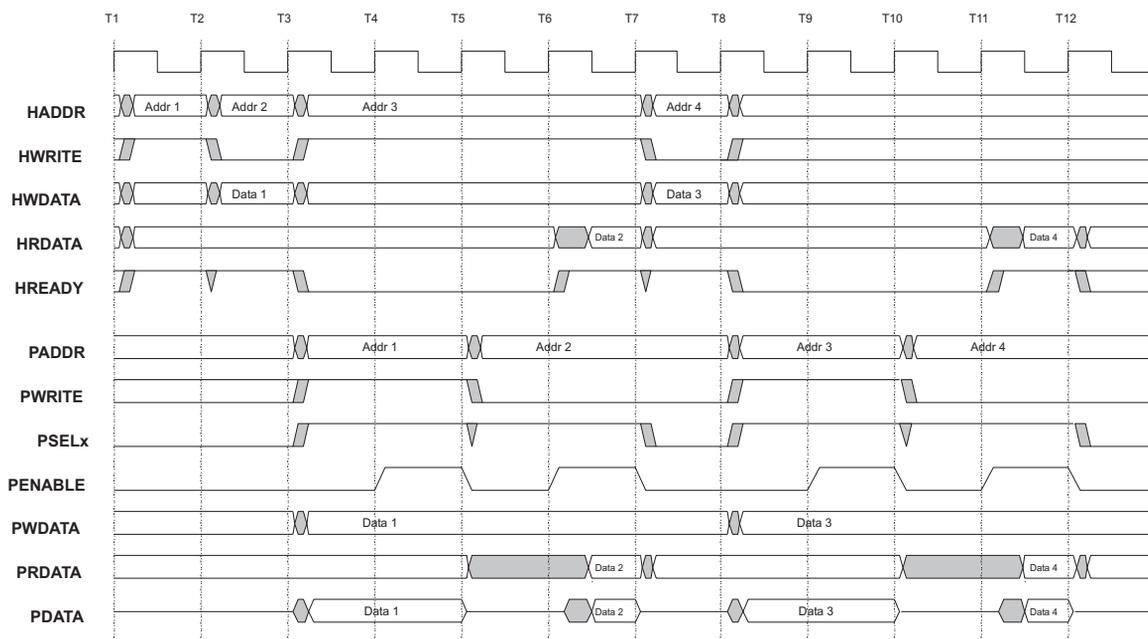


Figure 5-14 Tristate data bus

## 5.7 Interfacing APB to ASB

Interfacing the AMBA APB to the ASB is described in:

- *Write transfer*
- *Read transfer* on page 5-21.

### 5.7.1 Write transfer

Figure 5-15 illustrates how an interface from ASB to APB can be constructed. The write transfer can occur with zero wait-states, although an additional wait state is required for a burst of writes.

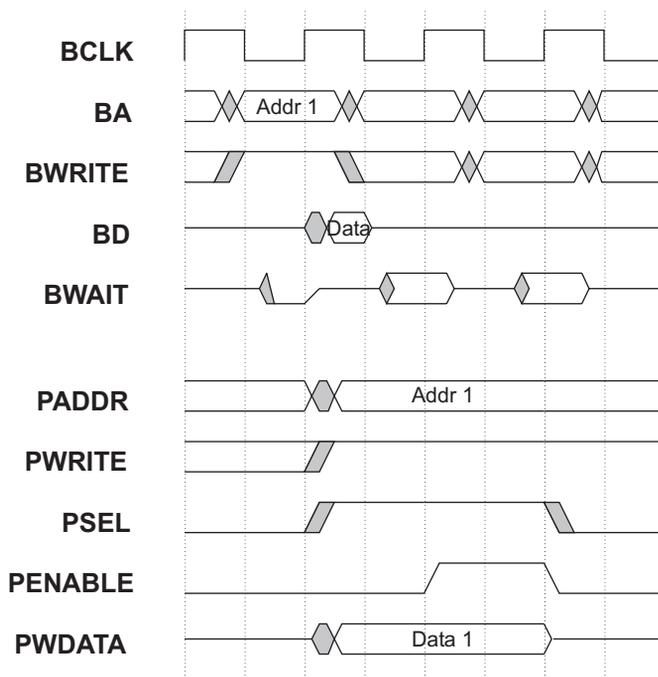


Figure 5-15 Write transfer from ASB

## 5.7.2 Read transfer

The read transfer will always require a single wait state (see Figure 5-16). In systems with a high clock frequency it may be necessary to insert an additional wait state to ensure that the read data has adequate time to pass through the bridge and become valid on the ASB.

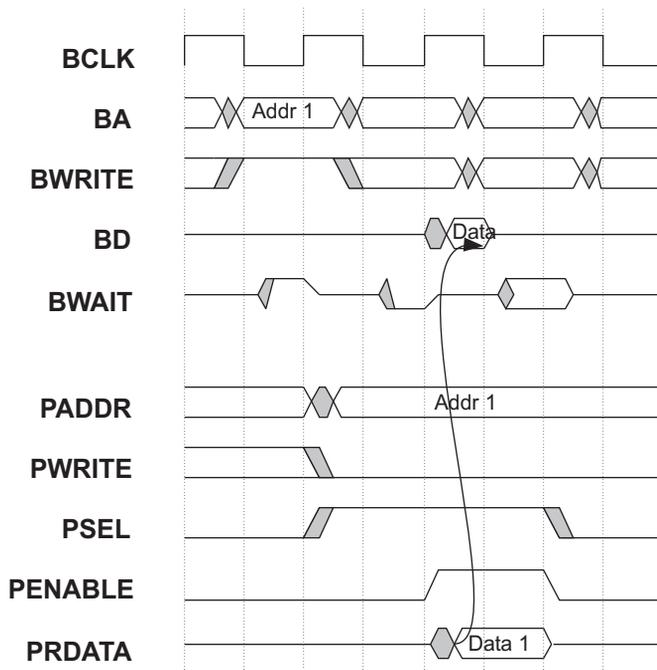


Figure 5-16 Read transfer to ASB

## 5.8 Interfacing rev D APB peripherals to rev 2.0 APB

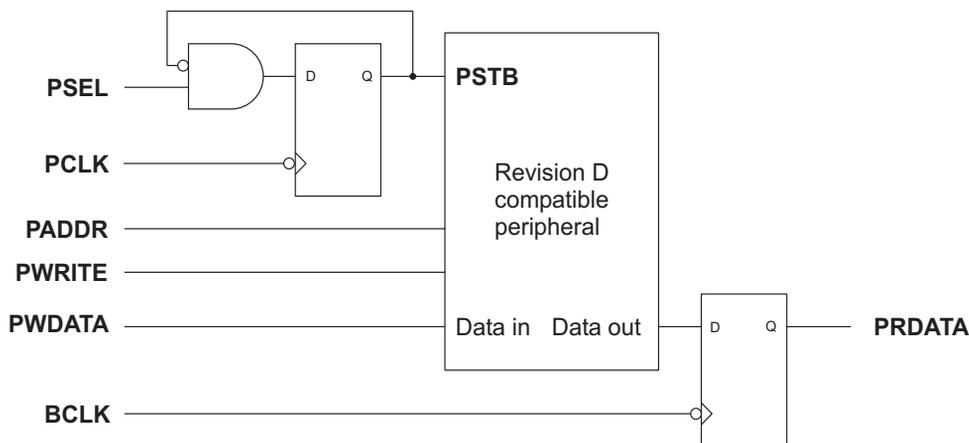
When using a combination of peripherals, some designed to the revision 2.0 specification and others designed to previous revisions, it is recommended that a revision 2.0 bridge is used and the earlier version peripherals are converted for use with the new bridge.

This section shows how a single revision D peripheral may be converted to the latest version of the APB. If a number of peripherals are to be converted it is more efficient to perform the conversion in a single centralized block.

There are two fundamental differences between the rev D and rev 2.0 APB specifications:

- the timing of the strobe signal compared to the enable signal
- the point at which read data is sampled.

To quickly determine whether a peripheral is designed to the rev D or rev 2.0 specification, see if it has a **PSTB** input (in which case it is rev D) or a **PENABLE** input (in which case it is rev 2.0). Figure 5-17 shows the two stages that are required to interface an existing revision D peripheral.



**Figure 5-17** Interfacing a rev D peripheral

Firstly, the **PSEL** signal may be used to generate a **PSTB** signal. A fed-back version of the **PSTB** signal can be used to ensure the signal is only asserted for a single clock cycle.

The second interface stage that may be required is a falling edge triggered register or transparent latch on the output data (read data) from the peripheral. This is only required if the peripheral changes the output data after the falling edge.

# Chapter 6

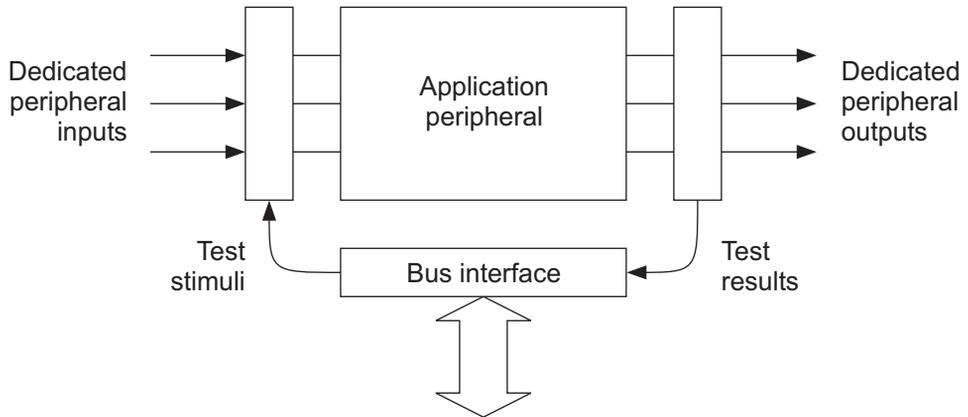
## AMBA Test Methodology

This chapter describes the test interface used with AMBA module designs. It contains the following sections:

- *About the AMBA test interface* on page 6-2
- *External interface* on page 6-4
- *Test vector types* on page 6-6
- *Test interface controller* on page 6-7
- *The AHB Test Interface Controller* on page 6-12
- *Example AMBA AHB test sequences* on page 6-17
- *The ASB test interface controller* on page 6-25
- *Example AMBA ASB test sequences* on page 6-27.

## 6.1 About the AMBA test interface

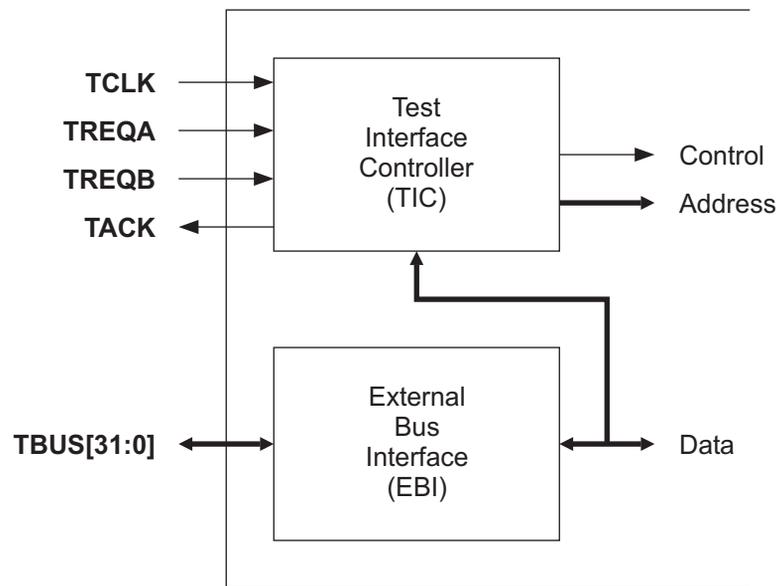
The AMBA test philosophy allows individual modules in the system to be tested in isolation. Each module is designed so it can be tested only using transfers from the bus and does not rely on the interaction of any other system element. Therefore it is necessary to have access to the inputs and outputs of the peripheral that are not directly connected to the bus and this is provided by a test harness.



