binho

APPLICATION NOTE

Measuring CMIS firmware update time with the Binho Supernova

AN0011

Document information

Keywords	AN0011, Supernova, Pulsar, CMIS, OIF, CDB, firmware update, LPL, EPL, tWRITE, tCDBF, tBPC, Nmax, optical module, hold-off, I2CMCI, I3C, SPIMCI, transactions per block
Abstract	Updating firmware on an optical module is slow, and the obvious remedy is a faster management bus. Working through the CMIS access limits and the Table 10-4 timing parameters shows that the wire is not where the time goes: for a 1 MiB image the bus clock accounts for a small fraction of the total and the per-block hold-offs account for most of it. This application note derives the budget from the specification, states plainly which numbers are arithmetic and which would have to be measured, and supplies a command that times the access phases of a real module against the specification ceilings so that a host can find out which hold-offs its module actually takes.

Binho Inc.
42 Dore Street, Unit A, San Francisco, CA 94103

REVISION 1.0
6 NOVEMBER 2026

> **What was tested.** The budget in Section 4 is arithmetic derived from the specification, and now carries all three management interfaces the specification and this series cover: I2CMCI, SPIMCI, and I3C as a Binho derivation. The `timing` output in Section 5 is a real run against firmware targets, not an optical module, and it turns out to measure the host path rather than the target. Section 5.2 measures two adapter firmware revisions across two host transports and finds the floor unmoved by either, Sections 5.3 and 5.4 trace that floor to the adapter's USB descriptor and show how much of it a host can hide, and Section 7 states what is outstanding.

1 Introduction

I3C single data rate runs roughly 31 times faster than I2C Fast-mode on the wire. A host engineer facing a slow module firmware update reasonably expects a faster management bus to make it roughly 31 times quicker.

Working the arithmetic through the CMIS access limits gives a different answer, and the difference is actionable: the choice of which Common Data Block command carries the payload matters more than the choice of bus.

This note derives the budget, and supplies a way to find out where a specific module actually spends its time rather than where the specification permits it to.

> **This document does not state a measured throughput.** Section 7 explains why, and what would have to be run to state one.

1.1 Scope

The document covers the arithmetic of a CMIS firmware download: the access length limits, the block sizes of the two Write Firmware Block commands, and the Table 10-4 hold-off parameters. It supplies a command that times the read, page change and Common Data Block access phases of a module against those ceilings.

It does **not** write firmware to a module, and the supplied utility has no command that does. It does not cover the firmware update state machine, image signing, or the Run and Commit sequence beyond noting their timing. AN0008 covers the register model and is a prerequisite.

1.2 Applicable devices

Table 1. Applicable devices

Item	Role	Basis
Binho Supernova	Host adapter	Exercised, Section 7
Binho Pulsar	Host adapter	Not exercised
Any CMIS managed module	Target	**Not exercised.** The measurement used an adapter in target mode

> **The budget in Section 4 is arithmetic, not measurement.** It is derived from OIF-CMIS-05.3 sections 5.2.2 and chapter 9 and from Table 10-4, and is reproducible with `python cmis_mci.py budget`. No row of it has been checked against a module.

1.3 Related documents

Table 2. Related documents

Document	Subject
OIF-CMIS-05.3	Sections 5.2.2.1 and 5.2.2.2, chapter 9 for the CDB, Table 10-4 for timing
AN0008	Bringing up a CMIS optical module over I2C with the Binho Supernova
AN0009	Managing a CMIS optical module over I3C with the Binho Supernova
AN0010	Bringing up a CMIS optical module over SPI with the Binho Pulsar

2 Required equipment and software

Table 3. Required equipment

Item
Binho Supernova or Binho Pulsar USB host adapter, for the `timing` command
A CMIS managed optical module, for the `timing` command
Nothing at all, for the `budget` command

```
pip install binhosupernova
python cmis_mci.py budget
```

> **A host adapter can serve two transports, and which one you get is decided by your host software, silently.** Binho adapter firmware from 4.5.0 carries transfers on vendor bulk endpoints as well as on the original USB HID path. The bulk one is opt-in: an adapter whose host does not negotiate it serves the HID path unchanged, with no error and no warning, which is what makes the firmware a non-breaking release. `binhosupernova`, which this note and the rest of the series use, is HID only, and so is any CosmicSDK older than 1.5.0. Ask the open device whether the transport is available rather than inferring it from the release you installed, because an SDK without it produces the older numbers and reports nothing unusual while doing so. Section 5.2 measures both, and the distinction matters more for the conclusion than for the numbers.

