AHB Protocol

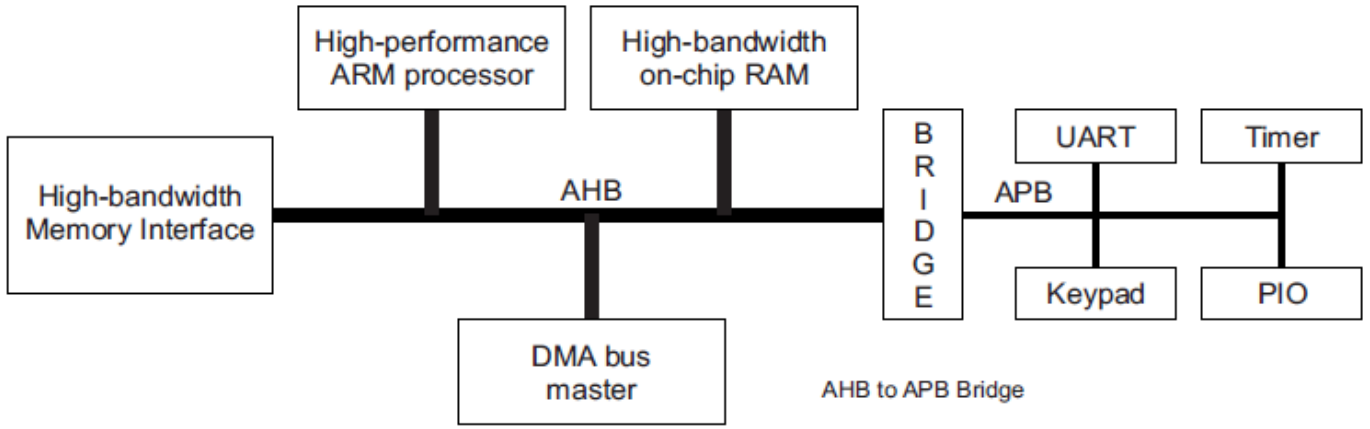
Mitesh Khadgi

For queries, contact: mitesh.khadgi2025@gmail.com

AHB 2.0

- 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).

A typical AMBA AHB-based system

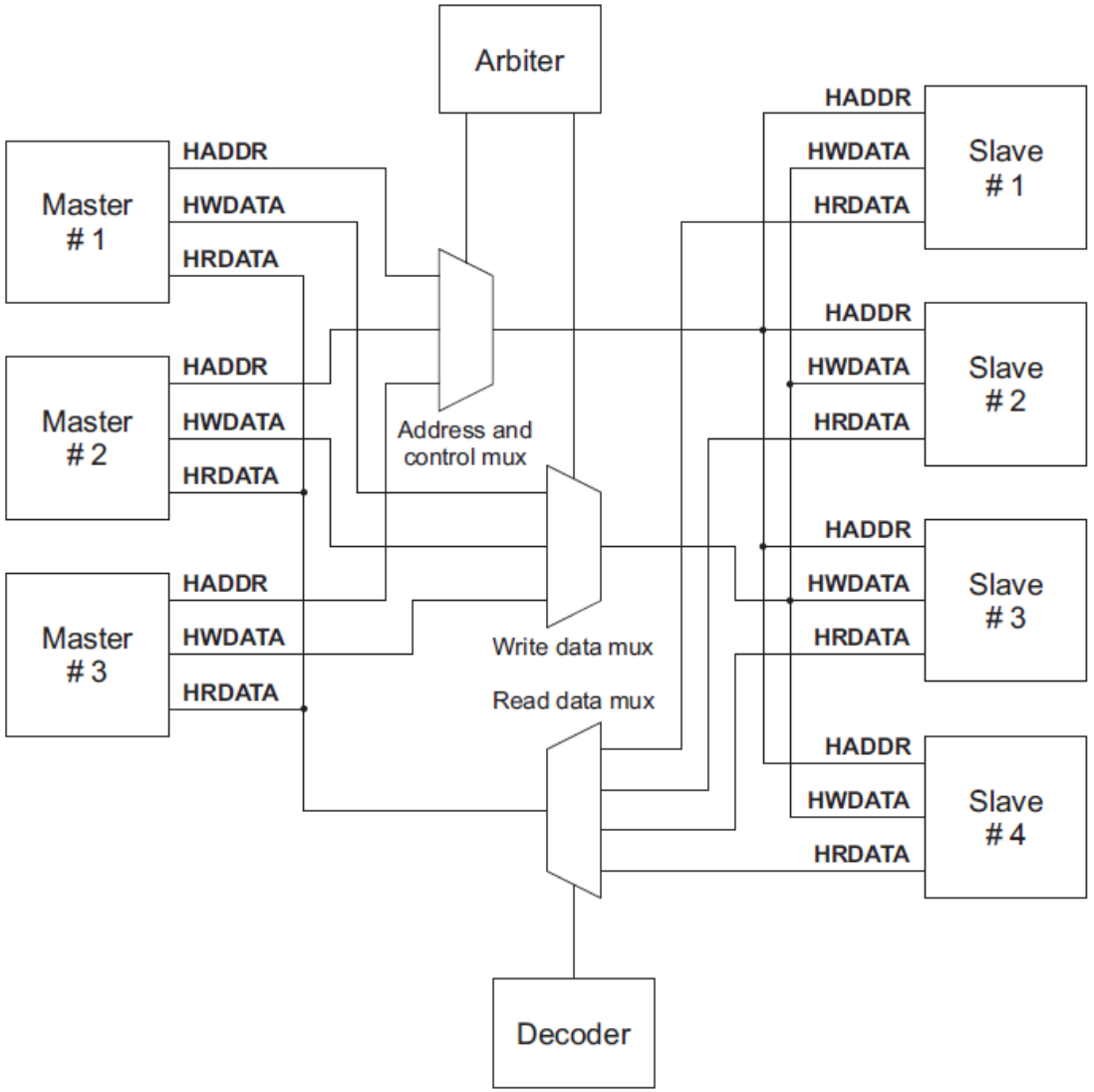


AHB to APB Bridge

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

Bus interconnection



Overview of 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.