**Figure 6-1 Peripheral test harness**

A low gate-count *Test Interface Controller* (TIC) bus master module is required in the system to allow externally applied test vectors to be converted into internal bus transfers.

The TIC uses a minimal three-wire handshake mechanism to control the application of test vectors and the data path of the *External Bus Interface* (EBI) is used to provide a high speed 32-bit parallel vector interface.



**Figure 6-2 TIC and external bus interface interaction**

To support this method of test vector application a 32-bit bidirectional port must be available during test access. For a system with an external data bus interface of 32-bits this is straightforward. 16-bit and 8-bit data bus designs require, for example, 16 or 24 address lines to be reconfigured as bidirectional test port signals for test mode access.

## 6.2 External interface

The external test interface consists of:

- a test clock
- three control signals
- a 32-bit test bus.

Only two dedicated signal pins are required (**TREQA** and **TACK**) to control the entry and exit of test mode. The remaining signals can be provided by reusing existing device pins.

### 6.2.1 Test bus request A

**TREQA** is the test bus request A input signal and is required as a dedicated device pin.

During normal system operation the **TREQA** signal is used to request entry into the test mode. This will cause the test bus to become tristated, allowing test vectors to be applied.

During test this signal is used, in combination with **TREQB**, to indicate the type of test vector that will be applied in the following cycle.

### 6.2.2 Test bus request B

**TREQB** is the test bus request B input signal.

During test this signal is used, in combination with **TREQA**, to indicate the type of test vector that will be applied in the following cycle.

### 6.2.3 Test acknowledge

**TACK** is the test bus acknowledge output signal and is required as a dedicated device pin.

The test bus acknowledge signal gives external indication that the test bus has been granted and also indicates when a test access has completed. When **TACK** is LOW the current test vector must be extended until **TACK** becomes HIGH. The **TREQA** and **TREQB** signals are only sampled by the TIC when **TACK** is HIGH.

Table 6-1 and Table 6-2 show the operation of the **TREQA**, **TREQB** and **TACK** signals. The signals have different functions depending on whether or not test mode has been entered.

Table 6-1 Test control signals during normal operation

TREQA	TREQB	TACK	Description
0	0	0	Normal operation
1	0	0	Enter test mode request
0	1	0	Reserved (for external master request)
-	-	1	Test mode entered

Table 6-2 Test control signals during test mode

TREQA	TREQB	TACK	Description
-	-	0	Current access incomplete
1	1	1	Address vector, control vector or turnaround vector
1	0	1	Write vector
0	1	1	Read vector
0	0	1	Exit test mode

#### 6.2.4 Test clock

**TCLK** is the test clock input signal.

In test mode, the internal bus clock is driven from the external **TCLK** source. This pin may be the normal clock oscillator source input or a port replacement signal. The system bus clock must not glitch when switching between normal and test mode.

On entry into test mode the TIC indicates that it has switched to the test clock input by asserting the **TACK** signal.

#### 6.2.5 Test bus

**TBUS[31:0]** is the 32-bit bidirectional test port.

The test bus is used as an input to apply address, control and write vectors. For read vectors the test bus is used as a device output. The test interface protocol ensures that a turnaround period is always provided when changing the direction of the test bus.

## 6.3 Test vector types

There are 5 types of test vector associated with the test interface:

- address vector
- write vector
- read vector
- control vector
- turnaround vector.

Address vector, control vector and turnaround vector are all indicated by the same value on the **TREQA** and **TREQB** signals. The following rules may be used to determine which type of vector is being applied.

- When a single address/control vector is applied it is an address vector.
- When a burst of address/control vectors are applied they are all address vectors, apart from the last which is a control vector.
- A read vector, or burst of read vectors, is always followed by a turnaround vector. This is the only occurrence of the turnaround vector. The ASB version of the test interface requires a single turnaround vector, while the AHB version requires two.

## 6.4 Test interface controller

The *Test Interface Controller (TIC)* is a bus master that accepts test vectors from the external test bus, **TBUS[31:0]**, and initiates bus transfers. The TIC latches address vectors and, when required, increments the address to allow read and write bursts of test vectors.

### 6.4.1 Test transfer parameters

The default TIC bus master operation when entering test mode is:

- 32-bit transfer width
- privileged system access.

This is sufficient for testing many embedded system designs and minimizes the on-chip test support logic. In the case of systems that require the above control signals to be dynamically changed, a control vector mechanism is used to update the control signals within the TIC.

Bit 0 of the control vector is used to indicate if the control vector is valid. Thus, if a control vector is applied with bit 0 LOW, the vector will be ignored and will not update the control information. This mechanism allows address vectors which have bit 0 LOW to be applied for many cycles without updating the control information.

### 6.4.2 Incremental addressing

In order to support burst accesses using the test interface the TIC may support incrementing of the bus address. The number of address bits that are incremented is dependent on the maximum burst access length that is required via the test interface. This is system-dependent but a typical implementation would use an 8-bit address incrementer, allowing burst access up to 1kB boundaries using word transfers.

The control vector also provides a mechanism to enable and disable the address incrementer within the TIC. This allows burst accesses to incremental addresses, as would be used for testing internal RAM. Alternatively, the address increment can be disabled, such that successive accesses of a burst occur to the same address, as would be required to continually read from a single peripheral register.

If the transfer size is changed dynamically then any address incrementer support for burst-mode accesses must be able to support increment by byte, halfword and word offsets, so adaptive address incrementer logic is required.

The address incrementer is disabled by default and must be enabled using a control vector prior to use.

### 6.4.3 Entering test mode

In normal operating mode **TREQA** will be LOW, indicating that test access is not required and the test bus will be used as required for normal operation, which will usually be part of the external bus interface. Entering test mode allows test vectors to be applied externally that will cause transfers on the internal bus.

The following sequence is required in order to enter test mode:

1. **TREQA** is asserted to request test bus access.
2. Test mode is entered when the TIC has been granted the internal bus and this is indicated by the assertion of the **TACK** signal.
3. At this point **TCLK** will become the source of the internal clock signal.
4. When test mode has been entered **TREQB** is asserted to initiate an address vector.

The TIC will not perform any internal transfers until a valid address vector has been applied.

A synchronous tester would not be expected to poll **TACK** for the bus. Normally the **TREQA** signal would be asserted for a minimum number of cycles to guarantee to gain access to the bus (completion of the longest wait-state peripheral access or the maximum number of cycles for all bus masters to have completed their current instruction).

### 6.4.4 Address vectors

An address vector must be applied before any read or write operations can occur. The following sequence is required in order to apply an address vector:

1. **TREQA** and **TREQB** are both asserted HIGH indicating an address vector next cycle.
2. In the next cycle the address is applied to **TBUS[31:0]**, while **TREQA** and **TREQB** change to reflect the type of test vector that will follow.

In some high-speed systems it may be necessary to apply more than one address vector in succession, to allow sufficient time for the address to propagate from the external test bus through to the internal address bus. In such a case the TIC can negate **TACK** for the first cycle of the address vector, forcing a second cycle of address vector to be applied.

### 6.4.5 Control vector

A control vector is always the last in a sequence of address vectors and is used to update control information within the TIC. The sequence is as follows:

1. **TREQA** and **TREQB** are asserted HIGH indicating an address vector next cycle.
2. In the next cycle the address is applied to **TBUS[31:0]**. **TREQA** and **TREQB** both remain HIGH as the control vector will occur in the following cycle.
3. In the next cycle the control information is applied to **TBUS[31:0]**, while **TREQA** and **TREQB** change to reflect the type of test vector that will follow.
4. Finally the transfer occurs on the internal bus.

It is possible to apply an invalid control vector, by setting bit 0 of the control vector LOW. This will not change the control information within the TIC.

### 6.4.6 Write test vectors

Once test mode has successfully been entered, read and write operations may be performed through the test interface. In order to perform a write operation internally it is necessary to supply an address followed by the write data.

The address used for the write transfer will depend on the preceding vectors and a write vector may occur after any of the following:

- a single address vector
- an address/control vector sequence
- another write test vector, forming a burst of writes
- a turnaround vector after a single read or burst of reads.

When an internal bus transfer is extended by the insertion of wait states this is indicated externally by the **TACK** signal going LOW. During the waited condition the **TREQA** and **TREQB** should change to indicate the vector type that will follow when the current vector has completed. However, it is important to note that in the case of a write vector the data should continue to be applied to **TBUS[31:0]**.

### 6.4.7 Read test vectors

In a similar manner to write test vectors, read test vectors may follow a number of different vectors, as listed below, and the address used for the transfer will depend on the preceding vectors:

- a single address vector
- an address/control vector sequence
- another read test vector, forming a burst of reads
- a single write or burst of writes.

A read, or burst of reads, must always be followed by a turnaround vector to prevent bus clash on the external **TBUS** signals. As for a write vector, if the external transfer is extended then this is indicated externally by the **TACK** signal going LOW. The read data should not be sampled externally until the internal transfer has completed.

### 6.4.8 Burst vectors

Multiple write vectors or read vectors may be joined together to form bursts of vectors. This enables test vectors to be applied at a much faster rate by removing the need for an address vector to be associated with each read or write vector.

Burst transfers may use either incrementing addresses or static addresses, depending on whether or not the TIC contains an address incrementer which is enabled. With no address incrementer the TIC will perform non-sequential transfers to a constant address.

If the TIC does contain an enabled address incrementer then the address used for each successive transfer will be incremented by the appropriate amount, which is dictated by the transfer size.

### 6.4.9 Changing a burst direction

It is possible to change the transfer direction of a burst, from read to write or write to read.

If changing from read to write it is still necessary to insert a turnaround vector. This will not load a new address but will internally cause a new burst to be started allowing internal slaves to observe that the direction of the burst has altered.

### 6.4.10 Exiting test mode

Test mode is exited using the following sequence:

1. Apply a single cycle of address vector, which ensures any internal transfers have been completed.
2. **TREQA** and **TREQB** are both driven LOW to indicate that test mode is to be exited.
3. When the test interface has been configured for normal system operation **TACK** will go LOW to indicate that test mode has been exited.

It is important that test mode can be entered and exited cleanly so that diagnostic testing may be performed during system operation.

## 6.5 The AHB Test Interface Controller

The following state diagram illustrates the operation of the TIC.

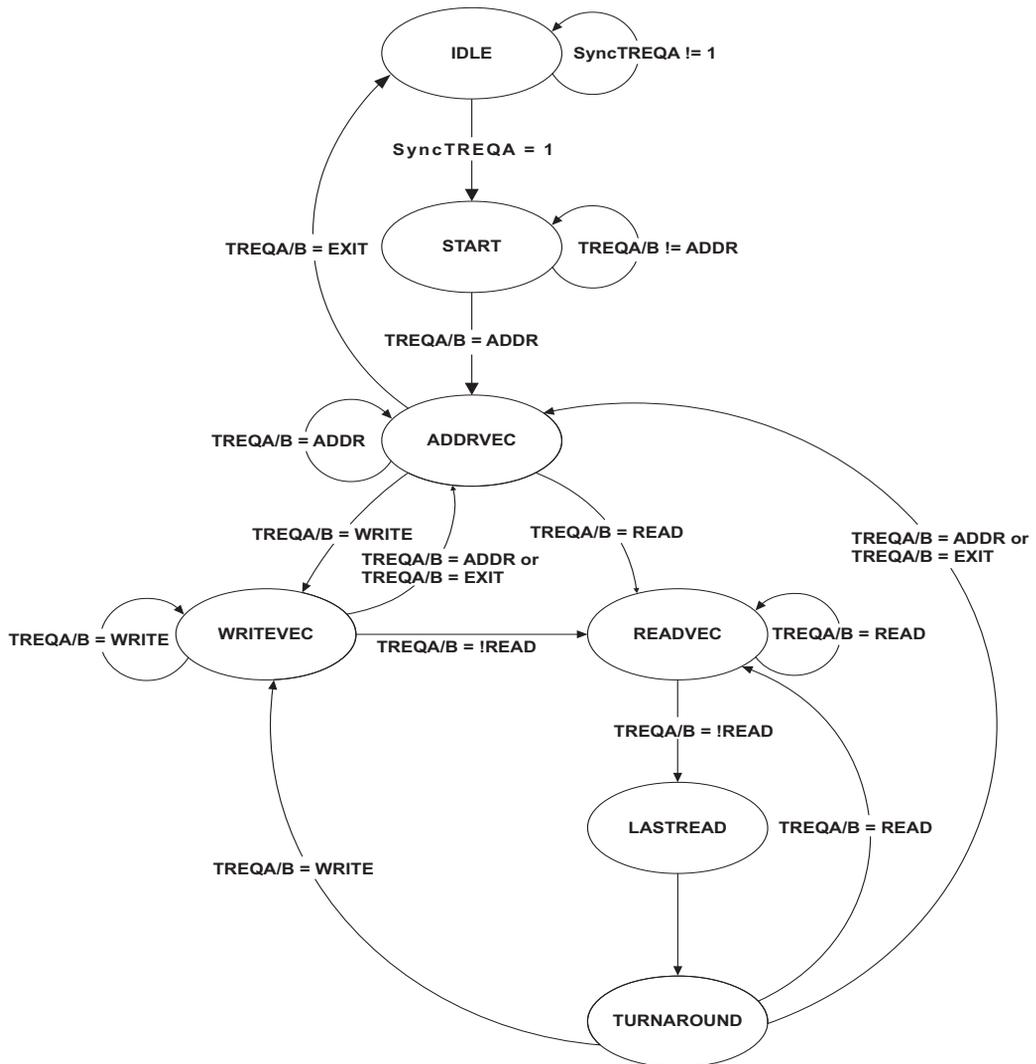


Figure 6-3 Test Interface Controller state diagram

The following points describe the TIC state diagram operation:

- At reset the TIC is in the IDLE state and will not be requesting use of the AHB. When in the IDLE state **TACK** is driven LOW to indicate that the test interface cannot be used.
- The **TACK** signal is used to control all transactions around the state machine, except for the transition from IDLE to START. In all other cases the state machine remains in the same state if the **TACK** signal is low.
- The **TREQA** signal is used to move from the IDLE state to the START state. This has been changed from the previous specification, which required **TREQA** to be high and **TREQB** to be low, and has the advantage that it is possible to use just **TREQA** to move from normal operation into test mode.
- In some system implementations it will be necessary to switch from an internal clock source to an external clock **TCLK** which is used during test mode. When **TREQA** first goes high this can be used as an indication that the clock source should be changed and a return signal that indicates when the clock switch has occurred successfully can be used to prevent the move into the START state until the test clock is in use.
- If clock switching is being used then it is possible that **TREQA** is asynchronous to the on-chip clock before test mode is entered and therefore a synchronizer is used to generate a synchronized version of **TREQA** to control the movement from the IDLE state to the START state.
- The START state is used to ensure that the first vector applied is an address vector to prevent read and write vectors occurring before the address has been initialized. The START state is only exited when **TREQA/B** indicate an address vector and the following state is ADDRVEC.
- In the ADDRVEC state the TIC registers the address on the **TBUS**. The ADDRVEC state is used for both address and control vectors, so additional logic is required to determine whether the value on **TBUS** should be considered as an address or as a control vector. If the previous cycle was an address vector and the following cycle (as indicated by **TREQA/B**) is not an address vector then the current cycle is a control vector.
- It is possible to stay in the ADDRVEC state for a number of cycles, but usually an address vector will be followed by either read or write transfers.
- If a write transfer is being performed the TIC moves into the WRITEVEC state at the same time that it initiates the transfer on the bus and multiple write transfers can be performed by remaining in the WRITEVEC state. Usually the WRITEVEC will be followed by an address vector, however it is also possible to move directly to read transfer by moving to the READVEC state.

- When a read, or a burst of reads is performed the TIC enters the READVEC state. This state indicates that the TIC is starting a read transfer on the bus and it is not until the following cycle that the read data will appear. When the READVEC state is first entered the **TBUS** will be tristate, but will become driven for further cycles in the READVEC state.
- All read vectors must be followed by two turnaround vectors. For the first of these cycles the TIC will move into the LASTREAD state, during which the last read of the transfer will complete and will be driven out on to the external **TBUS**. During the LASTREAD state no internal transfers will be started and the TIC will perform **IDLE** transfers on the bus.
- Following the LASTREAD state the TIC moves into the TURNAROUND state, during which time the external **TBUS** will be tristate. The TURNAROUND state will usually be followed by an address vector, but it is also possible to go immediately to a write vector or another read.
- The usual method to exit from test is to return to the ADDRVEC state and then set **TREQA/TREQB** both **LOW** to return to **IDLE** and effectively exit from test. In fact, at any point the test mode can be exited by setting both **TREQA** and **TREQB** **LOW** and eventually this will cause the TIC to exit from test.

---

**Note**

---

When applying TIC vectors it is theoretically possible to assert the **HLOCK** output and then exit from the test. If this happens and then the TIC is granted the bus under normal operation it will effectively lock up the bus. No protection is provided within the TIC to prevent this occurrence.

---

### 6.5.1 Control vector

A control vector is included within the TIC to determine the types of transfer it can perform. The control vector is used to set the values of **HSIZE**, **HPROT** and **HLOCK**.

The default TIC bus master operation when entering test mode is:

- 32-bit transfer width - **HSIZE[1:0]** signifies word transfer
- privileged system access - **HPROT[3:0]** signifies privileged data access, uncacheable and unbufferable.

Bit 0 of the control vector is used to indicate if the control vector is valid. Thus, if a control vector is applied with bit 0 **LOW**, the vector will be ignored and will not update the control information. This mechanism allows address vectors which have bit 0 **LOW** to be applied for many cycles without updating the control information.

Although the default settings will be sufficient for testing many embedded system designs, the control vector can be used both to change the control signals of the transfer and also to determine whether the TIC should generate fixed addresses or incrementing addresses.

Table 6-3 defines the bit positions of the control vector. The control vector bit definitions are designed to be backwards compatible with earlier versions of the TIC and therefore not all of the control bits are in obvious positions.

**Table 6-3 Control vector bit definitions**

<b>Bit position</b>	<b>Description</b>
0	Control vector valid
1	Reserved
2	<b>HSIZE[0]</b>
3	<b>HSIZE[1]</b>
4	<b>HLOCK</b>
5	<b>HPROT[0]</b>
6	<b>HPROT[1]</b>
7	Address increment enable
8	Reserved
9	<b>HPROT[2]</b>
10	<b>HPROT[3]</b>

There is no mechanism to control the types of burst that the TIC can perform and only incrementing bursts of an undefined length are supported. The TIC only supports 8-bit, 16-bit and 32-bit transfers and therefore **HSIZE[2]** cannot be altered and will always be low.

In order to support burst accesses using the test interface the Test Interface Controller may support incrementing of the bus address. The TIC increments 8 address bits and the address range that can be covered by this incremter is dependent on the size of the transfers being performed.

The control vector provides a mechanism to enable and disable the address incrementer within the TIC. This allows burst accesses to incremental addresses, as would be used for testing internal RAM. Alternatively, the address increment can be disabled such that successive accesses of a burst occur to the same address, as would be required to continually read from a single peripheral register.

If **HSIZE[1:0]** is changed dynamically then any address incrementer support for burst-mode accesses must be able to support increment by byte, halfword and word offsets, so adaptive address incrementer logic is required.

The address incrementer is disabled by default and must be enabled using a control vector prior to use.

———— **Note** —————

The control vector is primarily used to change signals which have the same timing as the address bus. However the control vector also allows the lock signal to be changed, which is actually required before the locked transfer commences. If the **HLOCK** signal is used during testing it should be set a transfer before it is required. This difference in timing on the **HLOCK** signal may in some cases cause an additional transfer to be locked both before and after the sequence that should in fact be locked.

---

## 6.6 Example AMBA AHB test sequences

Example AHB test sequences are described under the following headings:

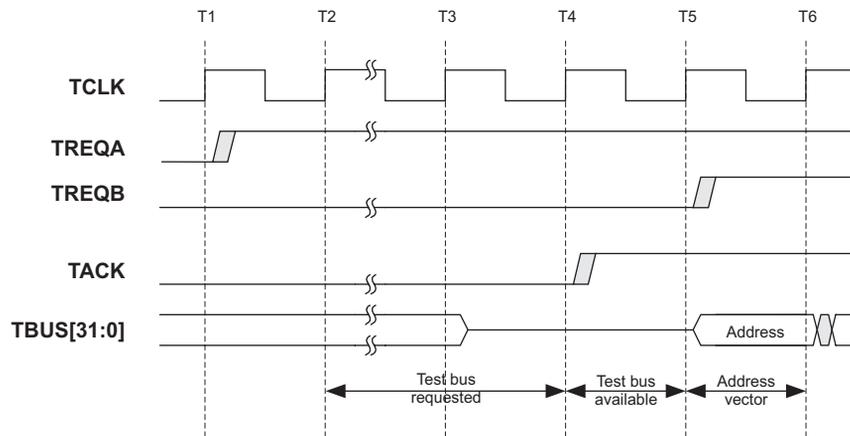
- *Entering test mode*
- *Write test vectors* on page 6-19
- *Read transfers* on page 6-20
- *Control vector* on page 6-21
- *Burst vectors* on page 6-22
- *Read-to-write and write-to-read* on page 6-23
- *Exiting test mode* on page 6-24.

### 6.6.1 Entering test mode

In normal operating mode **TREQA** will be LOW, indicating that test access is not required and the test bus will be used as required for normal operation, which will usually be part of the external bus interface. Entering test mode allows test vectors to be applied externally that will cause transfers on the internal bus.

The following sequence, as illustrated in Figure 6-4, is required in order to enter test mode:

1. **TREQA** is asserted to request test bus access.
2. Test mode is entered when the TIC has been granted the internal bus and this is indicated by the assertion of the **TACK** signal.
3. At this point **TCLK** will become the source of the internal **HCLK** signal.
4. When test mode has been entered **TREQB** is asserted to initiate an address vector.
5. The TIC will not perform any internal transfers until a valid address vector has been applied.



**Figure 6-4 Test start sequence**

A synchronous tester is not expected to poll **TACK** for the bus.

Normally the **TREQA** signal is asserted for a minimum number of cycles to guarantee access to the bus (completion of the longest wait-state peripheral access or the maximum number of cycles for all bus masters to have completed their current instruction).

## 6.6.2 Write test vectors

Figure 6-5 shows the sequence of events when applying a set of write test vectors. Initially an address vector is applied and this is followed by a write test vector.

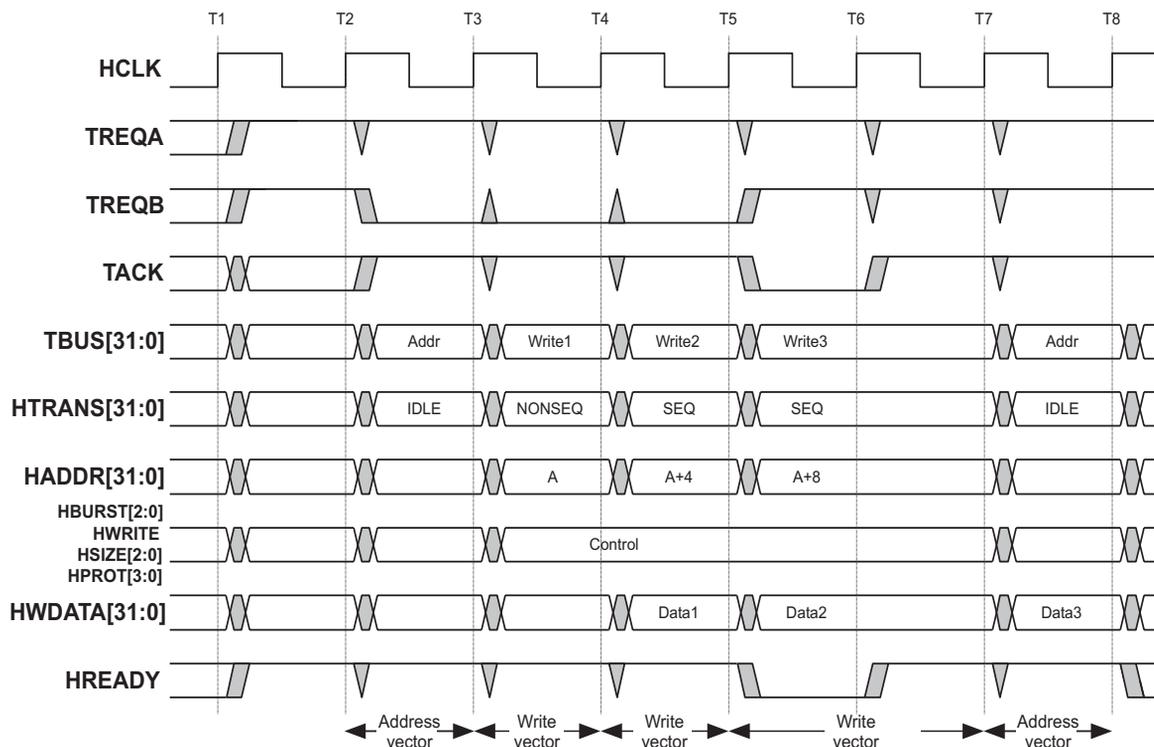


Figure 6-5 Write test vector

The following points apply when writing test vectors:

- The **TREQA** and **TREQB** signals are pipelined and are used to indicate what type of vector will be applied in the following cycle. Figure 6-5 shows an example of a number of write transfers being performed.
- The TIC samples the address and **TREQA/B** signals at time T3. Following this it can initiate the appropriate transfer on the AHB.
- In the following cycle the write data is driven on to the **TBUS** and it is then sampled on the following clock edge, T4, and driven on to the internal bus.
- If the internal transfer is not able to complete then the **TACK** signal is driven low and this indicates that the external test vector must be applied for another cycle.

