binho

應用筆記

使用 Binho Supernova 量測 CMIS 韌體更新時間

AN0011

文件資訊

關鍵詞	AN0011, Supernova, Pulsar, CMIS, OIF, CDB, 韌體更新, LPL, EPL, tWRITE, tCDBF, tBPC, Nmax, 光模組, 等待期（hold-off）, I2CMCI, I3C, SPIMCI, 每區塊交易數
摘要	在光模組上更新韌體很慢，而顯而易見的補救辦法是改用更快的管理匯流排。依照 CMIS 的存取限制與 Table 10-4 的時序參數逐項推算後可以看出，時間並不是花在線路上：對於 1 MiB 的映像檔，匯流排時脈 只佔總時間的一小部分，而每個區塊的等待期（hold-off）佔了絕大部分。本應用筆記從規格推導出時間預算， 明確說明哪些數字是算術推導、哪些必須實際量測，並提供一個指令，對照規格上限為實際模組的各存取階段計時， 讓主機能夠查明其模組實際需要哪些等待期。

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

修訂版 1.0
6 NOVEMBER 2026

測試範圍。第 4 節中的預算是由規格推導出的算術結果，現已涵蓋規格與本系列所涉及的 全部三種管理介面：I2CMCI、SPIMCI，以及作為 Binho 推導的 I3C。第 5 節中的 `timing` 輸出是 針對韌體目標裝置（而非光模組）的實際執行結果，而且事實證明它量測的是主機路徑而不是目標裝置。 第 5.2 節在兩種主機傳輸方式上量測了兩個轉接器韌體修訂版，發現兩者都沒有改變這個下限；第 5.3 與 5.4 節將該下限追溯到轉接器的 USB 描述元，並說明主機能夠隱藏其中多少；第 7 節說明尚未完成的工作。

1 簡介

在線路上，I3C 單倍資料速率（single data rate）大約比 I2C 快速模式（Fast-mode）快 31 倍。面對模組韌體 更新緩慢問題的主機工程師，自然會預期更快的管理匯流排能讓更新快大約 31 倍。

依照 CMIS 的存取限制推算，會得到不同的答案，而這個差異是可以付諸行動的：選擇哪個通用資料區塊 （Common Data Block）指令來承載酬載，比選擇哪種匯流排更重要。

本筆記推導出時間預算，並提供一種方法，用於查明特定模組實際把時間花在哪裡，而不是規格允許它花在哪裡。

本文件不提供實測傳輸量。 第 7 節說明原因，以及要提供該數值需要執行哪些測試。

1.1 範圍

本文件涵蓋 CMIS 韌體下載的算術：存取長度限制、兩個 Write Firmware Block 指令的區塊大小，以及 Table 10-4 中的等待期參數。本文件提供一個指令，對照這些上限為模組的讀取、頁切換與通用資料區塊 存取階段計時。

本文件**不會**將韌體寫入模組，所提供的公用程式也沒有執行此操作的指令。除記錄其時序外，本文件不涉及 韌體更新狀態機、映像檔簽章或 Run 與 Commit 序列。AN0008 涵蓋暫存器模型，是閱讀本文件的前提。

1.2 適用裝置

表 1. 適用裝置

項目	角色	依據
Binho Supernova	主機轉接器	已測試，第 7 節
Binho Pulsar	主機轉接器	未測試
任何 CMIS 受管模組	目標裝置	**未測試。** 量測使用的是處於目標裝置模式的轉接器

第 4 節中的預算是算術推導，不是量測結果。 它由 OIF-CMIS-05.3 的 5.2.2 節與第 9 章以及 Table 10-4 推導而來，可用 `python cmis_mci.py budget` 重現。其中沒有任何一列經過模組驗證。

1.3 相關文件

表 2. 相關文件

文件	主題
OIF-CMIS-05.3	5.2.2.1 與 5.2.2.2 節，第 9 章（CDB），Table 10-4（時序）
AN0008	使用 Binho Supernova 透過 I2C 啟動 CMIS 光模組
AN0009	使用 Binho Supernova 透過 I3C 管理 CMIS 光模組
AN0010	使用 Binho Pulsar 透過 SPI 啟動 CMIS 光模組