3 The four facts the budget rests on

3.1 A firmware block over the Local Payload is 116 bytes

The Common Data Block occupies Page 9Fh. Its Local Payload is 120 bytes, of which `Write Firmware Block LPL`, command `0103h`, spends 4 on the block offset. That leaves **116 bytes** of firmware per block.

The Extended Payload is pages A0h through AFh, 16 pages of 128 bytes, giving **2048 bytes** per block through `Write Firmware Block EPL`, command `0104h`.

3.2 A WRITE carries at most 8 bytes

Section 5.2.2.2 states it directly: "A successful WRITE writes a sequence of up to eight given byte values". `FullPageReadSupported` raises `Nmax` to 128 for **reads**, and does nothing for writes.

The only escape is the note immediately following: "A WRITE with more than 8 bytes may be allowed when specified explicitly in chapter 7.8", which is the Common Data Block Extended Payload region.

So a 116 byte Local Payload block is `ceil(116 / 8)` = **15 separate write transactions**, each with its own address phase.

3.3 The module then stops talking

Table 4. Hold-off parameters, CMIS Table 10-4 maxima

Parameter	Maximum	Applies after
tREAD	500 microseconds	A register read
tWRITE	10 ms	A write to volatile memory
tWRITENV	80 ms	A write to non-volatile memory
tBPC	10 ms	A bank or page change
tCDBF	4960 ms	A foreground CDB command. The default is 80 ms
tMgmtInit	2000 ms	Reset, until ModuleLowPwr is reached

These are maxima. A module may advertise anything up to them, and Section 5 is about finding out what a given module actually takes.

3.4 Almost all CMIS memory is volatile

The specification states it outright, so **tWRITE at 10 ms is the common case**, not the optimistic one.

The hold-off is paid **once per block**, not once per transaction. That is the whole lever: a 2048 byte block pays it 512 times over a 1 MiB image, and a 116 byte block pays it 9040 times.

4 The budget

```
python cmis_mci.py budget --image-kib 1024
```

```
Firmware update budget, 1024 KiB image, tWRITE 10 ms per block
  path                            blocks   per block       bus    hold-off     total
  I2CMCI 400 kHz, LPL 0103h         9040       15 txn   36.61 s     90.40 s   127.01
s
  I3C SDR 12.5 Mb/s, LPL 0103h      9040       15 txn    1.17 s     90.40 s    91.57
s
  SPIMCI 1 MHz, LPL 0103h           9040       15 txn   15.19 s     90.40 s   105.59
s
  I2CMCI 400 kHz, EPL 0104h          512       16 txn   24.33 s      5.12 s    29.45
s
  I3C SDR 12.5 Mb/s, EPL 0104h       512       16 txn    0.78 s      5.12 s     5.90
s
  SPIMCI 1 MHz, EPL 0104h            512        1 txn    8.41 s      5.12 s    13.53
s
  SPIMCI 50 MHz, EPL 0104h           512        1 txn    0.17 s      5.12 s     5.29
s

  Bus clock alone, 400 kHz to 12.5 Mb/s, about 31 times the wire speed: 1.39 times
faster.
  Payload size alone, LPL to EPL at 400 kHz: 4.3 times faster.
  Both together: 21.5 times faster.
  SPIMCI at 50 MHz, 125 times the I2C wire speed and one transaction per block: 24.0
times faster.
```

Read the first two rows against each other. Multiplying the wire speed by 31 removes 35 seconds. Changing which Common Data Block command carries the payload, at the original slow clock, removes 97.

