binho

애플리케이션 노트

Binho Supernova를 사용한 CMIS 펌웨어 업데이트 시간 측정

AN0011

문서 정보

키워드	AN0011, Supernova, Pulsar, CMIS, OIF, CDB, 펌웨어 업데이트, LPL, EPL, tWRITE, tCDBF, tBPC, Nmax, 광 모듈, 홀드오프, I2CMCI, I3C, SPIMCI, 블록당 트랜잭션 수
요약	광 모듈의 펌웨어 업데이트는 느리며, 가장 먼저 떠오르는 해결책은 더 빠른 관리 버스입니다. CMIS의 액세스 제한과 Table 10-4의 타이밍 파라미터를 따라 계산해 보면 시간이 선로에서 소비되는 것이 아님을 알 수 있습니다. 1 MiB 이미지의 경우 버스 클록은 전체의 작은 부분만 차지하고, 블록별 홀드오프(hold-off)가 대부분을 차지합니다. 이 애플리케이션 노트는 사양에서 시간 예산을 도출하고, 어떤 수치가 산술이며 어떤 수치를 측정해야 하는지 분명히 밝히며, 실제 모듈의 액세스 단계를 사양 상한과 비교하여 측정하는 명령을 제공합니다. 이를 통해 호스트는 자신의 모듈이 실제로 어떤 홀드오프를 사용하는지 확인할 수 있습니다.

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

개정 1.0
6 NOVEMBER 2026

검증 범위. 섹션 4의 예산은 사양에서 도출한 산술이며, 이제 사양과 이 시리즈가 다루는 세 가지 관리 인터페이스, 즉 I2CMCI, SPIMCI, 그리고 Binho의 도출로서의 I3C를 모두 포함합니다. 섹션 5의 `timing` 출력은 광 모듈이 아닌 펌웨어 타깃에 대한 실제 실행 결과이며, 타깃이 아니라 호스트 경로를 측정한다는 것이 밝혀졌습니다. 섹션 5.2는 두 가지 호스트 트랜스포트에서 두 가지 어댑터 펌웨어 리비전을 측정하여 어느 쪽도 하한을 움직이지 않음을 확인하고, 섹션 5.3과 5.4는 그 하한을 어댑터의 USB 디스크립터까지 추적하고 호스트가 그중 얼마를 숨길 수 있는지 보여 주며, 섹션 7은 미완료 사항을 기술합니다.

1 소개

I3C SDR(single data rate)은 선로상에서 I2C Fast-mode보다 약 31배 빠릅니다. 모듈 펌웨어 업데이트가 느린 상황에 직면한 호스트 엔지니어는 더 빠른 관리 버스가 업데이트를 약 31배 빠르게 해 줄 것이라고 기대하는 것이 당연합니다.

CMIS 액세스 제한을 따라 산술을 진행하면 다른 답이 나오며, 그 차이는 실제 조치로 이어집니다. 어떤 버스를 선택하느냐보다 어떤 Common Data Block(공통 데이터 블록) 명령으로 페이로드를 전달하느냐가 더 중요합니다.

이 노트는 예산을 도출하고, 특정 모듈이 사양상 허용되는 곳이 아니라 실제로 어디에서 시간을 소비하는지 확인하는 방법을 제공합니다.

이 문서는 측정된 처리량을 제시하지 않습니다. 섹션 7에서 그 이유와, 이를 제시하려면 무엇을 실행해야 하는지 설명합니다.

1.1 범위

이 문서는 CMIS 펌웨어 다운로드의 산술, 즉 액세스 길이 제한, 두 가지 Write Firmware Block 명령의 블록 크기, 그리고 Table 10-4의 홀드오프 파라미터를 다룹니다. 모듈의 읽기, 페이지 변경, Common Data Block 액세스 단계를 이러한 상한과 비교하여 측정하는 명령을 제공합니다.

이 문서는 모듈에 펌웨어를 기록하지 **않으며**, 제공되는 유틸리티에도 그런 명령이 없습니다. 펌웨어 업데이트 상태 머신, 이미지 서명, Run 및 Commit 시퀀스는 그 타이밍을 언급하는 것 이상으로 다루지 않습니다. 레지스터 모델은 AN0008에서 다루며, 이 문서의 선행 조건입니다.

1.2 적용 디바이스

표 1. 적용 디바이스

항목	역할	근거
Binho Supernova	호스트 어댑터	검증됨, 섹션 7
Binho Pulsar	호스트 어댑터	검증되지 않음
모든 CMIS 관리 모듈	타깃	**검증되지 않음.** 측정에는 타깃 모드의 어댑터를 사용했습니다

섹션 4의 예산은 측정이 아니라 산술입니다. OIF-CMIS-05.3의 섹션 5.2.2와 9장, 그리고 Table 10-4에서 도출했으며, `python cmis_mci.py budget`으로 재현할 수 있습니다. 어떤 행도 모듈에 대해 확인되지 않았습니다.

