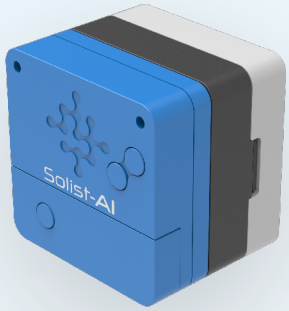


FIRMWARE EXTERNAL INTERFACE SPECIFICATION

Solist-AI モジュール ファームウェア 外部インターフェース 仕様書

ML63Q2537 (ROHM Solist-AI) を搭載した AI アクセラレータモジュール。本書は、モジュール出荷時に flash 済みのファームウェアが提供する外部インタフェースを規定します。

VERSION
Rev. 1.1

INTERFACE
SPI Slave

文書情報 / Document Information

対象製品 / Product	ML63Q2537（ROHM Solist-AI）— AI Accelerator
版 / Version	Rev. 1.1

本書の位置づけ / SCOPE
本書は、モジュール出荷時に flash 済みのファームウェアが SPI 経由で外部ホスト機器へ提供するインタフェースを規定します。**本書に記載のない挙動は変更される可能性があります。**

改訂履歴 / Revision History

Rev.	Date	変更内容 / Changes
1.1	2026-07	§4.5 CRC/形式検証エラー時の NAK 挙動記述を拡張。形式検証（header / 長さ / footer）または CRC 照合のいずれかで弾かれた場合に NAK（`0x02`）を返すこと、および当該経路の NAK の `data` 欄には §5.2 のエラーコードではなく内部 seq_counter が格納されることを明記。
1.0	2026-06	初版発行。SPI スレーブインタフェース全体（物理層・パケット形式・全コマンド）を規定。

目次 / Contents

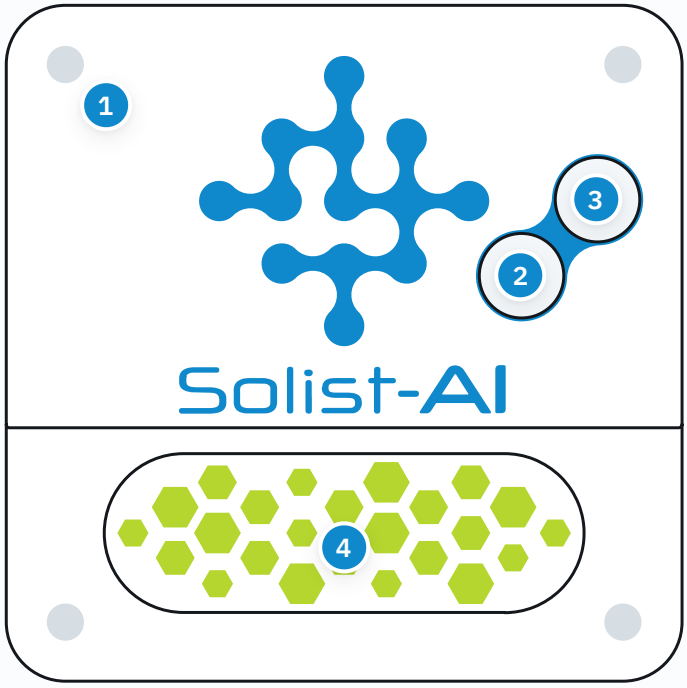
1

CHAPTER 01

製品概要

本モジュールの位置づけ、想定する利用シーン、AI機能の 2 種類の構成バリエーションを示します。まず本章で全体像を把握してください。

1 製品概要



1. **モジュール本体** – 上面から見た本体
2. **Reset ボタン** – 上面奥の 2 ボタンのうち右
3. **IAP Mode ボタン** – 上面奥の 2 ボタンのうち左
4. **gene ハニカムコネクタ** – 上面中央の白い基板とハニカム形状のパッド

図 1 Solist-AI モジュール本体 上面図。本体上面の各部名称を示す。

1.1 本モジュールとは

本モジュールは、別の MCU（以降「**ホスト**」と呼ぶ）に従属する **アクセラレータ MCU** です。ホストから SPI 経由で要求を受け取り、AI 推論・AI 学習・FFT・フラッシュメモリ操作といった機能をサービスとして提供します。

本モジュールは独立して動作するものではなく、ホストが SPI バスを駆動して初めて応答します。本モジュール自身が計時した時間に基づいて動作することはなく、ホストへ自発的に通知を送ることもありません。

