AMBA® Generic Flash Bus

Protocol Specification

ARM IHI 0083A.b (ID052622)

AMBA Generic Flash Bus
Protocol Specification

Release Information

The following changes have been made to this specification.

Change history

Date	Issue	Confidentiality	Change
06 July 2018	A	Non-Confidential	First release.
24 May 2022	A.b	Non-Confidential	Regularized terminology to be Manager and Subordinate. Clarification regarding reset.

Proprietary Notice

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
LES-PRE-21451 version 2.2

AMBA SPECIFICATION LICENCE

THIS END USER LICENCE AGREEMENT ("LICENCE") IS A LEGAL AGREEMENT BETWEEN YOU (EITHER A SINGLE INDIVIDUAL, OR SINGLE LEGAL ENTITY) AND ARM LIMITED ("ARM") FOR THE USE OF ARM'S INTELLECTUAL PROPERTY (INCLUDING, WITHOUT LIMITATION, ANY COPYRIGHT) IN THE RELEVANT AMBA SPECIFICATION ACCOMPANYING THIS LICENCE. ARM LICENSES THE RELEVANT AMBA SPECIFICATION TO YOU ON CONDITION THAT YOU ACCEPT ALL OF THE TERMS IN THIS LICENCE. BY CLICKING "I AGREE" OR OTHERWISE USING OR COPYING THE RELEVANT AMBA SPECIFICATION YOU INDICATE THAT YOU AGREE TO BE BOUND BY ALL THE TERMS OF THIS LICENCE.

"LICENSEE" means You and your Subsidiaries.

"Subsidiary" means, if You are a single entity, any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by You. A company shall be a Subsidiary only for the period during which such control exists.

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

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

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

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

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

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

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

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

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

4. THE RELEVANT AMBA SPECIFICATION IS PROVIDED "AS IS" WITH NO REPRESENTATION OR WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE, OR THAT ANY USE OR IMPLEMENTATION OF SUCH ARM TECHNOLOGY WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER INTELLECTUAL PROPERTY RIGHTS.

5. NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS AGREEMENT, TO THE FULLEST EXTENT PETMITTED BY LAW, THE MAXIMUM LIABILITY OF ARM IN AGGREGATE FOR ALL CLAIMS MADE AGAINST ARM, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS AGREEMENT (INCLUDING WITHOUT LIMITATION (I) LICENSEE'S USE OF THE ARM TECHNOLOGY; AND (II) THE IMPLEMENTATION OF THE ARM TECHNOLOGY IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS AGREEMENT) SHALL NOT EXCEED THE FEES PAID (IF ANY) BY LICENSEE TO ARM UNDER THIS AGREEMENT. THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

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

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

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

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
AMBA Generic Flash Bus Protocol Specification

Preface

This preface introduces the *AMBA Generic Flash Bus Protocol Specification*. It contains the following sections:

About this specification

This specification describes the *AMBA Generic Flash Bus (GFB) Protocol.* It describes the signals, parameters, and rules required by the protocol. Also, it contains example waveforms to show the operation of the bus.

Intended audience

This document targets the following audiences:

- Software developers using GFB
- Hardware designers implementing GFB

Using this specification

This specification is organized into the following chapters:

Chapter 1 *Introduction*

Introduction to the Generic Flash Bus.

Chapter 2 *Signal descriptions*

Description of the protocol signals and the configurable parameters that define the address bus widths and the data bus widths.

Chapter 3 *Protocol*

Description of the Flash commands and transfer rules. It also provides examples of basic GFB transfers.

Appendix A *Transfer examples*

Provides examples of GFB transfers.

Appendix B *Revisions*

Information about the technical changes between released issues of this specification.

Conventions

This section describes the conventions that this specification uses:

- *Typographic*
- *Signals*
- *Timing diagrams*
- *Numbers* on page xi

Typographic

italic	Highlights important notes, introduces special terminology, and denotes internal cross-references and citations.
bold	Denotes signal names, and is used for terms in descriptive lists, where appropriate.
`monospace`	Used for assembler syntax descriptions, pseudocode, and source code examples. Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.
SMALL CAPITALS	Used for a few terms that have specific technical meanings.

Signals

This specification does not define processor signals, but it does include some signal examples and recommendations.

The signal conventions are:

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

Lowercase n
At the start or end of a signal name denotes an active-LOW signal.

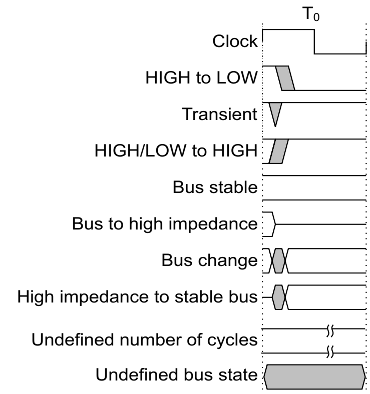
Prefix F
Denotes an *AMBA Generic Flash Bus* (GFB) signal.

Timing diagrams

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

When time periods are labeled, references are to the duration of a full clock cycle. The numbers that are associated with the cycle are only labels and do not indicate the ordinal of the clock cycle. An indefinite time period is indicated by the disconnect symbols that are shown as "Undefined number of cycles" in *Key to timing diagram conventions* on page xi.

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 during that time and does not affect normal operation.



Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in *Key to timing diagram conventions*. If a timing diagram shows a single-bit signal in this way, then its value does not affect the accompanying description.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a `monospace` font, for example `0xFFFF0000`.

Additional reading

This section lists relevant publications from Arm.

See *Arm Developer*, `https://developer.arm.com/docs`, for access to Arm documentation.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this specification, send an email to errata errata@arm.com. Give:

- The title, *AMBA Generic Flash Bus Protocol Specification*.
- The number, ARM IHI 0083A.b.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change. Previous issue of this document included terms that can be offensive. We have replaced these terms.

If you find offensive terms in this document, please contact terms@arm.com.

Chapter 1
Introduction

This chapter provides an overview of the AMBA Generic Flash Bus. This chapter contains the following sections:

1.1 Purpose of the Generic Flash Bus

The AMBA Generic Flash Bus (GFB) simplifies the integration of embedded Flash controllers in subsystems by providing a simple interface between the system and the Flash. GFB exists on the boundary between the Manager side of the Flash controller and the Subordinate side, as shown in Figure 1-1. The Manager side has a generic Flash controller, which has general functions that are supported by most eFlash macros.

The Subordinate side has the process-dependent Flash macro that is used for a specific implementation. GFB serves as the data path for accessing the flash memory resources, control related accesses are handled over other interfaces. This facilitates reusability of the general functions with different processes.