4.1 Three management interfaces, and the same answer from all of them

The SPIMCI rows carry the comparison the whole series is built to make, because SPIMCI is the only other management interface CMIS actually specifies. Appendix B.3.7.1 frames every transaction as a four byte control word, `N` flow control bytes and up to **2048 bytes** of payload, and `MciSpeedConfiguration` at `00h:27.3-0` runs from 1 MHz to **50 MHz**. That is 125 times the I2C clock and 16 times the largest access the paged interfaces can make.

It buys 24.0 times against I2CMCI's 21.5. Two facts are worth separating out of that.

At the speed a module actually starts at, SPIMCI is slower on the wire than I3C. The initial `MciSpeedConfiguration` on exit from MgmtInit is always 1 MHz, and at 1 MHz the Extended Payload path spends 8.41 s on the wire against I3C's 0.78 s. The clock has to be raised deliberately, by a host that read the module's advertisement and wrote the field.

What SPIMCI changes at any speed is the transaction count. An Extended Payload block is 16 transactions on I2CMCI and on I3C, both of which cap an access at one page, and **one** on SPIMCI. That column never appears in a bandwidth figure, and Section 5 shows it is the one a

host feels: every transaction costs a round trip through the host, and on the bench that round trip is about two milliseconds whatever the bus is doing.

So the three interfaces disagree by a factor of 125 on the wire and by a factor of 24 on the total, and they agree completely on where the time goes. A firmware update is not a bandwidth problem.

Table 5. What the budget model does and does not include

Included	Excluded
Transactions per block, from the access length limits	Host USB latency, which is not a bus property
Bus time per transaction, as 4 framing bytes plus payload at 9 bits each	tCDBF, which varies by module and command
tWRITE once per block	The Run Image and Commit Image steps
	Retries after a rejected access

> **The bus time column is an estimate of wire occupancy, not a measured throughput.** It is deliberately crude, because its only job is to show that it is small next to the hold-off column. A more careful model of it would not change the conclusion, and a measured value would characterize the host's USB path as much as the bus.

5 Measuring a real module

```
python cmis_mci.py timing
```

The command writes no firmware. It times the access phases an update is built from, a single byte read, an eight byte read at the write limit, a full page read where the module advertises one, a page change in both directions, and reaching the Common Data Block page, and prints each next to its Table 10-4 ceiling.

```
$ python cmis_mci.py timing
phase                                 median        max    CMIS max
single byte read                    2.058 ms    3.400 ms      0.5 ms
8 byte read, the WRITE limit        1.995 ms    2.064 ms           -
page change 00h to 01h to 00h       7.988 ms    9.019 ms     20.0 ms
select CDB page 9Fh                 4.043 ms    5.913 ms     10.0 ms

One 116 byte LPL firmware block is 15 write accesses, about 29.9 ms of bus time,
against a tWRITE hold-off of up to 10 ms that the module pays once per block.
```

Read that as a measurement of the host, not of the target. Every row is roughly two milliseconds per USB round trip: one access costs about 2 ms, the two register writes of a CDB page selection cost about 4, and the four writes of a page change in both directions cost about 8. A single byte read sits four times above tREAD, and an eight byte read is not slower than a one byte read, which is only possible if the bytes on the wire are not what is being timed.

The target in this run is an adapter in I2C target mode, which answers immediately and never holds the bus off. So what the numbers show is the floor the host path imposes before a module has contributed anything at all, and that floor is around 2 ms per access on this setup. Section 7.2 draws the conclusion.

What to look for in the result:

A median far below the specification maximum means the module is faster than it is permitted to be, and a host that waits the maximum after every write is leaving most of the update time unclaimed. This is the most valuable thing the measurement can tell you.

A maximum close to the specification ceiling on a page change means the module really does take tBPC sometimes. A host that assumes the median will fail intermittently, which is the worst failure mode to debug.