Overview of AHB operation

- 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]**

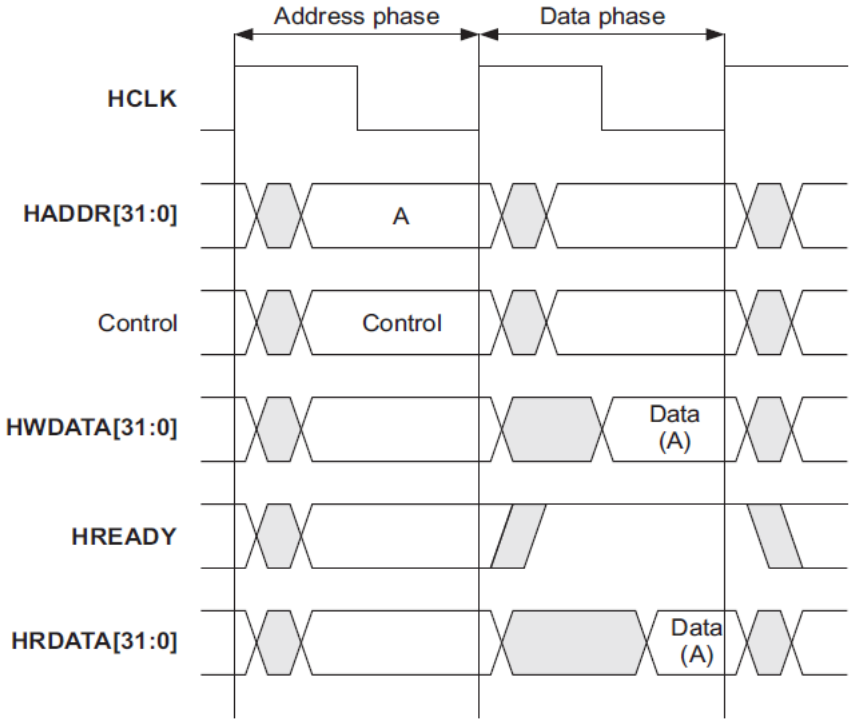
HRESP[1:0]	Description
OKAY	The OKAY response is used to indicate that the transfer is progressing normally and when **HREADY** goes HIGH this shows the transfer has completed successfully.
ERROR	The ERROR response indicates that a transfer error has occurred and the transfer has been unsuccessful.
RETRY 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.

Overview of AHB operation

- 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.
- 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.

Basic Transfer

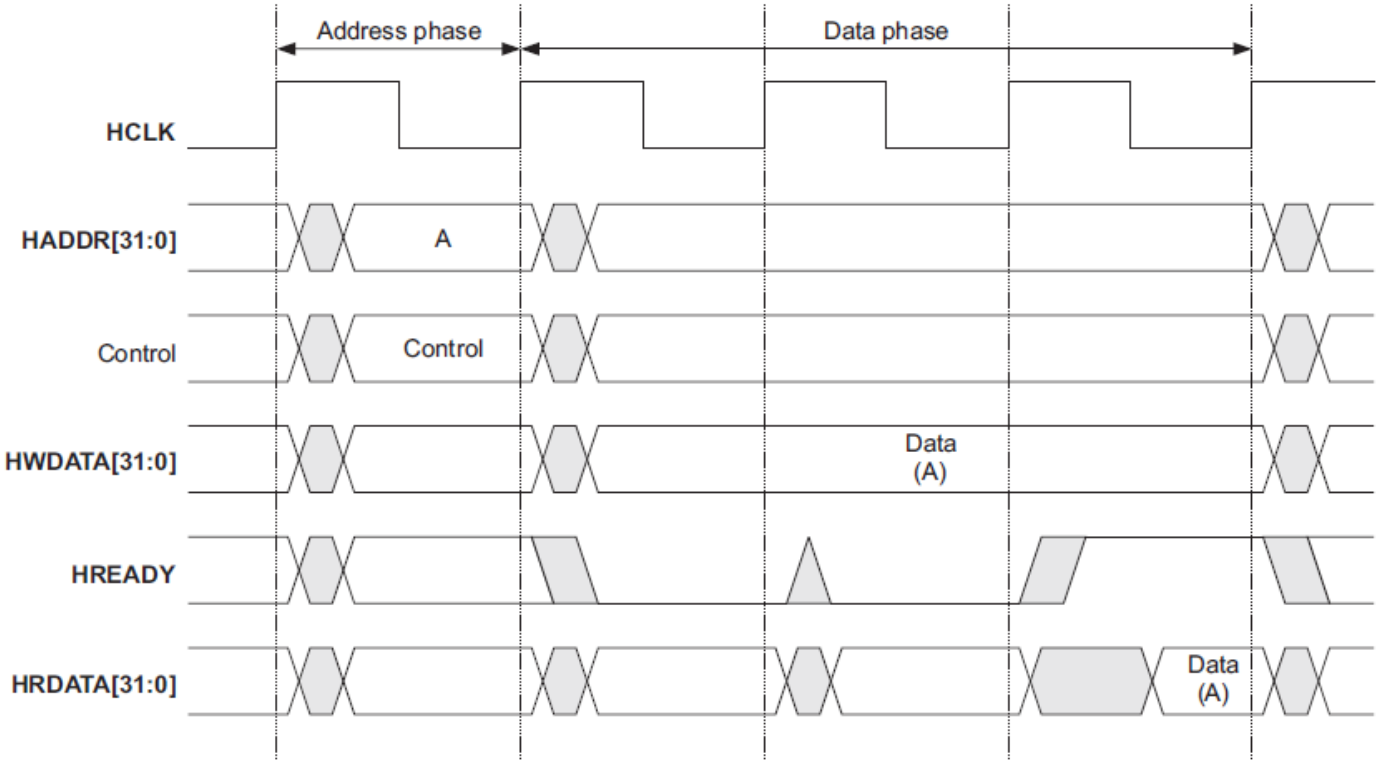
- An AHB transfer consists of two distinct sections:
  - The address phase, which lasts only a single cycle.
  - The data phase, which may require several cycles. This is achieved using the **HREADY** signal.
- With no wait states



Explanation

- 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

Transfer with wait states issued by slave



Multiple transfers

- 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.