> **NOTE / 注記**
> 本モジュールからホストへの非同期通知機能は、将来バージョンでの提供を検討中です。本リリースには含まれません（付録 B 参照）。

1.2 利用シーンの例

- ホスト MCU が推論・学習・FFT を本モジュールへオフロードする
- ホストが収集したセンサーデータを本モジュール上の学習モデルに入力し、結果を取り出す

1.3 構成バリエーション

本モジュールは AI 機能の制限が異なる 2 種類の構成があります。AI 関連の上限値はこの構成に依存します。各構成の具体的な上限値（AI モデル数・入力次元・隠れ層ユニット数など）については、**Solist-AI データシート**を参照してください。

> **CAUTION / 注意**
> ホストの実装者は導入時にどちらの構成かを把握してください。構成をランタイムで問い合わせるコマンドは本リリースでは提供していません。

2

CHAPTER 02

通信モデルと用語

本モジュールとホストがどのように対話するかという概念モデルを導入します。物理層（第 3 章）やパケットのバイト形式（第 4 章）よりも前に、まずこのモデルを理解してください。

2 通信モデルと用語

2.1 ホストとモジュールの関係

本モジュールとホストは SPI バスで接続されます。通信は常に **ホスト起動** です。ホストが何も送らなければ、本モジュールは応答を返さず、データを生成することもありません。

● **ホスト**	SPI Master。クロックとチップセレクト（CS）を駆動し、通信を起動する側。
● **モジュール**	SPI Slave。ホストからの要求にのみ応答する側（＝本モジュール）。

2.2 コマンドとは何か

ホストが本モジュールを動かす唯一の手段は **コマンド** の発行です。コマンドは「本モジュールに特定の動作を依頼する 1 個の要求」を指します。各コマンドは以下の構成要素を持ちます。

構成要素	役割
コマンドコード（8 bit）	どの機能を呼び出すかを識別する。例: `0x20` = AI 初期化
パラメータ（32 bit）	数値パラメータ。書込アドレス、AI モデル番号、サイズなど
データペイロード（任意長）	重み、学習サンプル、書込内容など、まとまったバイト列を伴う場合に付随（省略可）

本モジュールはコマンドを受け取ると、**ACK**（成功確認。必要に応じ 32 bit 戻り値を含む）、**NAK**（失敗。エラーコードを含む、§ 5.2）、または大きな返値を返す **データパケット** のいずれかで応答します。コマンドによっては中間 ACK を返す 2 段階応答を取るものがあり、この組み立て方を **実行パターン** と呼びます（§ 5.3）。

2.3 パケットとは何か

「コマンド」「応答」「データ」はすべて、SPI 上では **パケット** という単位で送受信されます。本仕様は 2 種類のパケットを定めます。

パケット種別	用途	長さ
CONTROL パケット	コマンド本体、ACK / NAK 応答、PING など制御目的	9 バイト固定長
DATA パケット	バイト列のペイロード送受信	可変、最大 1033 バイト

両パケットには CRC が含まれており、伝送エラーを検出できます。バイト単位の形式は §4 で規定します。

2.4 機能カテゴリとコマンドコード

本モジュールが提供するアプリケーションコマンド（CONTROL パケットの `layer_id = 0x01`）は機能ごとに分類され、`code`（下位 8 bit）はカテゴリごとに範囲が割り当てられています。各コマンドの個別仕様は §6 に記述します。`layer_id = 0x00`（Transport 層）のコードは §4.2 で規定します。

機能カテゴリ	コマンドコード範囲	提供する機能
Buffer	0x0A–0x0C	1 パケットに収まらないペイロードのステージング領域操作
Flash	0x10–0x1E	内蔵フラッシュメモリの読出 / 消去 / 書込
AI/ML	0x20–0x2D	オンデバイス学習および推論
FFT	0x30–0x33	高速フーリエ変換
System	0xD0–0xD3	バージョン / 状態 / 能力 / 経過時間の取得

2.5 用語集

用語	定義
ホスト	本モジュールを SPI 経由で制御する外部 MCU。SPI Master
本モジュール	本仕様書が規定する SPI スレーブ側のモジュール（ML63Q2537 を搭載）
コマンド	ホストが本モジュールに発行する 1 件の要求（§2.2）
コマンドコード	Application 層コマンドを識別する 1 バイトの値（`code`。CONTROL パケット `command[7:0]`、`layer_id = 0x01` の下で意味を持つ）
`layer_id`	CONTROL パケット `command[15:8]`。Transport 層（`0x00`）と Application 層（`0x01`）を区別する識別子（§4.2）