A single byte read above tREAD is host side or adapter side overhead, not module hold-off. tREAD is an abstraction over I2C clock stretching, and clock stretching is visible on a bus capture in a way that a host side timer cannot distinguish from anything else.

5.1 The same measurement against an I3C CMIS target

The run above is over I2CMCI against an adapter in target mode. Repeating it over I3C against the CMIS target firmware of AN0009, which implements the register model rather than a flat memory, moves nothing that matters:

```
$ python cmis_mci.py timing --transport i3c --address 0x09 --push-pull
PUSH_PULL_2_5_MHZ_25_DC
phase                                 median        max    CMIS max
single byte read                    2.059 ms    3.008 ms      0.5 ms
8 byte read, the WRITE limit        2.062 ms    2.760 ms           -
page change 00h to 01h to 00h       7.997 ms    9.170 ms     20.0 ms
select CDB page 9Fh                 3.998 ms    4.120 ms     10.0 ms

One 116 byte LPL firmware block is 15 write accesses, about 30.9 ms of bus time,
against a tWRITE hold-off of up to 10 ms that the module pays once per block.
```

A 31 times faster bus, and a target that answers from firmware rather than from a flat arena, leave every row where it was. That is the same result the budget predicts, arrived at from the other direction.

5.2 The floor belongs to the host stack, and this one cannot leave it

Supernova firmware 4.5.0 moves adapter transfers onto vendor bulk endpoints, raising the transfer ceiling and reworking the USB path. Measured through the utility this note supplies, it changes nothing.

Table 6. The same four phases on two adapter firmware revisions, median of three runs

Phase	4.4.0	4.5.0
single byte read	2.006 ms	2.036 ms
8 byte read, the WRITE limit	2.003 ms	2.036 ms
page change 00h to 01h to 00h	7.993 ms	8.061 ms
select CDB page 9Fh	3.987 ms	4.035 ms

The board was taken from 4.4.0 to 4.5.0 and back to 4.4.0 and forward again, and measured in each state, because a single comparison across a reflash cannot tell a firmware change from anything else that moved.

> **Both rows above are the compatibility path.** The newer transport is opt-in at the host: an adapter whose host does not negotiate it serves the older path unchanged, which is what makes the firmware a non-breaking release. `binhosupernova` 4.2.0, the SDK this note and the whole series run on, transports over USB HID and carries no vendor bulk endpoint code at all. So the pair says the compatibility path did not regress, and on its own it says nothing about the transport 4.5.0 added.

Repeating the same four phases through a host stack that does negotiate it answers the question the table cannot. The phases do not move.

Table 7. The same four phases on the compatibility transport and on the bulk transport, both on firmware 4.5.0

Phase	USB HID	Bulk transport
single byte read	2.036 ms	2.009 ms
8 byte read, the WRITE limit	2.036 ms	2.006 ms
page change 00h to 01h to 00h	8.061 ms	8.019 ms
select CDB page 9Fh	4.035 ms	3.993 ms

The right hand column was taken with the transport reporting itself available and advertising a 32768 byte ceiling, so it is not a second compatibility run wearing a different name. The transport arrived in CosmicSDK 1.5.0; an older SDK produces the left hand column while reporting nothing unusual, which is the failure this measurement had to rule out rather than assume. The check is a capability query against the open device, not a release number. What it

is not is a negative result about the transport, because in the same session the transport does exactly what it was built to do:

Table 8. Clocking bytes out of SPI, same adapter and session, by transfer size

Transfer	Request path	Bulk path
8 bytes	2.05 ms	2.00 ms
1024 bytes	35.0 ms	5.0 ms
32768 bytes	not available	114.8 ms, 285 kB/s

A wider transport is a payload lever, not a latency lever. At a kilobyte it is seven times faster. At eight bytes it is the same speed, because what an eight byte transfer costs is one round trip, and a round trip does not get shorter when the pipe gets wider.

