Chapter 1
Introduction to the AMBA Buses

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

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).

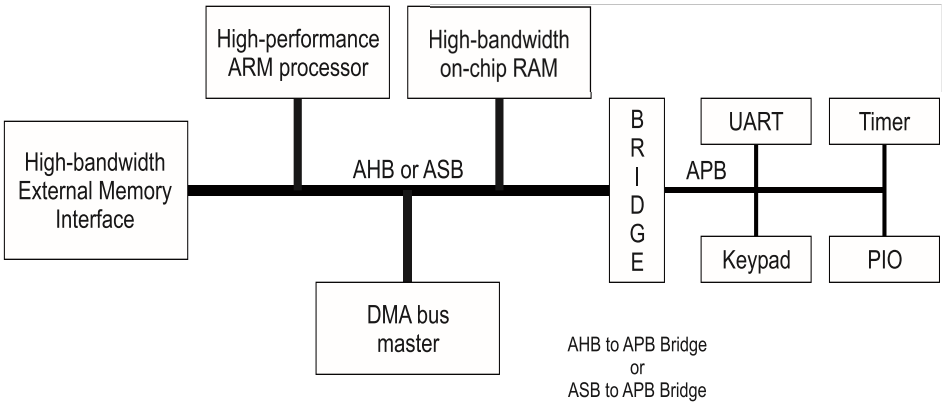

Figure 1-1 A typical AMBA system

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:

AHB master 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.

AHB slave 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.

AHB arbiter 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 *highest priority* or *fair* access can be implemented depending on the application requirements.

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.

1.7 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 (unclocked 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:

Table 2-1 AMBA AHB signals (continued)

Name	Source	Description
HWDATA[31:0] 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.
HSELx 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.
HRDATA[31:0] 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.
HREADY Transfer done	Slave	When HIGH the **HREADY** 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 **HREADY** as both an input and an output signal.
HRESP[1:0] 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.

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

Name	Description
PCLK Bus clock	The rising edge of **PCLK** is used to time all transfers on the APB.
PRESETn APB reset	The APB bus reset signal is active LOW and this signal will normally be connected directly to the system bus reset signal.
PADDR[31:0] 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.
PSELx 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 **PSELx** signal for each bus slave.
PENABLE 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 **PENABLE** occurs in the middle of the APB transfer.
PWRITE APB transfer direction	When HIGH this signal indicates an APB write access and when LOW a read access.
PRDATA APB read data bus	The read data bus is driven by the selected slave during read cycles (when **PWRITE** is LOW). The read data bus can be up to 32-bits wide.
PWDATA APB write data bus	The write data bus is driven by the peripheral bus bridge unit during write cycles (when **PWRITE** 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:

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.

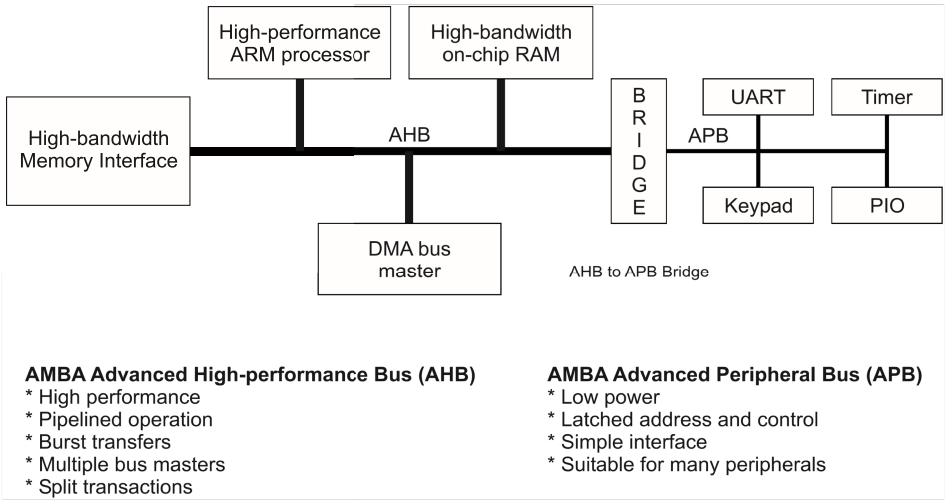

Figure 3-1 A typical AMBA AHB-based system

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.

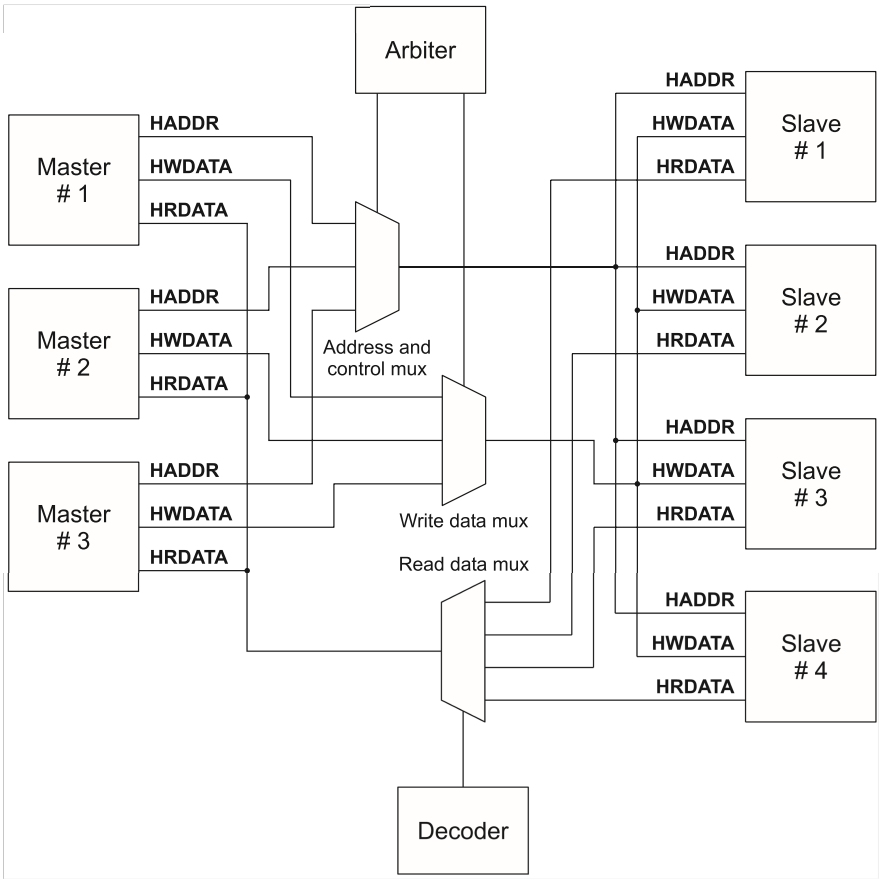

Figure 3-2 Multiplexor interconnection

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 **HREADY** 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.