用語	定義
パラメータ	CONTROL パケットの `data` フィールドに載せる 32 bit 値
ペイロード	DATA パケットの本体バイト列
パケット	SPI 上のやりとりの基本単位。CONTROL または DATA（§ 2.3, § 4）
ACK / NAK	コマンドへの成功 / 失敗応答
実行パターン	1 コマンドあたりの通信往復構造の分類。A〜F の 6 種類（§ 5.3）
エラーコード	NAK 応答が伝える失敗理由を示す 1 バイトの値（§ 5.2）
インスタンス	AI 機能内で独立に保持される 1 個の学習モデル。0 または 1 で識別（§ 6.3）
セッション	連続する Flash 操作をまとめる文脈。`SESSION_BEGIN/END` で境界付ける（§ 6.2）
転送バッファ	1024 バイト超を段階転送するための内部 2048 バイト記憶領域（§ 5.6）
bfloat16	AI 機能で使用する 16 bit 浮動小数点形式（1 sign + 8 exp + 7 mantissa）（§ 6.3）
アクセスコード	Flash 書込・消去時に要求される秘密値。本モジュールがセッション開始時に内部保持するため、ホストが意識する必要はない

3

CHAPTER 03

物理層 / 配線

SPI の設定、信号線の割り当て、フロー制御に用いる READY 信号の利用手順、およびホスト基板 gene とをつなぐコネクタのピンマップを規定します。本章はホスト側の SPI 周辺回路とドライバ設計の前提となります。

3 物理層 / 配線

3.1 SPI 設定

役割	Slave
データ幅	8 bit
ビット順	MSB first
CPOL / CPHA	CPOL=0, CPHA=0（アイドル時 Low、第 1 エッジでサンプル、第 2 エッジでシフト）
最大クロック周波数	2.8 MHz
全二重	対応（ただし本プロトコルでは半二重的に使用）

> ⚠ **CAUTION / 注意**
> 最大クロック周波数 **2.8 MHz** を超えると受信側の取り込みが間に合わず、データ取りこぼしが発生します。

3.2 信号線

信号	方向	用途
SCLK	ホスト → 本モジュール	クロック
MOSI	ホスト → 本モジュール	データ送信
MISO	本モジュール → ホスト	データ送信
CS (SS)	ホスト → 本モジュール	チップセレクト（アクティブ Low）

信号	方向	用途
READY	**本モジュール → ホスト**	受信準備状態（HIGH = ready, LOW = busy）。ホスト側はプルアップ付き入力で受ける

3.3 READY 信号の利用手順

READY 信号は、本モジュールが新たなコマンドを受け付けられる状態にあるかをホストへ知らせるフロー制御線です。

1. **送出前に READY = HIGH を確認**
   ホストはコマンド送出前に READY = HIGH を確認します。
2. **本モジュールが READY = LOW を出力**
   本モジュールは受信処理を開始すると READY = LOW を出力します。
3. **準備完了で READY = HIGH に復帰**
   応答準備が整い次第、本モジュールは READY = HIGH に戻します。
4. **HIGH 観測後に CS をアサート**
   ホストは READY = HIGH を観測してから CS をアサートし、応答を読み出します。

> **TIP / 推奨**
> READY を観測しない実装でも動作する場合がありますが推奨されません（内部 FIFO の取りこぼしの原因となります）。

3.4 モジュールのコネクタ構成

本モジュールは、ホスト基板 **gene**（P板.com のセンサーモジュール化サービス基板、MCU は ESP32-S3）に装着して使用します。基板裏面・側面に次の 2 つのコネクタを備えます。

16 ピン gene コネクタ	主インタフェース。gene のセンサーモジュール用コネクタに勘合し、SPI 4 線・READY・電源を接続する（裏面・2×08）。
4 ピン UART コネクタ	スレーブ MCU 単体のデバッグ用 UART。ホスト側とは独立した別チャネル（側面）。