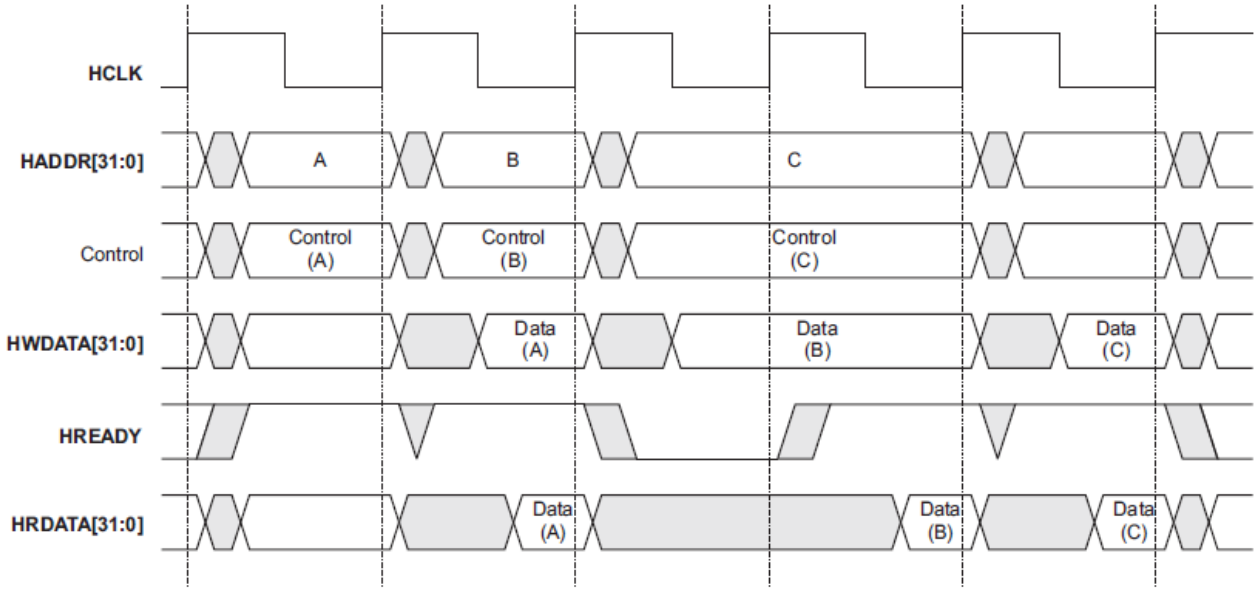


Figure 3-5 Multiple transfers

Explanation

- 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.

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

Transfer type encoding

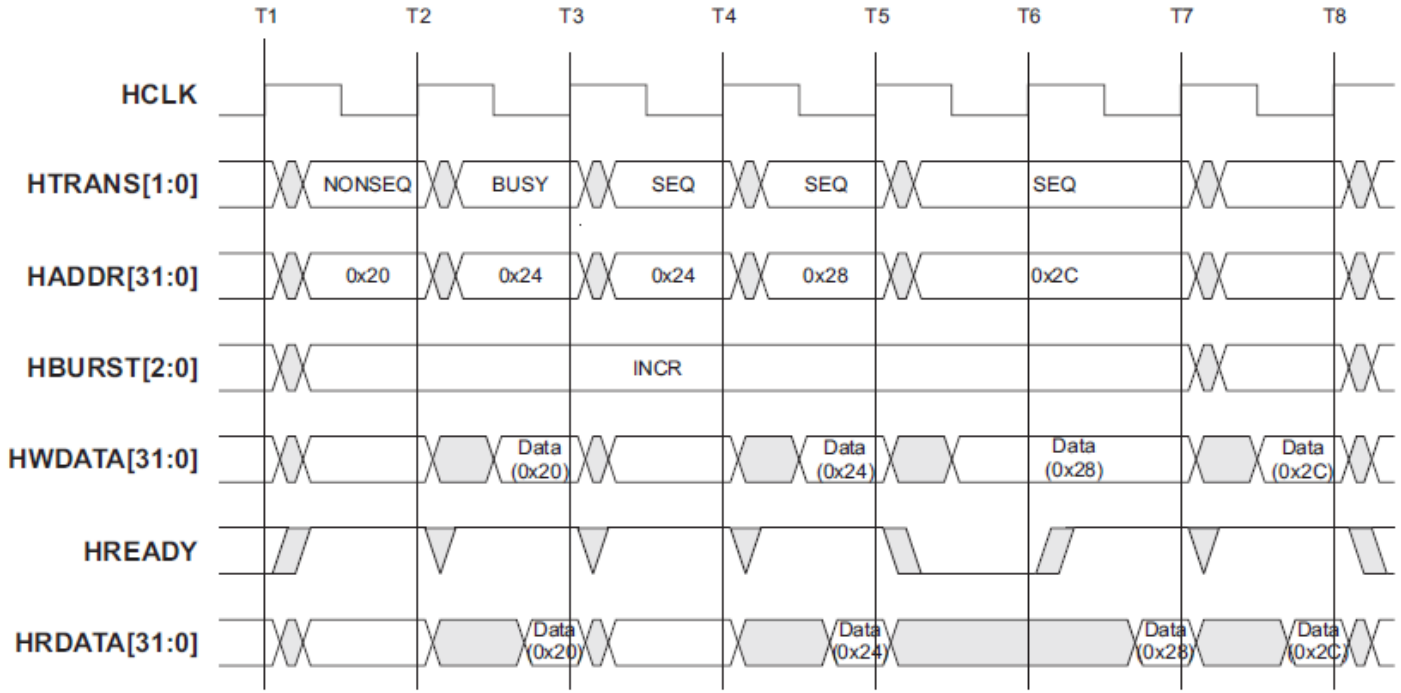
HTRANS[1:0]	Type	Description
00	IDLE	Indicates that no data transfer is required. The IDLE transfer type is used when a bus master is granted the bus, but does not wish to perform a data transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer should be 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).

Explanation

- 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.

Usage example of transfer types

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



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

Burst encoding

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

About Bursts

- 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 ransferred. 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).

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.