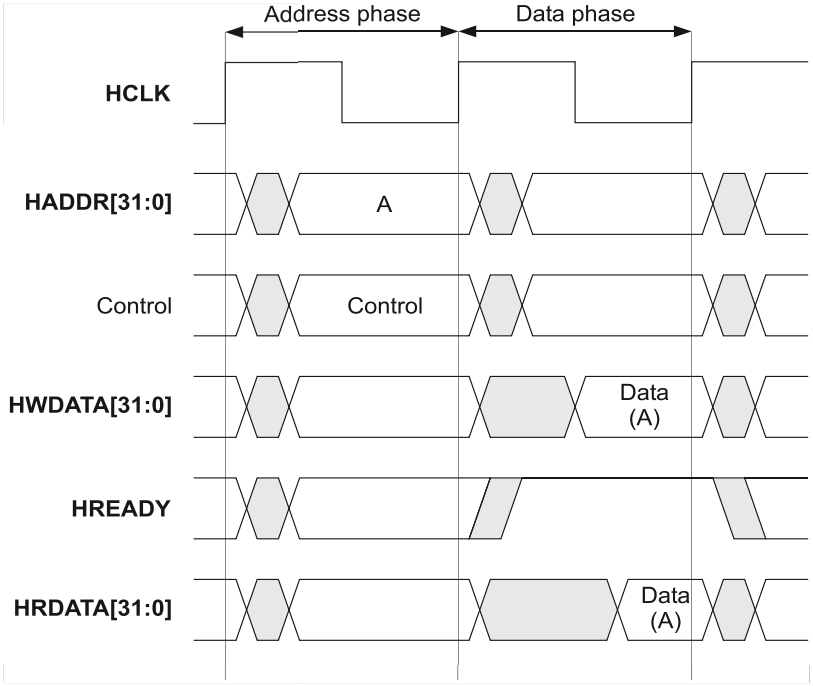

Figure 3-3 Simple transfer

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.

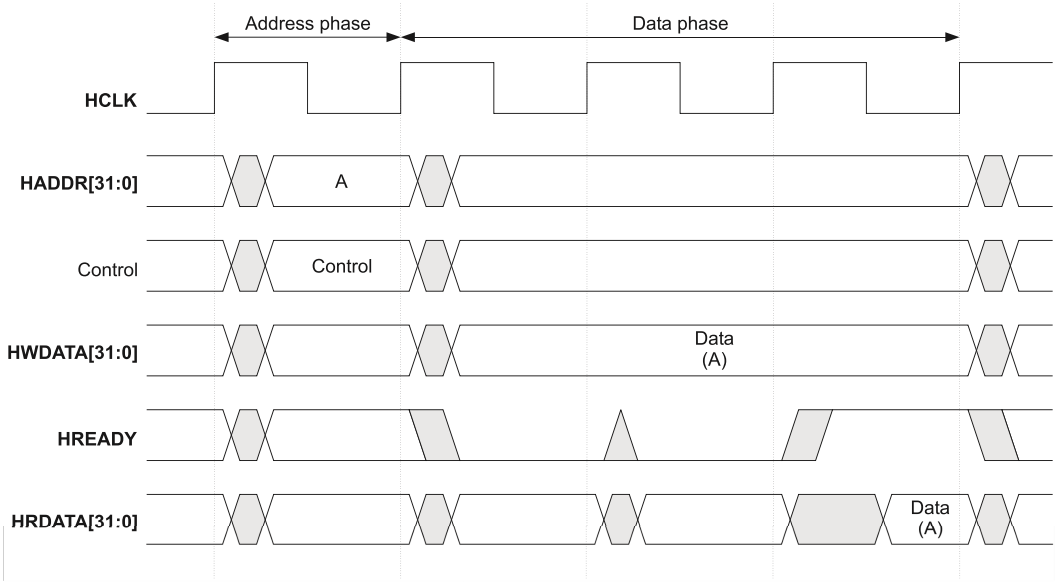

Figure 3-4 Transfer with wait states

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.

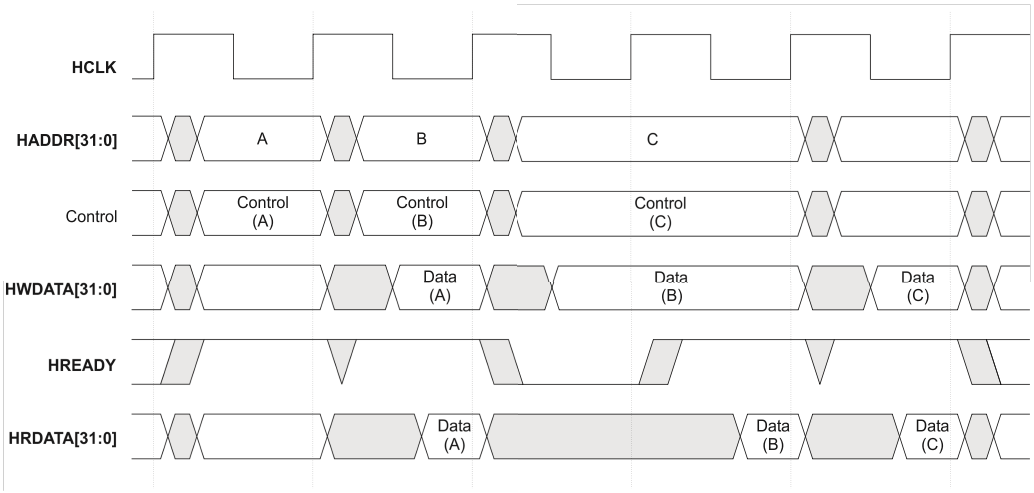

Figure 3-5 Multiple transfers

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.

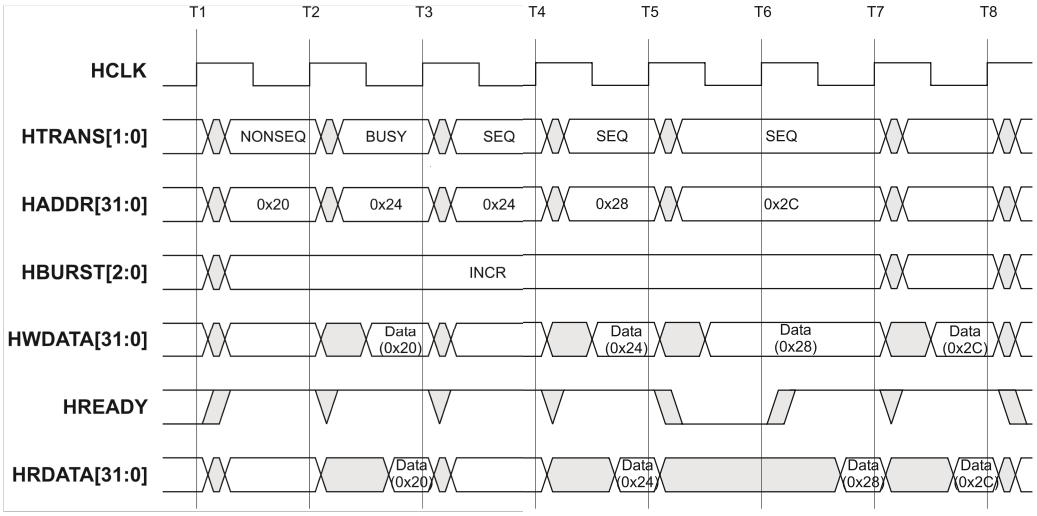

Figure 3-6 Transfer type examples

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

HBURST[2:0]	Type	Description
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.