That is the whole of Section 4 restated on the host side, and it is why the two sections agree. `5.2.2.2` caps a WRITE at eight bytes and an access at one page, so **CMIS on a paged management interface never issues a transfer large enough for any of this to help.** The interface that does issue one is SPIMCI, at 2048 bytes, which is the row in Section 4 that collapses sixteen transactions per block into one. No SPIMCI target exists on this bench, so that row remains arithmetic.

5.3 Where the two milliseconds come from

Two independent host stacks agreeing on a figure to three digits is not a coincidence, and the cause is in the adapter's USB descriptor rather than in either of them.

Table 9. The adapter's endpoints, from its configuration descriptor

Interface	Endpoints	Transfer type	Max packet	Interval
Human Interface Device	0x01 OUT, 0x81 IN	Interrupt	64 bytes	1 ms
Vendor specific	0x02 OUT, 0x82 IN	Bulk	64 bytes	none

The command plane is the interrupt pair, and an interrupt endpoint is polled on a schedule. The interval here is one millisecond, so a request cannot leave before the next frame and its answer cannot arrive before the one after. That accounts for most of the two milliseconds, and it is not work any SDK is doing: it is when the bus lets a transfer happen. What remains is adapter firmware turnaround and host stack, which this measurement does not separate.

The bulk pair carries payload, not commands. That is the whole of the earlier result: widening the payload path leaves the command round trip exactly where it was, so an eight byte access, which is one command and one answer and nothing else, cannot get faster.

The same model predicts the other column, which is the check worth doing. A 1024 byte transfer on the request path travels inside the commands themselves, 64 bytes at a time, so it is sixteen frames of payload plus the command round trip: about 34 ms, against 35.0 ms

measured. On the bulk path the payload leaves the schedule entirely and only the command round trip remains: about 2 ms plus the transfer, against 5.0 ms measured.

So the floor has a name, and it tells a host what to do about it. What moves it is the number of commands, and how many of them are waiting at once. Neither is the size of the pipe they travel in.

5.4 Half of the floor is the answer, and a host can hide it

If a command and its answer cost one frame each, then a host that waits for every answer before sending the next command is paying for both, and one that keeps commands in flight should be paying for only the first. That is testable. The same 1 byte read, issued 200 times through the non blocking interface, once waiting for each answer and once not:

Table 10. The same 200 reads, waiting for each answer against keeping them in flight

	Total	Per command
one at a time	400.0 ms	2.000 ms
all in flight	200.9 ms	1.005 ms

Pipelining halves it, and the half it removes is the answer. What remains is one command per frame, which is the outbound polling interval and the thing a host cannot get under without sending fewer commands. Every one of the 200 was answered, so this is not a measurement of commands being dropped.

Two consequences for a host that wants them. The saving is real but bounded: it takes the cost of an Extended Payload block from about 32 ms of host time to about 16 ms, against a tWRITE of 10 ms, so the host stops dominating the block without becoming negligible in it. And it is only available where the accesses do not depend on each other, which inside one block of firmware they do not, because they are consecutive writes into the same Extended Payload region.

It is worth saying plainly that **the utility this note supplies does not do this.** It uses the blocking interface, one command at a time, which is why every number in Section 5 is the two millisecond column. The measurement above is a separate harness against the non blocking interface, and it exists to price the option rather than to describe the tool.

That is worth putting next to Section 4. An Extended Payload block is sixteen accesses, about 32 ms of host time against a 10 ms tWRITE, so on this bench the host dominates the block rather than the module does. Pipelining takes that to about 16 ms and **SPIMCI takes it to one transaction rather than sixteen**, which is the largest change in this note that reaches the term Section 5 measures. A host built around the assumption that its own round trips are negligible is wrong here, and on a paged management interface the best it can do is halve them.

Which is why Sections 4 and 5 arrive at the same place from opposite directions.

6 What to do with the finding

For host software. Query `Firmware Management Features`, CDB command `0041h`, once at the start of a download and prefer `Write Firmware Block EPL` (`0104h`) over `0103h` where the module supports it. Do not assume either way. Then issue the accesses within a block without waiting for each answer: Section 5.4 measures that at half the host cost, it is available because the accesses inside one block do not depend on each other, and it needs a non blocking interface rather than a faster one.