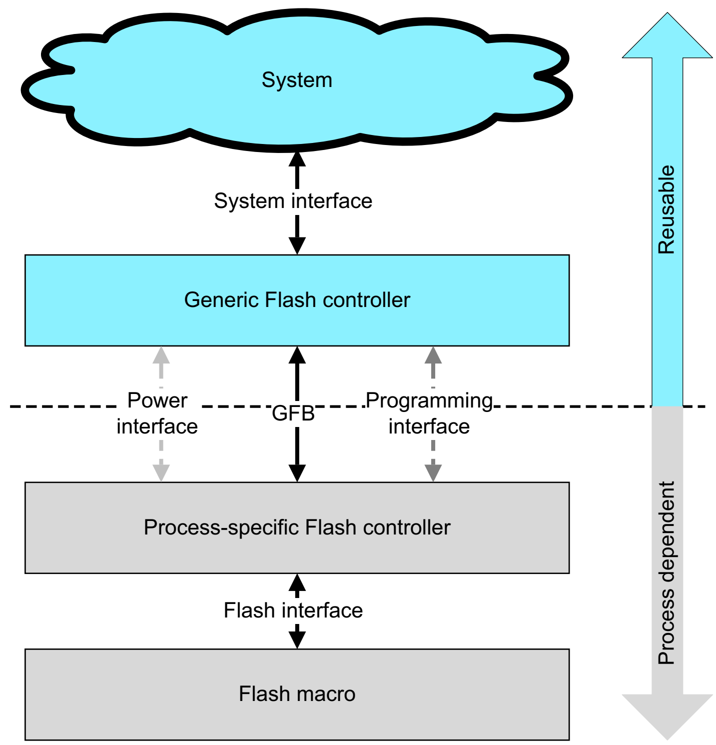


Figure 1-1 System architecture using GFB

1.2 General description

GFB is a point-to-point connection between a single Manager and a single Subordinate. It has parallel address and data buses, with additional control signals, that enable the Manager to send commands to the Subordinate. Figure 1-2 shows the connections that are required.

The Manager can send the following commands:

- IDLE
- READ
- WRITE
- ROW WRITE
- ERASE
- MASS ERASE

If the Manager needs to stop a long-running command due to an emergency in the system, then the Manager can cancel a command that it has sent.

If the Subordinate needs more time to execute a command, then it can delay the command.

The Manager sends the transfers to the Subordinate in a pipeline sequence, with a separate address and data phase for each transaction.

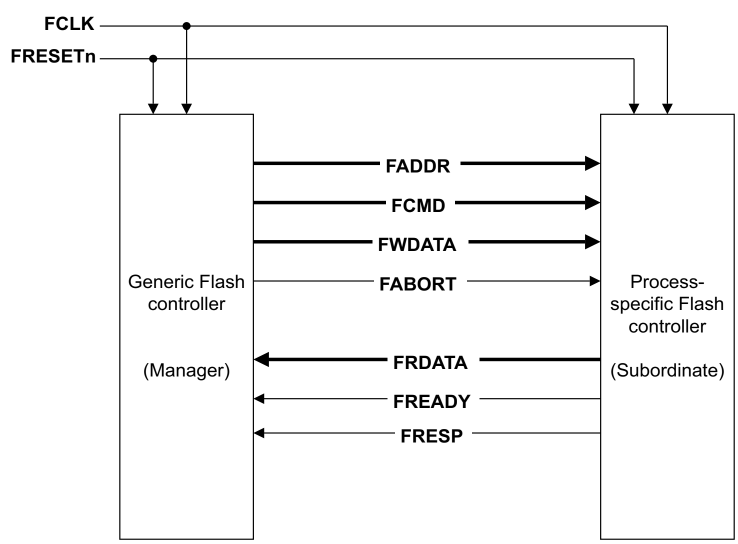


Figure 1-2 Generic Flash Bus connections

Chapter 2
Signal descriptions

This chapter describes the protocol signals and the configurable parameters that define the address width and the data widths. It contains the following sections:

- *GFB signals* on page 2-20
- *Signal parameters* on page 2-22

2.1 GFB signals

Table 2-1 lists the GFB signals.

Table 2-1 Signal list

Signal	Source	Width	Description
FCLK	Clock source	1	Clock input for both Manager and Subordinate.
FRESETn	Reset source	1	Active-LOW reset input for both Manager and Subordinate. The reset signal can be asserted asynchronously, but de-assertion must be synchronous with a rising edge of **FCLK**.
FADDR	Manager	**FADDR_WIDTH**	Byte-based address. • For READ commands, this must be aligned to the read data width. • For WRITE and ROW WRITE commands, this must be aligned to the write data width. • For IDLE, ERASE, and MASS ERASE commands, FADDR alignment is not necessary. **FADDR_WIDTH** determines the size of the address space that is accessible. A minimum width of 12 bits is recommended.
FCMD	Manager	3	Command for the Subordinate to execute: `0b000` IDLE command. `0b001` READ command. `0b010` WRITE command. `0b011` ROW WRITE command. `0b100` ERASE command. `0b111` MASS ERASE command. The `0b101` and `0b110` commands are invalid.
FWDATA	Manager	**FWDATA_WIDTH**	Write data to the Subordinate. **FWDATA_WIDTH** determines the width of the write data path that is defined by the attached Flash macro. The width must be a power of 2. A minimum width of 32 bits is recommended.
FABORT	Manager	1	Request to cancel a previously accepted command. **FABORT** is an OPTIONAL signal for the Manager and the Subordinate. While the Subordinate responds to a command, it also samples the **FABORT** signal. If the Subordinate detects a HIGH value, then it initiates an abort. Aborting can take several cycles before it returns to a state when it can accept new commands. If **FABORT** is not used in a system, it can be tied to LOW for Subordinate devices and can be left open for Manager devices.

Table 2-1 Signal list (continued)

Signal	Source	Width	Description
FRDATA	Subordinate	**FRDATA_WIDTH**	Read data from the Subordinate. **FRDATA_WIDTH** determines the width of the read data path that is defined by the attached Flash macro. The width must be a power of 2. A minimum width of 32 bits is recommended.
FREADY	Subordinate	1	Command complete indication. **FREADY** HIGH means that the Subordinate completed the previous command and the result is available. It also indicates that the Subordinate is ready to accept a command. **FREADY** LOW means that Subordinate is processing an ongoing command or doing housekeeping activities.
FRESP	Subordinate	1	Error indication from the Subordinate for a previously accepted command.

2.2 Signal parameters

GFB is configurable so it can support a wide variety of Flash macros.

2.2.1 FADDR_WIDTH

The memory size of the Flash is not required to be power-of-two-based, however the system needs to be able to address the full flash memory. Therefore, the **FADDR_WIDTH** parameter needs to be set to provide a power-of-two-based address range that fully addresses the whole Flash content. The width must be set to include all memory regions in the Subordinate that are available for the Manager to access.

Figure 2-1 shows an example where a 1MB flash memory with 8KB of extra information requires a 2MB address space. **FADDR_WIDTH** is therefore set to 21. In this case, **FADDR[20]** can be used to select between main flash memory and extra memory area.