### 6.6.3 Read transfers

Read transfers are more complex because they require the **TBUS** to be driven in the opposite direction and therefore additional cycles are required to prevent bus clash when changing between different drivers of **TBUS**. Figure 6-6 shows a typical test sequence for reads.

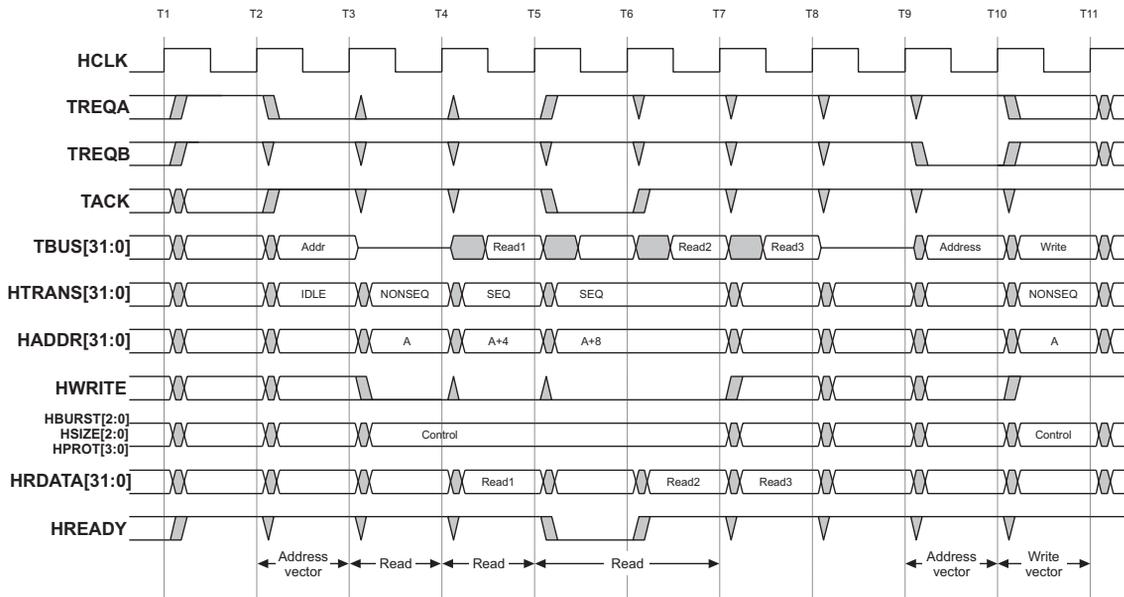


Figure 6-6 Read test vector

The following points apply when reading test vectors:

- The **TREQA** and **TREQB** signals are used in the same way as for a write transfer. Initially, **TREQA/B** are used to apply an address vector, in the following cycle they are used to indicate that a read transfer is required. For the first cycle of a read the **TBUS** must be tristate, which ensures that the external equipment driving **TBUS** has an entire cycle to tristate its buffers before the TIC will enable the on-chip buffers to drive out the read data.
- At the end of a burst of reads it is also necessary to allow time for bus turnaround. In this case the TIC must turn off the internal buffers and an entire cycle is allowed before the external test equipment starts to drive.
- The end of a burst of reads is indicated by both **TREQA** and **TREQB** being HIGH, as for an address vector. In fact they must indicate address vector for two cycles, which allows for both the turnaround cycle at the start of the burst and also the turnaround cycle at the end of the burst.

### 6.6.4 Control vector

The operation of the TIC may be modified by the use of a control vector. Whenever more than one address vector is applied in succession then the last vector is considered to be a control vector and is not latched as the address. Bit 0 of the control vector is used to determine whether or not the control vector should be considered valid, which allows multiple address vectors to be applied without changing the control information,

Figure 6-7 shows the process of inserting a control vector.

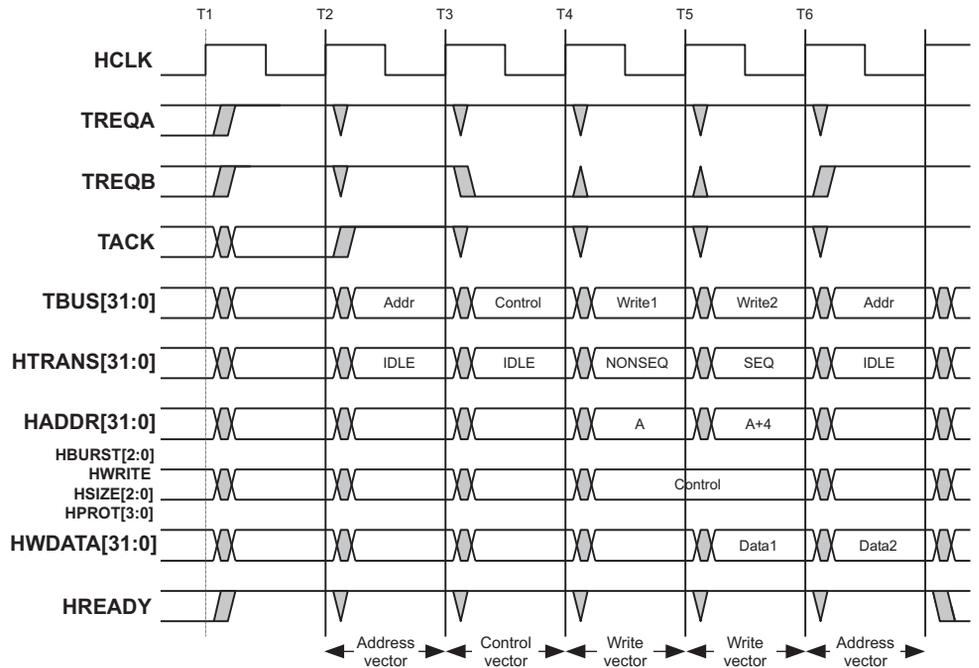


Figure 6-7 Control vector

At time T4 the TIC can determine that the **TBUS** contains a control vector. This is because the previous cycle was an address vector and **TREQA/B** are indicating that the following cycle is either a read or a write and therefore the current cycle must be a control vector.

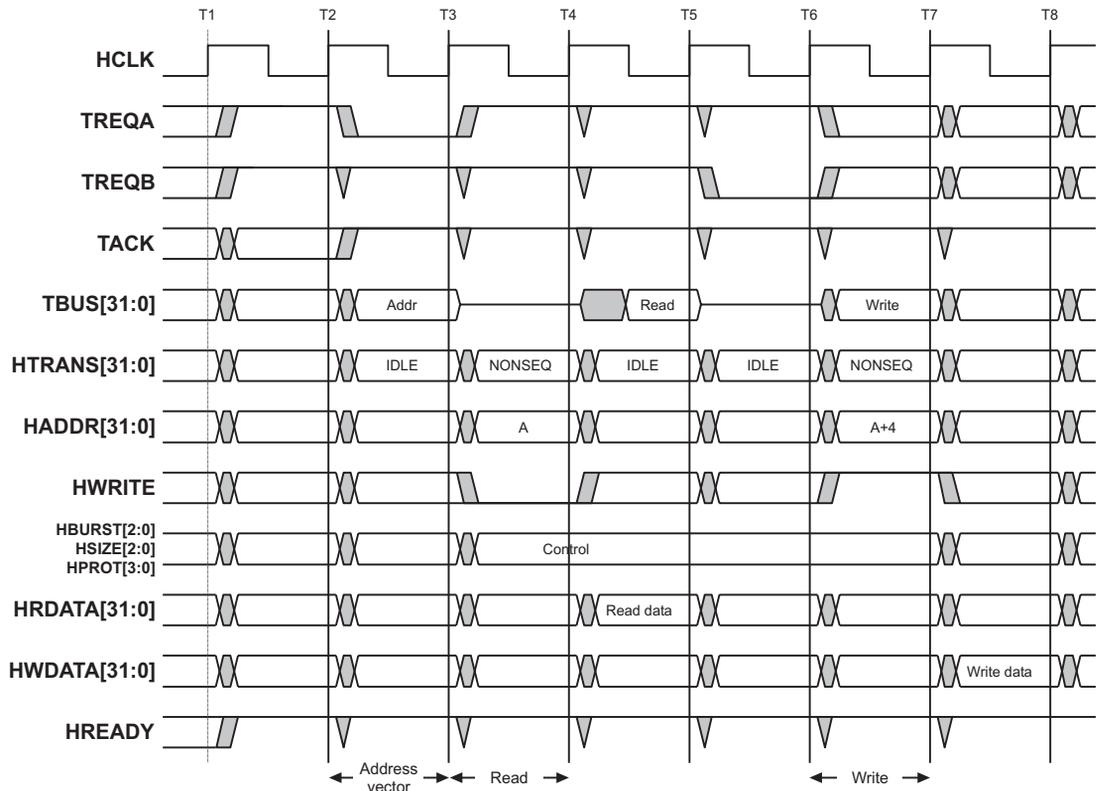
### 6.6.5 Burst vectors

The examples of read and write transfers in Figure 6-5 on page 6-19 and Figure 6-6 on page 6-20 also show how additional transfers can be used to form burst transfers on the bus. The TIC has limited capabilities for burst transfers and can only perform undefined-length incrementing bursts.

The TIC contains an 8-bit incrementer and if an attempt is made to perform a burst which crosses the incrementer boundary then the address will wrap and the TIC will signal the transfer as NONSEQUENTIAL. The exact boundary at which this will occur is dependent on the size of the transfer. For word transfers the incrementer will overflow at 1kB boundaries, for halfword transfers it will overflow at 512-byte boundaries and for byte transfers the overflow will occur at 256-byte boundaries.

### 6.6.6 Read-to-write and write-to-read

It is possible to switch between read transfers and write transfers without applying a new address vector. Usually this would be done with the address incremter disabled, so that both the read transfers and the write transfers would be to the same address. It is also possible to do this with the incremter enabled if the test circumstances require it.



**Figure 6-8** Read then write transfers

When moving from a read transfer to a write transfer it is also necessary to allow the two cycles for bus handover and therefore **TREQA** and **TREQB** should signal address vector for two cycles after the read. This will not cause the address to be changed unless it is followed by a third address vector. Figure 6-8 illustrates the sequence of events.

### 6.6.7 Exiting test mode

Test mode is exited using the following sequence:

1. Apply a single cycle of address vector, which causes an IDLE cycle internally, which ensures any internal transfers have been completed.
2. **TREQA** and **TREQB** are both driven LOW to indicate that test mode is to be exited.
3. When the test interface has been configured for normal system operation **TACK** will go LOW to indicate that test mode has been exited.