4-beat Wrapping burst

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

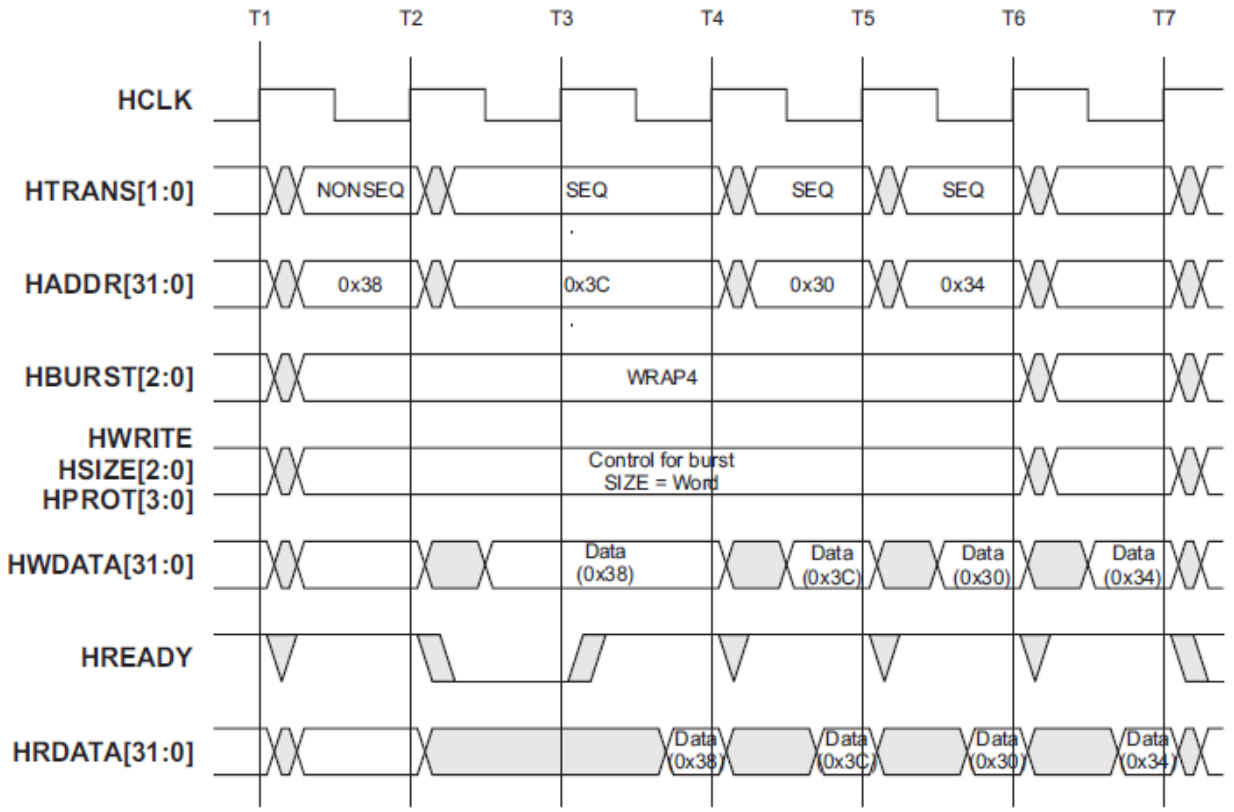


Figure 3-7 Four-beat wrapping burst

4-beat incrementing burst

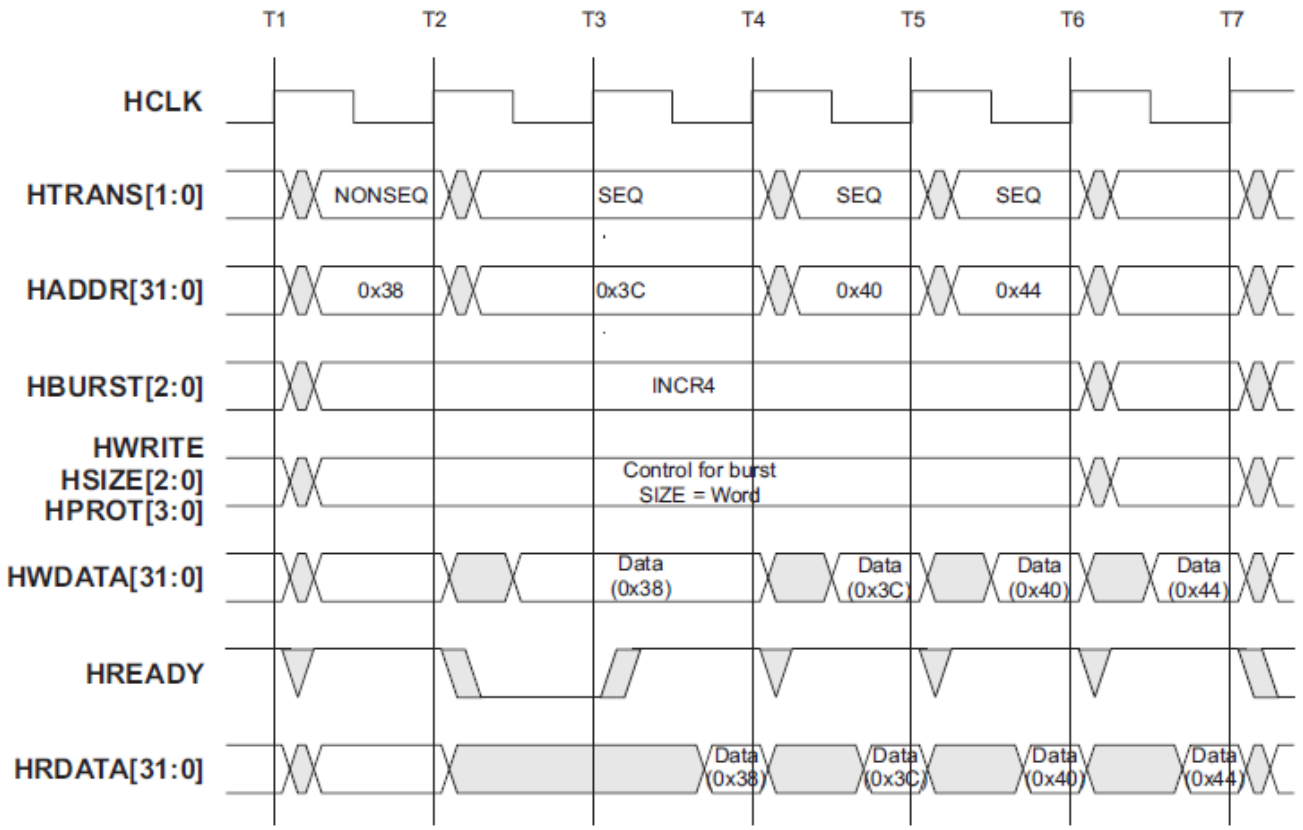


Figure 3-8 Four-beat incrementing burst

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]**.

Transfer Size

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

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.

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
- 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.

Protection encoding

Table 3-4 Protection signal encodings

HPROT[3] cacheable	HPROT[2] bufferable	HPROT[1] privileged	HPROT[0] data/opcode	Description
-	-	-	0	Opcode fetch
-	-	-	1	Data access
-	-	0	-	User access
-	-	1	-	Privileged access
-	0	-	-	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.