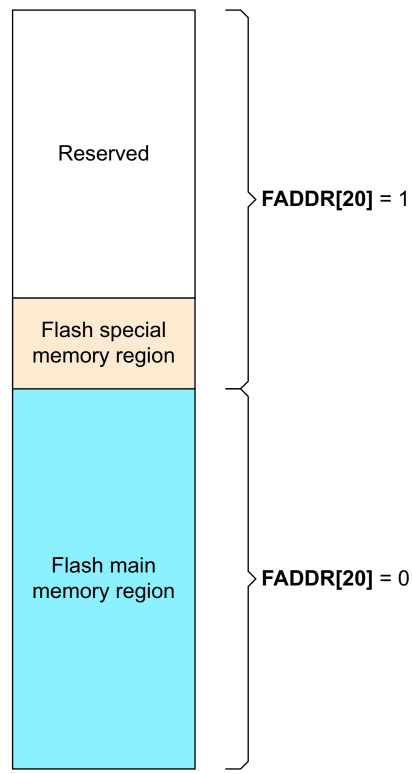


Figure 2-1 Address map example

2.2.2 FRDATA_WIDTH and FWDATA_WIDTH

Flash macros can have input and output data widths that are different. For example, 128-bit wide memory is readable in 128-bit words, but writable in 32-bit words. For this reason, the **FRDATA** and **FWDATA** widths can be set to different values.

Chapter 3
Protocol

This chapter describes the Flash commands and transfer rules. It also provides examples of basic GFB transfers.

3.1 Flash commands

The protocol includes a set of general commands to accommodate the capabilities of the Flash devices. All commands are issued by the Manager using **FCMD[2:0]** and accepted by the Subordinate when **FREADY** is asserted.

IDLE

When **FCMD** indicates the IDLE command, it means that the Manager is not initiating any new commands to the Subordinate.

READ

The READ command is used to read the Flash contents at the address on **FADDR**. The Subordinate can signal when the data is available using **FREADY**.

Only full data width reads are allowed, no smaller, or larger data widths are possible. If the Subordinate responds with an error, then the read data is invalid.

WRITE

The WRITE command is used to program the Flash word at the address on **FADDR**. The Manager provides the data on **FWDATA** in the data phase of the transfer. The Subordinate can extend the period in which write data is made available, using the **FREADY** signal. Only full data width writes are allowed, no smaller, or larger data widths are possible. If the Subordinate responds with an error, it indicates that the programming failed and the data that is stored at address location **FADDR** is UNDEFINED.

ROW WRITE

The ROW WRITE is identical to the WRITE command, except that the Manager can indicate this as a hint to the Subordinate to keep programming active after it completes. A Manager issuing a series of writes might use the ROW WRITE command to improve timing and power efficiency. If a ROW WRITE is being executed by the Subordinate and the next command is also a ROW WRITE, the Subordinate continues programming. In all other cases, the Subordinate must stop programming and perform a recovery, which might mean that the beginning of the execution of the next command is delayed. There are no restrictions on **FADDR** for consecutive ROW WRITE commands. Depending on the internal structure of the Subordinate, it might cease programming and start a new execution when consecutive ROW WRITE commands are accepted.

ERASE

Flash memory normally supports erasing the memory content that is within a set of data words, referred to as pages. When this command is asserted, the page indicated by **FADDR** is erased by the Subordinate. **FADDR** is not necessarily aligned to the page size since it depends on the internal structure of the Subordinate. The Manager might point to any location that is in the page that is to be deleted. The ERASE command needs to program flash memory to its initial content. The execution time depends on the Subordinate, but it is typically longer than the WRITE command. If the Subordinate responds to ERASE with **FRESP** asserted, it indicates to the Manager that ERASE has failed and the page content is UNDEFINED.

MASS ERASE

This command clears the whole memory area, and is an alternative method for erasing the content of the flash memory. This command allows all pages to be erased in parallel, which can speed up an update procedure when the full Flash content needs to be rewritten. This command clears the whole memory area that is mapped to the interface.

3.2 Basic transactions

Figure 3-1 shows some examples of simple read operations.

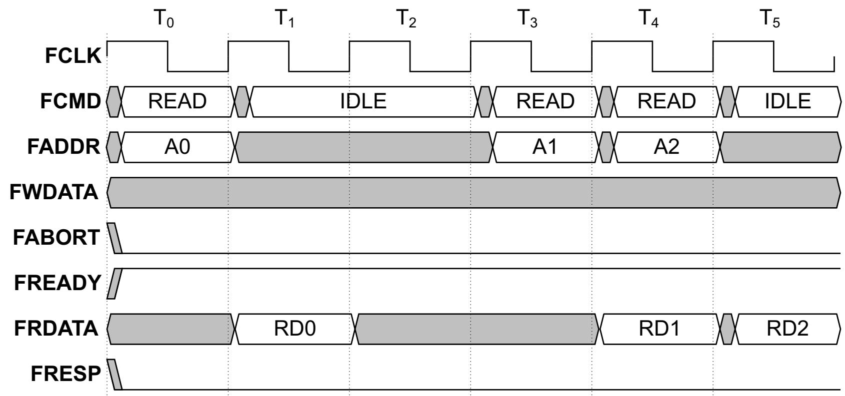


Figure 3-1 Basic read transactions

During time:

T_0 The first READ command is driven by the Manager for address A0. The first READ is accepted by the Subordinate with **FREADY** HIGH. This is the address phase of the READ command.

T_1 The Subordinate finishes the transfer by driving **FREADY** HIGH. It provides the read data RD0 which is sampled by the Manager. This is the data phase of the READ command. The Manager has no other transaction to initiate towards the Subordinate and therefore sets **FCMD** to IDLE.

T_2 The Manager continues to send an IDLE.

T_3 The Manager initiates a READ command to address A1, which is accepted because **FREADY** is HIGH.

T_4 The READ command during T3 is executed in the Subordinate within one clock cycle, and the resulting data, RD1 is present on **FRDATA**.

The Manager sends a READ for the address A2 and it is accepted because **FREADY** is still HIGH.

This cycle is the data phase of the second read and the address phase of the third read.

T_5 The RD2 response from the Subordinate is received. This is sampled by the Manager at the rising edge of the clock of the next cycle.

Figure 3-2 on page 3-26 shows an example of a write transaction. For WRITE commands, the time duration of T_3 might be several thousand **FCLK** cycles.

