GBANA DESIGN DOC

Version 0.1.0

April 25, 2025

Contents

[1] ARM DDI 0029G 3-4

[2] ARM DDI 0029G 3-5

[3] ARM DDI 0029G 3-6

[4] ARM DDI 0029G 3-7

1. Preface

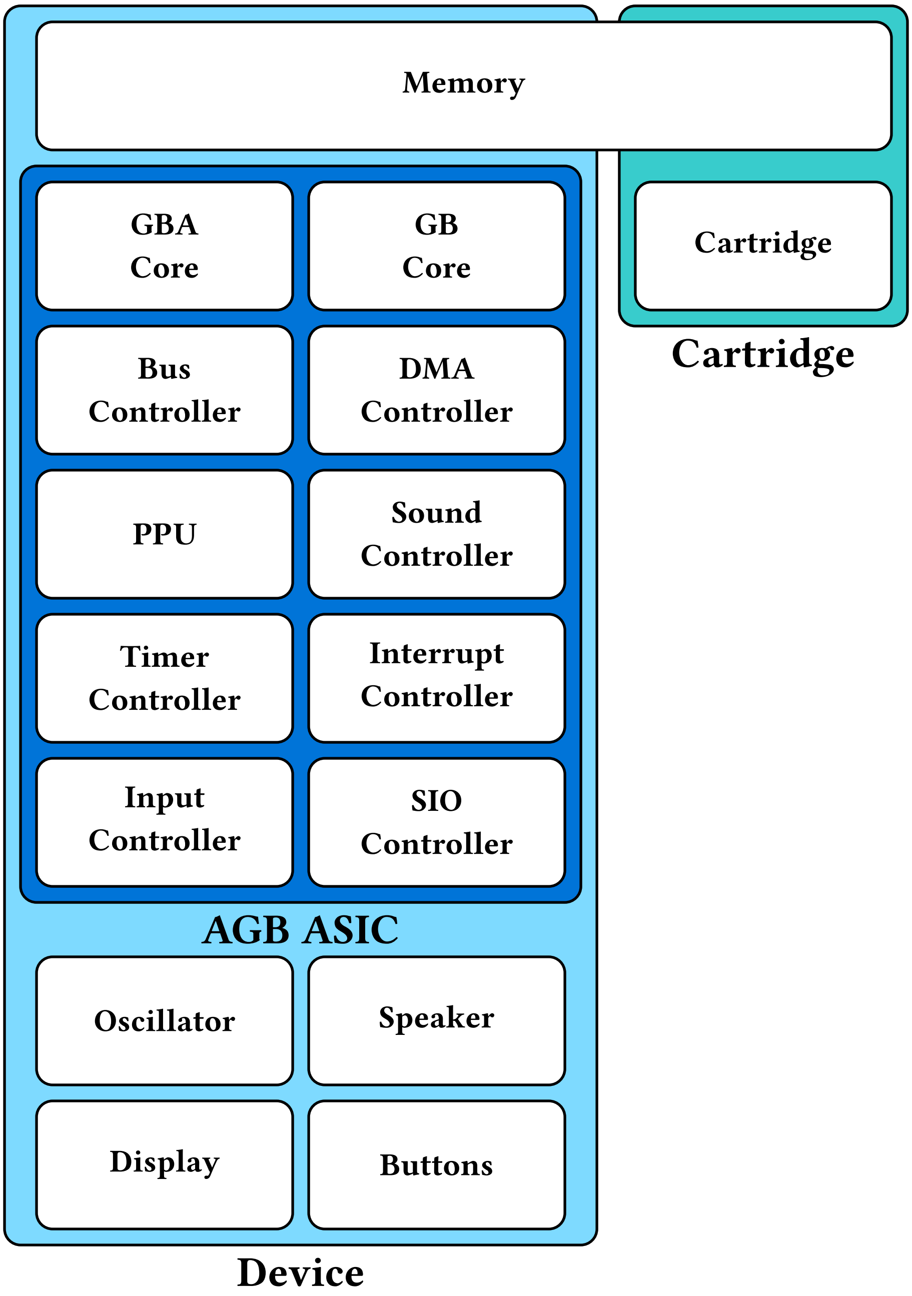
GBANA is a Game Boy Advance emulator. Other GBA emulators exist and they make games look and feel as good as they did on the original GBA. GBANA will make them look and feel *better* than they did on the original GBA.

2. Classes

Each file represents a class. The *singleton classes* correspond to the components in the block diagram on the next page. The *helper classes* contain procedures that are used by the singleton classes. The *instance classes* contain objects that are instantiated and used by the other classes.