2 所需設備與軟體

表 3. 所需設備

項目
Binho Supernova 或 Binho Pulsar USB 主機轉接器，用於 `timing` 指令
一個 CMIS 受管光模組，用於 `timing` 指令
`budget` 指令不需要任何設備

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

一個主機轉接器可以提供兩種傳輸方式，而您得到哪一種由主機軟體決定，且不會有任何提示。 從 4.5.0 起，Binho 轉接器韌體除了原有的 USB HID 路徑外，還透過廠商自訂批次（bulk）端點承載傳輸。 批次傳輸需要主動啟用：如果主機沒有協商啟用它，轉接器會照常提供 HID 路徑，不報錯也不警告， 這正是該韌體屬於非破壞性版本的原因。本筆記及本系列其餘筆記使用的 `binhosupernova` 僅支援 HID， 低於 1.5.0 的任何 CosmicSDK 也是如此。請向已開啟的裝置查詢該傳輸方式是否可用，而不要根據您安裝的 版本來推斷，因為不支援它的 SDK 會給出舊的數字，並且在此過程中不回報任何異常。第 5.2 節對兩者都 進行了量測，這個區別對結論的影響大於對數字的影響。

3 預算所依據的四個事實

3.1 透過本地酬載傳輸的韌體區塊為 116 位元組

通用資料區塊位於 Page 9Fh。其本地酬載（Local Payload）為 120 位元組，其中 `Write Firmware Block LPL`（指令 `0103h`）用 4 位元組表示區塊偏移量。因此每個區塊剩下 **116 位元組** 韌體。

擴充酬載（Extended Payload）為 A0h 到 AFh 頁，共 16 頁、每頁 128 位元組，透過 `Write Firmware Block EPL`（指令 `0104h`）每個區塊可傳輸 **2048 位元組**。

3.2 一次 WRITE 最多攜帶 8 位元組

5.2.2.2 節直接寫明："A successful WRITE writes a sequence of up to eight given byte values"（一次成功的 WRITE 寫入最多八個給定位元組值的序列）。`FullPageReadSupported` 將 **讀取** 的 `Nmax` 提高到 128，但對寫入沒有任何作用。

唯一的例外是緊接在後的註記："A WRITE with more than 8 bytes may be allowed when specified explicitly in chapter 7.8"（當第 7.8 章明確規定時，可以允許超過 8 位元組的 WRITE）， 也就是通用資料區塊擴充酬載區域。

因此，一個 116 位元組的本地酬載區塊需要 `ceil(116 / 8)` = **15 次獨立的寫入交易**，每次都有自己的位址階段。

3.3 接著模組停止回應

表 4. 等待期參數，CMIS Table 10-4 最大值

參數	最大值	適用於以下操作之後
tREAD	500 微秒	暫存器讀取
tWRITE	10 ms	寫入揮發性記憶體
tWRITENV	80 ms	寫入非揮發性記憶體
tBPC	10 ms	記憶庫（bank）或頁切換
tCDBF	4960 ms	前景 CDB 指令。預設值為 80 ms
tMgmtInit	2000 ms	重設後，直到進入 ModuleLowPwr

這些都是最大值。模組可以宣告不超過這些值的任意數值，而第 5 節的內容就是查明特定模組實際需要多長時間。

3.4 幾乎所有 CMIS 記憶體都是揮發性的

規格明確說明了這一點，因此 **10 ms 的 tWRITE 是常見情況**，而不是樂觀情況。

等待期是**每個區塊支付一次**，而不是每次交易支付一次。這就是全部的槓桿所在：對於 1 MiB 的映像檔， 2048 位元組的區塊需要支付 512 次，而 116 位元組的區塊需要支付 9040 次。