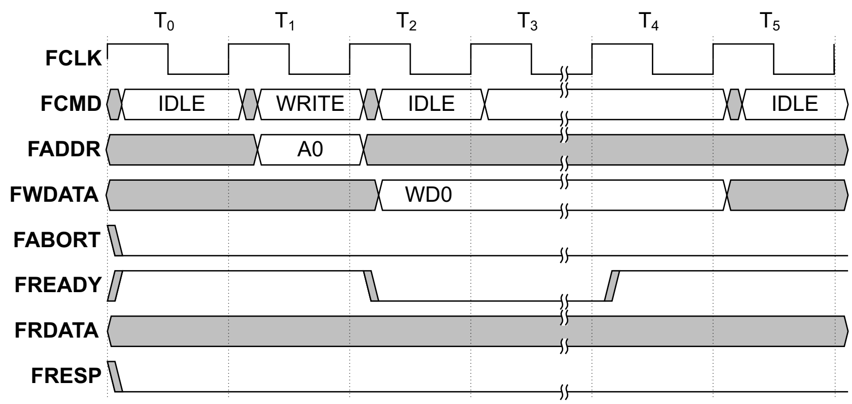


Figure 3-2 Basic write transaction

During time:

T_0 IDLE command is accepted by the Subordinate, so no operation is done.

T_1 **FREADY** signal remains HIGH.

The Manager initiates a WRITE command at address A0.

T_2 The Subordinate pulls **FREADY** LOW to indicate that it requires additional cycles to complete the command.

The Manager provides the write data of WD0 on **FWDATA**.

The Manager has no other command to initiate so it drives the **FCMD** bus IDLE.

The Subordinate can sample the **FWDATA** at the rising edge of the clock of T3. This is the start of the data phase of the WRITE command.

T_3 The Subordinate keeps the **FREADY** LOW for a long period, extending the data phase of the transfer.

The Manager must keep **FWDATA** stable through this period.

T_4 The write completes successfully, and the Subordinate asserts **FREADY** to indicate that it is ready to accept new commands.

T_5 The Manager has no further commands to send, so drives **FCMD** to IDLE.

3.3 Protocol operation and rules

This section describes GFB operation and protocol rules. Bulleted statements are protocol rules; other text is informative.

3.3.1 Reset and initialization

FRESETn is asserted to reset both the Manager and Subordinate interfaces. During the reset period, the Flash device can be initialized. When **FRESETn** is deasserted, the Subordinate can hold **FREADY** LOW to prevent acceptance of any commands until Flash initialization is complete.

FRESETn can be asserted at any time, which cancels any current activity on the GFB.

When **FRESETn** is asserted:

- **FCMD** must be IDLE.
- **FABORT** must be LOW.
- **FREADY** must be LOW.
- **FRESP** must be LOW.

The earliest point after reset when **FCMD**, **FABORT**, **FREADY**, and **FRESP** can change is at a rising **FCLK** edge after **FRESETn** is HIGH.

3.3.2 Normal operation

All signals are synchronous to the rising edge of **FCLK**, except **FRESETn** which can be asserted asynchronously.

FADDR **FADDR** is sampled in the same cycle as **FCMD**.

- **FADDR** must have a legal value when **FCMD** is not IDLE.
- **FADDR** must be aligned to the read data width for READ commands.
- **FADDR** must be aligned to the write data width for WRITE and ROW WRITE commands.
- When **FCMD** is not IDLE, **FADDR** must not change in the next cycle if **FREADY** is LOW and **FRESP** is LOW.

FCMD Commands are signaled by the Manager using **FCMD** and accepted by the Subordinate when **FREADY** is HIGH. A command is then ongoing until the next cycle when **FREADY** is HIGH.

- **FCMD** must always signal a legal value, see Table 2-1 on page 2-20.
- If **FCMD** is not IDLE, it must not change in the next cycle if **FREADY** is LOW and **FRESP** is LOW.

FWDATA **FWDATA** is active in the cycle after **FREADY** is HIGH and **FCMD** is WRITE or ROW WRITE.

- **FWDATA** must be a legal value when it is active.
- When **FWDATA** is active, it must not change in the next cycle if **FREADY** is LOW and **FRESP** is LOW.

FRDATA After a READ command has been accepted, **FRDATA** is active in the next cycle that **FREADY** is HIGH.

- **FRDATA** must be a legal value when it is active.

FREADY The **FREADY** signal is always valid as it can be sampled every cycle.

3.3.3 Error response

When the Subordinate encounters an error condition while executing a command, it can signal an error response using **FRESP**. This cancels the current command.

If an error response is received for a WRITE, ROW WRITE, ERASE, or MASS ERASE command, then the programming failed and the content of the memory is UNDEFINED.

A Subordinate is not required to support every GFB command. The Subordinate can respond with an error response if it receives a command which is not supported. It can also send an error response for an access to an invalid address location.

An error response always takes two cycles. The first cycle has **FREADY** LOW and **FRESP** HIGH; the second cycle has **FREADY** HIGH and **FRESP** HIGH.

- **FRESP** can only be asserted if there is an ongoing non-IDLE command.
- If **FRESP** is HIGH and **FREADY** is LOW, then **FRESP** must be HIGH and **FREADY** must be HIGH in the next cycle.
- When **FRESP** is asserted, **FRDATA** is not required to be legal.
- When **FRESP** and **FREADY** are asserted, **FWDATA** is not required to be legal.

3.3.4 Aborting commands

The Manager can request to abort an ongoing command by asserting **FABORT**.

The Subordinate can ignore the request or accept the request and issue an error response. If **FABORT** goes HIGH in the same cycle as **FREADY** is HIGH, then the abort request is ignored by the Subordinate.

- **FABORT** can only be asserted if there is an ongoing non-IDLE command.
- If **FABORT** is HIGH, it must be HIGH in the next cycle if **FREADY** is LOW.

3.3.5 Housekeeping

The Subordinate might need to execute internal processes that are not initiated by the Manager. For example, these activities can include initialization, configuration, self-test, and are commonly referred to as housekeeping.

During housekeeping, the Subordinate can keep **FREADY** LOW. This is typically done when the ongoing command is IDLE.

Appendix A
Transfer examples

This appendix describes examples of GFB transfers.

A.1 Successful transfers

The following sections provide examples of successful transfers.

A.1.1 ERASE

Figure A-1 shows an ERASE, READ, and WRITE sequence of commands.

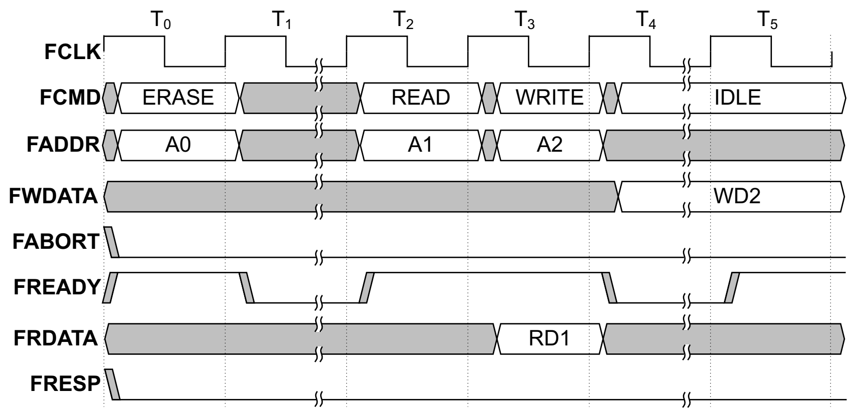


Figure A-1 ERASE command

In Figure A-1, the duration in T_1 might be several million **FCLK** cycles for an ERASE command and several thousand **FCLK** cycles in T_4 for a WRITE command.