1.3 관련 문서

표 2. 관련 문서

문서	주제
OIF-CMIS-05.3	섹션 5.2.2.1 및 5.2.2.2, CDB는 9장, 타이밍은 Table 10-4
AN0008	Binho Supernova를 사용한 I2C 기반 CMIS 광 모듈 초기 구동
AN0009	Binho Supernova를 사용한 I3C 기반 CMIS 광 모듈 관리
AN0010	Binho Pulsar를 사용한 SPI 기반 CMIS 광 모듈 초기 구동

2 필요한 장비 및 소프트웨어

표 3. 필요한 장비

항목
Binho Supernova 또는 Binho Pulsar USB 호스트 어댑터, `timing` 명령용
CMIS 관리 광 모듈, `timing` 명령용
`budget` 명령에는 아무것도 필요하지 않음

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

호스트 어댑터는 두 가지 트랜스포트를 제공할 수 있으며, 어느 쪽을 사용하게 되는지는 호스트 소프트웨어가 아무런 표시 없이 결정합니다. Binho 어댑터 펌웨어 4.5.0부터는 기존 USB HID 경로뿐 아니라 벤더 전용 벌크 엔드포인트로도 전송을 처리합니다. 벌크 쪽은 옵트인 방식입니다. 호스트가 이를 협상하지 않는 어댑터는 오류나 경고 없이 HID 경로를 그대로 제공하며, 이 덕분에 이 펌웨어는 호환성을 깨지 않는 릴리스가 됩니다. 이 노트와 시리즈의 나머지 노트가 사용하는 `binhosupernova`는 HID 전용이며, 1.5.0보다 오래된 CosmicSDK도 마찬가지입니다. 설치한 릴리스로 추론하지 말고 열린 디바이스에 트랜스포트를 사용할 수 있는지 질의하십시오. 이를 지원하지 않는 SDK는 이전 수치를 내면서도 아무런 이상을 보고하지 않기 때문입니다. 섹션 5.2는 둘 다 측정하며, 이 구분은 수치보다 결론에 더 중요합니다.

3 예산의 근거가 되는 네 가지 사실

3.1 Local Payload를 통한 펌웨어 블록은 116바이트

Common Data Block은 Page 9Fh를 차지합니다. 그 Local Payload(로컬 페이로드)는 120바이트이며, 그 중 `Write Firmware Block LPL`(명령 `0103h`)이 블록 오프셋에 4바이트를 사용합니다. 따라서 블록당 펌웨어는 **116바이트**가 남습니다.

Extended Payload(확장 페이로드)는 A0h부터 AFh까지의 페이지로, 128바이트 페이지 16개이며 `Write Firmware Block EPL`(명령 `0104h`)을 통해 블록당 **2048바이트**가 됩니다.

3.2 WRITE 한 번은 최대 8바이트를 전달

섹션 5.2.2.2는 이를 직접 명시합니다. "A successful WRITE writes a sequence of up to eight given byte values"(성공한 WRITE는 주어진 최대 8개의 바이트 값 시퀀스를 기록한다).
`FullPageReadSupported`는 **읽기**의 `Nmax`를 128로 높이지만 쓰기에는 아무런 영향이 없습니다.

유일한 예외는 바로 뒤의 주석입니다. "A WRITE with more than 8 bytes may be allowed when specified explicitly in chapter 7.8"(7.8장에서 명시적으로 지정된 경우 8바이트를 초과하는 WRITE가 허용될 수 있다). 이는 Common Data Block Extended Payload 영역을 말합니다.

따라서 116바이트 Local Payload 블록은 `ceil(116 / 8)` = **15개의 개별 쓰기 트랜잭션**이며, 각각 고유한 주소 단계를 가집니다.

3.3 그 후 모듈은 응답을 멈춤

표 4. 홀드오프 파라미터, CMIS Table 10-4 최댓값

파라미터	최댓값	적용 시점
tREAD	500마이크로초	레지스터 읽기 후
tWRITE	10 ms	휘발성 메모리에 쓰기 후
tWRITENV	80 ms	비휘발성 메모리에 쓰기 후
tBPC	10 ms	뱅크 또는 페이지 변경 후
tCDBF	4960 ms	포그라운드 CDB 명령 후. 기본값은 80 ms
tMgmtInit	2000 ms	리셋 후 ModuleLowPwr에 도달할 때까지

이 값들은 최댓값입니다. 모듈은 이 값 이하의 어떤 값이든 광고할 수 있으며, 섹션 5는 특정 모듈이 실제로 얼마나 걸리는지 확인하는 내용입니다.

3.4 CMIS 메모리는 거의 모두 휘발성

사양이 이를 명시적으로 밝히고 있으므로, **10 ms의 tWRITE가 일반적인 경우**이며 낙관적인 경우가 아닙니다.

홀드오프는 트랜잭션마다가 아니라 **블록당 한 번** 발생합니다. 이것이 레버의 전부입니다. 1 MiB 이미지에서 2048바이트 블록은 이를 512번 치르고, 116바이트 블록은 9040번 치릅니다.

4 예산

```
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.
```

처음 두 행을 서로 비교해 보십시오. 선로 속도를 31배 높이면 35초가 줄어듭니다. 원래의 느린 클록에서 페이로드를 전달하는 Common Data Block 명령을 바꾸면 97초가 줄어듭니다.