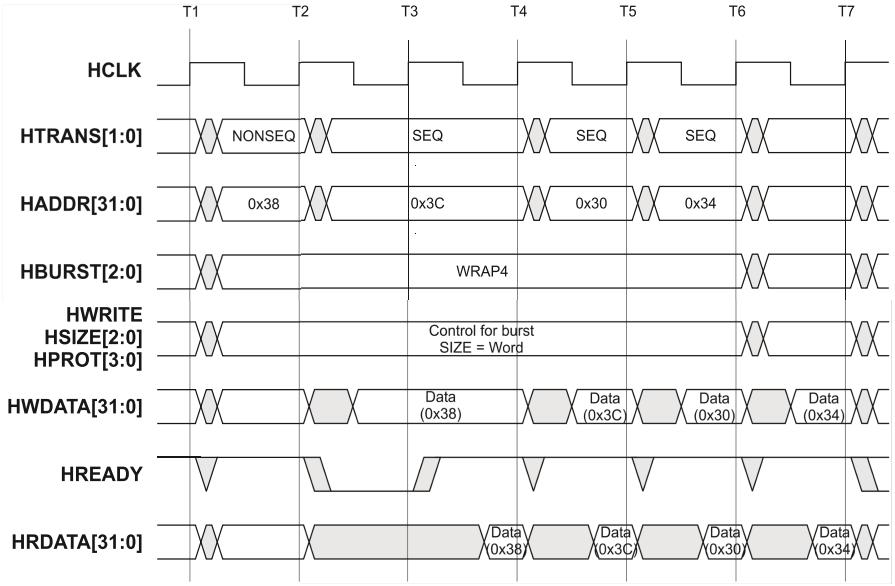

Figure 3-7 Four-beat wrapping burst

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.

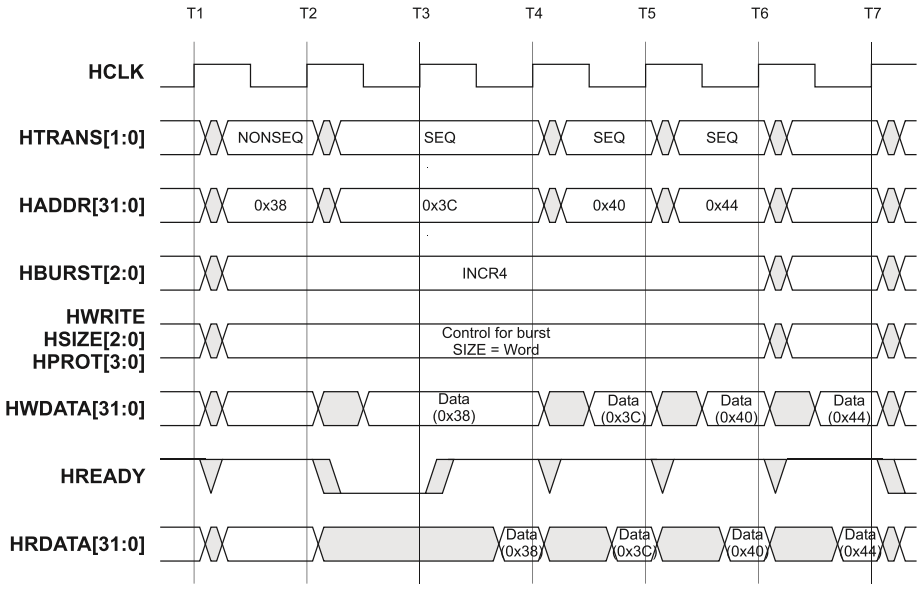

Figure 3-8 Four-beat incrementing burst

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

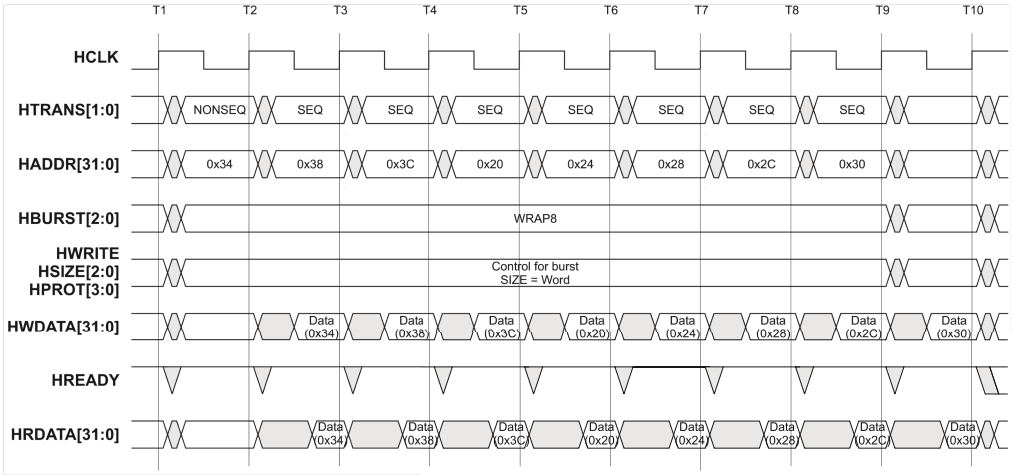

Figure 3-9 Eight-beat wrapping burst

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.