4 預算

```
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.
```

請將前兩列對照來看。線路速度提高 31 倍，節省 35 秒。在原本的低速時脈下，改變承載酬載的 通用資料區塊指令，節省 97 秒。

4.1 三種管理介面，給出同一個答案

SPIMCI 列承載了整個系列旨在進行的比較，因為 SPIMCI 是 CMIS 實際規定的唯一另一種管理介面。Appendix B.3.7.1 將每筆交易組框為一個四位元組控制字、`N` 個流量控制位元組與最多 **2048 位元組** 的 酬載，而 `00h:27.3-0` 處的 `MciSpeedConfiguration` 範圍為 1 MHz 到 **50 MHz**。這是 I2C 時脈的 125 倍，也是分頁介面所能進行的最大存取的 16 倍。

它帶來 24.0 倍的提速，而 I2CMCI 為 21.5 倍。其中有兩個事實值得單獨指出。

在模組實際啟動時的速度下，SPIMCI 在線路上比 I3C 慢。 離開 MgmtInit 時，`MciSpeedConfiguration` 的初始值一律為 1 MHz，而在 1 MHz 下，擴充酬載路徑在線路上花費 8.41 s，I3C 則為 0.78 s。 時脈必須由主機刻意提高：主機需讀取模組的能力宣告並寫入該欄位。

無論速度如何，SPIMCI 改變的是交易數。 在 I2CMCI 與 I3C 上（兩者都將一次存取限制為一頁），一個擴充酬載區塊需要 16 次交易，而在 SPIMCI 上只需**一次**。這一欄從不出現在頻寬數據中， 而第 5 節顯示，它正是主機感受得到的那一欄：每次交易都要經過主機來回一次，而在測試台上，無論匯流排在做什麼，這個來回大約都是兩毫秒。

因此，這三種介面在線路上相差 125 倍，在總時間上相差 24 倍，但在時間花在哪裡這一點上完全一致。 韌體更新不是頻寬問題。

表 5. 預算模型包含與不包含的內容

包含	不包含
每區塊交易數，由存取長度限制得出	主機 USB 延遲，它不是匯流排屬性
每次交易的匯流排時間，以 4 個組框位元組加酬載、每位元組 9 位元計算	tCDBF，因模組與指令而異
每個區塊一次 tWRITE	Run Image 與 Commit Image 步驟
	存取被拒絕後的重試

> **匯流排時間欄是對線路佔用的估計，不是實測傳輸量。** 它刻意做得很粗略，因為它唯一的作用是顯示它與等待期欄相比很小。更精細的模型不會改變結論，而實測值所表徵的主機 USB 路徑與匯流排一樣多。

5 量測實際模組

```
python cmis_mci.py timing
```

該指令不寫入任何韌體。它為構成更新的各存取階段計時：單位元組讀取、達到寫入限制的八位元組讀取、 模組宣告支援時的整頁讀取、雙向頁切換，以及切換到通用資料區塊頁，並將每一項與其 Table 10-4 上限並列輸出。

```
$ 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.
```

應將其解讀為對主機的量測，而不是對目標裝置的量測。 每一列大約都是每次 USB 來回兩毫秒： 一次存取約 2 ms，CDB 頁選擇的兩次暫存器寫入約 4 ms，雙向頁切換的四次寫入約 8 ms。單位元組讀取 是 tREAD 的四倍，而八位元組讀取並不比單位元組讀取慢，只有在計時對象不是線路上的位元組時才可能如此。

此次執行中的目標裝置是處於 I2C 目標裝置模式的轉接器，它會立即應答，從不讓匯流排等待。因此這些 數字顯示的是在模組尚未貢獻任何時間之前，主機路徑施加的下限，在此配置下該下限約為每次存取 2 ms。 第 7.2 節提出結論。

結果中需要留意的內容：

中位數遠低於規格最大值，表示模組比規格允許的更快，而每次寫入後都等待最大值的主機，會讓大部分 更新時間白白浪費。這是該量測所能告訴您的最有價值的資訊。

頁切換的最大值接近規格上限，表示模組有時確實需要 tBPC。假設中位數的主機會間歇性失敗， 這是最難除錯的失敗模式。

單位元組讀取高於 tREAD，是主機端或轉接器端的額外負擔，而不是模組的等待期。tREAD 是對 I2C 時脈延展 （clock stretching）的抽象，時脈延展在匯流排擷取中是可見的，而主機端計時器無法將其與其他因素區分開來。