4.1 세 가지 관리 인터페이스, 모두 같은 답

SPIMCI 행은 이 시리즈 전체가 하려는 비교를 담고 있습니다. SPIMCI는 CMIS가 실제로 규정하는 유일한 다른 관리 인터페이스이기 때문입니다. Appendix B.3.7.1은 모든 트랜잭션을 4바이트 제어 워드, `N` 개의 흐름 제어 바이트, 최대 **2048바이트**의 페이로드로 프레이밍하며, `00h:27.3-0` 의 `MciSpeedConfiguration` 은 1 MHz부터 **50 MHz**까지입니다. 이는 I2C 클록의 125배이며, 페이지 방식 인터페이스가 수행할 수 있는 가장 큰 액세스의 16배입니다.

이를 통해 I2CMCI의 21.5배에 비해 24.0배의 속도 향상을 얻습니다. 여기서 따로 짚어 볼 만한 사실이 두 가지 있습니다.

모듈이 실제로 시작하는 속도에서는 SPIMCI가 선로상에서 I3C보다 느립니다. MgmtInit을 빠져나올 때 `MciSpeedConfiguration`의 초깃값은 항상 1 MHz이며, 1 MHz에서 Extended Payload 경로는 선로상에서 8.41 s를 소비하는 반면 I3C는 0.78 s입니다. 클록은 모듈의 광고를 읽고 해당 필드에 기록하는 호스트가 의도적으로 높여야 합니다.

어떤 속도에서든 SPIMCI가 바꾸는 것은 트랜잭션 수입니다. Extended Payload 블록은 둘 다 액세스를 한 페이지로 제한하는 I2CMCI와 I3C에서는 16개 트랜잭션이고, SPIMCI에서는 **1개**입니다. 이 열은 대역폭 수치에 결코 나타나지 않지만, 섹션 5가 보여 주듯이 호스트가 체감하는 것은 바로 이 열입니다. 모든 트랜잭션은 호스트를 거치는 왕복 비용을 수반하며, 벤치에서는 버스가 무엇을 하든 그 왕복이 약 2밀리초입니다.

따라서 세 인터페이스는 선로상에서 125배, 전체에서 24배 차이가 나지만, 시간이 어디에 소비되는지에 대해서는 완전히 일치합니다. 펌웨어 업데이트는 대역폭 문제가 아닙니다.

표 5. 예산 모델에 포함되는 것과 포함되지 않는 것

포함	제외
액세스 길이 제한에서 산출한 블록당 트랜잭션 수	호스트 USB 지연 시간(버스 속성이 아님)
트랜잭션당 버스 시간(4바이트 프레이밍과 페이로드, 바이트당 9비트로 계산)	tCDBF(모듈과 명령에 따라 다름)
블록당 한 번의 tWRITE	Run Image 및 Commit Image 단계
	액세스가 거부된 후의 재시도

> **버스 시간 열은 선로 점유의 추정치이며 측정된 처리량이 아닙니다.** 이는 의도적으로 대략적입니다. 그 유일한 역할은 홀드오프 열에 비해 작다는 것을 보여 주는 것이기 때문입니다. 더 정교한 모델로도 결론은 바뀌지 않으며, 측정값은 버스만큼이나 호스트의 USB 경로를 특성화하게 됩니다.

5 실제 모듈 측정

```
python cmis_mci.py timing
```

이 명령은 펌웨어를 기록하지 않습니다. 업데이트를 구성하는 액세스 단계, 즉 단일 바이트 읽기, 쓰기 제한과 같은 8바이트 읽기, 모듈이 광고하는 경우 전체 페이지 읽기, 양방향 페이지 변경, Common Data Block 페이지 도달을 측정하고, 각각을 Table 10-4 상한과 나란히 출력합니다.

```
$ 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.
```

이를 타깃이 아니라 호스트의 측정으로 읽으십시오. 모든 행은 USB 왕복당 약 2밀리초입니다. 액세스 한 번은 약 2 ms, CDB 페이지 선택의 레지스터 쓰기 두 번은 약 4 ms, 양방향 페이지 변경의 쓰기 네 번은 약 8 ms입니다. 단일 바이트 읽기는 tREAD의 4배이며, 8바이트 읽기는 1바이트 읽기보다 느리지 않습니다. 이는 측정 대상이 선로상의 바이트가 아닐 때만 가능합니다.

이 실행의 타깃은 I2C 타깃 모드의 어댑터로, 즉시 응답하며 버스를 홀드오프하지 않습니다. 따라서 이 수치가 보여 주는 것은 모듈이 아무것도 기여하기 전에 호스트 경로가 부과하는 하한이며, 이 구성에서 그 하한은 액세스당 약 2 ms입니다. 결론은 섹션 7.2에서 제시합니다.

결과에서 살펴볼 사항은 다음과 같습니다.

중앙값이 사양 최댓값보다 훨씬 낮으면 모듈이 허용된 것보다 빠르다는 뜻이며, 매번 쓰기 후 최댓값만큼 기다리는 호스트는 업데이트 시간의 대부분을 낭비하고 있습니다. 이것이 이 측정이 알려 줄 수 있는 가장 가치 있는 정보입니다.

페이지 변경의 최댓값이 사양 상한에 가까우면 모듈이 실제로 때때로 tBPC를 사용한다는 뜻입니다. 중앙값을 가정하는 호스트는 간헐적으로 실패하며, 이는 디버깅하기 가장 어려운 실패 모드입니다.