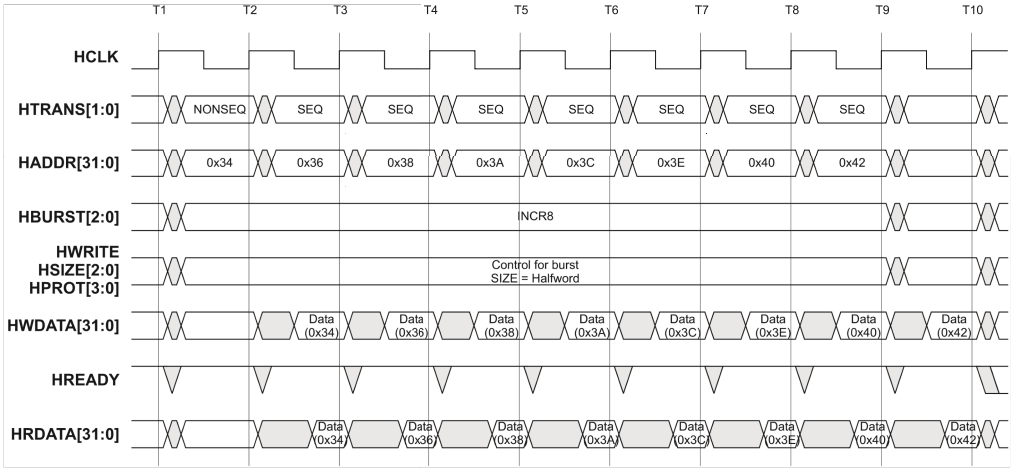

Figure 3-10 Eight-beat incrementing burst

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

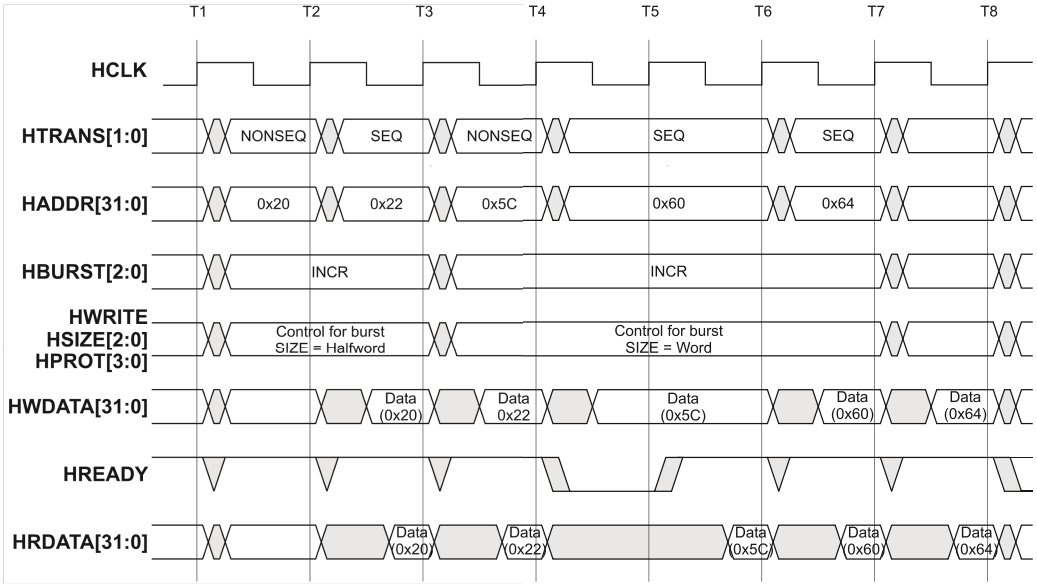

Figure 3-11 Undefined-length bursts

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

HSIZE[2]	HSIZE[1]	HSIZE[0]	Size	Description
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).

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 **HREADY** is HIGH this shows the transfer has completed successfully. The OKAY response is also used for any additional cycles that are inserted, with **HREADY** 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.

3.10 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.

Chapter 4
AHB Modules

This chapter describes the data sheets for the modules that are connected to the *Advanced High Performance Bus* (AHB). It contains the following sections:

4.1 APB bridge

The AHB to APB bridge is an AHB slave, providing an interface between the high-speed AHB and the low-power APB. Read and write transfers on the AHB are converted into equivalent transfers on the APB. As the APB is not pipelined, then wait states are added during transfers to and from the APB when the AHB is required to wait for the APB. Figure 4-1 shows the block diagram of the APB bridge module.

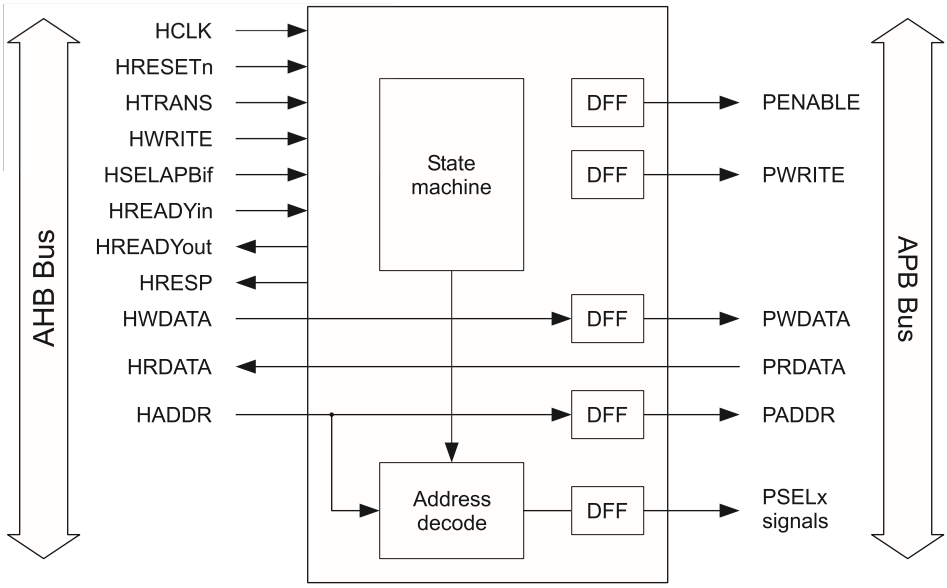

Figure 4-1 Block diagram of bridge module

The main sections of this module are:

- AHB slave bus interface
- APB transfer state machine, which is independent of the device memory map
- APB output signal generation.

To add new APB peripherals, or alter the system memory map, only the address decode sections need to be modified.

4.1.1 Signal descriptions

The APB bridge module signals are described in Table 4-1.

Table 4-1 Signal descriptions for bridge module

Signal	Type	Direction	Description
HCLK	Bus clock	Input	This clock times all bus transfers.
HRESETn	Reset	Input	The bus reset signal is active LOW, and is used to reset the system and the bus.
HADDR[31:0]	Address bus	Input	The 32-bit system address bus.
HTRANS[1:0]	Transfer type	Input	This indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE	Transfer direction	Input	When HIGH this signal indicates a write transfer, and when LOW, a read transfer.
HWDATA[31:0]	Write data bus	Input	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.
HSELAPBif	Slave select	Input	Each APB slave has its own slave select signal, and this signal indicates that the current transfer is intended for the selected slave. This signal is a combinatorial decode of the address bus.
HRDATA[31:0]	Read data bus	Output	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.
HREADYin **HREADYout**	Transfer done	Input/output	When HIGH the **HREADY** signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.
HRESP[1:0]	Transfer response	Output	The transfer response provides additional information on the status of a transfer. This module will always generate the OKAY response.
PRDATA[31:0]	Peripheral read data bus	Input	The peripheral read data bus is driven by the selected peripheral bus slave during read cycles (when **PWRITE** is LOW).

Table 4-1 Signal descriptions for bridge module (continued)

Signal	Type	Direction	Description
PWDATA[31:0]	Peripheral write data bus	Output	The peripheral write data bus is continuously driven by this module, changing during write cycles (when **PWRITE** is HIGH).
PENABLE	Peripheral enable	Output	This enable signal is used to time all accesses on the peripheral bus. **PENABLE** goes HIGH on the second clock rising edge of the transfer, and LOW on the third (last) rising clock edge of the transfer.
PSELx	Peripheral slave select	Output	There is one of these signals for each APB peripheral present in the system. The signal indicates that the slave device is selected, and that a data transfer is required. It has the same timing as the peripheral address bus. It becomes HIGH at the same time as **PADDR**, but will be set LOW at the end of the transfer.
PADDR[31:0]	Peripheral address bus	Output	This is the APB address bus, which may be up to 32 bits wide and is used by individual peripherals for decoding register accesses to that peripheral. The address becomes valid after the first rising edge of the clock at the start of the transfer. If there is a following APB transfer, then the address will change to the new value, otherwise it will hold its current value until the start of the next APB transfer.
PWRITE	**Peripheral transfer direction**	Output	This signal indicates a write to a peripheral when HIGH, and a read from a peripheral when LOW. It has the same timing as the peripheral address bus.