It is important that test mode can be entered and exited cleanly so that the TIC can also be used for diagnostic test during system operation, as well as production testing.

## 6.7 The ASB test interface controller

Figure 6-9 shows the ASB test interface controller state diagram.

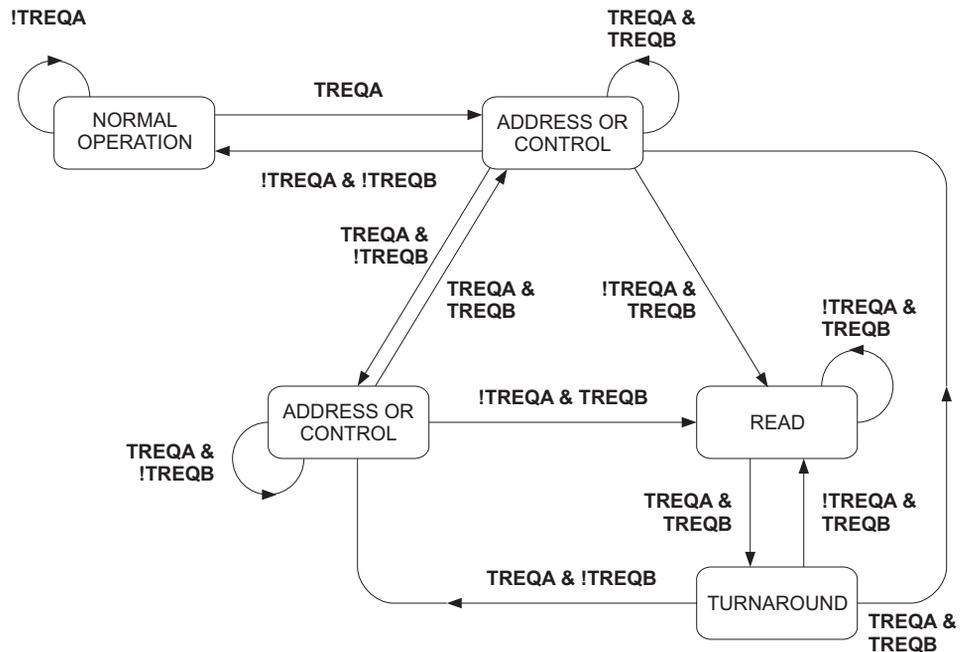


Figure 6-9 Test interface controller state diagram

**TREQA** and **TREQB** are sampled on the falling edge of **TCLK** when **TACK** is HIGH, except in the **NORMAL OPERATION** state where **TREQA** is used asynchronously to transition into the **ADDRESS OR CONTROL** state. The reset state is **NORMAL OPERATION**.

### 6.7.1 Control vector bit definitions

A control vector is included within the TIC to determine the types of transfer it can perform. The control vector is used to set the values of **BSIZE**, **BPROT** and **BLOK** and to control address incrementing.

Byte 0 of the control packet is used to define the access that will occur on the internal system bus. Byte 1 of the control packet is reserved for clock control and debug.

Table 6-4 shows the control vector bit assignments.

**Table 6-4 Control vector bit definitions**

<b>Bit position</b>	<b>Description</b>
0	Control vector valid
1	Reserved
2	<b>BSIZE[0]</b>
3	<b>BSIZE[1]</b>
4	<b>BLOK</b>
5	<b>BPROT[0]</b>
6	<b>BPROT[1]</b>
7	Address increment enable

## 6.8 Example AMBA ASB test sequences

Example ASB test sequences are described under the following headings:

- *Entering test mode*
- *Address vectors* on page 6-28
- *Control vectors* on page 6-29
- *Write test vectors* on page 6-31
- *Changing burst direction* on page 6-36
- *Exiting test mode* on page 6-37.

### 6.8.1 Entering test mode

Test mode is entered, as shown in Figure 6-10, using the following sequence:

1. **TREQA** is asserted to request test bus access.
2. Test mode is entered when the TIC has been granted the internal bus and this is indicated by the assertion of the **TACK** signal.
3. At this point **TCLK** will become the source of the internal **BCLK** signal.
4. When test mode has been entered **TREQB** is asserted to initiate an address vector.

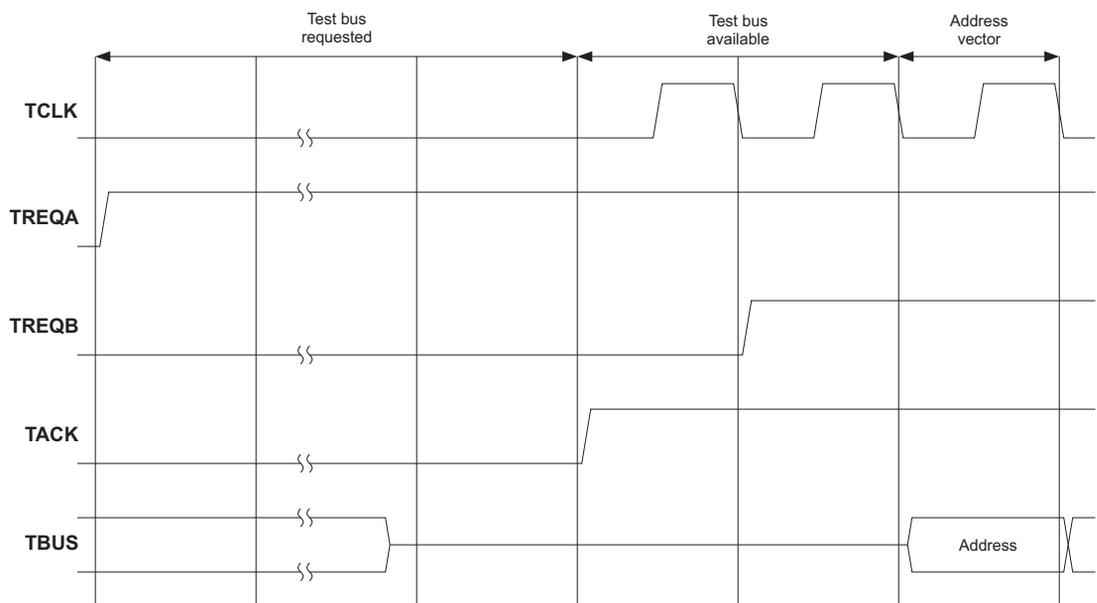


Figure 6-10 Test start sequence

## 6.8.2 Address vectors

An address vector must be applied before a read or write operation can occur. Figure 6-11 shows an example of a single address vector followed by a write vector, the following sequence occurs:

1. **TREQA** and **TREQB** are both asserted HIGH to indicate an address vector next cycle.
2. In the next cycle the address is applied, while **TREQA** and **TREQB** change to indicate the type of test vector that will follow. During this cycle the address appears on the address bus.
3. In the next cycle the write (or read) vector is applied.

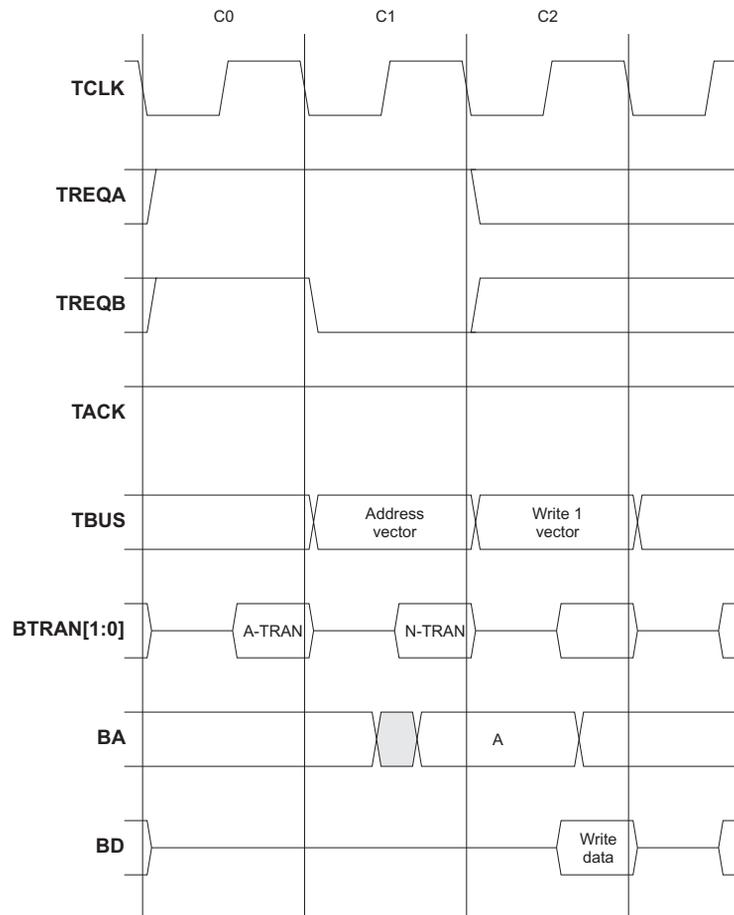


Figure 6-11 Address vector

### 6.8.3 Control vectors

A control vector must always follow an address vector. Figure 6-12 shows an address and control vector sequence followed by a write vector. The following sequence occurs:

1. **TREQA** and **TREQB** both remain HIGH after the address vector has ended to indicate a control vector next cycle.
2. In the next cycle control information is applied to **TBUS[31:0]**, while **TREQA** and **TREQB** change to reflect the type of test vector that will follow. During this cycle any internal signals, which have been affected by the control vector, will change.

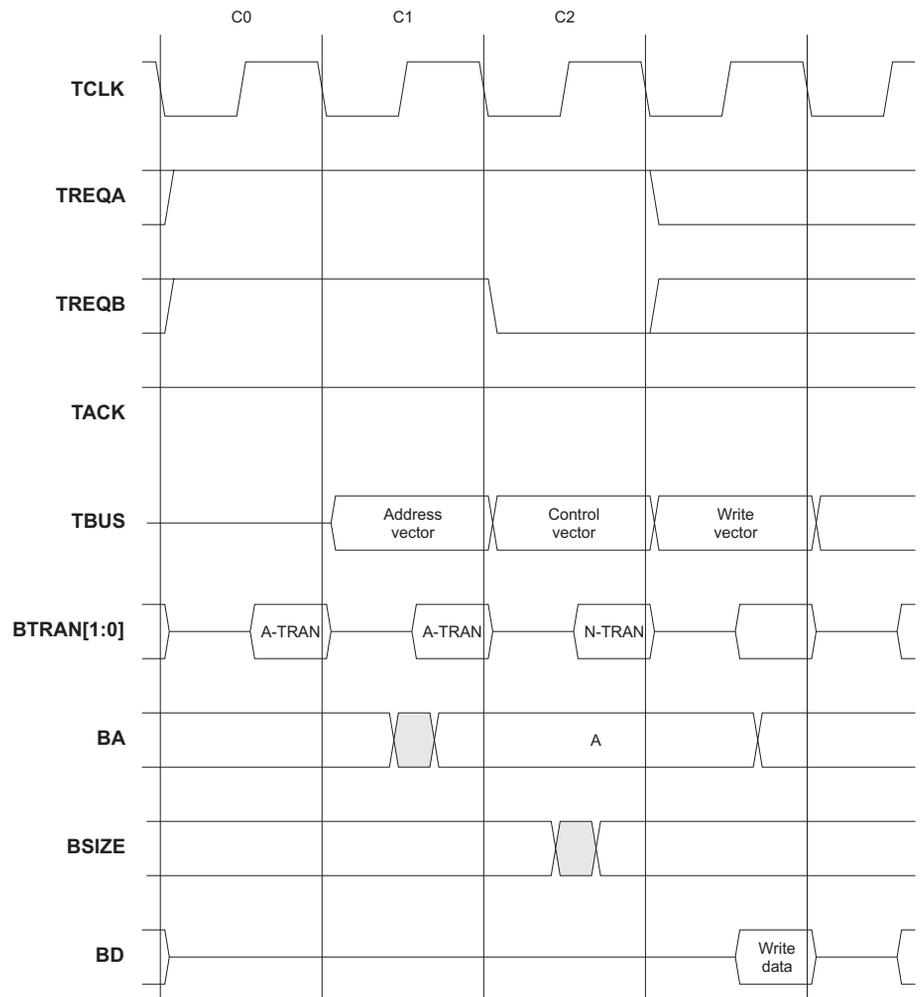


Figure 6-12 Control vector

Figure 6-13 shows an example of a transfer following an invalid control vector. The TIC performs a SEQUENTIAL transfer on the internal bus because the control signals have not changed.