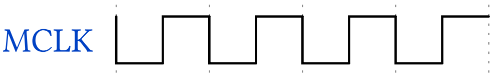
Name	File	Type	Description
GBA Core	`gba_core.odin`	Singleton Class	Emulating the ARM7TDMI core inside the AGB chip.
GB Core	`gb_core.odin`	Singleton Class	Emulating the SM83 core inside the AGB chip.
Bus Controller	`bus_controller.odin`	Singleton Class	Emulating the bus control logic inside the AGB chip.
DMA Controller	`dma_controller.odin`	Singleton Class	Emulating the DMA controller inside the AGB chip.
Memory Controller	`memory_controller`	Singleton Class	Emulating the memory controller inside the AGB chip.
Cpu	`cpu.odin`	Helper Class	Emulating the shared logic between the components inside the AGB chip.
Memory	`memory.odin`	Singleton Class	Emulating both the internal and external memory of the gba.
Buttons	`buttons.odin`	Singleton Class	Emulating the buttons of the GBA.
Cartridge	`cartridge.odin`	Singleton Class	Emulating a GBA cartridge.
Display	`display.odin`	Singleton Class	Emulating the display of the GBA.
GBA Isa	`gba_isa.odin`	Helper Class	Defining the ISA of the GBA core.
GB Isa	`gb_isa.odin`	Helper Class	Defining the ISA of the GB core.
PPU	`ppu.odin`	Singleton Class	Emulating the GBA PPU.
Line and Bus	`line_and_bus.odin`	Instance Class	Emulating the behavior of a line and a bus.
SIO Controller	`sio_controller.odin`	Singleton Class	Emulating the SIO controller.
Speakers	`speakers.odin`	Singleton Class	Emulating the speakers.
Util	`util.odin`	Helper Class	General utilities.

3. Block Diagram



4. Emulating the Clock & Cycle

GBANA is phase-accurate. Every phase of every cycle is simulated (one tick simulates one phase). Synchronization of events within the phase need not match the real GBA, but at the end of each phase, the correct phase must be produced.



Main clock frequency: 16 MHz, ie approximately 16E6 cycles per second, or 62.5 ns per cycle. This is plenty time to emulate a single cycle on a modern computer. Each cycle has a low phase and a high phase. Each phase is one emulator tick. Each tick has two parts: a *start* part, where all the signals are updated and their callback functions are called, and an *interior* part, where the components execute their logic based on their internal state and the updated signals.

5. Signals

Components communicate by means of two types of interface: lines and buses. A line is just a boolean bus. There are two ways to affect a line/bus: (1) by *putting* data on it, and (2) by *forcing* data on it. Forcing updates the output value immediately. Putting schedules an update to the output value, to occur after a certain number of ticks.

Out of the signals defined in the ARM DDI 0029G, `D` is the only bidirectional signal so I got rid of it, in favor of `DIN` and `DOUT`.

Signal classes:

- clock
- address
- request
- response
- control

Name	Class	Component	Description
`A`	Memory Interface	**Memory**	The 32-bit address bus. The CPU writes an address to this but, for memory access requests.
`ABE`	Bus Controls	**Bus Controller**	
`ABORT`	Memory Management Interface		The memory sets this to *high* to tell the CPU that the memory access request cannot be fulfilled.
`ALE`	Bus Controls	**Bus Controller**	
`APE`	Bus Controls	**Bus Controller**	
`BIGEND`	Bus Controls		
`BL`	Memory Interface	**Memory**	Byte latch control. A 4-bit bus where each bit corresponds to one of the bytes in a word. Used to indicate which part of the requested word is to be read/written.
`BUSDIS`	Bus Controls		
`BUSEN`	Bus Controls		
`DBE`	Bus Controls		
`DIN`	Memory Interface	**GBA Core**	Unidirectional input data bus.

Name	Class	Component	Description
DOUT	Memory Interface	**Memory**	Unidirectional output data bus.
ENIN	Bus Controls	**Bus Controller**	
ENOUT	Bus Controls	**Bus Controller**	
FIQ	Interrupts	**GBA Core**	
ECLK	Clocks and Timing		MCLK exported from the core, for debugging. Has a small latency. Irrelevant for the emulator.
HIGHZ	Bus Controls	**Bus Controller**	
ISYNC	Interrupts	**GBA Core**	
IRQ	Interrupts	**GBA Core**	
LOCK	Memory Interface	**Memory**	Locks the memory, giving exclusive access to it to the CPU. This is effectively a mutex.
MAS	Memory Interface	**Memory**	Memory access size.
MCLK	Clocks and Timing		The main clock. Has two phases: a low phase and a high phase. Procedures can be constrained to any combination of these four: (1) the *start* of the low phase, (2) the *interior* of the low phase, (3) the *start* of the high phase, and (4) the *interior* of the high phase.
M	Processor Mode	**GBA Core**	
MREQ	Memory Interface	**Memory**	Set to *high* to indicate that the next cycle will be used to execute a memory request.
OPC	Memory Interface	**Memory**	This signal is used to distinguish between next-instruction-fetch and data-read/data-write. Set to *high* for instruction fetch request, set to *low* for data read/write requests.
RESET	Bus Controls		Used to start the processor. Must be held *high* for at least 2 cycles, with WAIT set to *low*.
RW	Memory Interface	**Memory**	This signal is used to distinguish between memory read and memory write. Set to *high* for read requests, set to *low* for write requests.

Name	Class	Component	Description
`SEQ`	Memory Interface	**Memory**	Set to *high* to indicate that the address of the next memory request will be in the same word that was accessed in the previous memory access or the word immediately after it. Sequential reads require fewer memory cycles.
`TBE`	Bus Controls		
`TBIT`	Processor State	**GBA Core**	Set to *high* for Thumb mode, set to *low* for ARM mode.
`TRANS`	Memory Management Interface		This signal is used to enable address translation in the memory management system. Irrelevant for the emulator.
`WAIT`	Clocks and Timing		This signal is used to insert wait cycles. Different memory regions have different access latency, which determines how many wait cycles need to be inserted.