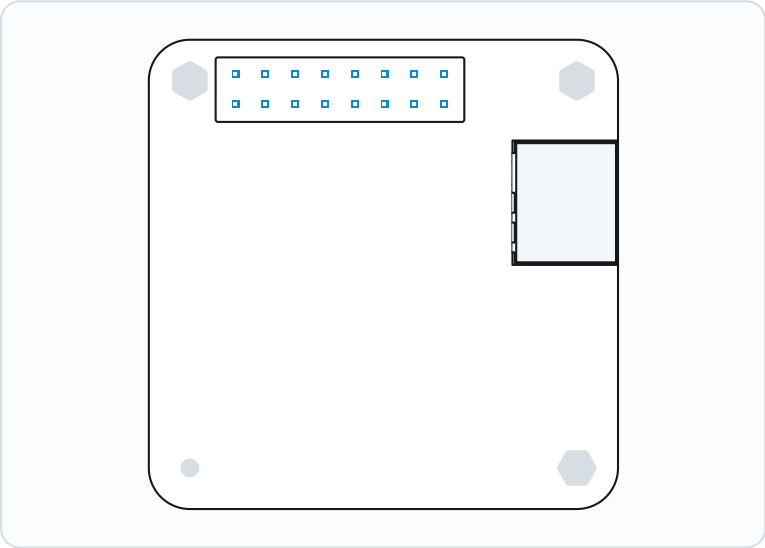


図 3-1 モジュール裏面 ― 上辺に 16 ピン gene コネクタ（2×08）、右辺に 4 ピン UART コネクタ。四隅は取付穴。

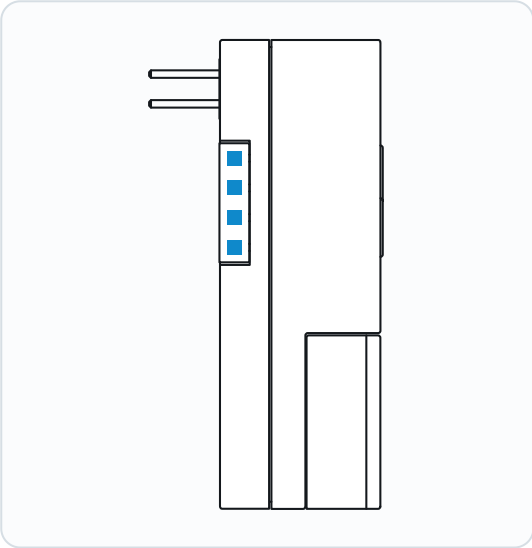


図 3-2 モジュール側面 ― 4 ピン UART コネクタ。上から VDD / RX / TX / GND。

コネクタ接点　取付穴・コネクタ外形

3.5 gene コネクタ（16 ピン）

gene のセンサーモジュール用コネクタは ESP32-S3 の 12 本（`IO1`–`IO3` / `IO7`–`IO9` / `IO21` / `IO35`–`IO38` / `IO46`）と電源（`3V3` / `GND`）を引き出し、SPI・I2C・GPIO・ADC として利用できます。本モジュールはこのうち **SPI 4 線と READY、電源**を使用します。

表 3-1 gene ⇔ Solist-AI 結線（SPI / フロー制御 / 電源）

信号	gene 側 (ESP32-S3)	Solist-AI 側 (ML63Q2537)	方向
SCLK	GPIO36 (IO36)	P40 / SSIOF0_SCK	gene → Solist-AI
MOSI	GPIO37 (IO37)	P42 / SSIOF0_SI	gene → Solist-AI
MISO	GPIO35 (IO35)	P41 / SSIOF0_SO	Solist-AI → gene
CS	GPIO21 (IO21)	P43 / SSIOF0_SS	gene → Solist-AI
READY	GPIO38 (IO38)	P82	Solist-AI → gene
3V3	3V3	VDD	gene → Solist-AI（電源 3.3 V）
GND	GND	GND	—

3.6 UART コネクタ（4 ピン）

側面の 4 ピンコネクタは、スレーブ MCU（ML63Q2537）の UART シリアルポートを引き出したものです。コンソール出力・ログ取得や、外部機器とのシリアル通信などに利用できます。SPI のホスト側とは独立した別チャネルで、gene の GPIO とは結線されていません。

表 3-2 UART コネクタ ピン配列（側面・上から）

ピン	信号	接続先 (ML63Q2537)	方向
1	VDD	–（電源 3.3 V）	–
2	RX	P32 / UARTF0_RX	入力（モジュールへ）
3	TX	P33 / UARTF0_TX	出力（モジュールから、起動時アイドル HIGH）
4	GND	–	–

通信設定: 115200 bps @ 48 MHz。本チャネルは任意であり、SPI 通信には関与せず、未接続でも動作します。

4

CHAPTER 04

パケット形式

第 2 章で導入した 2 種類のパケット（CONTROL / DATA）の、バイト単位の形式を規定します。エンディアン・CRC・コマンドコードの並びまで、ホスト側のパーサ実装に必要な情報をすべて含みます。