Shows a typical address decoding system and the slave select signals

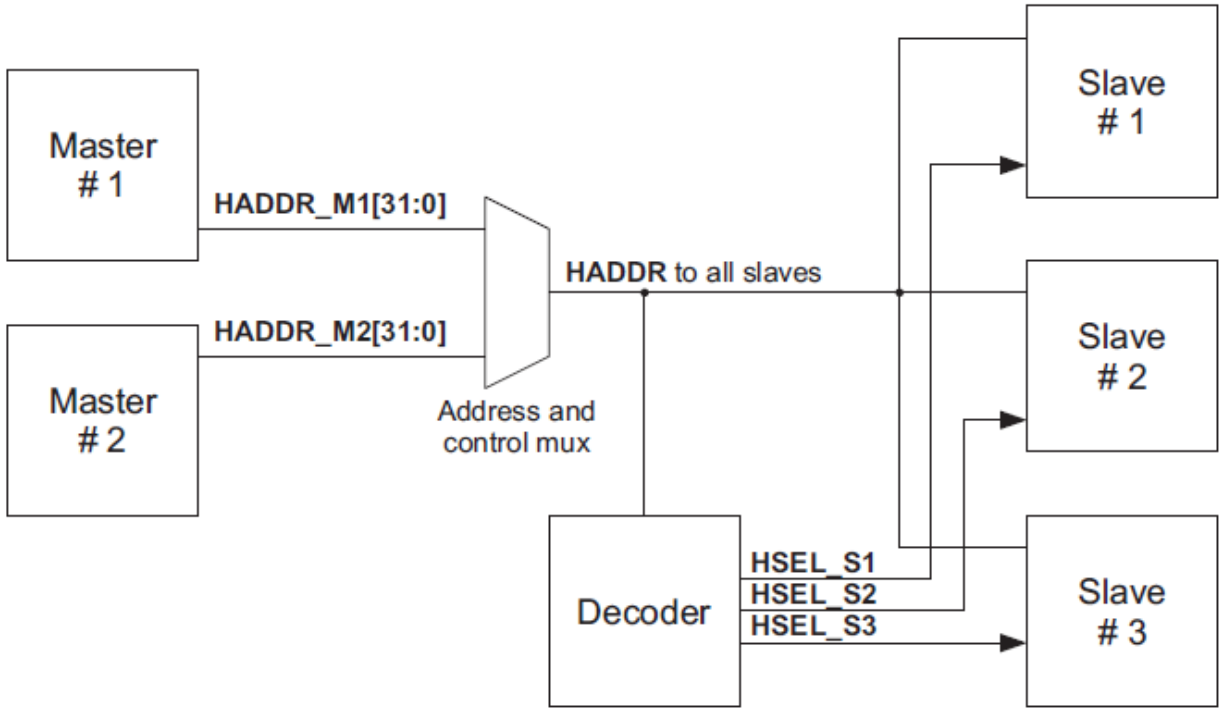


Figure 3-12 Slave select signals

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.

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.

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

Transfer responses encoding

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.

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 twocycle 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.

An example of a RETRY operation

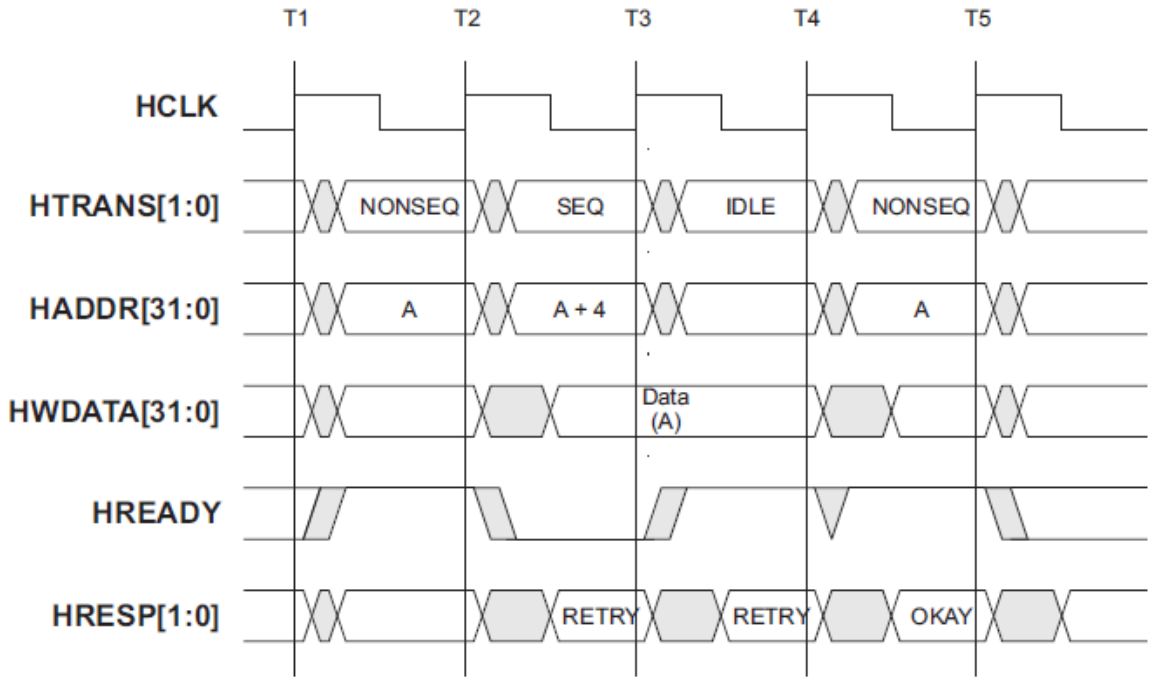


Figure 3-13 Transfer with retry response

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.