During time:

T_0 The Manager issues an ERASE command at address A0.

T_1 Subordinate accepts the ERASE command and sets **FREADY** to LOW, indicating that it requires additional cycles to complete the operation.

The ERASE command takes a long time to execute.

T_2 The Manager presents a READ command on the bus at address A1.

When the ERASE is finished successfully, the READ is also accepted by the Subordinate.

T_3 The Manager issues a WRITE command at address A2.

The Subordinate returns the data for the READ.

T_4 The WRITE command takes long time to execute. The Manager holds the WD2 data on **FWDATA** while **FREADY** is LOW.

T_5 The Subordinate finishes the WRITE command successfully and the Manager has no more commands to send.

A.1.2 READ delayed

Figure A-2 shows READ commands which are delayed by the Subordinate by differing numbers of cycles.

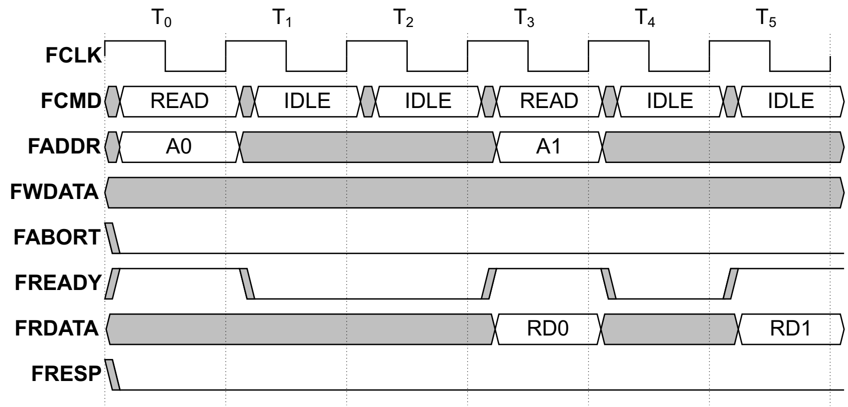


Figure A-2 READ command that is delayed by Subordinate

During time:

T_0 The Manager sends a READ command that is accepted by the Subordinate at the rising edge of the clock at the end of T_0.

T_1 The Subordinate delays the READ by two cycles by taking **FREADY** LOW. The Manager has no other commands to send.

T_3 The Subordinate returns the data on **FRDATA** successfully and takes **FREADY** HIGH. The Manager sends another READ command at the same time.

T_4 The Subordinate delays the READ by one cycle.

T_5 The READ completes with **FREADY** HIGH. The Manager has no further commands to send.

A.1.3 Initialization and back-to-back read

Figure A-3 shows the Subordinate holding **FREADY** LOW while it initializes after reset. The Manager signals a READ command during this period.

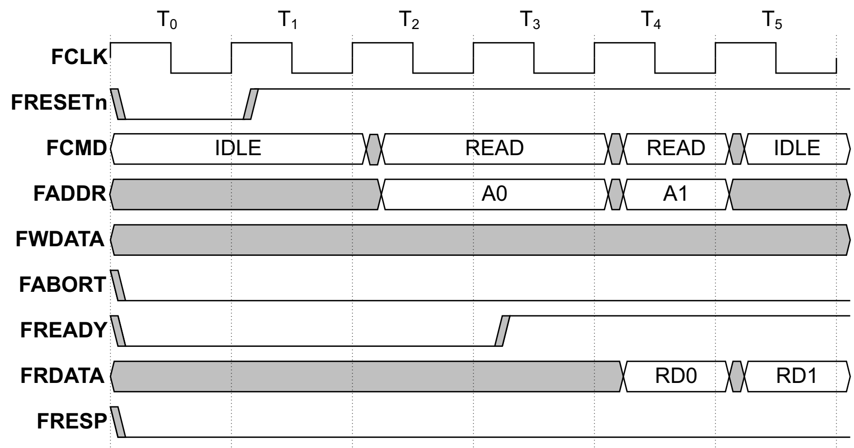


Figure A-3 Initialization and back-to-back READ

During time:

T_0 **FRESETn** is LOW so both Manager and Subordinate are reset. All signals are in their initial state.

T_1 **FRESETn** is deasserted. The Manager and Subordinate are still in reset until the rising edge of the clock of at the start of T_2.

T_2 The Manager can send the first READ command. The Subordinate still needs some initialization time, so it holds **FREADY** LOW.

T_3 The Subordinate has finished initialization so normal operation can start, READ commands are accepted and executed by the Subordinate.

A.1.4 IDLE with housekeeping

Figure A-4 shows when the Subordinate device performs internal housekeeping due to an external event. For example, this event could be a self-test request or a reconfiguration of internal registers.

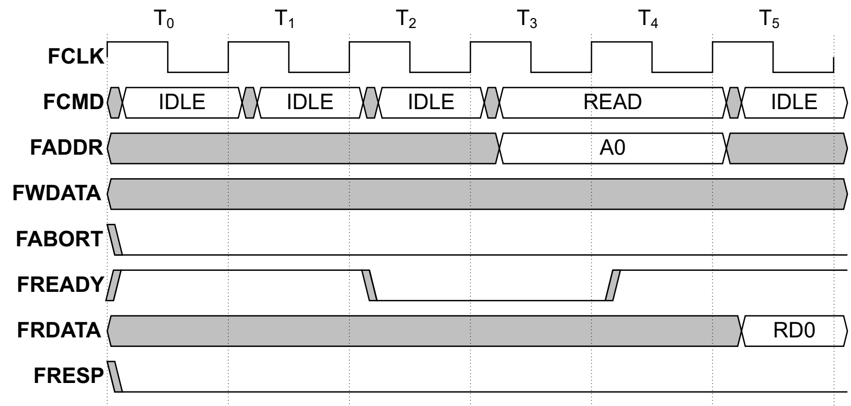


Figure A-4 Housekeeping IDLE command

During time:

T_0 The Manager sends IDLE commands and the Subordinate accepts them with **FREADY** HIGH.

T_2 The Subordinate must do housekeeping and pulls **FREADY** LOW after an accepted IDLE command. The Manager still has no commands to send.

T_3 The Manager sends a READ command, but it is delayed by the Subordinate due to the ongoing housekeeping activities.

T_4 The housekeeping is finished, and the pending READ command is accepted.

T_5 The Subordinate returns a successful READ response.

A.1.5 ROW WRITE command back-to-back

Figure A-5 shows a sequence of ROW WRITE commands sent by the Manager. The commands arrive in time for the Subordinate so it might use this sequence to program the memory in a more efficient way.