6. Timing

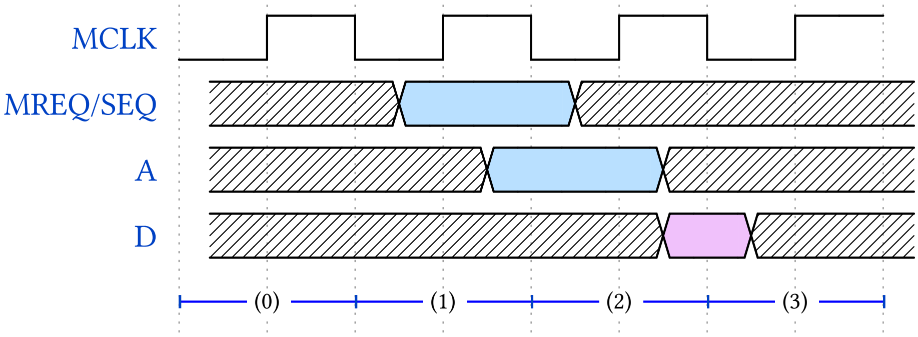
Types of intervals in a timing diagram[5]:

- *Open Unshaded* - The line/bus is expected to remain stable throughout this interval.
  - *Writing* occurs at the *start* and is prohibited in the *interior*.
  - *Reading* is allowed at the *start* (by signals succeding it in the tick order) and in the *interior*.
- *Open Shaded* - The line/bus is expected to change at an arbitrary time during this interval.
  - *Writing* is prohibited at the *start* and allowed in the *interior*.
  - Reading is allowed at the *start* and prohibited in the *interior*.
- *Closed* - The line/bus is disabled.
  - *Writing* is prohibited at the *start* and prohibited in the *interior*.
  - *Reading* is prohibited at the *start* and in the *interior*.

In request/response contexts, request data is in displayed in blue, and respone data is displayed in pink.

6.1. Simple Memory Cycle[6]

This is what a general memory cycle looks like:



Cycle (0) is the *pre-cycle*, cycle (1) is the *request cycle*, cycle (2) is the *response cycle*, and cycle (3) is the *post-cycle*. The term *memory cycle* refers to cycle (2).

General logic of a memory cycle:

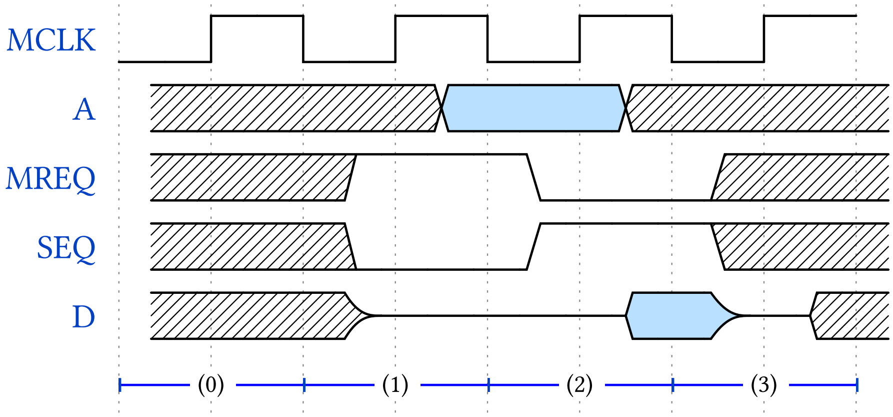
- The CPU must write `MREQ` and `SEQ` during the interior of phase 1 of the request cycle.
- The Memory may read `MREQ` and `SEQ` during phase 2 of the request cycle and/or at the start of phase 1 of the response cycle.
- The CPU must write `A` during the interior of phase 2 of the request cycle.
- The Memory may read `A` during phase 1 of the response cycle and/or at the start of phase 2 of the response cycle.
- The Memory must write `D` during the interior of phase 2 of the response cycle.
- The CPU may read `D` at the start of phase 1 of the post cycle.

[5] ARM DDI 0029G xix

[6] ARM DDI 0029G 3-4

6.2. N-Cycle[7]

This is what a Nonsequential Memory Cycle (N-cycle) looks like:



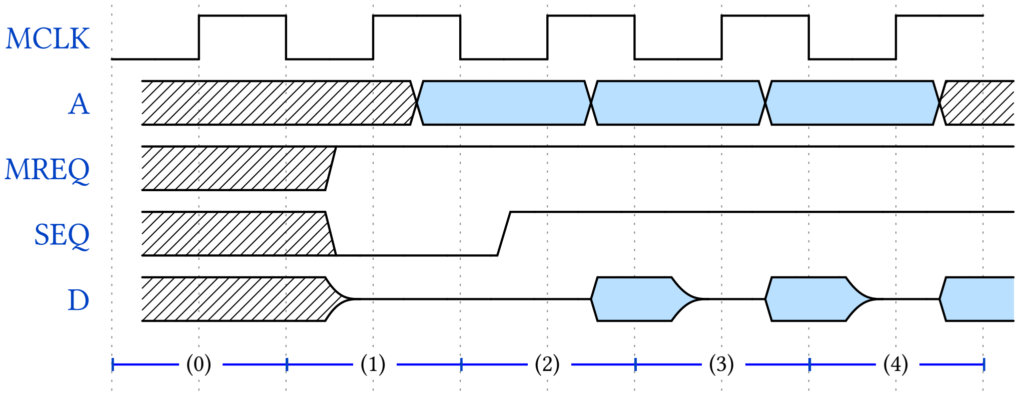
Cycle (0) is the *pre-cycle*, cycle (1) is the *request cycle*, cycle (2) is the *response cycle*, and cycle (3) is the *post-cycle*. The term *nonsequential memory cycle* refers to cycle (2).

Specific logic of an N-Cycle:

- General memory cycle logic.
- The CPU must set `MREQ` and `SEQ` to low during the interior of phase 1 of the request cycle.
- The Memory may extend phase 1 of the response cycle by setting the `WAIT` signal.

6.3. S-Cycle[8]

This is what a Sequential Memory Cycle (S-cycle) looks like:



Cycle (0) is the *pre-cycle*, cycle (1) is the *request cycle*, cycle (2) is the *N-response cycle*, cycle (3) is the 1st *S-response cycle*, cycle (4) is the 2nd *S-response cycle*, etc. The term *sequential memory cycle* refers to cycles (3), (4), etc.

Specific logic of an N-Cycle:

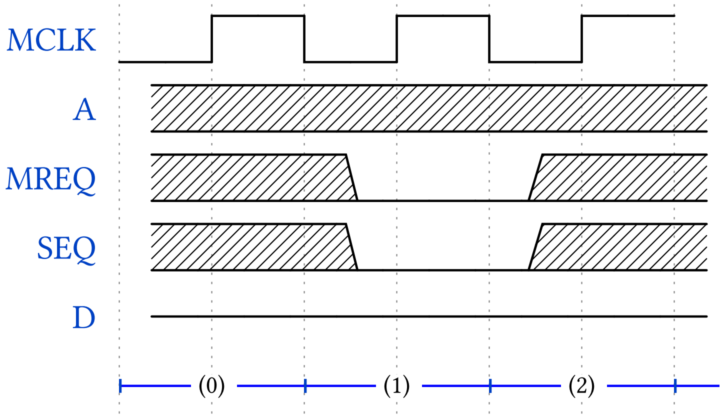
[7] ARM DDI 0029G 3-5

[8] ARM DDI 0029G 3-6

- General memory cycle logic.
- The CPU must set MREQ and SEQ to low during the interior of phase 1 of the request cycle.
- The CPU must set SEQ to high during the interior of phase 1 of the N-response cycle.
- The Memory may extend phase 1 of the response cycle by setting the WAIT signal.

6.4. I-Cycle[9]

This is what an Internal Memory Cycle (I-cycle) looks like:



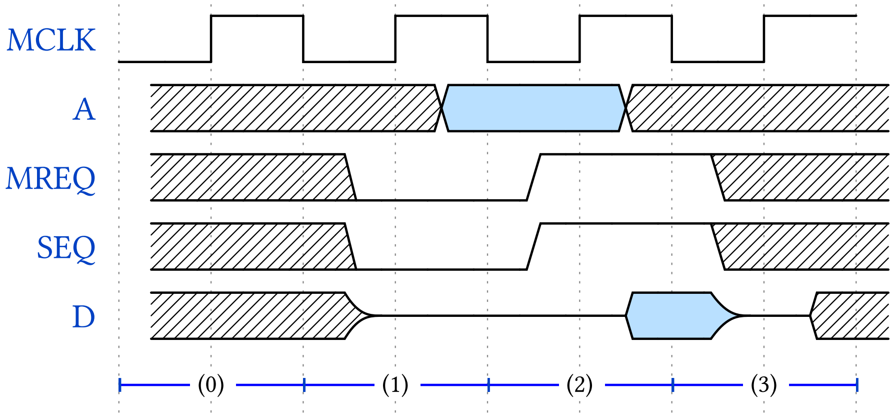
Cycle (0) is the *pre-cycle*, cycle (1) is the *internal cycle*, and cycle (2) is the *post-cycle*.

Specific logic of an N-Cycle:

- The CPU must set MREQ and SEQ to low during the interior of phase 1 of the request cycle.
- D must remain disabled.