5.1 針對 I3C CMIS 目標裝置的相同量測

上述執行是透過 I2CMCI 針對處於目標裝置模式的轉接器進行的。透過 I3C 針對 AN0009 的 CMIS 目標裝置韌體（它實作的是暫存器模型，而不是平面記憶體）重複量測，沒有任何重要變化：

```
$ 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.
```

快 31 倍的匯流排，加上一個由韌體而非平面記憶區應答的目標裝置，所有列都保持不變。這與預算的預測結果 相同，只是從另一個方向得出。

5.2 下限屬於主機軟體堆疊，而目前的軟體堆疊無法擺脫它

Supernova 韌體 4.5.0 將轉接器傳輸移到廠商自訂批次端點上，提高了傳輸上限並重構了 USB 路徑。透過本筆記提供的公用程式量測，它沒有帶來任何變化。

表 6. 兩個轉接器韌體修訂版上的相同四個階段，三次執行的中位數

階段	4.4.0	4.5.0
單位元組讀取	2.006 ms	2.036 ms
8 位元組讀取，即 WRITE 限制	2.003 ms	2.036 ms
頁切換 00h 到 01h 再到 00h	7.993 ms	8.061 ms
選擇 CDB 頁 9Fh	3.987 ms	4.035 ms

該電路板從 4.4.0 升到 4.5.0，再回到 4.4.0，然後再次升級，並在每種狀態下都進行了量測，因為跨一次重新 燒錄的單次比較無法區分韌體變化與其他發生變化的因素。

> **上表中的兩者都是相容路徑。** 較新的傳輸方式需要主機主動啟用：如果主機沒有協商啟用它，轉接器會照常提供舊路徑，這正是該韌體屬於非破壞性版本的原因。本筆記及整個系列所執行的 SDK `binhosupernova` 4.2.0 透過 USB HID 傳輸，完全不含廠商自訂批次端點的程式碼。因此這組數據說明 相容路徑沒有退化，但單憑它無法說明 4.5.0 新增的傳輸方式的任何情況。

透過確實會協商啟用它的主機軟體堆疊重複相同的四個階段，就能回答上表無法回答的問題。各階段沒有變化。

表 7. 相容傳輸方式與批次傳輸方式上的相同四個階段，皆在韌體 4.5.0 上

階段	USB HID	批次傳輸方式
單位元組讀取	2.036 ms	2.009 ms
8 位元組讀取，即 WRITE 限制	2.036 ms	2.006 ms
頁切換 00h 到 01h 再到 00h	8.061 ms	8.019 ms
選擇 CDB 頁 9Fh	4.035 ms	3.993 ms

右側一欄是在傳輸方式回報自身可用並宣告 32768 位元組上限的情況下量得的，因此它不是換了名字的第二次 相容路徑執行。該傳輸方式從 CosmicSDK 1.5.0 起提供；較舊的 SDK 會給出左側一欄的結果，同時不回報 任何異常，這正是本次量測必須排除而不能假定的失敗。檢查方式是向已開啟的裝置進行能力查詢，而不是看 版本號碼。不過，這並不是關於該傳輸方式的負面結果，因為在同一工作階段中，該傳輸方式完全達成了其設計目的：

表 8. 從 SPI 輸出位元組，同一轉接器與工作階段，依傳輸大小

傳輸大小	請求路徑	批次路徑
8 位元組	2.05 ms	2.00 ms
1024 位元組	35.0 ms	5.0 ms
32768 位元組	不可用	114.8 ms，285 kB/s

更寬的傳輸方式是酬載的槓桿，而不是延遲的槓桿。 在一千位元組時，它快七倍。在八位元組時速度相同， 因為八位元組傳輸的成本就是一次來回，而管道變寬並不會使來回變短。