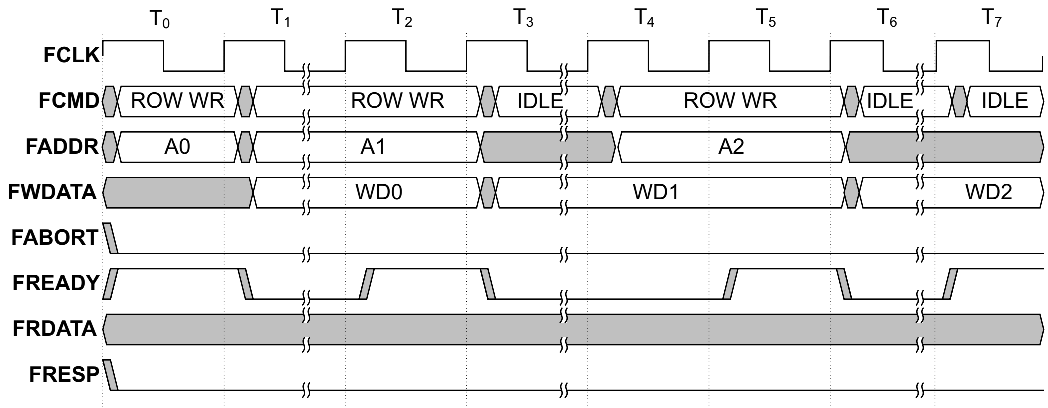


Figure A-5 ROW WRITE command back-to-back

During time:

T_0 The Manager sends a ROW WRITE command which is accepted by the Subordinate.

T_1 The Subordinate executes the programming and pulls **FREADY** LOW. The Manager prepares the next ROW WRITE command on the bus.

T_2 The Subordinate finishes the ROW WRITE and accepts the next ROW WRITE command to **FADDR** A1.

T_3 The Subordinate executes the programming and pulls **FREADY** LOW. The Manager sends IDLE commands in the meantime.

T_5 The Subordinate finishes the ROW WRITE to **FADDR** A1.

The Subordinate accepts the next ROW WRITE command that is presented by the Manager. This might still be interpreted by the Subordinate as part of the same ROW WRITE sequence.

T_6 The Subordinate executes the third ROW WRITE.

The Manager has no other commands to send.

T_7 The Subordinate finishes the programming and accepts the IDLE command sent by the Manager.

The ROW WRITE sequence is finished.

A.1.6 ROW WRITE interrupted by other command

Figure A-6 shows a ROW WRITE sequence that is interrupted by a READ command. The ROW WRITE following the READ must be interpreted as the start of a new ROW WRITE sequence.

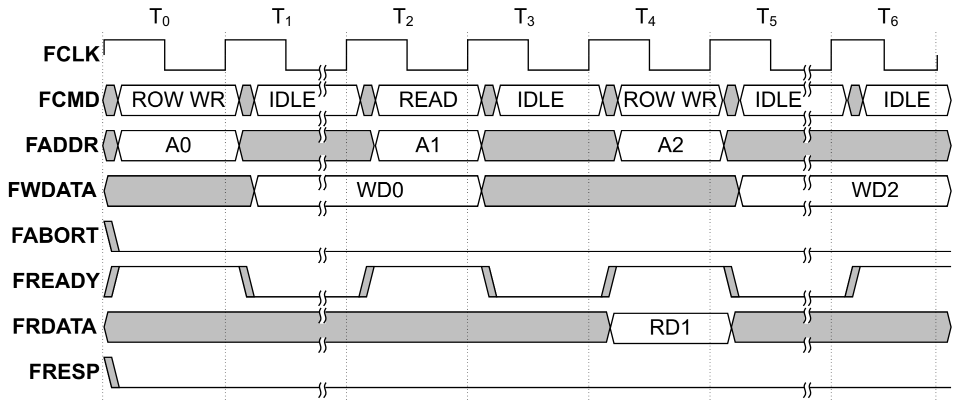


Figure A-6 ROW WRITE command that is interrupted by READ

During time:

T_0 The Manager sends a ROW WRITE command, which is accepted by the Subordinate.

T_1 The Subordinate executes the programming and pulls **FREADY** LOW.

The Manager sends IDLE commands on the **FCMD**.

T_2 The Subordinate finishes the ROW WRITE and the Manager sends a READ command at the same time. The Subordinate accepts the READ which terminates the ROW WRITE sequence.

T_3 The Subordinate executes the READ command and pulls **FREADY** LOW.

The Manager sends IDLE commands in the meantime.

T_4 The Subordinate finishes the READ and the Manager sends a new ROW WRITE command to **FADDR** A2. This needs to be interpreted by the Subordinate as the start of a new ROW WRITE sequence.

T_5 The Subordinate executes and finishes the new ROW WRITE command while the Manager has no other commands to send.

A.2 Errored transfers

The following sections provide examples of transfers when the Subordinate indicates an error response.

A.2.1 ERASE error

Figure A-7 shows an ERASE command that is responded with a 2-cycle error response by the Subordinate. The READ command following the error response is executed successfully.

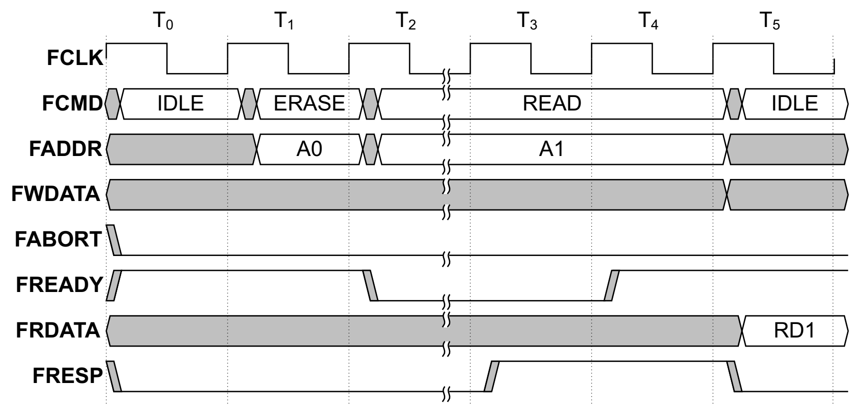


Figure A-7 ERASE command error

During time:

T_1 The Manager sends an ERASE command. The ERASE is accepted by the Subordinate.

T_2 The Subordinate executes the ERASE but it fails.

T_3 The Subordinate sets **FRESP** HIGH, to start the 2-cycle error response.

T_4 The Subordinate sets **FREADY** HIGH to complete the 2-cycle error response. This response indicates that the Subordinate failed to properly erase the page that contains address A0. The memory content of the flash is therefore UNDEFINED. The following READ command is executed successfully and **FRDATA** is valid in T_5.

A.2.2 WRITE error

Figure A-8 shows a WRITE command that fails in the Subordinate. Due to the error, the Manager changes the following WRITE command to an IDLE.