Timing diagrams showing the relationship between AHB and APB transfers can be found in the *APB Specification*.

4.1.2 Peripheral memory map

The APB bridge controls the memory map for the peripherals, and generates a select signal for each peripheral. The default system memory map is shown in Figure 4-2.

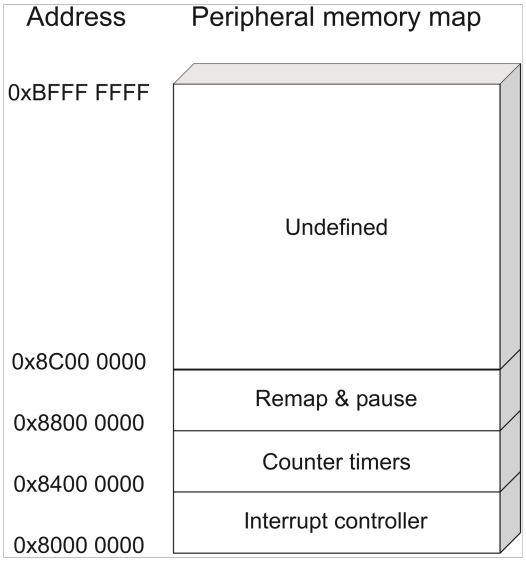

Figure 4-2 Peripheral memory map

4.1.3 Function and operation of module

The APB bridge responds to transaction requests from the currently granted AHB master. The AHB transactions are then converted into APB transactions. The state machine, shown in Figure 4-3 on page 4-6, controls:

- the AHB transactions with the **HREADYout** signal
- the generation of all APB output signals.

The individual **PSELx** signals are decoded from **HADDR**, using the state machine to enable the outputs while the APB transaction is being performed.

If an undefined location is accessed, operation of the system continues as normal, but no peripherals are selected.

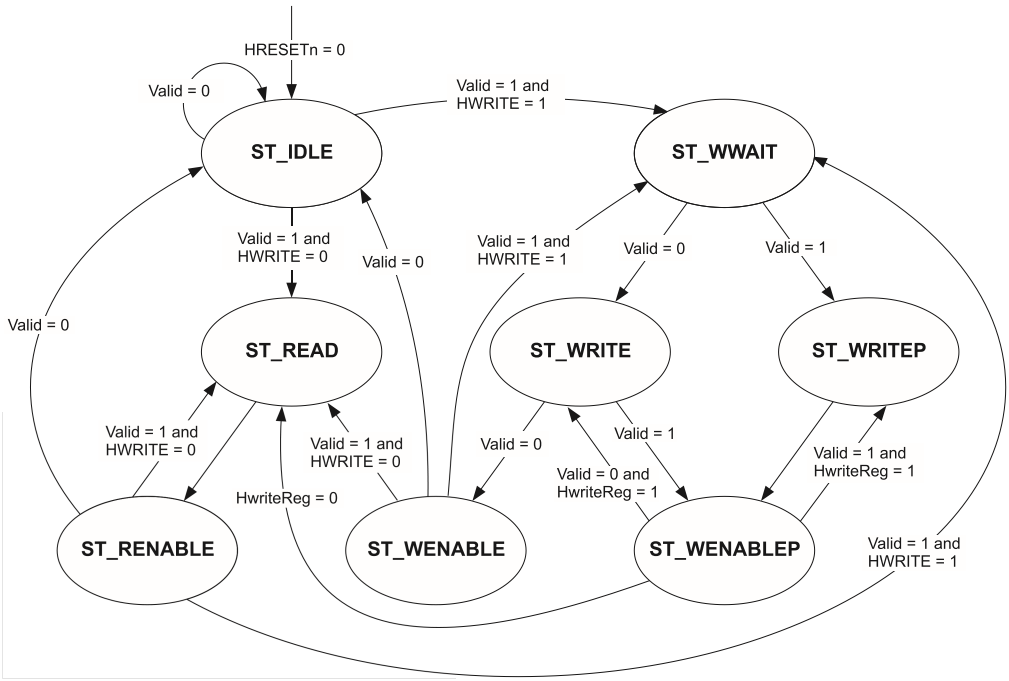

Figure 4-3 State machine for AHB to APB interface

The individual states of the state machine operation are described in the following sections:

- *ST_IDLE* on page 4-7
- *ST_READ* on page 4-7
- *ST_WWAIT* on page 4-7
- *ST_WRITE* on page 4-8
- *ST_WRITEP* on page 4-8
- *ST_RENABLE* on page 4-9
- *ST_WENABLE* on page 4-9
- *ST_WENABLEP* on page 4-9.

ST_IDLE

During this state the APB buses and **PWRITE** are driven with the last values they had, and **PSEL** and **PENABLE** lines are driven LOW.

The ST_IDLE state is entered from:

- reset, when the system is initialized
- ST_RENABLE, ST_WENABLE, or ST_IDLE, when there are no peripheral transfers to perform.

The next state is:

- ST_READ, for a read transfer, when the AHB contains a valid APB read transfer
- ST_WWAIT, for a write transfer, when the AHB contains a valid APB write transfer.

ST_READ

During this state the address is decoded and driven onto **PADDR**, the relevant **PSEL** line is driven HIGH, and **PWRITE** is driven LOW. A wait state is always inserted to ensure that the data phase of the current AHB transfer does not complete until the APB read data has been driven onto **HRDATA**.

The ST_READ state is entered from ST_IDLE, ST_RENABLE, ST_WENABLE, or ST_WENABLEP during a valid read transfer.

The next state will always be ST_RENABLE.

ST_WWAIT

This state is needed due to the pipelined structure of AHB transfers, to allow the AHB side of the write transfer to complete so that the write data becomes available on **HWDATA**. The APB write transfer is then started in the next clock cycle.

The ST_WWAIT state is entered from ST_IDLE, ST_RENABLE, or ST_WENABLE, during a valid write transfer.

The next state will always be ST_WRITE.

ST_WRITE

During this state the address is decoded and driven onto **PADDR**, the relevant **PSEL** line is driven HIGH, and **PWRITE** is driven HIGH.

A wait state is not inserted, as a single write transfer can complete without affecting the AHB.

The ST_WRITE state is entered from:

- ST_WWAIT, when there are no further peripheral transfers to perform
- ST_WENABLEP, when the currently pending peripheral transfer is a write, and there are no further transfers to perform.

The next state is:

- ST_WENABLE, when there are no further peripheral transfers to perform
- ST_WENABLEP, when there is one further peripheral write transfer to perform.

ST_WRITEP

During this state the address is decoded and driven onto **PADDR**, the relevant **PSEL** line is driven HIGH, and **PWRITE** is driven HIGH. A wait state is always inserted, as there must only ever be one pending transfer between the currently performed APB transfer and the currently driven AHB transfer. See the write transfer timing diagrams in the *AMBA Specification (Rev 2.0)* for more details.

The ST_WRITEP state is entered from:

- ST_WWAIT, when there is a further peripheral transfer to perform.
- ST_WENABLEP, when the currently pending peripheral transfer is a write, and there is a further transfer to perform.