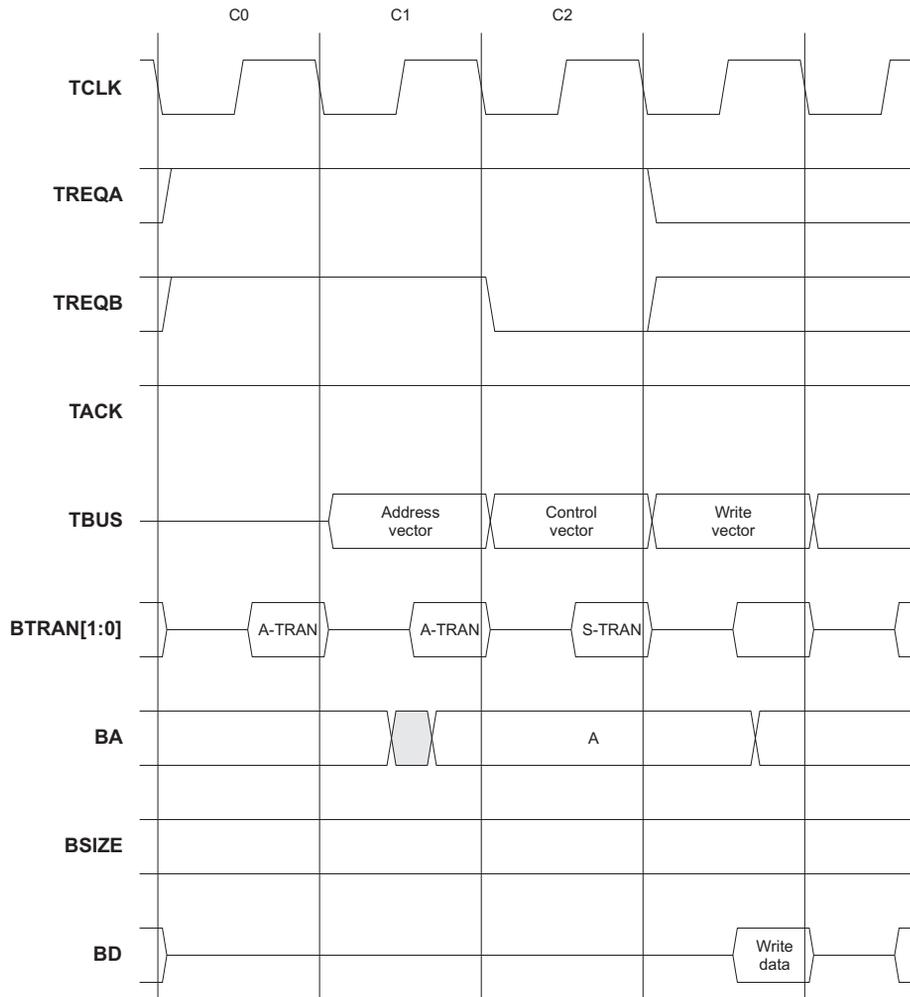


Figure 6-13 Invalid control vector

### 6.8.4 Write test vectors

Figure 6-14 shows an example of a single write vector following a single address vector.

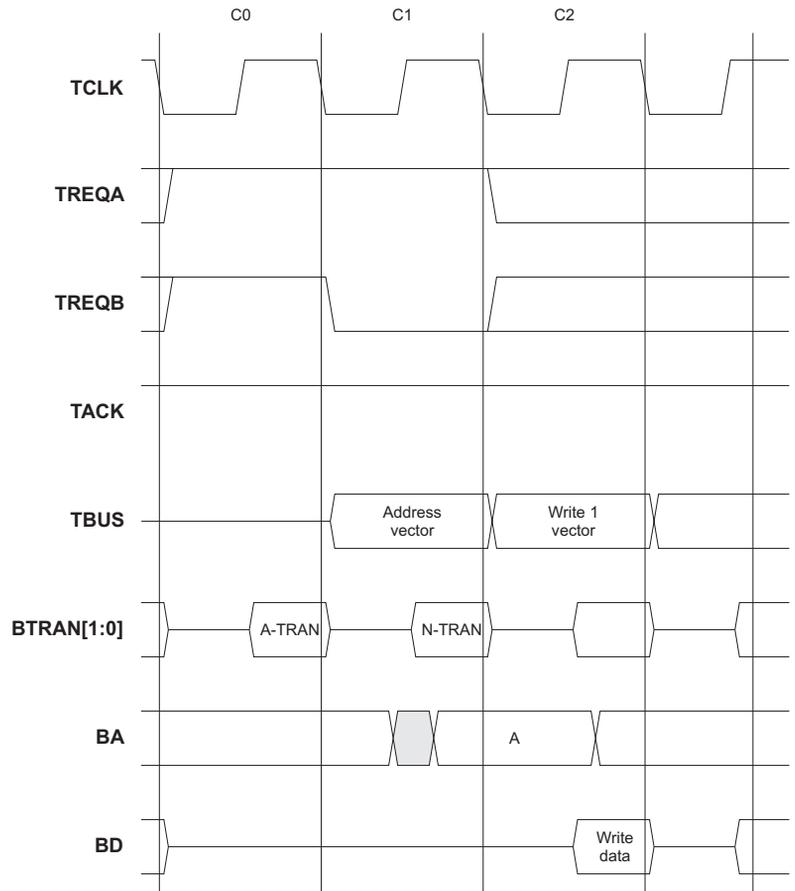


Figure 6-14 Write test vectors

Figure 6-15 shows an example of extended write vectors following a single address vector.

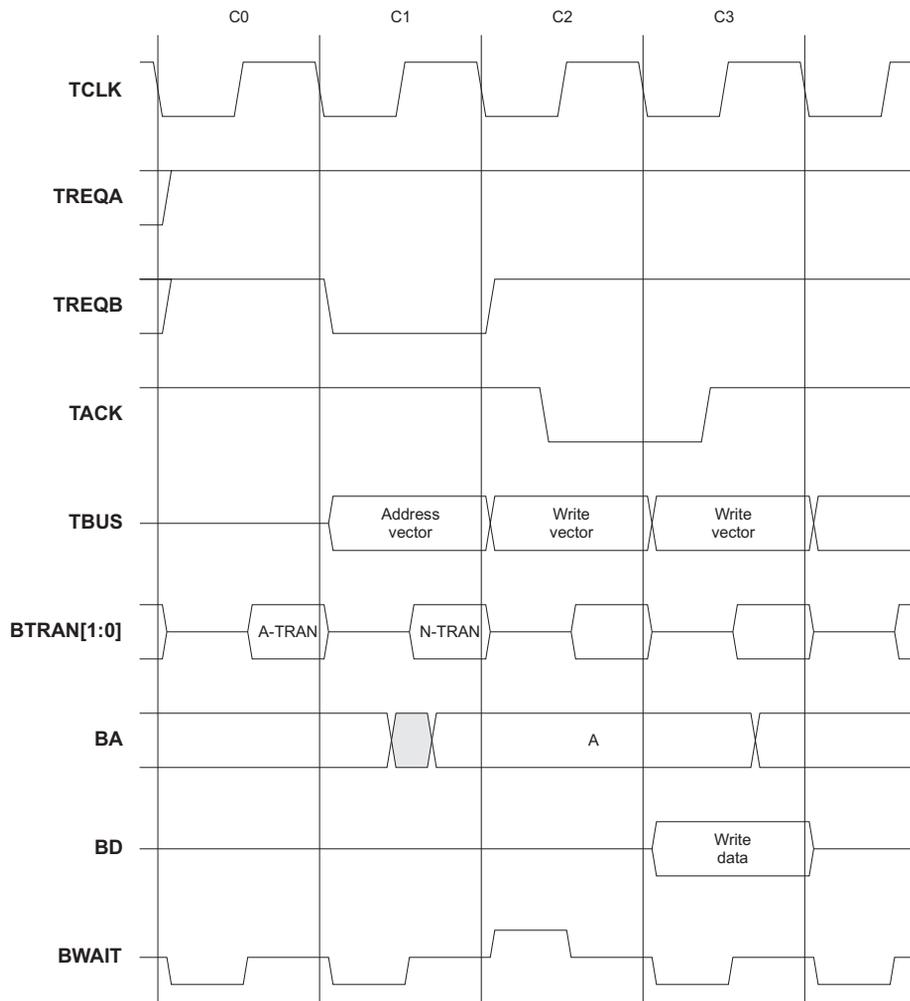
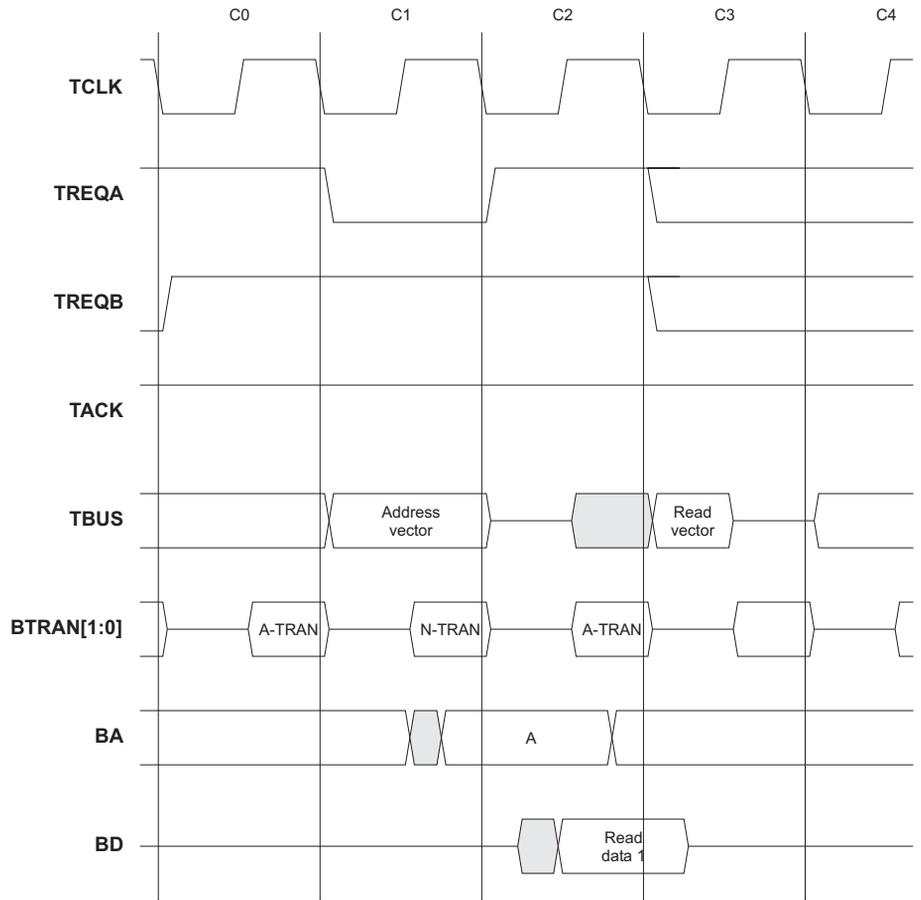


Figure 6-15 Extended write test vectors

Figure 6-16 shows an example of a single address vector, followed by a single read vector and terminated with a single turnaround vector.



**Figure 6-16 Read test vector**

Figure 6-17 shows SEQUENTIAL transfers to non-incrementing addresses.