6.5. Merged IS-Cycle

This is what a Merged Internal-Sequential Memory Cycle (merged IS-cycle) looks like:



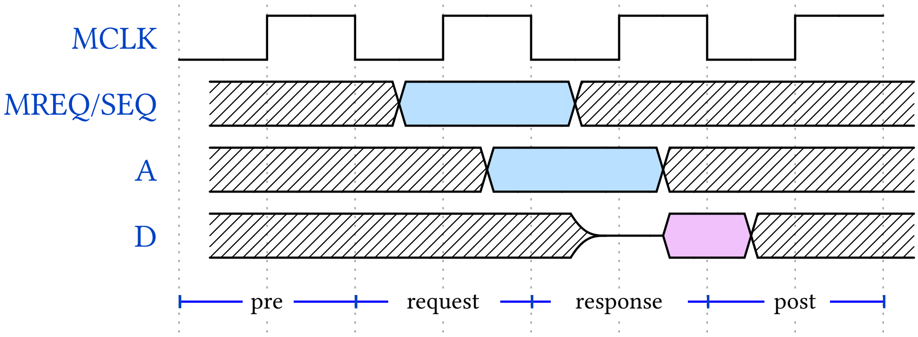
This looks the same as an N-Cycle, except the request cycle is merged with an I-cycle.

Specific logic of an N-Cycle:

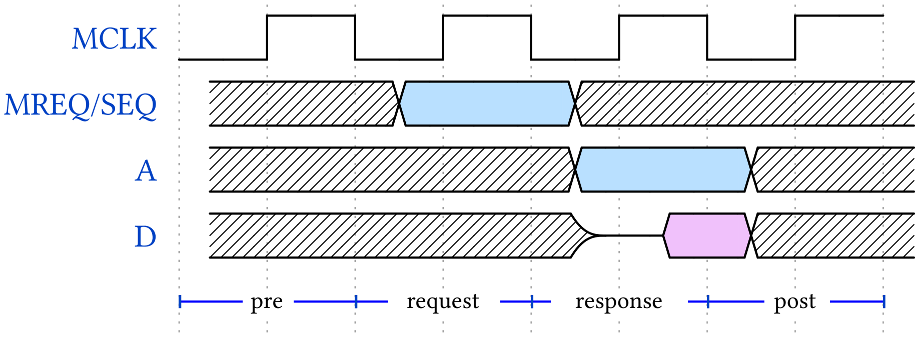
[9] ARM DDI 0029G 3-7

- The CPU may put the address on the bus a cycle earler, to give more time to the Memory to decode it.