For module firmware. Advertise hold-off times you can honor. tWRITE is a maximum, not a target, and the measurement in Section 5 is how a host discovers what you left unclaimed.

For anyone specifying a new MCI binding. SPIMCI already carries up to 2048 bytes in one transaction, which Section 4.1 prices, and I3C negotiates the same thing natively through SETMWL, SETMRL and GETMXDS. A binding that inherits I2CMCI's 8 byte write limit inherits this entire problem with it, and inherits it whatever clock it runs at.

> **Budget the Run Image step separately.** CDB command `0109h` takes as long as a full module reboot, "mainly governed by the tMgmtInit timing parameter", so allow roughly two seconds for it rather than a CDB timeout.

7 Measured results

7.1 Test configuration

Table 11. Test configuration, the I2CMCI run of Section 5

Item	Value
Controller	Binho Supernova, serial `1912DEB5…79FA`, firmware 4.4.0, hardware revision C
Target	Binho Supernova, serial `ED7BF8F3…F7BB`, firmware 4.4.0, armed as an I2C target at 0x50
Target contents	CMIS 5.3 image, the Appendix C 400G QSFP-DD worked example
SDK	`binhosupernova` 4.2.0, Python 3.10.12
Host	Ubuntu 22.04, x86-64
Bus	3.3 V, 400 kHz
Repeats	20 per phase, median and maximum reported
Date	1 September 2026

Table 12. Test configuration, the I3C runs of Sections 5.1 and 5.2

Item	Value
Controller	Binho Supernova, serial `4F970418…1111`, firmware 4.4.0 and 4.5.0
Host transports	`binhosupernova` 4.2.0 over USB HID, and CosmicSDK 1.5.0 over the vendor bulk endpoints, confirmed available by capability query and advertising a 32768 byte ceiling
Target	NXP FRDM-MCXN947 running the AN0009 CMIS target firmware, dynamic address 0x09
Target contents	CMIS 5.3 image, the Appendix C 400G QSFP-DD worked example
SDK	`binhosupernova` 4.2.0, Python 3.14
Host	Arch Linux, x86-64
Bus	3.3 V, I3C SDR, push-pull 2.5 Mbit/s at 25% duty cycle, open drain 1 Mbit/s
Repeats	20 per phase, median and maximum reported, three runs per firmware
Section 5.4	A separate harness against the non blocking interface, 200 reads per arm, three runs
Date	3 September 2026

The two configurations are different benches, different hosts and different Python versions. They are not compared against each other anywhere in this note, and only the second one is used for the firmware comparison in Section 5.2, where the host and the target are held fixed and the adapter firmware is the only thing that changes.

7.2 Results

Derived. The budget in Section 4 is reproducible from the supplied utility and rests only on the specification citations in Section 3. It is arithmetic, and this document does not present it as anything else.

Measured, and it measures the host. The Section 5 run establishes a floor of about 2 ms per register access on this host, adapter and SDK, with a target that imposes no hold-off of its own. That number belongs to the USB path, not to the bus and not to a module: a single byte read exceeds tREAD fourfold, and an eight byte read is no slower than a one byte read, both of which are only possible if wire time is not the term that dominates.

This is the reason **no end to end throughput figure appears anywhere in this document.** A figure measured this way would characterize the host, and quoting it as a bus or module property would be wrong in a way that is hard to detect later.

It does, however, sharpen the note's own argument rather than weakening it. Section 4 argues that wire time is not where a firmware update goes. The measurement adds that host round trip cost is not where it goes either. Both are small next to a per-block hold-off, and both shrink for

the same reason when a block gets larger: **fewer accesses and fewer blocks.** A 2 ms per access floor costs 30 ms per Local Payload block and 32 ms per Extended Payload block, but the Extended Payload block carries 17 times more firmware.