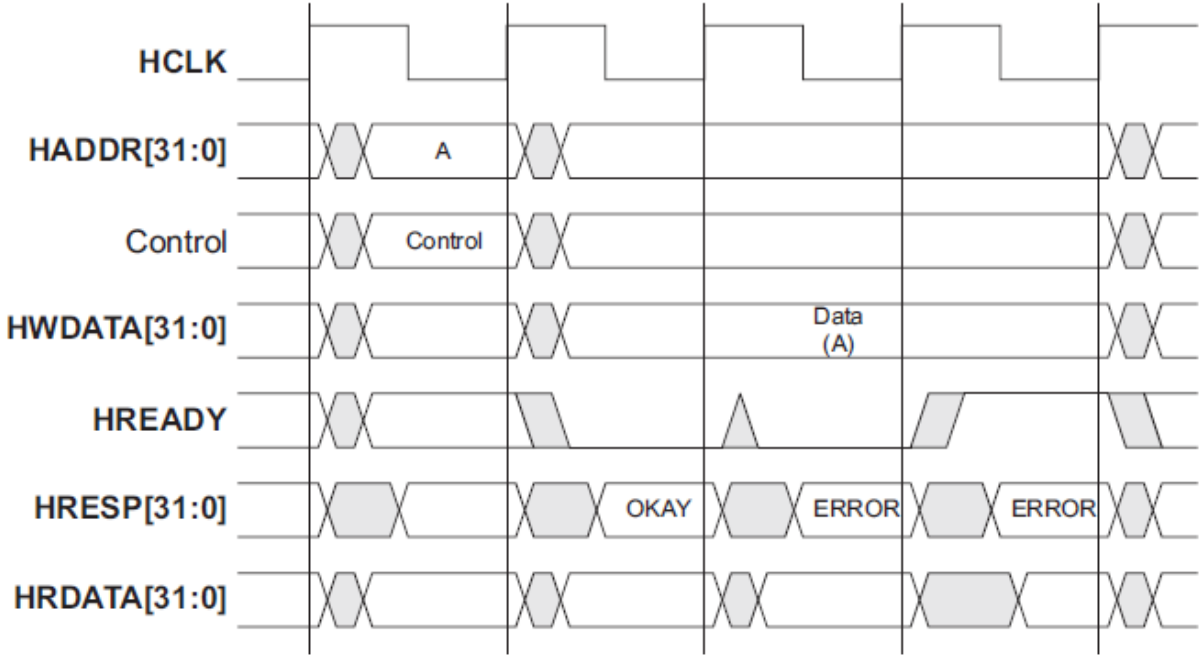


Figure 3-14 Error response

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.

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.

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.