단일 바이트 읽기가 tREAD보다 길면 모듈 홀드오프가 아니라 호스트 측 또는 어댑터 측 오버헤드입니다. tREAD는 I2C 클록 스트레칭을 추상화한 것이며, 클록 스트레칭은 버스 캡처에서는 보이지만 호스트 측 타이머로는 다른 요인과 구별할 수 없습니다.

5.1 I3C CMIS 타깃에 대한 동일한 측정

위의 실행은 타깃 모드의 어댑터에 대해 I2CMCI로 수행한 것입니다. 평면 메모리가 아니라 레지스터 모델을 구현하는 AN0009의 CMIS 타깃 펌웨어에 대해 I3C로 반복해도 중요한 것은 아무것도 바뀌지 않습니다.

```
$ 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.
```

31배 빠른 버스와 평면 영역이 아닌 펌웨어에서 응답하는 타깃을 사용해도 모든 행이 그대로입니다. 이는 예산이 예측하는 것과 같은 결과이며, 반대 방향에서 도달한 것입니다.

5.2 하한은 호스트 스택에 속하며, 이 스택은 이를 벗어날 수 없음

Supernova 펌웨어 4.5.0은 어댑터 전송을 벤더 전용 벌크 엔드포인트로 옮겨 전송 상한을 높이고 USB 경로를 재작업했습니다. 이 노트가 제공하는 유틸리티로 측정하면 아무것도 바뀌지 않습니다.

표 6. 두 가지 어댑터 펌웨어 리비전에서의 동일한 네 단계, 세 번 실행한 중앙값

단계	4.4.0	4.5.0
단일 바이트 읽기	2.006 ms	2.036 ms
8바이트 읽기, WRITE 제한	2.003 ms	2.036 ms
페이지 변경 00h에서 01h, 다시 00h	7.993 ms	8.061 ms
CDB 페이지 9Fh 선택	3.987 ms	4.035 ms

보드는 4.4.0에서 4.5.0으로, 다시 4.4.0으로 되돌렸다가 다시 올렸으며, 각 상태에서 측정했습니다. 재플래시를 사이에 둔 단일 비교로는 펌웨어 변경과 그 밖에 바뀐 요인을 구별할 수 없기 때문입니다.

> **위의 두 가지 모두 호환 경로입니다.** 새 트랜스포트는 호스트에서 옵트인합니다. 호스트가 이를 협상하지 않는 어댑터는 이전 경로를 그대로 제공하며, 이 덕분에 이 펌웨어는 호환성을 깨지 않는 릴리스가 됩니다. 이 노트와 시리즈 전체가 실행되는 SDK인 `binhosupernova` 4.2.0은 USB HID로 전송하며 벤더 전용 벌크 엔드포인트 코드를 전혀 포함하지 않습니다. 따라서 이 쌍은 호환 경로가 퇴행하지 않았음을 보여 주지만, 그것만으로는 4.5.0이 추가한 트랜스포트에 대해 아무것도 말해 주지 않습니다.

이를 협상하는 호스트 스택으로 동일한 네 단계를 반복하면 표가 답할 수 없는 질문에 답할 수 있습니다. 단계는 움직이지 않습니다.

표 7. 호환 트랜스포트와 벌크 트랜스포트에서의 동일한 네 단계, 모두 펌웨어 4.5.0

단계	USB HID	벌크 트랜스포트
단일 바이트 읽기	2.036 ms	2.009 ms
8바이트 읽기, WRITE 제한	2.036 ms	2.006 ms
페이지 변경 00h에서 01h, 다시 00h	8.061 ms	8.019 ms
CDB 페이지 9Fh 선택	4.035 ms	3.993 ms

오른쪽 열은 트랜스포트가 스스로 사용 가능하다고 보고하고 32768바이트 상한을 광고하는 상태에서 측정한 것이므로, 이름만 바꾼 두 번째 호환 경로 실행이 아닙니다. 이 트랜스포트는 CosmicSDK 1.5.0에서 도입되었으며, 이전 SDK는 아무런 이상도 보고하지 않으면서 왼쪽 열을 출력합니다. 이것이 바로 이 측정이 가정하지 않고 배제해야 했던 실패입니다. 확인 방법은 릴리스 번호가 아니라 열린 디바이스에 대한 기능 질의입니다. 그렇다고 이것이 트랜스포트에 대한 부정적인 결과는 아닙니다. 같은 세션에서 트랜스포트는 설계된 대로 정확히 동작하기 때문입니다.

표 8. SPI에서 바이트 출력, 동일한 어댑터와 세션, 전송 크기별

전송	요청 경로	벌크 경로
8바이트	2.05 ms	2.00 ms
1024바이트	35.0 ms	5.0 ms
32768바이트	사용 불가	114.8 ms, 285 kB/s

더 넓은 트랜스포트는 페이로드 레버이지 지연 시간 레버가 아닙니다. 1킬로바이트에서는 7배 빠릅니다. 8바이트에서는 같은 속도입니다. 8바이트 전송의 비용은 왕복 한 번이며, 파이프가 넓어진다고 왕복이 짧아지지는 않기 때문입니다.