The next state will always be ST_WENABLEP.

ST_RENABLE

During this state the **PENABLE** output is driven HIGH, enabling the current APB transfer. All other APB outputs remain the same as the previous cycle.

The ST_RENABLE state is always entered from ST_READ.

The next state is:

- ST_READ, when there is a further peripheral read transfer to perform
- ST_WWAIT, when there is a further peripheral write transfer to perform
- ST_IDLE, when there are no further peripheral transfers to perform.

ST_WENABLE

During this state the **PENABLE** output is driven HIGH, enabling the current APB transfer. All other APB outputs remain the same as the previous cycle.

The ST_WENABLE state is always entered from ST_WRITE.

The next state is:

- ST_READ, when there is a further peripheral read transfer to perform
- ST_WWAIT, when there is a further peripheral write transfer to perform
- ST_IDLE, when there are no further peripheral transfers to perform.

ST_WENABLEP

A wait state is inserted if the pending transfer is a read because, when a read follows a write, an extra wait state must be inserted to allow the write transfer to complete on the APB before the read is started.

The ST_WENABLEP state is entered from:

- ST_WRITE, when the currently driven AHB transfer is a peripheral transfer
- ST_WRITEP, when there is a pending peripheral transfer following the current write.

The next state is:

- ST_READ, when the pending transfer is a read
- ST_WRITE, when the pending transfer is a write, and there are no further transfers to perform
- ST_WRITEP, when the pending transfer is a write, and there is a further transfer to perform.

4.1.4 System description

This section describes how the HDL code for the APB bridge is set out. A simple system block diagram, with information about the main parts of the HDL code, is followed by details of the registers, inputs, and outputs used in the module. This should be read in conjunction with the HDL code.

Figure 4-4 shows the APB bridge module block diagram.

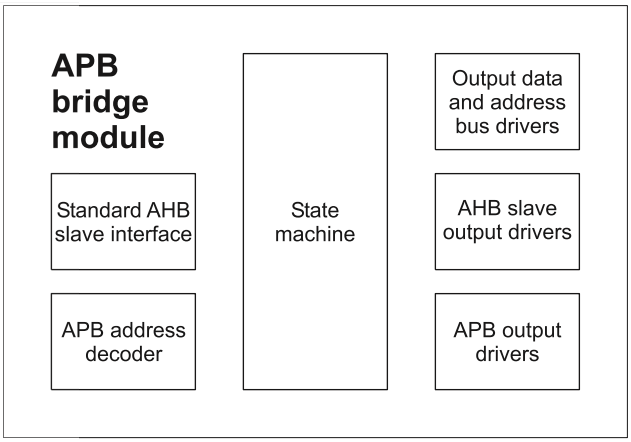

Figure 4-4 APB bridge module block diagram

The AHB to APB bridge comprises a state machine, which is used to control the generation of the APB and AHB output signals, and the address decoding logic which is used to generate the APB peripheral select lines.

All registers used in the system are clocked from the rising edge of the system clock **HCLK**, and use the asynchronous reset **HRESETn**.

Figure 4-5 on page 4-11 shows the APB bridge HDL file.

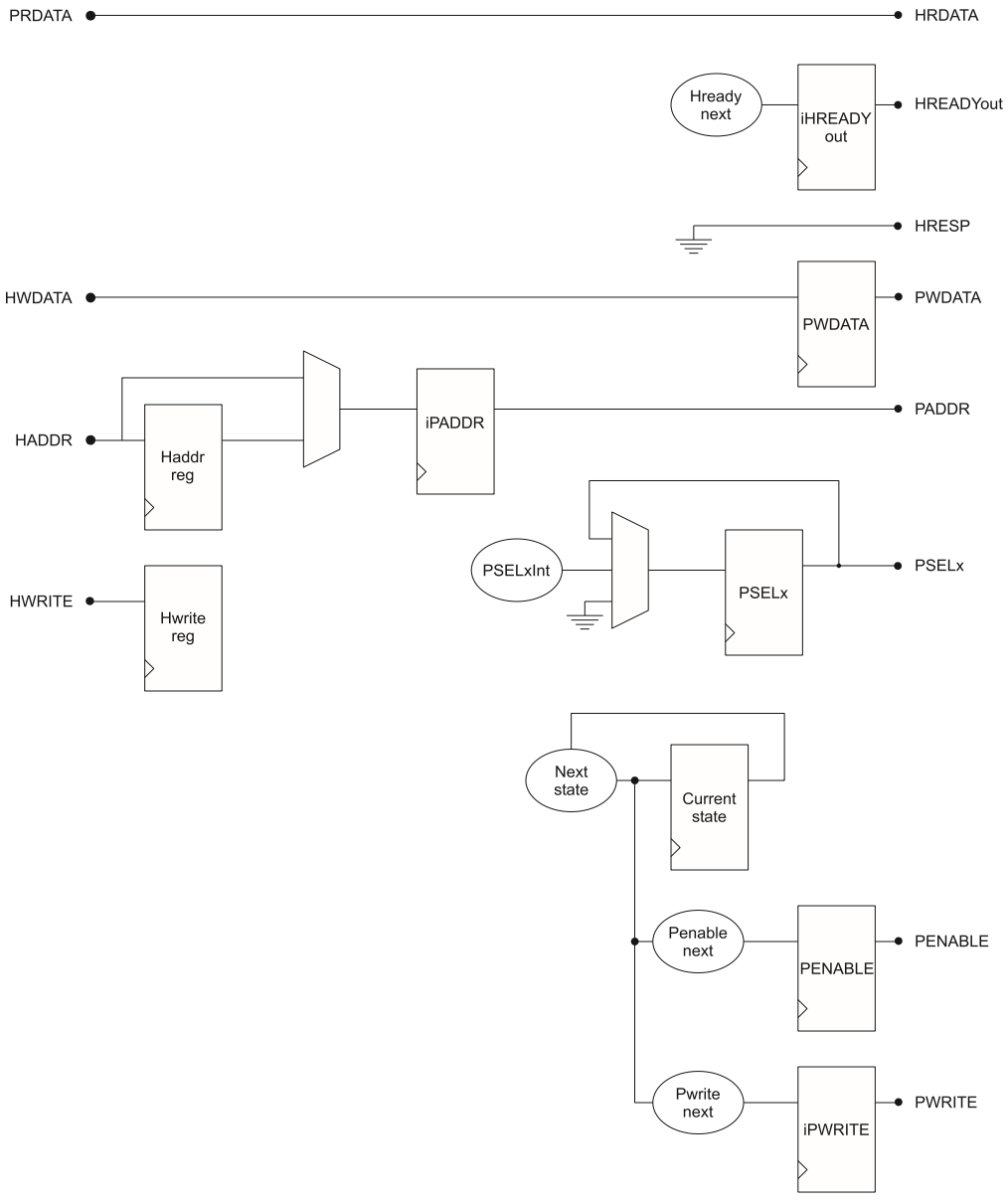

Figure 4-5 APB bridge module system diagram

The main sections in this module are explained in the following paragraphs:

- *Constant definitions*
- *AHB slave bus interface*
- *APB transfer state machine*
- *APB output signal generation* on page 4-13
- *AHB output signal generation* on page 4-13.