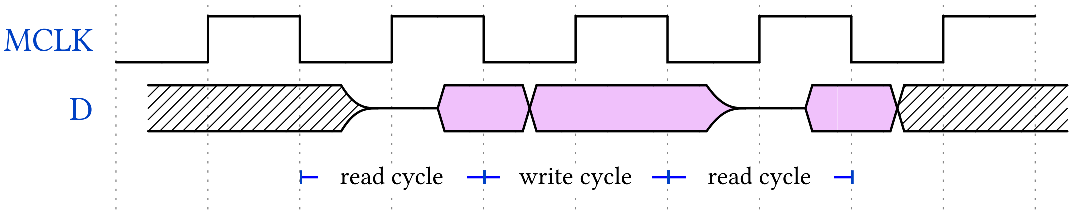
6.6. Pipelined Addresses



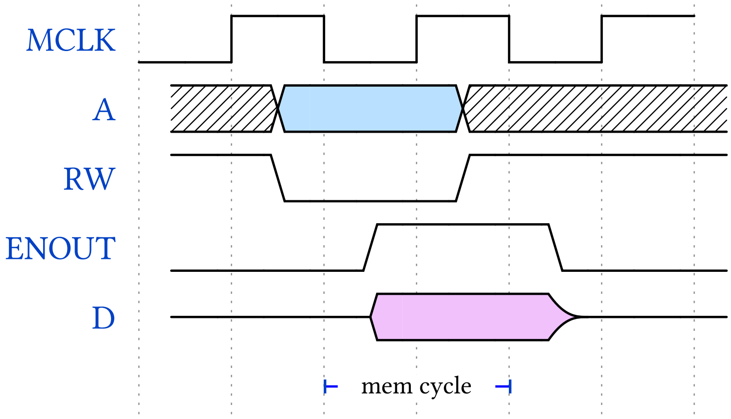
6.7. Depipelined Addresses



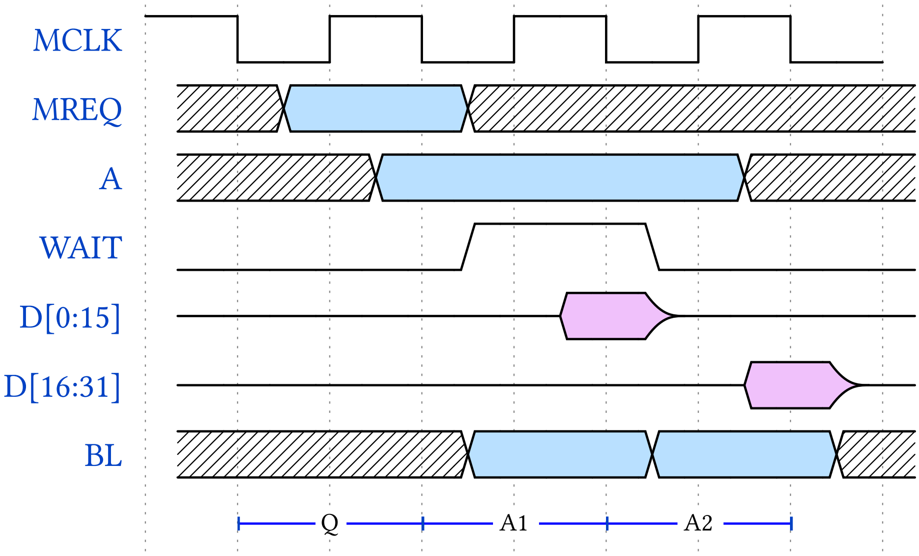
6.8. Bidirectional Bus Cycle



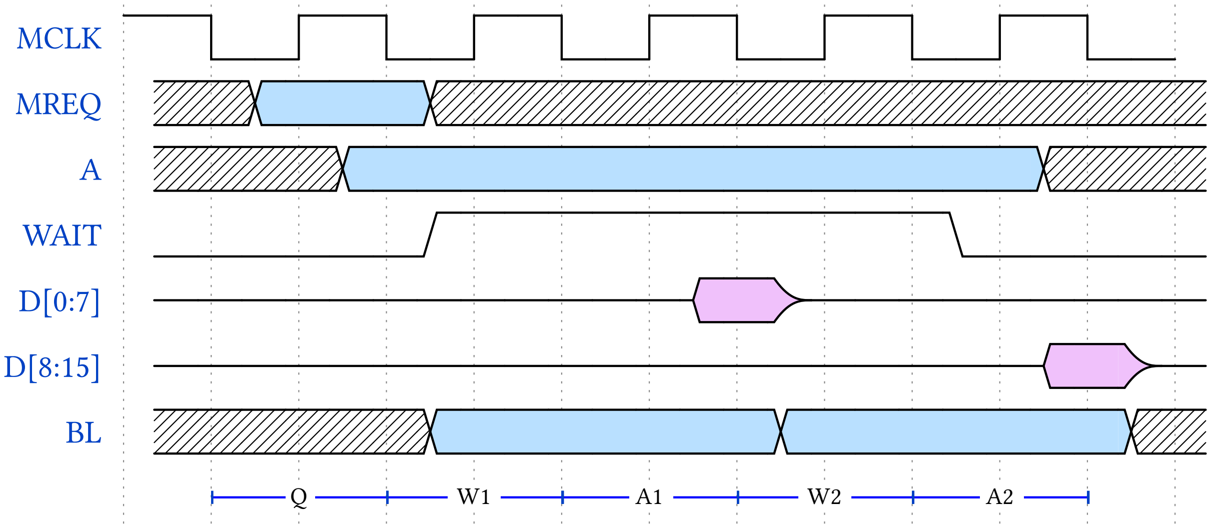
6.9. Data Write Bus Cycle



6.10. Halfword Bus Cycle

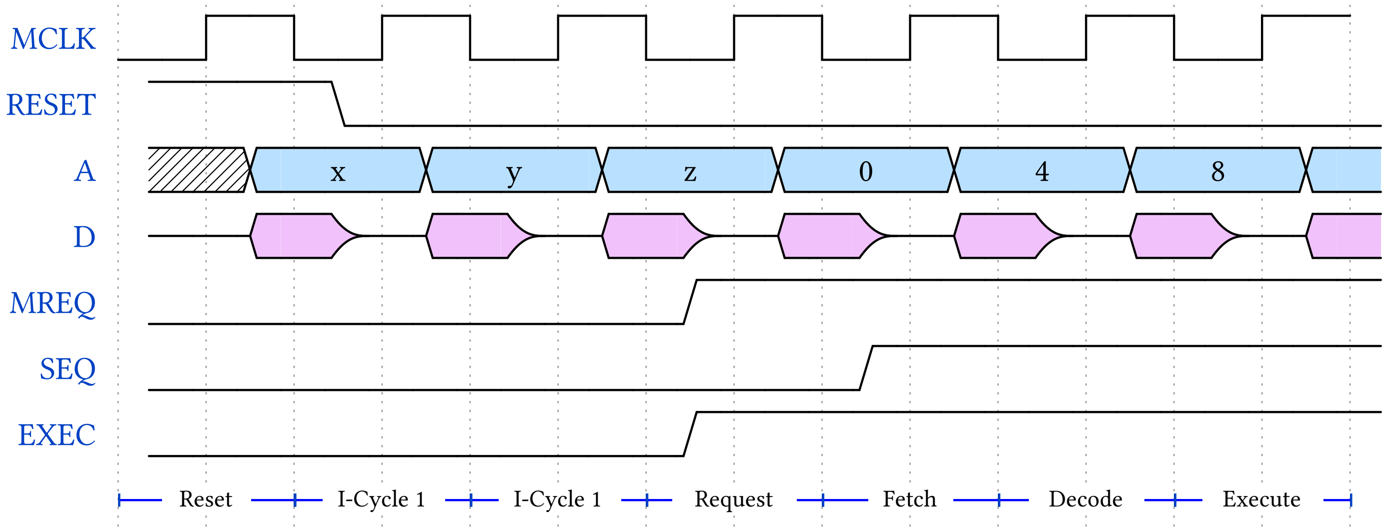


6.11. Byte Bus Cycle

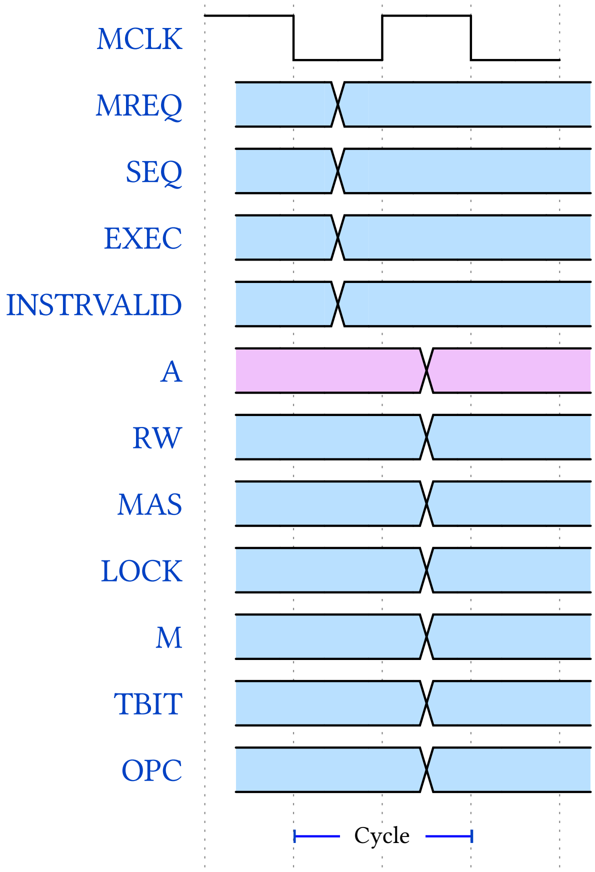


6.12. Reset Sequence

The reset sequence should look like this:



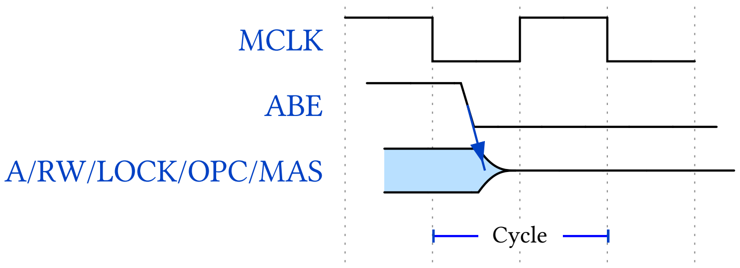
6.13. General Timing

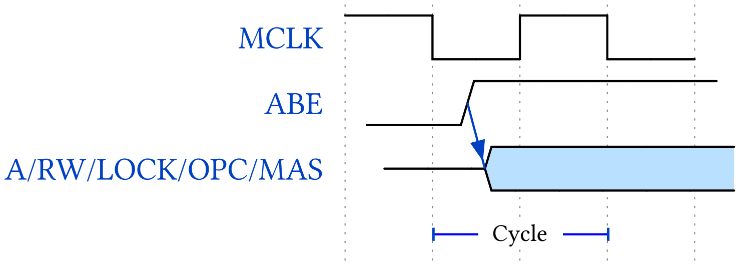