這正是第 4 節在主機端的重述，也是這兩節結論一致的原因。`5.2.2.2` 將一次 WRITE 限制為八位元組、 一次存取限制為一頁，因此 **在分頁管理介面上，CMIS 從不發出足夠大的傳輸，讓這些改進派上用場。** 確實會發出這種傳輸的介面是 SPIMCI，可達 2048 位元組，這對應第 4 節中將每區塊十六次交易壓縮為一次的 那一列。本測試台上沒有 SPIMCI 目標裝置，因此該列仍是算術推導。

5.3 兩毫秒從何而來

兩個獨立的主機軟體堆疊在一個數值上三位有效數字都一致，這不是巧合，原因在於轉接器的 USB 描述元， 而不在於其中任何一個軟體堆疊。

表 9. 轉接器的端點，取自其組態描述元

介面	端點	傳輸類型	最大封包大小	間隔
人機介面裝置（Human Interface Device）	0x01 OUT, 0x81 IN	中斷	64 位元組	1 ms
廠商自訂	0x02 OUT, 0x82 IN	批次	64 位元組	無

指令平面是那對中斷端點，而中斷端點是依排程輪詢的。 這裡的間隔是一毫秒，因此請求不能在下一個訊框 之前送出，其應答也不能在再下一個訊框之前抵達。這解釋了兩毫秒中的大部分，而且這並不是任何 SDK 在做的 工作：它是匯流排允許傳輸發生的時機。剩下的是轉接器韌體的周轉時間與主機軟體堆疊，本量測沒有將兩者分開。

批次端點對承載酬載，而不是指令。這就是前面結果的全部原因：加寬酬載路徑後，指令來回維持原樣， 因此一個八位元組存取（只有一個指令與一個應答，別無其他）無法變得更快。

同一模型還能預測另一欄，這才是值得做的驗證。請求路徑上的 1024 位元組傳輸是在指令本身中攜帶的， 每次 64 位元組，因此它是十六個訊框的酬載加上指令來回：約 34 ms，實測為 35.0 ms。在批次路徑上， 酬載完全脫離排程，只剩下指令來回：約 2 ms 加上傳輸時間，實測為 5.0 ms。

因此這個下限有了名字，它也告訴主機該如何因應。 能改變它的是指令的數量，以及同時有多少指令在等待。兩者都與指令所經過的管道大小無關。

5.4 下限的一半是應答，主機可以隱藏它

如果一個指令與它的應答各佔一個訊框，那麼等到每個應答回來才送出下一個指令的主機要為兩者付出代價，而讓 指令保持在途（in flight）的主機應該只為前者付出代價。這是可以驗證的。透過非阻塞介面發出同樣的 1 位元組讀取 200 次，一次等待每個應答，一次不等待：

表 10. 同樣的 200 次讀取，等待每個應答與讓它們保持在途的比較

	總計	每個指令
逐一執行	400.0 ms	2.000 ms
全部在途	200.9 ms	1.005 ms

管線化使其減半，被消除的那一半就是應答。 剩下的是每個訊框一個指令，這是輸出方向的輪詢間隔，主機 除了少發指令之外無法低於這個值。200 個指令全部得到應答，因此這不是對指令被丟棄的量測。

對需要它的主機而言有兩個推論。節省是真實的但有限：它將一個擴充酬載區塊的主機時間成本從約 32 ms 降到約 16 ms，而 tWRITE 為 10 ms，因此主機不再主導該區塊，但也沒有變得可以忽略。而且只有在各存取 彼此不相依時才能使用，而在一個韌體區塊內部它們確實不相依，因為它們是寫入同一擴充酬載區域的連續寫入。

需要明確指出的是，**本筆記提供的公用程式並不這樣做。** 它使用阻塞介面，一次一個指令，這就是第 5 節 中每個數字都是兩毫秒那一欄的原因。上面的量測是針對非阻塞介面的獨立測試工具，其目的是

評估這個選項 的價值，而不是描述該公用程式。

這一點值得與第 4 節放在一起看。一個擴充酬載區塊是十六次存取，約 32 ms 的主機時間，而 tWRITE 為 10 ms，因此在本測試台上，主導該區塊的是主機而不是模組。管線化將其降到約 16 ms，而 **SPIMCI 將其 變為一次交易而不是十六次**，這是本筆記中能影響第 5 節所量測項目的最大變化。一個假設自身來回成本 可以忽略的主機在這裡是錯誤的，而在分頁管理介面上，它最多只能將這些成本減半。