Constant definitions

The constant PADDRWIDTH sets the width of the peripheral address bus that is used, up to a maximum of 32 bits. This size depends on the size of address that is needed by the peripherals in the system. The default value is a 16-bit address bus.

The next two constants define the state machine states, and the top four address bits that are used to decode the peripheral select outputs. If the peripheral address map is changed from the default, then these constants must be modified to match the changes.

AHB slave bus interface

This module uses the standard AHB slave bus interface, which comprises:

- the valid transfer detection logic which is used to determine when a valid transfer is accessing the slave
- the address and control registers, which are used to store the information from the address phase of the transfer for use in the data phase.

Due to the different AHB to APB timing of read and write transfers, either the current or the previous address input value is needed to correctly generate the APB transfer. A multiplexor is therefore used to select between the current address input or the registered address, for read and write transfers respectively.

APB transfer state machine

The transfer state machine is used to control the application of APB transfers based on the AHB inputs. The state diagram in Figure 4-3 on page 4-6 shows the operation of the state machine, which is controlled by its current state and the AHB slave interface signals.

APB output signal generation

The generation of all APB output signals is based on the status of the transfer state machine:

- **PWDATA** is a registered version of the **HWDATA** input, which is only enabled during a write transfer. As the bridge is the only bus master on the APB, then it can drive **PWDATA** continuously.
- **PENABLE** is only set HIGH during one of three enable states, in the last cycle of an APB transfer. A register is used to generate this output from the next state of the transfer state machine.
- The **PSELx** outputs are decoded from the current transfer address. They are only valid during the read, write and enable states, and are all driven LOW at all other times so that no peripherals are selected when no transfers are being performed.
- **PADDR** is a registered version of the currently selected address input (**HADDR** or the address register) and only changes when the read and write states are entered at the start of the APB transfer.
- **PWRITE** is set HIGH during a write transfer, and only changes when a new APB transfer is started. A register is used to generate this output from the next state of the transfer state machine.
- The **APBen** signal is used as an enable on the **PSEL**, **PWRITE** and **PADDR** output registers, ensuring that these signals only change when a new APB transfer is started, when the next state is ST_READ, ST_WRITE, or ST_WRITEP.

AHB output signal generation

A standard AHB slave interface consists of the following three outputs:

- **HRDATA** is directly driven with the current value of **PRDATA**. APB slaves only drive read data during the enable phase of the APB transfer, with **PRDATA** set LOW at all other times, so bus clash is avoided on **HRDATA** (assuming OR bus connections for both the AHB and APB read data buses).
- **HREADYout** is driven with a registered signal to improve the output timing. Wait states are inserted by the APB bridge during the ST_READ and ST_WRITEP states, and during the ST_WENABLEP state when the next transfer to be performed is a read.
- **HRESP** is continuously held LOW, as the APB bridge does not generate SPLIT, RETRY or ERROR responses.

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.

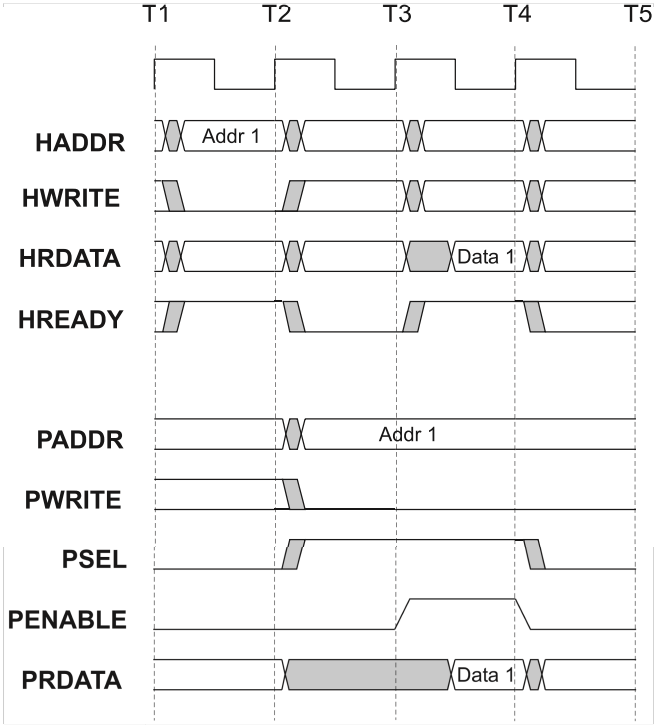

Figure 5-9 Read transfer to AHB

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.

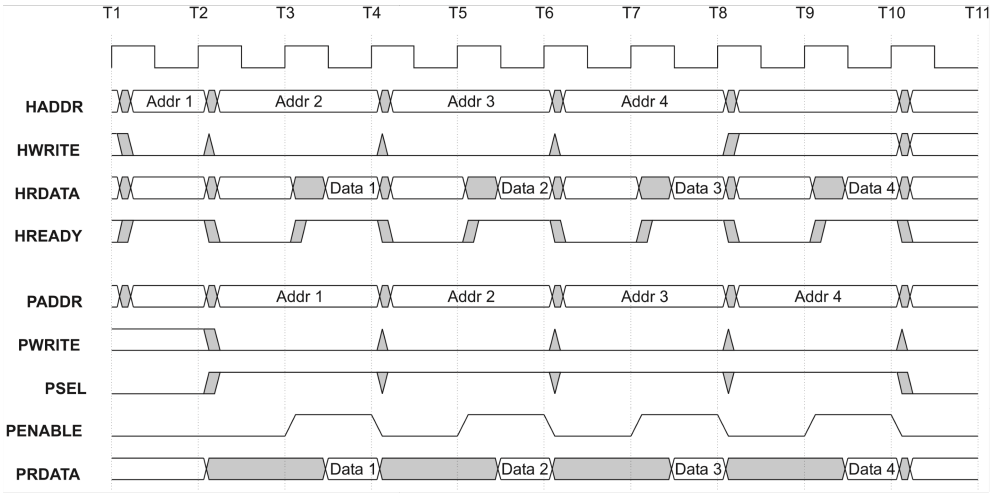

Figure 5-10 Burst of read transfers

5.6.2 Write transfers

Figure 5-11 shows a write transfer.

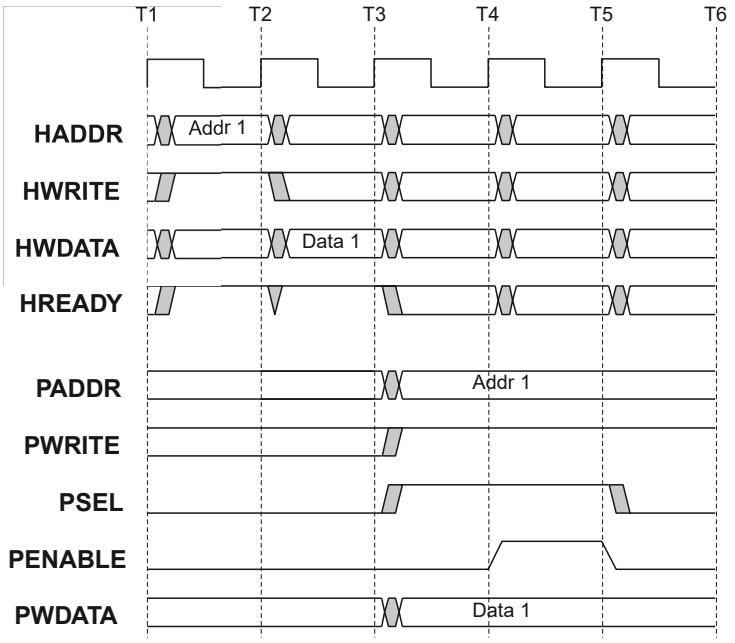

Figure 5-11 Write transfer from AHB

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.