이것이 섹션 4 전체를 호스트 측에서 다시 말한 것이며, 두 섹션이 일치하는 이유입니다. `5.2.2.2` 는 WRITE를 8바이트로, 액세스를 한 페이지로 제한하므로, **페이지 방식 관리 인터페이스상의 CMIS는 이 중 어느 것이든 도움이 될 만큼 큰 전송을 결코 발행하지 않습니다.** 그런 전송을 발행하는 인터페이스는 2048바이트의 SPIMCI이며, 섹션 4에서 블록당 16개 트랜잭션을 1개로 줄이는 행입니다. 이 벤치에는 SPIMCI 타깃이 없으므로 그 행은 산술로 남아 있습니다.

5.3 2밀리초는 어디에서 오는가

독립된 두 호스트 스택이 세 자리까지 같은 수치로 일치하는 것은 우연이 아니며, 원인은 둘 중 어느 스택이 아니라 어댑터의 USB 디스크립터에 있습니다.

표 9. 어댑터의 엔드포인트, 구성 디스크립터에서 발췌

인터페이스	엔드포인트	전송 유형	최대 패킷	간격
Human Interface Device	0x01 OUT, 0x81 IN	인터럽트	64바이트	1 ms
벤더 전용	0x02 OUT, 0x82 IN	벌크	64바이트	없음

명령 플레인은 인터럽트 쌍이며, 인터럽트 엔드포인트는 일정에 따라 폴링됩니다. 여기서 간격은 1밀리초이므로 요청은 다음 프레임 전에 나갈 수 없고, 그 응답은 그다음 프레임 전에 도착할 수 없습니다. 이것이 2밀리초의 대부분을 설명하며, 어떤 SDK가 하는 작업도 아닙니다. 버스가 전송을 허용하는 시점입니다. 나머지는 어댑터 펌웨어의 처리 시간과 호스트 스택이며, 이 측정은 이 둘을 분리하지 않습니다.

벌크 쌍은 명령이 아니라 페이로드를 전달합니다. 이것이 앞의 결과의 전부입니다. 페이로드 경로를 넓혀도 명령 왕복은 그대로이므로, 명령 하나와 응답 하나뿐인 8바이트 액세스는 빨라질 수 없습니다.

같은 모델로 다른 열도 예측할 수 있으며, 이것이 해 볼 만한 검증입니다. 요청 경로의 1024바이트 전송은 명령 자체에 실려 64바이트씩 이동하므로, 페이로드 16프레임에 명령 왕복을 더한 것입니다. 약 34 ms이며, 측정값은 35.0 ms입니다. 벌크 경로에서는 페이로드가 일정에서 완전히 벗어나고 명령 왕복만 남습니다. 약 2 ms에 전송 시간을 더한 것이며, 측정값은 5.0 ms입니다.

따라서 이 하한에는 이름이 있으며, 그것은 호스트가 무엇을 해야 하는지 알려 줍니다. 이를 움직이는 것은 명령의 수와 동시에 대기 중인 명령의 수입니다. 둘 다 명령이 지나가는 파이프의 크기가 아닙니다.

5.4 하한의 절반은 응답이며, 호스트는 이를 숨길 수 있음

명령과 그 응답이 각각 한 프레임의 비용이라면, 다음 명령을 보내기 전에 모든 응답을 기다리는 호스트는 둘 다 지불하고, 명령을 진행 중(in flight)으로 유지하는 호스트는 첫 번째만 지불해야 합니다. 이는 검증할 수 있습니다. 동일한 1바이트 읽기를 논블로킹 인터페이스를 통해 200번 발행하되, 한 번은 각 응답을 기다리고 한 번은 기다리지 않았습니다.

표 10. 동일한 200번의 읽기, 각 응답을 기다리는 경우와 진행 중으로 유지하는 경우 비교

	합계	명령당
한 번에 하나씩	400.0 ms	2.000 ms
모두 진행 중	200.9 ms	1.005 ms

파이프라이닝은 이를 절반으로 줄이며, 제거되는 절반은 응답입니다. 남는 것은 프레임당 명령 하나이며, 이는 송신 방향 폴링 간격이고 호스트가 명령을 덜 보내지 않고는 그 아래로 내려갈 수 없는 값입니다. 200개 모두 응답을 받았으므로, 이는 명령이 누락된 것을 측정한 것이 아닙니다.

이를 원하는 호스트에게는 두 가지 결과가 있습니다. 절감은 실제적이지만 한계가 있습니다. Extended Payload 블록의 호스트 시간 비용을 약 32 ms에서 약 16 ms로 줄이지만 tWRITE는 10 ms이므로, 호스트는 블록을 지배하지 않게 되지만 무시할 수 있을 정도가 되지는 않습니다. 또한 이는 액세스가 서로 의존하지 않는 경우에만 가능하며, 하나의 펌웨어 블록 안에서는 의존하지 않습니다. 같은 Extended Payload 영역에 대한 연속 쓰기이기 때문입니다.

분명히 말해 둘 것은, **이 노트가 제공하는 유틸리티는 이렇게 하지 않는다는 것입니다.** 유틸리티는 블로킹 인터페이스를 사용하여 한 번에 명령 하나씩 처리하며, 이 때문에 섹션 5의 모든 수치가 2밀리초 열에 해당합니다. 위의 측정은 논블로킹 인터페이스에 대한 별도의 하네스이며, 도구를 설명하기 위한 것이 아니라 이 옵션의 가치를 가늠하기 위한 것입니다.