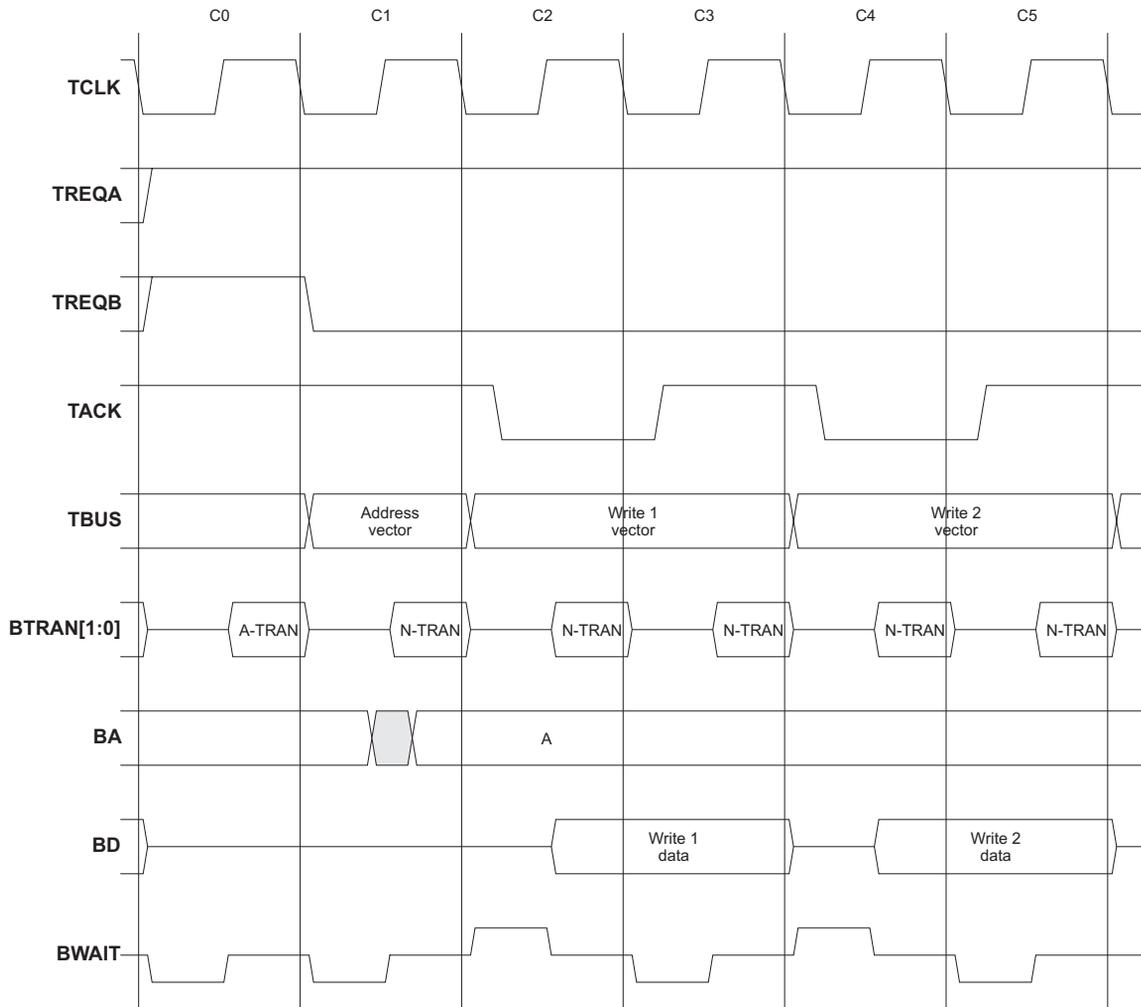


Figure 6-17 Burst write vectors with increment disabled

Figure 6-18 shows SEQUENTIAL transfers to incrementing addresses.

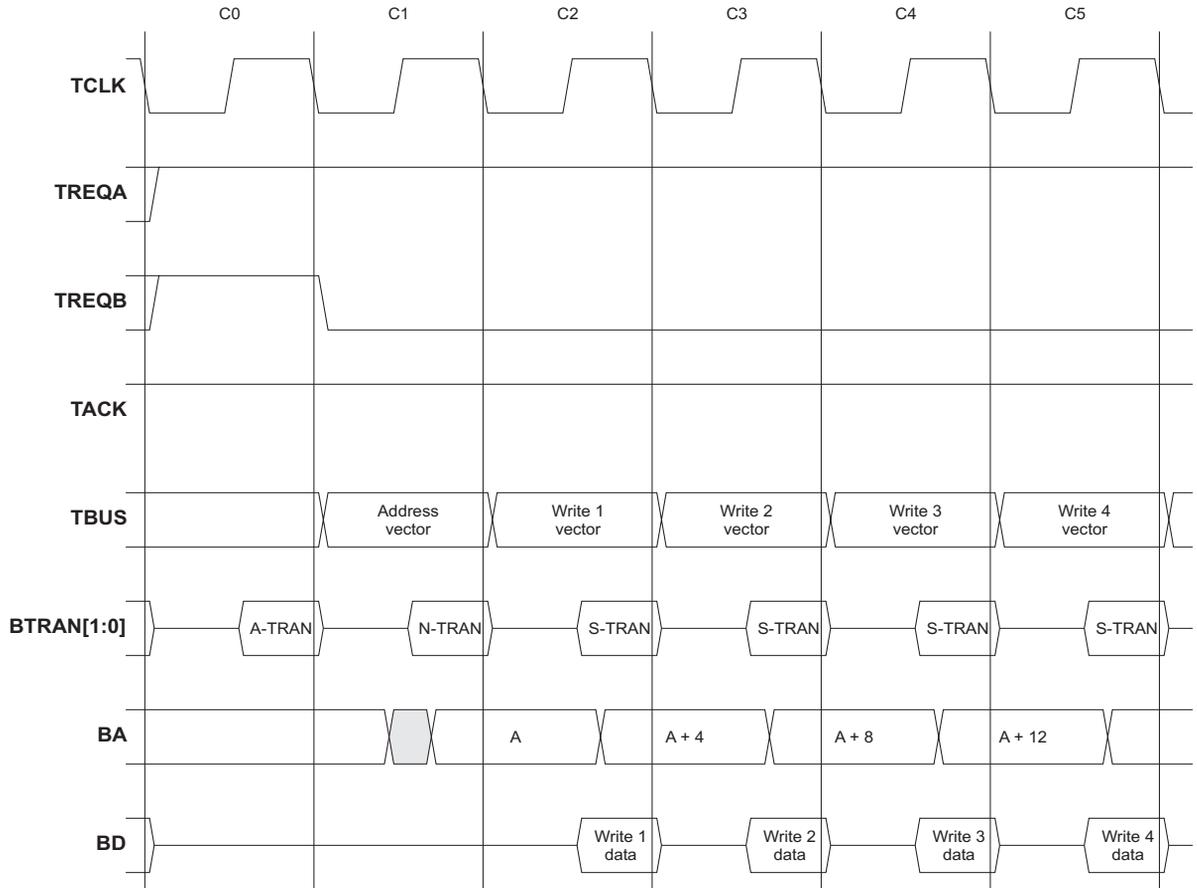


Figure 6-18 Burst write vectors with increment enabled

### 6.8.5 Changing burst direction

Figure 6-19 below shows a burst changing direction from read to write.

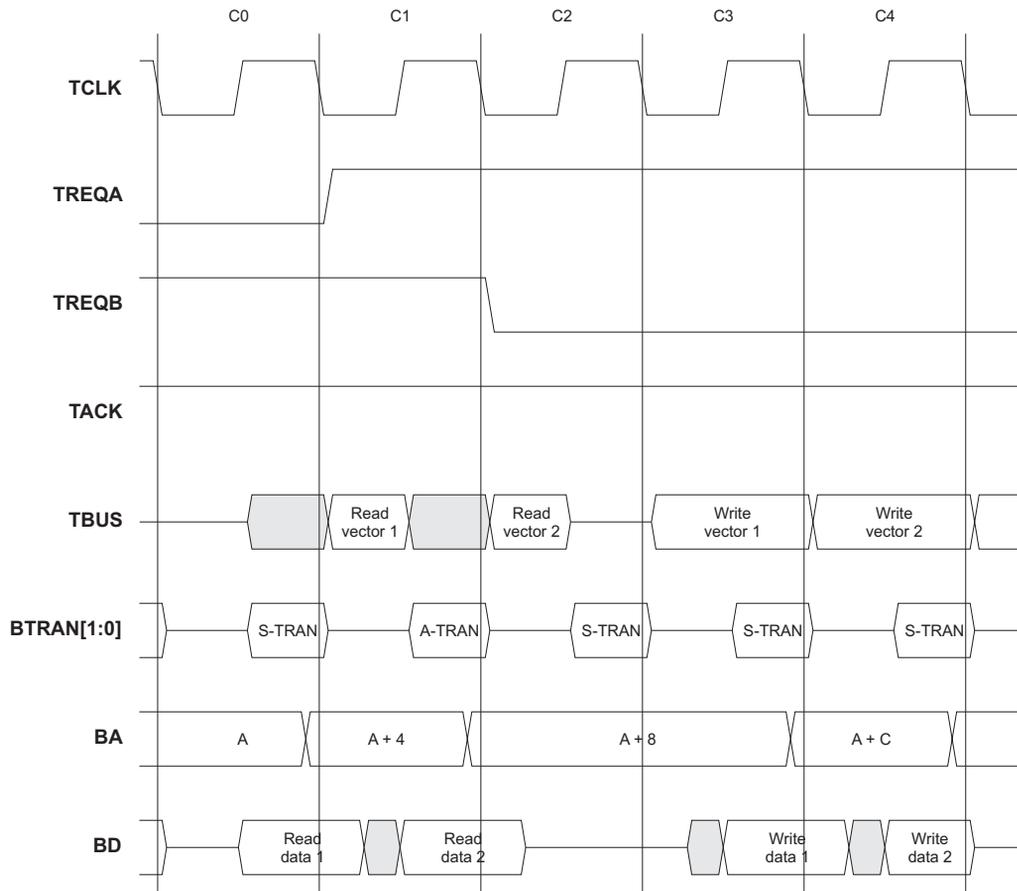


Figure 6-19 Changing burst direction

### 6.8.6 Exiting test mode

Figure 6-20 shows an exit from test mode.

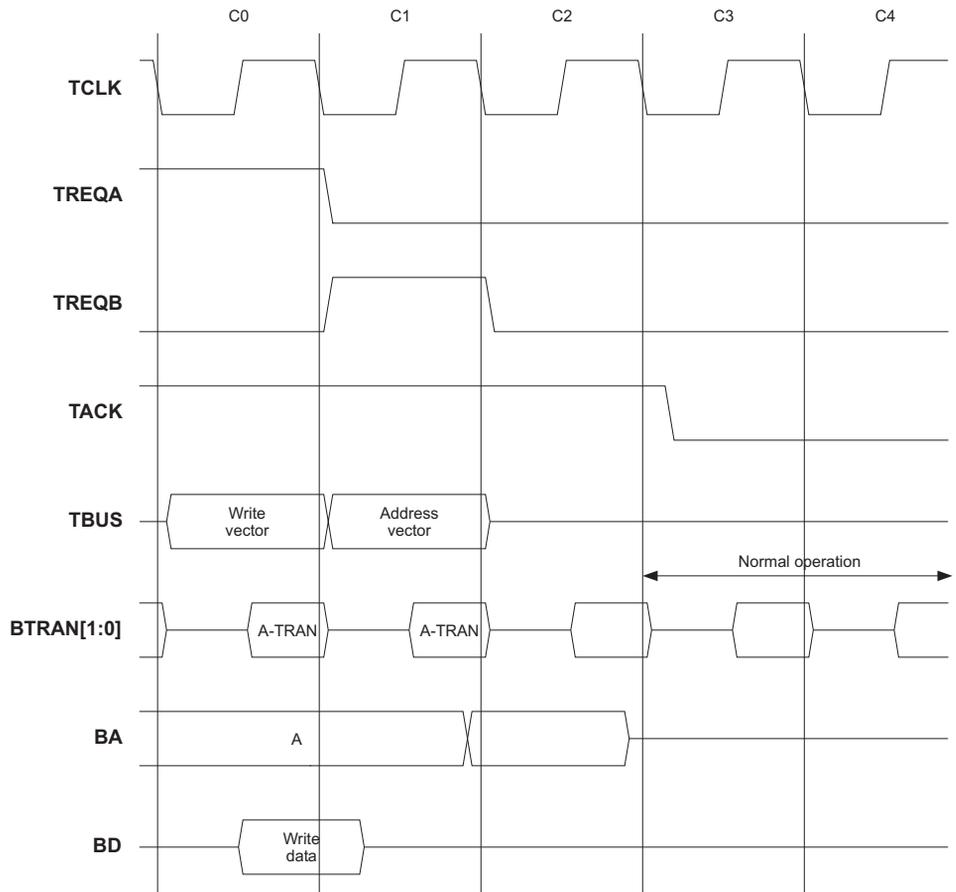


Figure 6-20 Exiting test mode



# Index

The items in this index are listed in alphabetic order. The references given are to page numbers.