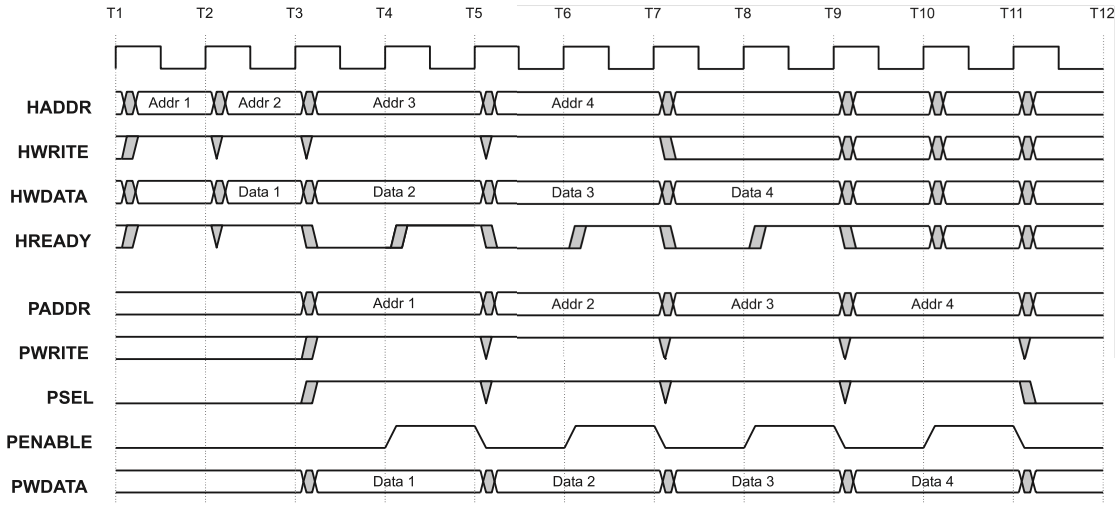

Figure 5-12 Burst of write transfers

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.

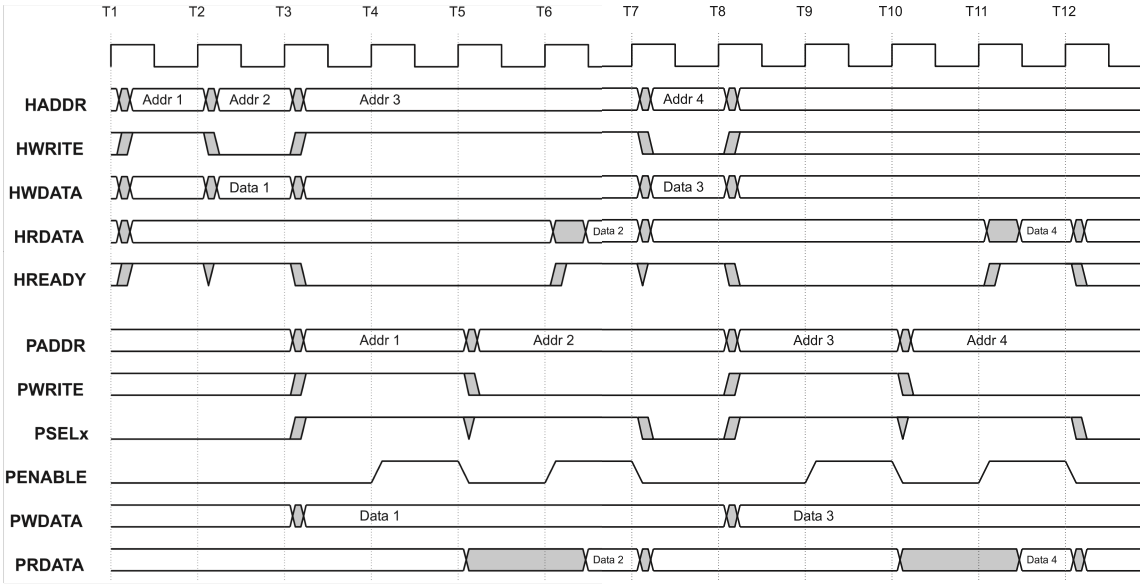

Figure 5-13 Back to back transfers

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.

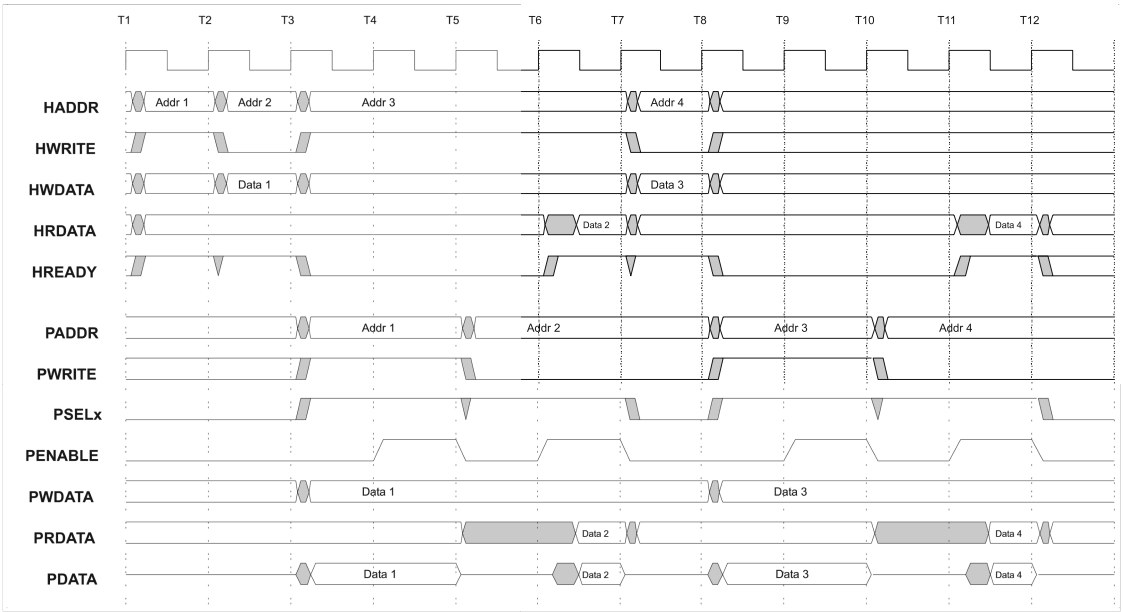

Figure 5-14 Tristate data bus