Measured again on a newer adapter firmware and a newer transport, and it did not move. Supernova firmware 4.5.0 leaves all four phases where 4.4.0 had them, and so does the bulk transport that firmware adds, measured with the transport reporting itself available. In the same session that transport is seven times faster on a kilobyte transfer, so this is a bounded result rather than a failed setup: a wider transport is a payload lever and not a latency lever, and an eight byte access costs one round trip either way.

That is Section 4 restated from the other end. `5.2.2.2` caps a WRITE at eight bytes and an access at one page, so CMIS on a paged management interface never issues a transfer large enough for a wider pipe to help. The levers that reach this table are the number of round trips, which is what the transactions per block column counts and what SPIMCI changes, and how many of them are outstanding at once.

The second of those is worth half, and it is available today. Section 5.4 issues the same access 200 times without waiting for each answer and measures 1.005 ms against 2.000 ms, every command answered. What is left is one command per USB frame. So a host writing a firmware block can halve its own contribution by not waiting inside the block, where the accesses are independent, and after that it is out of levers on a paged interface.

That is also why the transactions per block column in Section 4 is the one to read. SPIMCI carries an Extended Payload block in a single transaction where the paged interfaces need sixteen, and at about two milliseconds of host cost per transaction that is 2 ms against 32 ms per block before the bus has done anything at all.

On hardware, against a module. Nothing here. Outstanding at revision 1.0: the `timing` command against a real module, which is the only thing that establishes the hold-offs a module actually takes; a complete firmware download over both the Local Payload and Extended Payload paths, which is the only thing that would support an end to end figure; and any measurement at all on SPIMCI, for which no target exists on this bench. Every SPIMCI number in Section 4 is arithmetic from Appendix B.3 and is labelled as such.

Until that has been run, the claim this note supports is about **where the time is permitted to go**, and about what the host costs before a module contributes anything.

8 Troubleshooting

Table 13. Troubleshooting

Symptom	Probable cause and action
`timing` reports a single byte read far above tREAD	Host or adapter overhead, not the module. Compare against a bus capture
Every phase is close to its ceiling	The module may be genuinely slow, or another controller is contending for the bus
The 94 byte read is absent from the output	The module does not advertise full page read, so `Nmax` stayed at 8
Medians vary widely between runs	Something else is using the USB path. Close other host tools and repeat
The numbers look the same on a newer adapter firmware	Expected, and Sections 5.2 and 5.3 explain it. Also confirm which transport you are on: if the SDK does not negotiate the bulk one, the adapter serves the HID path with no error. Ask the SDK, do not infer it from a version number
Every access costs about 2 ms and nothing you change helps	Most of that is the USB schedule: the command plane is an interrupt endpoint polled at 1 ms, so a command and its answer span two frames. Section 5.3 has the descriptor. Reduce the number of commands rather than the size of them
A page change exceeds twice tBPC	The module is taking longer than the specification permits. Capture the bus before blaming the host

9 Acronyms

Table 14. Acronyms

Term	Meaning
CDB	Common Data Block
CMIS	Common Management Interface Specification
EPL	Extended Payload, CDB pages A0h to AFh
LPL	Local Payload, in CDB page 9Fh
MCI	Management Communication Interface
Nmax	Maximum number of bytes in one register access
OIF	Optical Internetworking Forum
SDR	Single data rate

10 Note about the source code in the document

Example code shown in this document and supplied in the accompanying archive is provided for evaluation and reference. It is released under the MIT license, the text of which is included in the archive.

Table 15. Contents of the assets archive

File	Description
`AN0011.md`	Markdown source of this document
`assets/cmis_mci.py`	Reference utility, shared by AN0008 through AN0011
`assets/LICENSE`	MIT license covering the example code