這就是第 4 節與第 5 節從相反方向得出同一結論的原因。

6 如何運用這項發現

對於主機軟體。 在下載開始時查詢一次 `Firmware Management Features`（CDB 指令 `0041h`），並在模組支援時優先使用 `Write Firmware Block EPL`（`0104h`）而不是 `0103h`。不要對任何一方預設立場。 接著在一個區塊內發出各存取時不要等待每個應答：第 5.4 節量得這樣做的主機成本減半，之所以可行是因為 一個區塊內的存取彼此不相依，而它需要的是非阻塞介面，而不是更快的介面。

對於模組韌體。 宣告您能夠遵守的等待期時間。tWRITE 是最大值，不是目標值，而第 5 節中的量測就是 主機發現您留下多少未用餘裕的方法。

對於制定新 MCI 繫結規格的任何人。 SPIMCI 已經能在一次交易中承載最多 2048 位元組，第 4.1 節對此 做了估算，而 I3C 透過 SETMWL、SETMRL 與 GETMXDS 原生協商同樣的能力。繼承了 I2CMCI 8 位元組寫入 限制的繫結，也會隨之繼承這一整個問題，而且無論以什麼時脈運作都會繼承。

> **單獨為 Run Image 步驟編列預算。** CDB 指令 `0109h` 所需時間與一次完整的模組重新開機相同，"mainly governed by the tMgmtInit timing parameter"（主要由 tMgmtInit 時序參數決定），因此應為它 預留大約兩秒，而不是一個 CDB 逾時時間。

7 量測結果

7.1 測試配置

表 11. 測試配置，第 5 節的 I2CMCI 執行

項目	值
控制器	Binho Supernova，序號 `1912DEB5…79FA`，韌體 4.4.0，硬體修訂版 C
目標裝置	Binho Supernova，序號 `ED7BF8F3…F7BB`，韌體 4.4.0，設定為位址 0x50 的 I2C 目標裝置
目標裝置內容	CMIS 5.3 映像檔，Appendix C 中的 400G QSFP-DD 範例
SDK	`binhosupernova` 4.2.0，Python 3.10.12
主機	Ubuntu 22.04，x86-64
匯流排	3.3 V，400 kHz
重複次數	每個階段 20 次，回報中位數與最大值
日期	2026 年 9 月 1 日

表 12. 測試配置，第 5.1 與 5.2 節的 I3C 執行

項目	值
控制器	Binho Supernova，序號 `4F970418…1111`，韌體 4.4.0 與 4.5.0
主機傳輸方式	透過 USB HID 的 `binhosupernova` 4.2.0，以及透過廠商自訂批次端點的 CosmicSDK 1.5.0，後者經能力查詢確認可用，並宣告 32768 位元組上限
目標裝置	執行 AN0009 CMIS 目標裝置韌體的 NXP FRDM-MCXN947，動態位址 0x09
目標裝置內容	CMIS 5.3 映像檔，Appendix C 中的 400G QSFP-DD 範例
SDK	`binhosupernova` 4.2.0，Python 3.14
主機	Arch Linux，x86-64
匯流排	3.3 V，I3C SDR，推挽 2.5 Mbit/s、25% 工作週期，開汲極 1 Mbit/s
重複次數	每個階段 20 次，回報中位數與最大值，每個韌體執行三次
第 5.4 節	針對非阻塞介面的獨立測試工具，每組 200 次讀取，執行三次
日期	2026 年 9 月 3 日

這兩種配置使用不同的測試台、不同的主機與不同的 Python 版本。本筆記中沒有任何地方將它們相互比較， 而且只有第二種配置用於第 5.2 節中的韌體比較，在該比較中主機與目標裝置保持不變，轉接