이는 섹션 4와 나란히 놓고 볼 만합니다. Extended Payload 블록은 16번의 액세스로, 10 ms의 tWRITE에 비해 약 32 ms의 호스트 시간이 들므로, 이 벤치에서는 모듈이 아니라 호스트가 블록을 지배합니다. 파이프라이닝은 이를 약 16 ms로 줄이고, **SPIMCI는 이를 16개가 아닌 1개의 트랜잭션으로 만듭니다.** 이는 이 노트에서 섹션 5가 측정하는 항에 도달하는 가장 큰 변화입니다. 자신의 왕복이 무시할 만하다는 가정 위에 구축된 호

스트는 여기서 틀렸으며, 페이지 방식 관리 인터페이스에서 할 수 있는 최선은 그것을 절반으로 줄이는 것입니다.

이것이 섹션 4와 5가 반대 방향에서 같은 곳에 도달하는 이유입니다.

6 이 결과의 활용

호스트 소프트웨어의 경우. 다운로드 시작 시 `Firmware Management Features` (CDB 명령 `0041h`)를 한 번 질의하고, 모듈이 지원하는 경우 `0103h` 보다 `Write Firmware Block EPL` (`0104h`)을 우선 사용하십시오. 어느 쪽으로도 가정하지 마십시오. 그런 다음 블록 내 액세스는 각 응답을 기다리지 않고 발행하십시오. 섹션 5.4는 이 방식에서 호스트 비용이 절반이 됨을 측정했으며, 이는 한 블록 내의 액세스가 서로 의존하지 않기 때문에 가능하고, 더 빠른 인터페이스가 아니라 논블로킹 인터페이스가 필요합니다.

모듈 펌웨어의 경우. 지킬 수 있는 홀드오프 시간을 광고하십시오. tWRITE는 목표가 아니라 최댓값이며, 섹션 5의 측정은 호스트가 여러분이 남겨 둔 여유를 발견하는 방법입니다.

새로운 MCI 바인딩을 규정하는 모든 분께. SPIMCI는 이미 한 트랜잭션에서 최대 2048바이트를 전달하며 섹션 4.1에서 이를 산정했고, I3C는 SETMWL, SETMRL, GETMXDS를 통해 같은 것을 기본적으로 협상합니다. I2CMCI의 8바이트 쓰기 제한을 물려받는 바인딩은 이 문제 전체를 함께 물려받으며, 어떤 클록으로 동작하든 물려받습니다.

> **Run Image 단계는 별도로 예산을 잡으십시오.** CDB 명령 `0109h` 는 모듈 전체 재부팅만큼 시간이 걸리며, "mainly governed by the tMgmtInit timing parameter"(주로 tMgmtInit 타이밍 파라미터에 의해 결정됨)이므로 CDB 타임아웃이 아니라 약 2초를 잡아 두십시오.

7 측정 결과

7.1 테스트 구성

표 11. 테스트 구성, 섹션 5의 I2CMCI 실행

항목	값
컨트롤러	Binho Supernova, 시리얼 `1912DEB5…79FA` , 펌웨어 4.4.0, 하드웨어 리비전 C
타깃	Binho Supernova, 시리얼 `ED7BF8F3…F7BB` , 펌웨어 4.4.0, 주소 0x50의 I2C 타깃으로 설정
타깃 내용	CMIS 5.3 이미지, Appendix C의 400G QSFP-DD 예제
SDK	`binhosupernova` 4.2.0, Python 3.10.12
호스트	Ubuntu 22.04, x86-64
버스	3.3 V, 400 kHz
반복	단계당 20회, 중앙값과 최댓값 보고
날짜	2026년 9월 1일

표 12. 테스트 구성, 섹션 5.1 및 5.2의 I3C 실행

항목	값
컨트롤러	Binho Supernova, 시리얼 `4F970418…1111`, 펌웨어 4.4.0 및 4.5.0
호스트 트랜스포트	USB HID를 통한 `binhosupernova` 4.2.0, 그리고 벤더 전용 벌크 엔드포인트를 통한 CosmicSDK 1.5.0(기능 질의로 사용 가능함을 확인, 32768바이트 상한 광고)
타깃	AN0009 CMIS 타깃 펌웨어를 실행하는 NXP FRDM-MCXN947, 동적 주소 0x09
타깃 내용	CMIS 5.3 이미지, Appendix C의 400G QSFP-DD 예제
SDK	`binhosupernova` 4.2.0, Python 3.14
호스트	Arch Linux, x86-64
버스	3.3 V, I3C SDR, 푸시풀 2.5 Mbit/s 25% 듀티 사이클, 오픈 드레인 1 Mbit/s
반복	단계당 20회, 중앙값과 최댓값 보고, 펌웨어당 3회 실행
섹션 5.4	논블로킹 인터페이스에 대한 별도의 하네스, 조건당 200회 읽기, 3회 실행
날짜	2026년 9월 3일

두 구성은 벤치, 호스트, Python 버전이 모두 다릅니다. 이 노트의 어디에서도 서로 비교하지 않으며, 섹션 5.2의 펌웨어 비교에는 두 번째 구성만 사용합니다. 거기서는 호스트와 타깃을 고정하고 어댑터 펌웨어만 바꿉니다.

7.2 결과

도출 결과. 섹션 4의 예산은 제공된 유틸리티로 재현할 수 있으며 섹션 3의 사양 인용에만 근거합니다. 이는 산술이며, 이 문서는 이를 다른 무엇으로도 제시하지 않습니다.