11 Revision history

Table 16. Revision history

Revision	Date	Description
0.1	1 September 2026	Draft for internal review. Budget derived from the specification; no measurement taken.
0.2	1 September 2026	`timing` run against a Supernova I2C target fixture. Output pasted, and reported as a measurement of the host path rather than of a module.
0.3	2 September 2026	Renumbered from AN0010. No change to the measurements or the budget.
0.4	3 September 2026	Budget extended to SPIMCI, at its initial 1 MHz and at the 50 MHz top of `MciSpeedConfiguration`, with the transactions per block column called out as the one a host feels. Section 5 gains the same measurement over I3C against the AN0009 CMIS target, and a comparison of Supernova firmware 4.4.0 against 4.5.0 across two host transports, neither of which moves the floor, with a SPI transfer size sweep in the same session showing the wider transport seven times faster at a kilobyte and identical at eight bytes. Both transport runs repeated against CosmicSDK 1.5.0 once the transport landed in it, with no change. New Section 5.3 traces the two millisecond floor to the 1 ms interrupt endpoints the command plane runs on, and checks that model against the transfer size sweep. Utility at 1.1.
0.5	3 September 2026	New Section 5.4 prices keeping commands in flight, measured at half the per command cost with every command answered, and Section 6 now tells a host to do it inside a block. The floor that remains is one command per frame.
1.0	6 November 2026	Published.

Legal information

Definitions

Draft

A document marked draft is under internal review and subject to formal approval. Binho Inc. makes no representation or warranty as to its accuracy or completeness and accepts no liability arising from its use.

Disclaimers

Limited warranty and liability

Information in this document is believed to be accurate and reliable. Binho Inc. makes no representation or warranty, express or implied, as to its accuracy or completeness, accepts no liability for the consequences of its use, and takes no responsibility for content originating outside Binho Inc.

In no event shall Binho Inc. be liable for any indirect, incidental, punitive, special or consequential damages, including without limitation lost profits, lost savings, business interruption, or costs of removing or replacing any product, whether or not based on tort, warranty, breach of contract or any other legal theory. Notwithstanding any damages a customer might incur, the aggregate and cumulative liability of Binho Inc. is limited in accordance with its terms and conditions of commercial sale.

Right to make changes

Binho Inc. reserves the right to change the information in this document, including specifications and product descriptions, at any time and without notice. This document supersedes all information supplied prior to its publication.

Suitability for use

Binho Inc. products are not designed, authorized or warranted for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure of a Binho Inc. product can reasonably be expected to result in personal injury, death, or severe property or environmental damage. Any such inclusion or use is at the customer's own risk and Binho Inc. accepts no liability for it.

Applications

Applications described here are illustrative only. Binho Inc. makes no warranty that they are suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their own applications and products, for appropriate design and operating safeguards, and for determining whether a Binho Inc. product is suitable and fit for their purpose. Binho Inc. accepts no liability for assistance with applications or customer product design.

Third-party products and specifications

This document refers to products, boards, specifications and documentation supplied by third parties, which remain the responsibility of their respective owners. Binho Inc. makes no warranty regarding them and their behavior may change without notice. Register maps, protocol details and device identifiers reproduced here were observed on the specific hardware and firmware versions stated in this document and must be confirmed against the vendor's own documentation for the reader's part.

Example code

Source code supplied with this document is provided as an example, for evaluation and reference. It is not qualified for production use, has not been developed under a functional safety or security process, and is licensed under the terms accompanying it in the distribution archive.

Terms and conditions of commercial sale

Binho Inc. products are sold subject to the general terms and conditions of commercial sale published by Binho Inc., unless otherwise agreed in a valid written individual agreement, in which case only the terms of that agreement apply.

Export control

This document and the items it describes may be subject to export control regulations. Export may require prior authorization from the competent authorities.

Security

All products may be subject to unidentified vulnerabilities or may support security standards with known limitations. The customer is responsible for the design and operation of its applications and products throughout their lifecycle to reduce the effect of such vulnerabilities, and for compliance with all legal, regulatory and security requirements concerning its products.

Trademarks

All referenced brands, product names, service names and trademarks are the property of their respective owners. **Binho** and the Binho logo are trademarks of Binho Inc. **CMIS** and **OIF** are trademarks or registered trademarks of the Optical Internetworking Forum. **I3C** and **MIPI** are trademarks or registered trademarks of MIPI Alliance, Inc.