Implementing a narrow slave on a wider bus

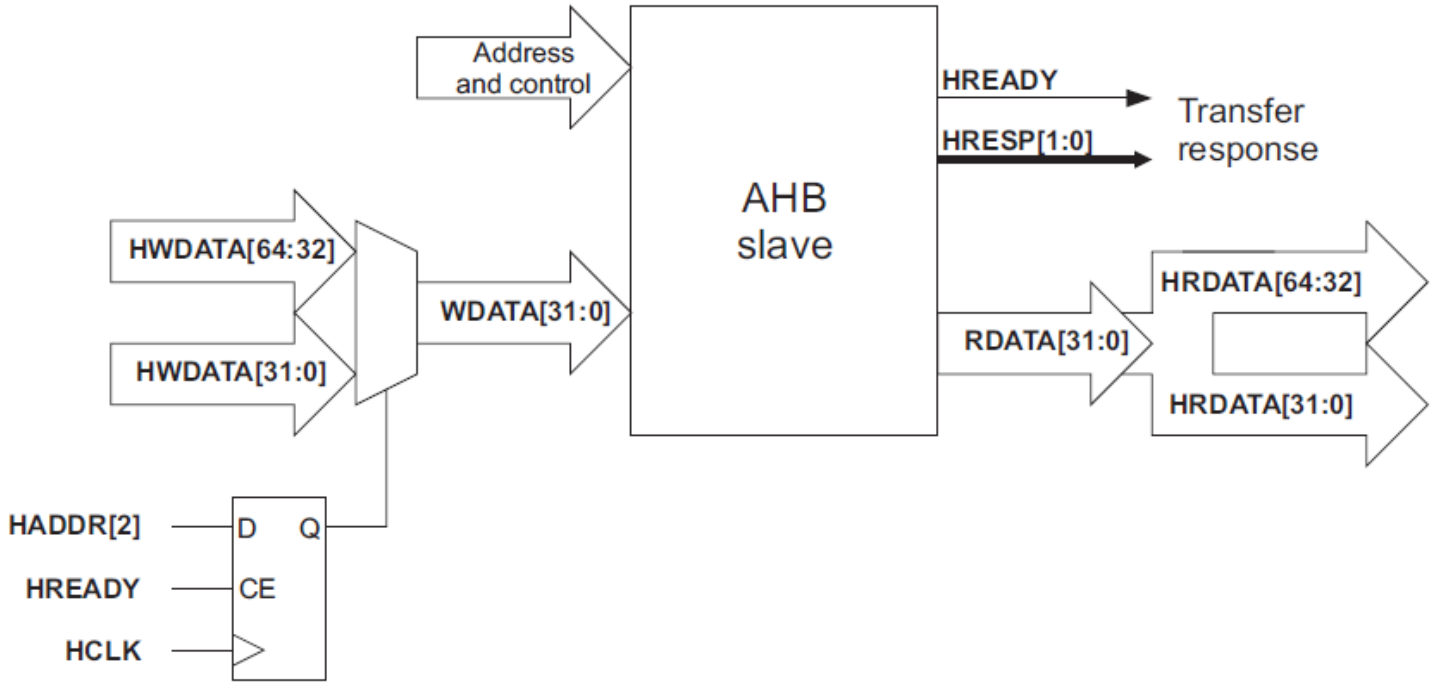


Figure 3-21 Narrow slave on a wide bus

Implementing a wide slave on a narrow bus

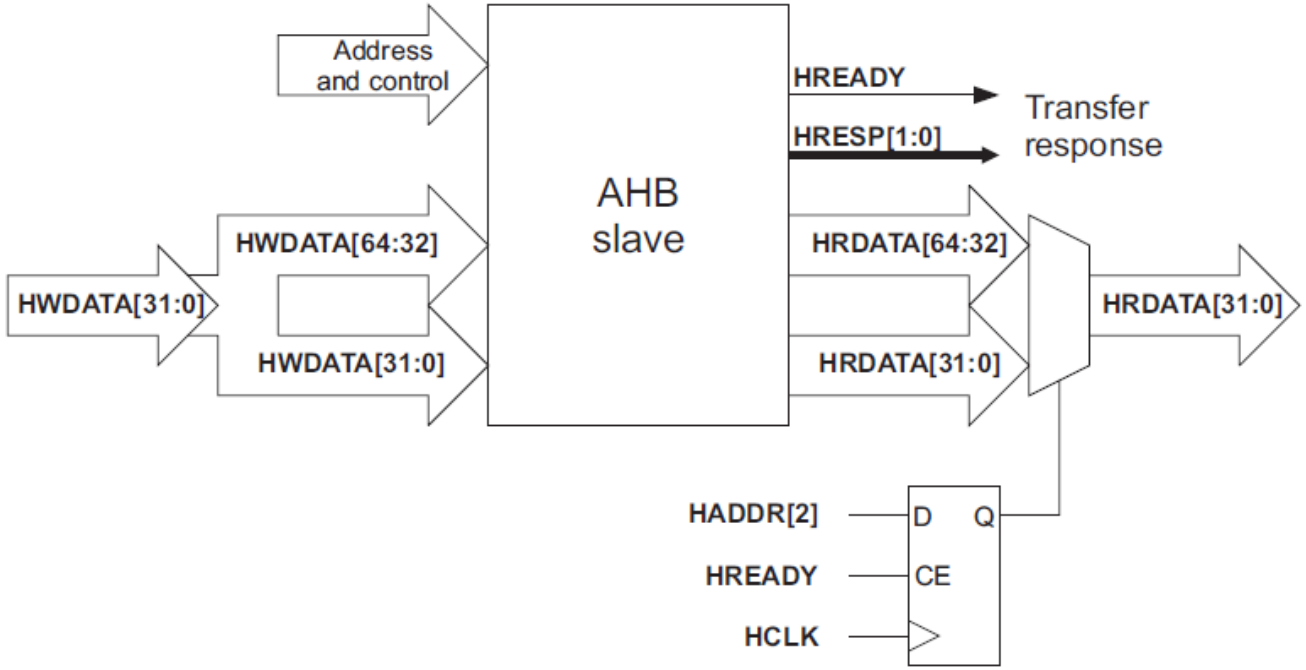


Figure 3-22 Wide slave on a narrow bus

Interface diagram – AHB bus slave

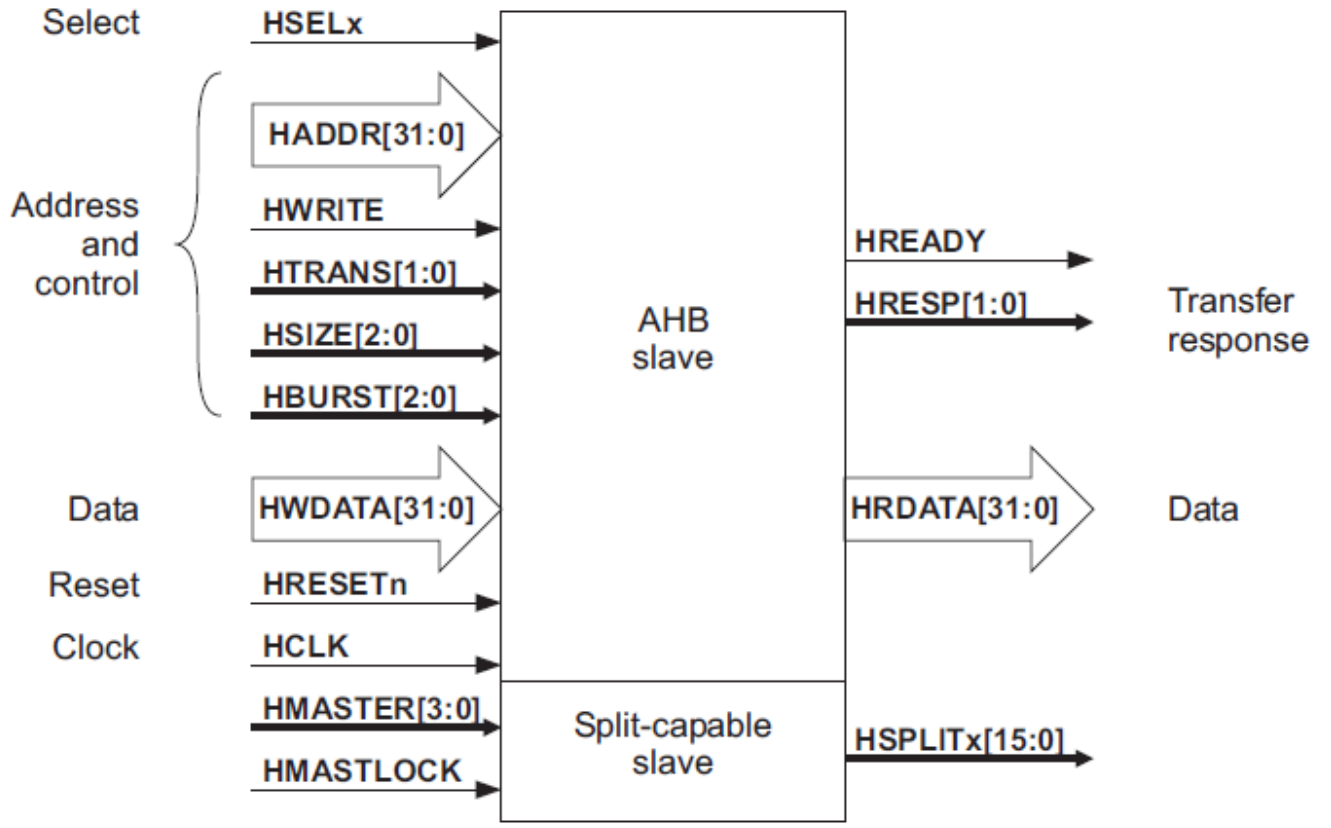


Figure 3-23 AHB bus slave interface