器韌體是唯一 變化的因素。

7.2 結果

推導結果。 第 4 節中的預算可用所提供的公用程式重現，並且僅依據第 3 節中引用的規格條文。它是算術推導，本文件不將其表述為其他任何東西。

實測結果，而它量測的是主機。 第 5 節的執行確立了在此主機、轉接器與 SDK 上，每次暫存器存取約 2 ms 的下限，而所用目標裝置本身不施加任何等待期。這個數字屬於 USB 路徑，而不屬於匯流排，也不屬於 模組：單位元組讀取超過 tREAD 四倍，而八位元組讀取並不比單位元組讀取慢，只有在線路時間不是主導項時， 這兩點才可能成立。

這就是 **本文件中任何地方都沒有出現端對端傳輸量數據** 的原因。以這種方式量得的數值表徵的是主機， 若將其作為匯流排或模組的屬性引用，就會造成一種日後難以察覺的錯誤。

不過，它反而強化了本筆記自身的論點，而不是削弱它。第 4 節論證韌體更新的時間不是花在線路上。 量測進一步顯示，時間也不是花在主機來回成本上。兩者與每個區塊的等待期相比都很小，並且當區塊變大時， 兩者都會因同一原因而縮小：**更少的存取與更少的區塊。** 每次存取 2 ms 的下限，使每個本地酬載區塊成本 30 ms、每個擴充酬載區塊成本 32 ms，但擴充酬載區塊承載的韌體多 17 倍。

在較新的轉接器韌體與較新的傳輸方式上再次量測，結果沒有變化。 Supernova 韌體 4.5.0 使全部四個 階段維持在 4.4.0 時的水準，該韌體新增的批次傳輸方式也是如此，量測時該傳輸方式回報自身可用。在同一 工作階段中，該傳輸方式在一千位元組的傳輸上快七倍，因此這是一個有界的結果，而不是配置失敗：更寬的 傳輸方式是酬載的槓桿而不是延遲的槓桿，而一個八位元組存取無論哪種方式都只需一次來回。

這是從另一端重述第 4 節。`5.2.2.2` 將一次 WRITE 限制為八位元組、一次存取限制為一頁，因此在分頁管理 介面上，CMIS 從不發出足夠大的傳輸，讓更寬的管道派上用場。能影響此表的槓桿是來回次數（即每區塊交易數 一欄所統計的、也是 SPIMCI 所改變的），以及同時有多少來回處於未完成狀態。

其中第二個槓桿值一半，而且現在就能使用。 第 5.4 節在不等待每個應答的情況下發出同一存取 200 次， 量得 1.005 ms 對比 2.000 ms，每個指令都得到了應答。剩下的是每個 USB 訊框一個指令。因此，寫入韌體區塊 的主機可以在區塊內（各存取相互獨立之處）不等待應答，從而將自身的成本減半，而在此之後，在分頁介面上 它就沒有其他槓桿可用了。

這也是第 4 節中每區塊交易數這一欄值得關注的原因。SPIMCI 在單次交易中承載一個擴充酬載區塊，而分頁 介面需要十六次，以每次交易約兩毫秒的主機成本計算，這就是在匯流排做任何事情之前，每個區塊 2 ms 對比 32 ms。

在硬體上，針對模組。 此處沒有任何結果。截至修訂版 1.0 尚未完成的工作：針對實際模組執行 `timing` 指令，這是確立模組實際需要哪些等待期的唯一方法；透過本地酬載與擴充酬載兩條路徑完成一次完整的 韌體下載，這是能夠支持端對端數據的唯一方法；以及在 SPIMCI 上進行任何量測，本測試台上不存在 SPIMCI 目標裝置。第 4 節中的每個 SPIMCI 數字都是根據 Appendix B.3 的算術推導，並已如此標示。