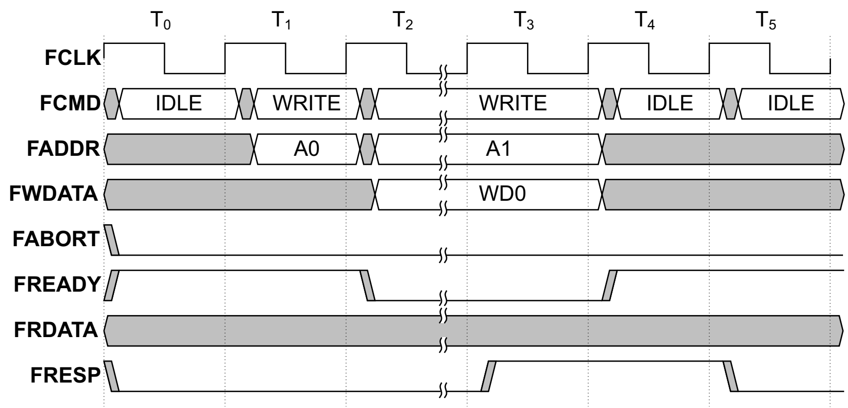


Figure A-8 WRITE command error

During time:

T_0 The Manager sends an IDLE.

T_1 The Manager sends a WRITE command for address A0.

T_2 The Subordinate executes the WRITE command, but it fails.

T_3 The Subordinate sets **FRESP** HIGH, to start the 2-cycle error response.

T_4 The Subordinate sets **FREADY** HIGH to complete the 2-cycle error response. This response indicates that the Subordinate failed to properly write A0, so the memory content is UNDEFINED.

The Manager issues an IDLE command when detecting **FRESP** HIGH and **FREADY** LOW, although the protocol permits it to issue any command.

A.3 Aborted transfers

The following sections provide examples of transfers that were successfully aborted.

A.3.1 READ aborted

Figure A-9 shows a READ command that is requested to be aborted by the Manager. The Subordinate aborts the command and returns a 2-cycle error response. Due to the error, the Manager changes the following WRITE command to IDLE.

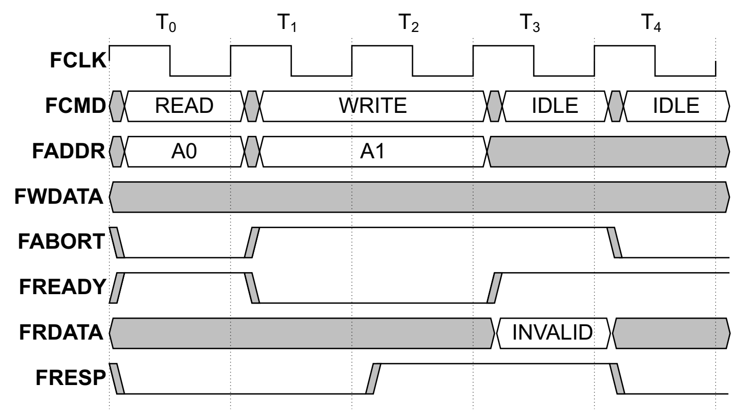


Figure A-9 READ command aborted

During time:

T_0 The Manager issues a READ command which is accepted by the Subordinate.

T_1 The Manager asserts **FABORT** to request that the READ is canceled.

T_2 The Subordinate aborts the command and takes **FRESP** HIGH to signal an error.

T_3 The Subordinate completes the 2-cycle error response by taking **FREADY** HIGH. The Manager cancels the pending WRITE to A1 by changing **FCMD** to IDLE.

A.3.2 WRITE aborted

Figure A-10 shows a WRITE command that is aborted by the Manager. The Subordinate extends the abort request while it determines whether the WRITE can safely be aborted.

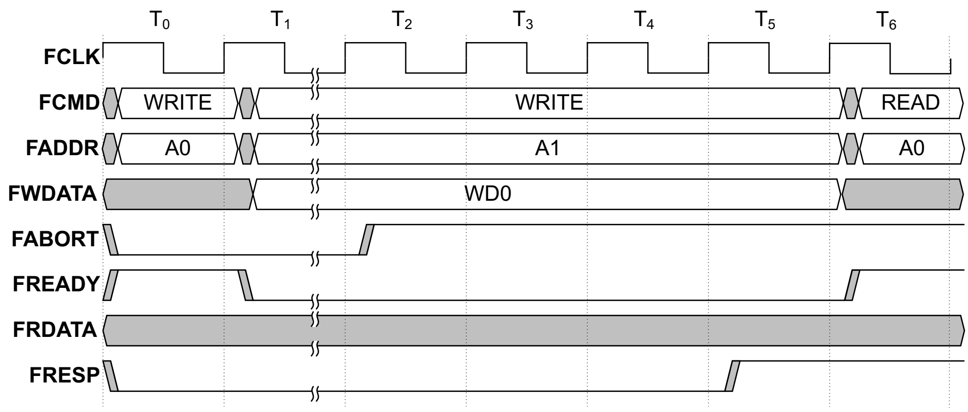


Figure A-10 WRITE command aborted

During time:

T_0 The Manager issues a WRITE command which is accepted by the Subordinate.

T_1 The Subordinate takes time to execute the WRITE and it pulls **FREADY** LOW.

T_2 The Manager sets **FABORT** HIGH to abort the transfer. The Subordinate detects that **FABORT** is HIGH and starts its abort mechanism. The Manager must keep the WD0 valid even if the **FABORT** is asserted in the data phase of the transfer.

T_3 The abort might take several clock cycles to finish properly.

T_5 The Subordinate sets **FRESP** HIGH to start a two-cycle error response.

T_6 The Manager issues a READ command of address A0, to check the results of the aborted WRITE.

The Manager is not required to hold **FWDATA** stable after **FRESP** is asserted.

A.3.3 WRITE aborted immediately

Figure A-11 shows a WRITE command that is immediately aborted by the Manager.

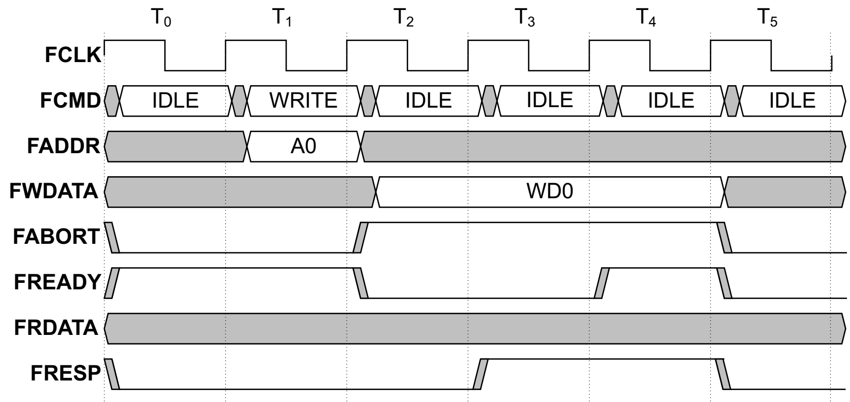


Figure A-11 WRITE command aborted immediately

During time:

T_1 The Manager issues a WRITE command which is accepted by the Subordinate.