1. MREQ, SEQ, EXEC, and INSTRVALID may only be updated at the start of or in the interior of phase 1.

2. `A`, `RW`, `MAS`, `LOCK`, `M`, `TBIT`, and `OPC` may only be updated at the start of or in the interior of phase 2.

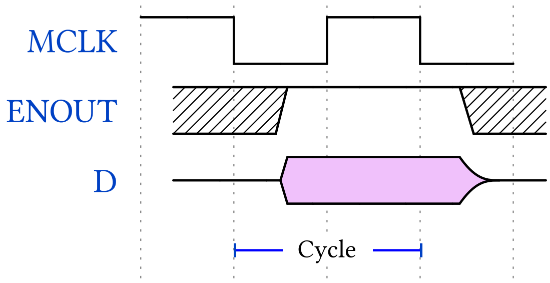
6.14. Address Bus Enable Control





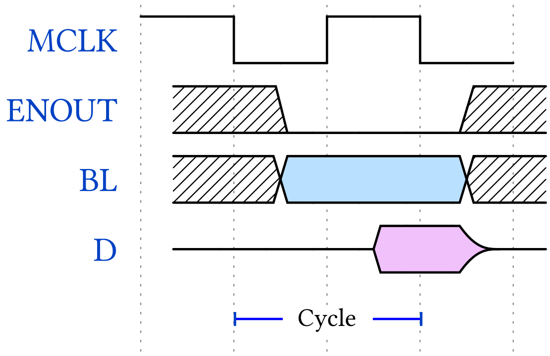
- `ABE` can change during phase 1.
- `A`, `RW`, `LOCK`, `OPC`, and `MAS` are enabled/disabled immediately when `ABE` switches to high/low.
- `A`, `RW`, `LOCK`, `OPC`, and `MAS` must be stable at the starts of both phases.