在完成上述執行之前，本筆記所支持的結論是關於 **時間被允許花在哪裡**，以及在模組貢獻任何時間之前 主機的成本是多少。

8 疑難排解

表 13. 疑難排解

症狀	可能原因與處理措施
`timing` 回報單位元組讀取遠高於 tREAD	主機或轉接器額外負擔，而非模組所致。請與匯流排擷取結果比較
每個階段都接近其上限	模組可能確實很慢，或者有另一個控制器在爭用匯流排
輸出中沒有 94 位元組讀取	模組未宣告支援整頁讀取，因此 `Nmax` 維持為 8
各次執行之間的中位數差異很大	有其他程式在使用 USB 路徑。關閉其他主機工具後重試
在較新的轉接器韌體上數字看起來相同	這是預期結果，第 5.2 與 5.3 節對此作了說明。同時請確認您使用的是哪種傳輸方式：如果 SDK 沒有協商啟用批次傳輸，轉接器會提供 HID 路徑而不報錯。請向 SDK 查詢，不要根據版本號碼推斷
每次存取約 2 ms，無論如何更改都沒有改善	其中大部分是 USB 排程：指令平面是以 1 ms 輪詢的中斷端點，因此一個指令及其應答跨越兩個訊框。第 5.3 節列出了描述元。應減少指令的數量，而不是指令的大小
頁切換超過 tBPC 的兩倍	模組所需時間超過規格允許的範圍。在歸咎於主機之前，請先擷取匯流排

9 縮寫

表 14. 縮寫

術語	含義
CDB	通用資料區塊（Common Data Block）
CMIS	通用管理介面規格（Common Management Interface Specification）
EPL	擴充酬載（Extended Payload），CDB 頁 A0h 到 AFh
LPL	本地酬載（Local Payload），位於 CDB 頁 9Fh
MCI	管理通訊介面（Management Communication Interface）
Nmax	一次暫存器存取中的最大位元組數
OIF	光網際網路論壇（Optical Internetworking Forum）
SDR	單倍資料速率（Single data rate）

10 關於文件中原始碼的說明

本文件中展示的以及隨附封存檔中提供的範例程式碼僅供評估與參考。這些程式碼依據 MIT 授權條款發布， 授權條款全文包含在封存檔中。

表 15. 資源封存檔的內容

檔案	說明
`AN0011.md`	本文件的 Markdown 原始檔
`assets/cmis_mci.py`	參考公用程式，由 AN0008 至 AN0011 共用
`assets/LICENSE`	涵蓋範例程式碼的 MIT 授權條款

11 修訂歷史

表 16. 修訂歷史

修訂版	日期	說明
0.1	2026 年 9 月 1 日	供內部審查的草稿。預算由規格推導；未進行量測。
0.2	2026 年 9 月 1 日	針對 Supernova I2C 目標裝置測試治具執行 `timing`。貼上了輸出，並將其回報為對主機路徑而非模組的量測。
0.3	2026 年 9 月 2 日	由 AN0010 重新編號。量測結果與預算均無變化。
0.4	2026 年 9 月 3 日	預算擴充到 SPIMCI，包括其初始的 1 MHz 與 `MciSpeedConfiguration` 最高的 50 MHz，並指出每區塊交易數一欄是主機感受得到的那一欄。第 5 節新增針對 AN0009 CMIS 目標裝置透過 I3C 進行的相同量測，以及 Supernova 韌體 4.4.0 與 4.5.0 在兩種主機傳輸方式上的比較，兩者都沒有改變下限；並在同一工作階段中進行了 SPI 傳輸大小掃描，顯示更寬的傳輸方式在一千位元組時快七倍、在八位元組時相同。在該傳輸方式加入 CosmicSDK 1.5.0 後，針對其重複了兩種傳輸方式的執行，結果沒有變化。新增第 5.3 節，將兩毫秒下限追溯到指令平面所使用的 1 ms 中斷端點，並以傳輸大小掃描驗證了該模型。公用程式版本為 1.1。
0.5	2026 年 9 月 3 日	新增第 5.4 節，評估讓指令保持在途的價值，量得每個指令的成本減半且每個指令都得到應答；第 6 節現在建議主機在區塊內這樣做。剩下的下限是每個訊框一個指令。
1.0	2026 年 11 月 6 日	正式發布。

法律資訊（英文）

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** 與 **OIF** 是 Optical Internetworking Forum 的商標或註冊商標。 **I3C** 與 **MIPI** 是 MIPI Alliance, Inc. 的商標或註冊商標。