Interface diagram – AHB bus Master

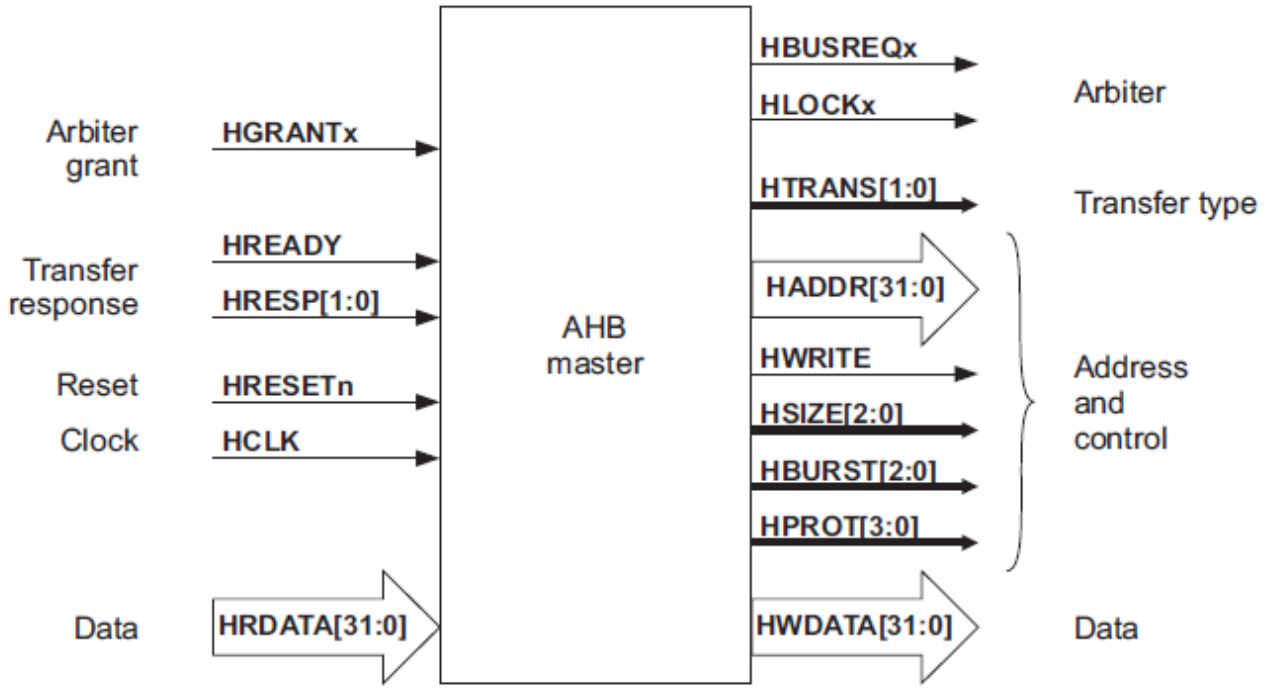


Figure 3-27 AHB bus master interface diagram

Interface diagram – AHB arbiter

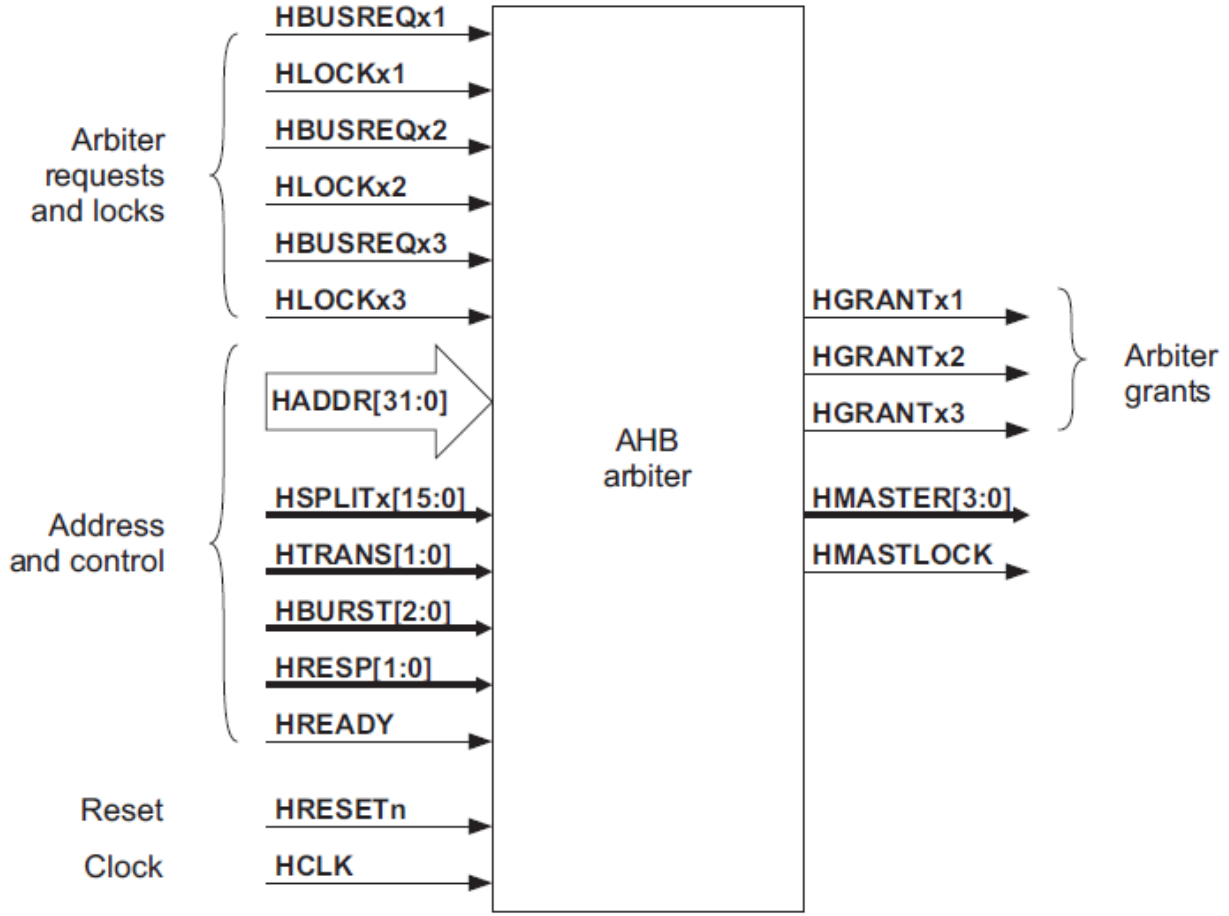


Figure 3-31 AHB arbiter interface diagram

Interface diagram – AHB decoder

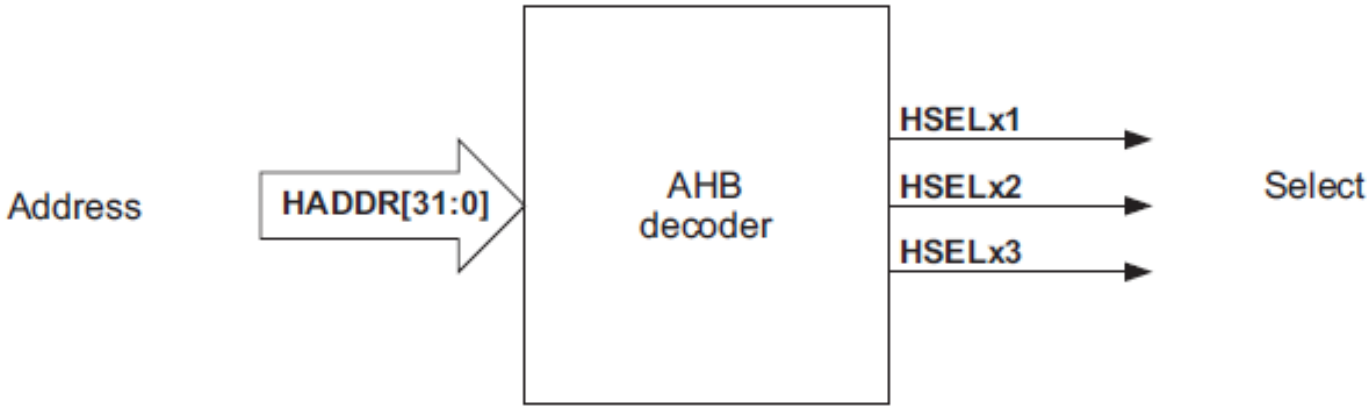


Figure 3-35 AHB decoder interface diagram

Thank You ☺