측정 결과, 그리고 이는 호스트를 측정합니다. 섹션 5의 실행은 이 호스트, 어댑터, SDK에서 자체 홀드오프를 부과하지 않는 타깃을 대상으로 레지스터 액세스당 약 2 ms의 하한을 확립합니다. 이 수치는 버스나 모듈이 아니라 USB 경로에 속합니다. 단일 바이트 읽기가 tREAD의 4배를 넘고 8바이트 읽기가 1바이트 읽기보다 느리지 않다는 것은 둘 다 선로 시간이 지배적인 항이 아닐 때만 가능합니다.

이것이 **이 문서의 어디에도 종단 간 처리량 수치가 나타나지 않는** 이유입니다. 이런 방식으로 측정한 수치는 호스트를 특성화하며, 이를 버스나 모듈의 속성으로 인용하면 나중에 발견하기 어려운 방식으로 틀리게 됩니다.

그러나 이는 노트 자체의 논지를 약화시키기보다 오히려 더 명확하게 합니다. 섹션 4는 펌웨어 업데이트의 시간이 선로 시간에 소비되지 않는다고 주장합니다. 측정은 호스트 왕복 비용에도 소비되지 않는다는 점을 더합니다. 둘 다 블록별 홀드오프에 비해 작으며, 블록이 커지면 같은 이유로 둘 다 줄어듭니다. **액세스가 줄고 블록이 줄기 때문입니다.** 액세스당 2 ms의 하한은 Local Payload 블록당 30 ms, Extended Payload 블록당 32 ms의 비용이 들지만, Extended Payload 블록은 17배 더 많은 펌웨어를 전달합니다.

더 새로운 어댑터 펌웨어와 더 새로운 트랜스포트에서 다시 측정했으며, 움직이지 않았습니다. Supernova 펌웨어 4.5.0은 네 단계 모두를 4.4.0과 같은 위치에 두며, 그 펌웨어가 추가한 벌크 트랜스포트도 트랜스포트가 스스로 사용 가능하다고 보고하는 상태에서 측정했을 때 마찬가지였습니다. 같은 세션에서 그 트랜스포트는 1

킬로바이트 전송에서 7배 빠르므로, 이는 설정 실패가 아니라 한계가 있는 결과입니다. 더 넓은 트랜스포트는 페이로드 레버이지 지연 시간 레버가 아니며, 8바이트 액세스는 어느 쪽이든 왕복 한 번의 비용이 듭니다.

이는 섹션 4를 반대쪽 끝에서 다시 말한 것입니다. `5.2.2.2` 는 WRITE를 8바이트로, 액세스를 한 페이지로 제한하므로, 페이지 방식 관리 인터페이스상의 CMIS는 더 넓은 파이프가 도움이 될 만큼 큰 전송을 결코 발행하지 않습니다. 이 표에 도달하는 레버는 왕복 횟수(블록당 트랜잭션 수 열이 세는 것이며 SPIMCI가 바꾸는 것)와 그중 동시에 미처리 상태인 것의 수입니다.

그중 두 번째는 절반의 가치가 있으며, 지금 바로 사용할 수 있습니다. 섹션 5.4는 각 응답을 기다리지 않고 같은 액세스를 200번 발행하여 2.000 ms 대비 1.005 ms를 측정했으며, 모든 명령이 응답을 받았습니다. 남는 것은 USB 프레임당 명령 하나입니다. 따라서 펌웨어 블록을 기록하는 호스트는 액세스가 독립적인 블록 내부에서 기다리지 않음으로써 자신의 기여를 절반으로 줄일 수 있으며, 그 이후에는 페이지 방식 인터페이스에서 더 이상 쓸 레버가 없습니다.

이것이 또한 섹션 4의 블록당 트랜잭션 수 열이 봐야 할 열인 이유입니다. SPIMCI는 페이지 방식 인터페이스가 16개를 필요로 하는 Extended Payload 블록을 단일 트랜잭션으로 전달하며, 트랜잭션당 약 2밀리초의 호스트 비용으로 보면 버스가 아무것도 하기 전에 블록당 2 ms 대 32 ms가 됩니다.

하드웨어에서, 모듈을 대상으로. 여기에는 아무것도 없습니다. 리비전 1.0 기준 미완료 사항은 다음과 같습니다. 실제 모듈에 대한 `timing` 명령 실행(모듈이 실제로 사용하는 홀드오프를 확립하는 유일한 방법), Local Payload와 Extended Payload 두 경로 모두를 통한 완전한 펌웨어 다운로드(종단 간 수치를 뒷받침할 수 있는 유일한 방법), 그리고 SPIMCI에서의 어떠한 측정(이 벤치에는 SPIMCI 타깃이 없음)입니다. 섹션 4의 모든 SPIMCI 수치는 Appendix B.3에서 도출한 산술이며 그렇게 표시되어 있습니다.

이것이 실행되기 전까지, 이 노트가 뒷받침하는 주장은 **시간이 어디에 소비되도록 허용되는지**, 그리고 모듈이 아무것도 기여하기 전에 호스트가 얼마의 비용을 차지하는지에 관한 것입니다.