6.15. Bidirectional Data Write Cycle



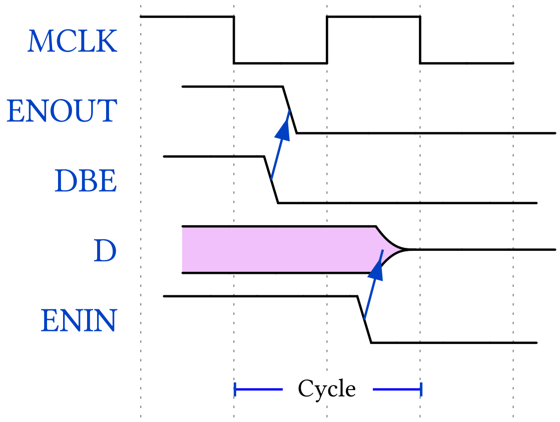
1. The CPU must enable `ENOUT` during the interior of phase 1.
2. The data must remain stable until the end of phase 1 of the post-cycle.

6.16. Bidirectional Data Read Cycle



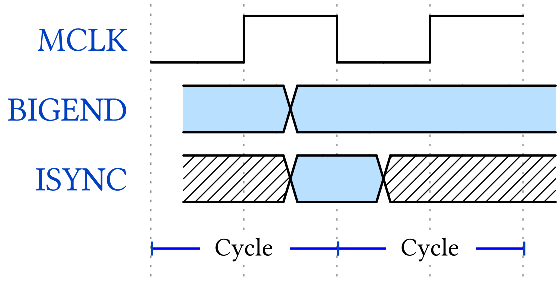
1. The CPU must disable ENOUT during the interior of phase 1.
2. The data must remain stable until the end of phase 1 of the post-cycle.

6.17. Data Bus Control



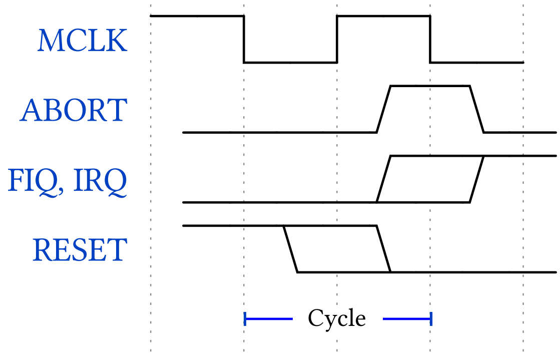
1. ENIN immediately disables D when it goes low.
2. ENOUT doesn't affect D .
3. DBE immediately disables ENOUT when it goes low.

6.18. Configuration Pin Timing

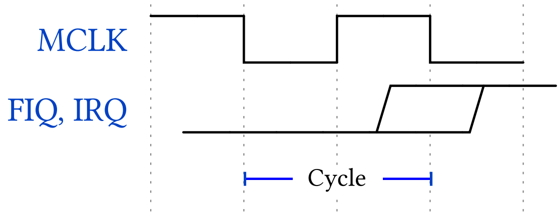


1. BIGEN may be updated during phase 2.
2. ISYNC must be stable at the start of phase 1, it may be written at any other time.

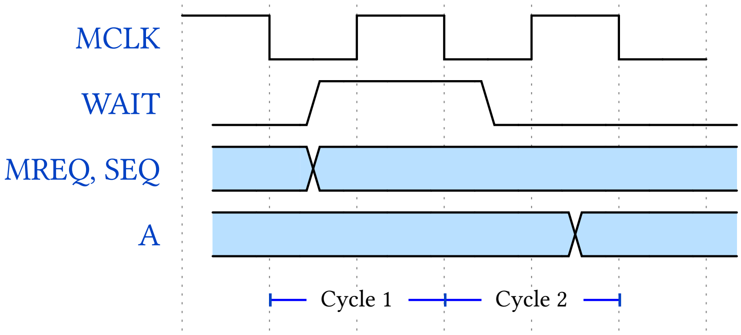
6.19. Exception Timing



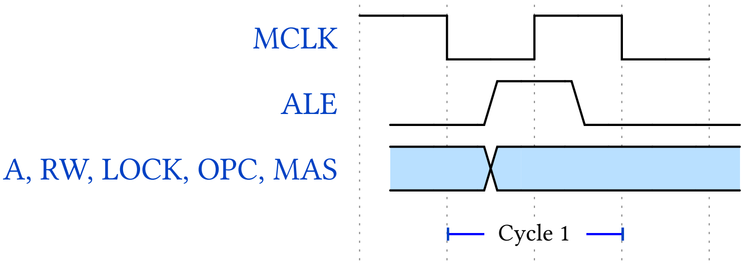
6.20. Synchronous Interrupt Timing



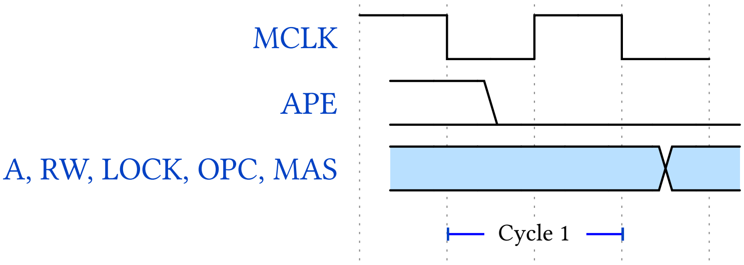
6.21. Memory Clock Timing



6.22. Address Latch Enable Control



6.23. Address Pipeline Enable Control


MCLK
APE
A, RW, LOCK, OPC, MAS
Cycle 1