T_2 The Manager issues an abort request by setting **FABORT** HIGH.

T_3 The Subordinate responds with a two-cycle error, indicated by **FRESP** HIGH.

A.4 Abort requests ignored

The following sections provide examples of transfers that are not successfully aborted.

A.4.1 READ abort ignored

Figure A-12 shows a READ command which the Manager requests to abort. The abort is not handled by the Subordinate and execution of further commands continues normally.

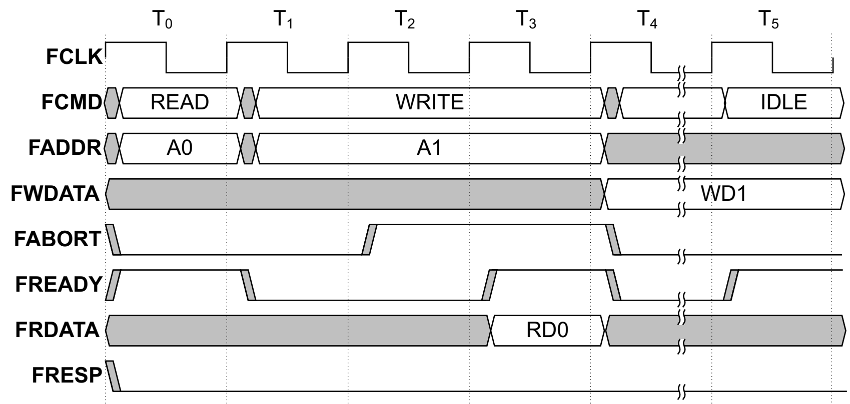


Figure A-12 READ command abort ignored

During time:

T_0 The Manager sends a READ command which is accepted by the Subordinate.

T_1 The Manager sends a WRITE command.

T_2 The Manager sets **FABORT** HIGH to abort the READ transfer to A0.

T_3 The Subordinate detects that **FABORT** is HIGH but it is too late to start its abort mechanism.

The Subordinate provides the read data, RD0.

T_4 The Manager receives the read data and was unsuccessful in aborting the READ command. The Manager sets **FABORT** LOW and continues to drive the write data on **FWDATA** until the Subordinate executes the WRITE.

Note

If the Manager asserts **FABORT** when **FREADY** is HIGH, then the Subordinate ignores the abort request.

A.4.2 WRITE abort ignored

Figure A-13 shows a write command that is already committed in the Subordinate before the Manager tries to abort it.

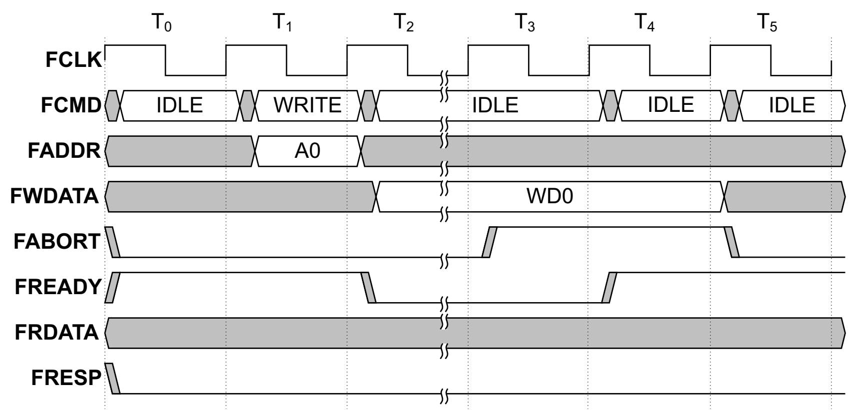


Figure A-13 WRITE command abort ignored

During time:

T_1 The Manager sends a WRITE command after an IDLE cycle.

T_2 The Manager sends IDLE commands. The Subordinate executes the WRITE and pulls **FREADY** LOW.

T_3 The Manager sets **FABORT** HIGH to request that the WRITE is aborted.

The Subordinate detects that **FABORT** is HIGH, but it is too late to start its abort mechanism, so **FRESP** remains LOW.

T_4 The WRITE operation to the memory succeeded, indicated by **FREADY** HIGH, so the content was updated. The Manager detects that **FRESP** is LOW so its abort request was unsuccessful.

T_5 The Manager sets **FABORT** LOW as there are no more commands to abort.

A.4.3 Aborting at FRESP

Figure A-14 shows the scenario when the Manager requests an abort but the Subordinate already started to respond with an error.

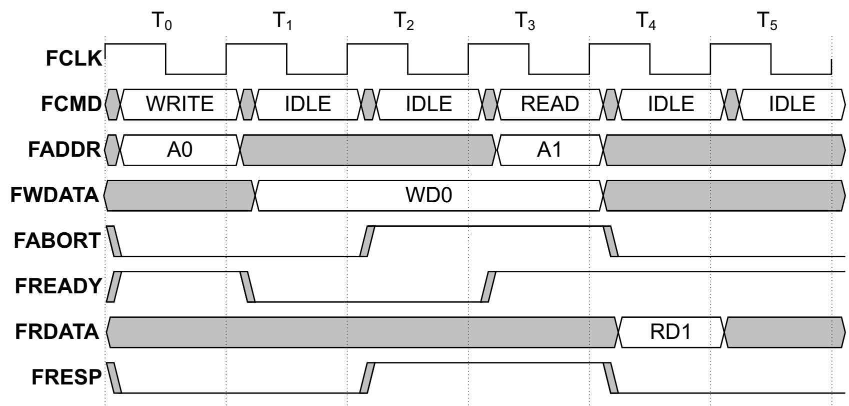


Figure A-14 Aborting too late with FRESP

During time:

T_0 The Manager sends a WRITE command which the Subordinate accepts.

T_1 The Subordinate starts executing the WRITE and takes **FREADY** LOW.

T_2 The Manager sets **FABORT** HIGH to abort the WRITE transfer to address A0.

The Subordinate sets **FRESP** HIGH to indicate a transfer failure, regardless of the abort request.

T_3 The Manager issues a READ command that is accepted by the Subordinate in the second cycle of the error response.

T_4 The Subordinate returns the data successfully. The Manager pulls **FABORT** LOW and has no other commands to send

Appendix B
Revisions

This appendix describes the technical changes between released issues of this specification.

Table B-1 Issue A

Change	Location
First release of Version A.	N/A

Table B-2 Differences between issue A and issue A.b

Change	Location
Terminology update. Regularized terminology to use Manager to indicate the agent that initiates transactions and Subordinate to indicate the agent that receives and responds to requests.	Throughout the specification.
Reformatting of the signal and properties tables.	*GFB signals* on page 2-20.
Clarification regarding reset.	*Reset and initialization* on page 3-27.