8 문제 해결

표 13. 문제 해결

증상	예상 원인 및 조치
`timing`이 tREAD보다 훨씬 긴 단일 바이트 읽기를 보고함	모듈이 아니라 호스트 또는 어댑터 오버헤드입니다. 버스 캡처와 비교하십시오
모든 단계가 상한에 가까움	모듈이 실제로 느리거나, 다른 컨트롤러가 버스를 경합하고 있을 수 있습니다
출력에 94바이트 읽기가 없음	모듈이 전체 페이지 읽기를 광고하지 않으므로 `Nmax`가 8로 유지되었습니다
실행마다 중앙값이 크게 다름	다른 무언가가 USB 경로를 사용하고 있습니다. 다른 호스트 도구를 닫고 다시 실행하십시오
더 새로운 어댑터 펌웨어에서도 수치가 같아 보임	예상된 결과이며, 섹션 5.2와 5.3에서 설명합니다. 또한 어떤 트랜스포트를 사용 중인지 확인하십시오. SDK가 벌크 트랜스포트를 협상하지 않으면 어댑터는 오류 없이 HID 경로를 제공합니다. 버전 번호로 추론하지 말고 SDK에 질의하십시오
모든 액세스에 약 2 ms가 걸리고 무엇을 바꿔도 개선되지 않음	그 대부분은 USB 일정입니다. 명령 플레인은 1 ms로 폴링되는 인터럽트 엔드포인트이므로 명령과 그 응답이 두 프레임에 걸칩니다. 디스크립터는 섹션 5.3에 있습니다. 명령의 크기가 아니라 수를 줄이십시오
페이지 변경이 tBPC의 두 배를 초과함	모듈이 사양이 허용하는 것보다 오래 걸리고 있습니다. 호스트를 탓하기 전에 버스를 캡처하십시오

9 약어

표 14. 약어

용어	의미
CDB	Common Data Block(공통 데이터 블록)
CMIS	Common Management Interface Specification(공통 관리 인터페이스 사양)
EPL	Extended Payload, CDB 페이지 A0h부터 AFh
LPL	Local Payload, CDB 페이지 9Fh 내
MCI	Management Communication Interface(관리 통신 인터페이스)
Nmax	한 번의 레지스터 액세스에서의 최대 바이트 수
OIF	Optical Internetworking Forum
SDR	Single data rate(단일 데이터 속도)

10 문서의 소스 코드에 관한 참고 사항

이 문서에 표시되고 첨부 아카이브에 제공되는 예제 코드는 평가 및 참고용으로 제공됩니다. MIT 라이선스로 배포되며, 라이선스 전문은 아카이브에 포함되어 있습니다.

표 15. 에셋 아카이브의 내용

파일	설명
`AN0011.md`	이 문서의 Markdown 소스
`assets/cmis_mci.py`	참조 유틸리티, AN0008부터 AN0011까지 공용
`assets/LICENSE`	예제 코드에 적용되는 MIT 라이선스

11 개정 이력

표 16. 개정 이력

리비전	날짜	설명
0.1	2026년 9월 1일	내부 검토용 초안. 예산은 사양에서 도출했으며 측정은 수행하지 않았습니다.
0.2	2026년 9월 1일	Supernova I2C 타깃 픽스처에 대해 `timing`을 실행했습니다. 출력을 붙여 넣고, 모듈이 아니라 호스트 경로의 측정으로 보고했습니다.
0.3	2026년 9월 2일	AN0010에서 번호를 변경했습니다. 측정이나 예산에는 변경이 없습니다.
0.4	2026년 9월 3일	예산을 SPIMCI로 확장하여 초기 1 MHz와 `MciSpeedConfiguration` 최대인 50 MHz를 포함하고, 블록당 트랜잭션 수 열을 호스트가 체감하는 열로 강조했습니다. 섹션 5에 AN0009 CMIS 타깃에 대한 I3C 기반의 동일한 측정과, 두 가지 호스트 트랜스포트에 걸친 Supernova 펌웨어 4.4.0 대 4.5.0 비교(둘 다 하한을 움직이지 않음)를 추가했으며, 같은 세션의 SPI 전송 크기 스윕으로 더 넓은 트랜스포트가 1킬로바이트에서 7배 빠르고 8바이트에서는 같음을 보였습니다. 트랜스포트가 CosmicSDK 1.5.0에 도입된 후 두 트랜스포트 실행을 이에 대해 반복했으며 변화가 없었습니다. 새 섹션 5.3에서 2밀리초 하한을 명령 플레인이 사용하는 1 ms 인터럽트 엔드포인트까지 추적하고, 전송 크기 스윕으로 그 모델을 검증했습니다. 유틸리티는 1.1입니다.
0.5	2026년 9월 3일	새 섹션 5.4에서 명령을 진행 중으로 유지하는 것의 가치를 산정하여, 모든 명령이 응답을 받은 상태에서 명령당 비용이 절반임을 측정했으며, 섹션 6은 이제 호스트에게 블록 내에서 이를 수행하도록 권고합니다. 남는 하한은 프레임당 명령 하나입니다.
1.0	2026년 11월 6일	공개.

법적 고지(영문)

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** 및 **OIF**는 Optical Internetworking Forum의 상표 또는 등록 상표입니다. **I3C** 및 **MIPI**는 MIPI Alliance, Inc.의 상표 또는 등록 상표입니다.