## A

- Active state 4-56
- Address and control signals
  - ASB 4-27
  - timing 4-29
- Address bus
  - AHB 2-3
  - APB 2-8
  - ASB 2-6, 4-27
- Address decoding
  - AHB 3-19
  - ASB 4-14
- Address vectors 6-8, 6-28
- Address-only transfers 4-10, 4-59
- AGNTx 2-6, 4-44
- AHB 3-1
  - arbiter 1-7
  - arbitration signals 2-5
  - decoder 1-8
  - introduction to 1-7
  - master 1-7
  - operation 3-5
  - signal list 2-3
  - signal prefixes 2-2
  - slave 1-7
- AHB/ASB or APB, when to use 1-13
- AMBA signal names 2-2
- AMBA system, typical 1-4
- AMBA test interface 6-2
- AMBA test methodology 6-1
- APB 5-1
  - address bus 2-8
  - bridge 5-8
  - bridge interface diagram 5-8
  - bridge transfer 5-9
  - components 5-7
  - in a typical AMBA system 5-3
  - introduction to 1-10
  - read data bus 2-8
  - read transfers 5-6
  - select 2-8
  - signal list 2-8
  - signal prefixes 2-2
  - slave 5-11
  - slave interface diagram 5-11
  - strobe 2-8
  - timing parameters 5-7, 5-10
  - transfer direction 2-8
  - write data bus 2-8
  - write transfers 5-5
- Arbiter
  - AHB 1-7
  - ASB 1-9, 4-20, 4-71
- Arbitration and reset signals 4-60

- Arbitration signals
  - AHB 2-5
  - ASB 4-44
- Arbitration, AHB 3-28
- AREQx 2-6, 4-44
- ASB 4-1
  - and APB 4-3
  - arbiter 1-9, 4-20, 4-71
  - arbiter interface diagram 4-71
  - arbiter timing parameters 4-73
  - bus master 4-52
  - bus master interface diagram 4-52
  - bus slave 4-47
  - bus slave interface 4-47
  - components 4-46
  - decoder 1-9, 4-63
  - decoder interface diagram 4-64
  - decoder timing diagrams 4-68
  - decoder timing parameters 4-69
  - description 4-4
  - introduction to 1-9
  - master 1-9
  - signal description 4-25
  - signal list 2-6
  - signal prefixes 2-2
  - slave 1-9
  - slave bus interface state machine 4-48
  - test sequence 6-27
  - transfers 4-6
- B**
- BA 2-6, 4-27
- Back to back transfers 5-18
- Backbone bus 1-4
- Basic transfers 3-6
- BCLK 2-6, 4-25
- BD 2-6, 4-40
- BERROR 2-6, 4-36
- BLAST 2-6, 4-36
- BLOK 2-6, 4-45
- BnRES 2-6, 4-23, 4-25
- BPROT 2-7, 4-28
  - encoding 4-28
- BSIZE 2-7, 4-28
  - encoding 4-28
- BTRAN 2-7, 4-26
  - encoding 4-26
  - timing 4-27
- Burst operation 1-6, 3-11
- Burst type, AHB 2-3
- Burst vectors 6-10, 6-22
- Bursts
  - incrementing 3-12
  - of read transfers 5-15
  - of write transfers 5-17
  - undefined-length wrapping 3-12
- Bus
  - backbone 1-4
  - choosing 1-12
  - peripheral 1-12
- Bus clock
  - AHB 2-3
  - APB 2-8
  - ASB 2-6
- Bus cycle 1-6
- Bus grant 4-44
  - AHB 2-5
  - ASB 2-6
- Bus interface state machine, ASB 4-54
- Bus lock 4-45
- Bus master
  - ASB 4-52
  - default 4-22
  - granted state machine 4-53
  - handover 3-29, 4-20
  - interface, ASB 4-52
  - main state machine 4-55
  - timing diagrams, ASB 4-57
  - timing parameters, ASB 4-60
- Bus request 4-44
  - AHB 2-5, 3-28
  - ASB 2-6
- Bus retract 4-36
- Bus slave interface, ASB 4-47
- Bus transfer 1-6
- Busidle state 4-56
- BWAIT 2-7, 4-36
- BWRITE 2-7, 4-27
  - encoding 4-27
- C**
- Choosing the right bus 1-12
- Clock, ASB 4-25
- Control signals 3-17
- Control vectors 6-9, 6-14, 6-21, 6-29
  - bit definitions 6-25
- D**
- Data bus
  - AHB 3-25
  - ASB 2-6, 4-40
- Deadlock 3-37
- Decode cycles 4-33, 4-65
- Decoder
  - AHB 1-8
  - ASB 1-9, 4-63
  - state machine 4-65, 4-67
  - with decode cycles 4-33, 4-65
  - without decode cycles 4-34, 4-67
- Default bus master 4-22
- Direction of transfer
  - APB 2-8
  - ASB 2-7
- Done response 4-16, 4-48
- DSEL 4-33
- DSELx 2-7
- E**
- Early burst termination 3-12
- Electrical characteristics 1-14
- Enable state 5-5
- Enter test mode 6-8, 6-17, 6-27
- Error response 4-16, 4-36, 4-48
  - ASB 2-6
- Exit from reset 4-23
- Exit test mode 6-11, 6-24, 6-37
- External test interface 6-4
- G**
- Grant signal, AHB 3-28
- Granted state machine 4-53
- H**
- HADDR 2-3
- Handover 3-29
- Handover state 4-56
- Handover, bus master 4-20
- HBURST 2-3
- HBUSREQx 2-5, 3-28
- HCLK 2-3
- HGRANTx 2-5, 3-28
- HLOCKx 2-5, 3-28
- HMASTER 2-5
- HMASTLOCK 2-5
- Hold state 4-56
- HPROT 2-3, 3-17
- HRDATA 2-4, 3-25
- HREADY 2-4, 3-20
- HRESETn 2-3
- HRESP 2-4, 3-20
- HSELx 2-4

HSIZE 2-3, 3-17  
 HSPLITx 2-5  
 HTRANS 2-3  
 HWDATA 2-4, 3-25  
 HWRITE 2-3, 3-17

**I**

Idle state 4-56, 5-4  
 Incremental addressing 6-7  
 Incrementing burst 3-12  
 Interfacing  
   APB to AHB 5-14  
   APB to ASB 5-20  
   revD peripherals 5-22

**L**

Last response 4-17, 4-36, 4-48  
   ASB 2-6  
 Lock signal, AHB 3-28  
 Locked sequence, AHB 2-5  
 Locked transfers  
   AHB 2-5  
   ASB 2-6, 4-22

**M**

Master  
   AHB 1-7  
   ASB 1-9  
 Master number  
   AHB 2-5  
 Multi-master operation, ASB 4-19  
 Multiple transfers 3-8

**N**

Nonsequential transfers 4-7, 4-57

**P**

PADDR 2-8  
 PCLK 2-8  
 PENABLE 2-8  
 Peripheral bus 1-12  
 Peripheral test harness 6-2  
 PRDATA 2-8  
 PRESETn 2-8  
 Protection control  
   AHB 2-3, 3-17  
   ASB 2-7  
 Protection signals  
   ASB 4-28

PSELx 2-8  
 PWDATA 2-8  
 PWRITE 2-8

**R**

Read data bus  
   AHB 2-4, 3-25  
   APB 2-8  
 Read test vectors 6-10  
 Read transfers 6-20  
   APB 5-6, 5-14  
   burst of 5-15  
   to ASB 5-21  
 Reset 4-25  
   AHB 2-3  
   APB 2-8  
   ASB 2-6  
   exit from 4-23  
 Reset operation, ASB 4-23  
 Response encoding 3-21  
 Retract response 4-17, 4-48  
 Retract state 4-56  
 Retry transfers 3-38  
 Rev D peripherals 5-22

**S**

Select, APB 2-8  
 Sequential transfers 4-8, 4-58  
 Setup state 5-4  
 Signal list  
   AHB 2-3  
   APB 2-8  
   ASB 2-6  
 Signal names  
   AMBA 2-2  
 Signal prefixes  
   AHB 2-2  
   APB 2-2  
   ASB 2-2  
 Size encoding 3-17  
 Size of transfer  
   ASB 2-7  
 Slave  
   AHB 1-7  
   ASB 1-9  
   transfer response 3-20  
 Slave select  
   AHB 2-4  
   ASB 2-7, 4-33  
 Split completion request, AHB 2-5  
 Split transfers 3-35, 3-37  
 State diagram  
   TIC 6-12

State machine  
   ASB slave bus interface 4-48  
   bus interface, ASB 4-54  
   bus master, main 4-55  
   decoder 4-65, 4-67  
 Strobe, APB 2-8

**T**

TACK 6-4  
 TBUS 6-5  
 TCLK 6-5  
 Technology independence 1-14  
 Termination, early burst 3-12  
 Terminology 1-6  
 Test  
   transfer parameters 6-7  
 Test acknowledge 6-4  
 Test bus 6-5  
 Test bus request 6-4  
 Test clock 6-5  
 Test harness 6-2  
 Test Interface Controller  
   ASB 6-25  
   ASB, state diagram 6-25  
 Test Interface Controller (TIC) 6-3, 6-7  
 Test Interface Controller state  
   diagram 6-12  
 Test mode  
   entering 6-8, 6-17, 6-27  
   exiting 6-11, 6-24, 6-37  
 Test sequence 6-17  
   ASB 6-27  
 Test vector types 6-6  
 TIC 6-3, 6-7  
 Timing diagrams  
   APB bridge 5-9  
   APB slave 5-12  
   ASB arbiter 4-72  
   ASB bus slave 4-49  
   ASB decoder 4-68  
 Timing parameters 4-69  
   APB 5-7, 5-10  
   APB slave 5-13  
   ASB 4-46  
   ASB arbiter 4-73  
   ASB bus master 4-60  
   ASB bus slave 4-50  
 Timing specification 1-14  
 Transfer direction  
   AHB 2-3, 3-17  
   APB 2-8  
   ASB 2-7  
 Transfer direction, ASB 4-27

## Index

- Transfer done
  - AHB 2-4, 3-20
- Transfer response 4-47
  - AHB 2-4, 3-20
  - ASB 4-16, 4-35
  - combinations 4-38
  - timing 4-39
- Transfer size
  - AHB 2-3, 3-17
  - ASB 2-7, 4-28
- Transfer type 3-9, 4-26
  - AHB 2-3
  - ASB 2-7
  - encoding 3-9
- Transfers
  - address-only 4-10, 4-59
  - back to back 5-18
  - basic 3-6
  - multiple 3-8
  - nonsequential 4-7, 4-57
  - sequential 4-8, 4-58
  - split 3-35
  - with retry response 3-22
  - with wait states 3-7
- TREQA 6-4
- TREQB 6-4
- Tristate
  - data bus 5-19
  - enable of address and control signals 4-32
- Two-cycle response 3-22
- Type of transfer 3-9
  - ASB 2-7
- Typical AMBA system 1-4, 5-3
  - AHB-based 3-3
  - ASB-based 4-2

## U

- Undefined-length burst 3-16

## W

- Wait response 4-16, 4-36, 4-47
  - ASB 2-7
- Wait states 3-7
- Wrapping burst 3-12
- Write data bus
  - AHB 2-4, 3-25
  - APB 2-8
- Write test vectors 6-9, 6-19, 6-31
- Write transfers
  - APB 5-5, 5-16
  - burst of 5-17
  - from ASB 5-20