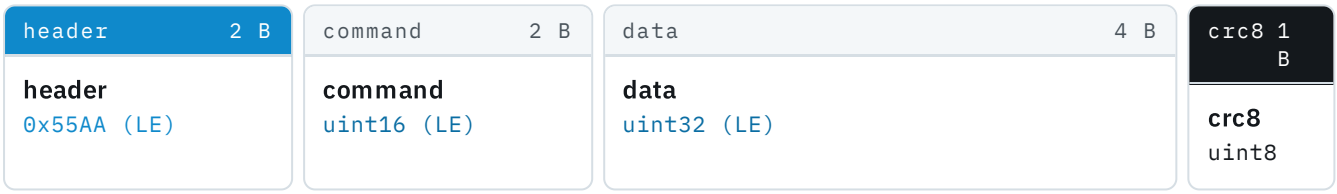
4 パケット形式

4.1 概要（2 種類のパケット）

両者の最初の 2 バイト（ヘッダ）は値が異なります。受信側はこの先頭ヘッダの値で種別を判別します。

パケット種別	ヘッダ値	用途
CONTROL	0x55AA	コマンド本体、ACK/NAK 応答、PING など短い制御用途
DATA	0xAA55	バイト列ペイロードの送受信

4.2 CONTROL パケット



合計: **9 バイト**（固定長）

field	内容
header	固定 `0x55AA`（CONTROL パケット識別子）
command	16 bit 識別子。**上位 8 bit =** `layer_id` **（プロトコル層）**、**下位 8 bit =** `code` **（層内のコード）**
data	コマンドごとに定義される 32 bit 値（パラメータ、または戻り値）
crc8	多項式 `0x07`、初期値 `0x00`。対象は header + command + data の 8 バイト

– command フィールドの層分離

`command` は 2 つのプロトコル層を `layer_id`（上位 8 bit）で区別します。同じ `code` 値でも `layer_id` により一意に識別されます。

layer_id	層	用途
0x00	Transport	プロトコル層の制御（ACK/NAK・PING/PONG・大規模転送境界通知など）
0x01	Application	アプリケーションコマンド（Buffer/Flash/AI/FFT/System）。コード体系は §2.4・付録 A を参照
0x02–0xFF	reserved	予約。受信した本モジュールは NAK（INVALID_COMMAND）を返す

– Transport 層コード（ `layer_id = 0x00` ）

code	種類	意味
0x01	応答	ACK（成功）
0x02	応答	NAK（失敗。`data[7:0]` にエラーコードを格納）
0x05	ホスト → 本モジュール	PING（接続確認）
0x06	本モジュール → ホスト	PONG（PING の `data` をそのまま返す）
0x10	ホスト → 本モジュール	START_TRANSFER（大規模転送開始通知。本モジュールは ACK を返す）
0x11	ホスト → 本モジュール	END_TRANSFER（大規模転送終了通知）
0x12	ホスト → 本モジュール	ABORT（現在の転送を中断）

Application 層コード（`layer_id = 0x01`）は §6 および付録 A で規定します。`code` の値域は `0x0A–0xD3`（未割当領域は付録 B）。

– ワイヤ上のバイト列（little-endian）

同じ `code = 0x10` でも `layer_id` によってワイヤ上の `command` バイト列が異なります。

Transport `START_TRANSFER` (0x00, 0x10)	`command = 10 00`
Application `FLASH_INIT` (0x01, 0x10)	`command = 10 01`

– 受信時の `layer_id` 判定

条件	応答
`layer_id = 0x00` かつ `code` が Transport 層コード表に含まれる	Transport 層が定義する応答（例: PING → PONG）
`0x00` かつ含まれない（例: `command = 0x0099`）	NAK（INVALID_COMMAND）
`layer_id = 0x01` かつ Application 層で定義済み	該当コマンドを実行（§ 6）
`0x01` かつ未定義（付録 B の予約範囲を含む）	NAK（INVALID_COMMAND）
`layer_id = 0x02` 以上（reserved）	NAK（INVALID_COMMAND）

> ⚠ **CAUTION / 破壊的変更**
> 本仕様バージョン以前は `command` 全体を単一の 8 bit コードとして扱い、`layer_id` は暗黙的に `0x00` でした。Application コマンドを従来どおり `command = 0x00NN` で送ると **INVALID_COMMAND NAK** が返ります。ホスト実装は Application コマンド送信時に上位バイトへ `layer_id = 0x01` を設定してください。

4.3 DATA パケット

合計: **9 + length バイト**（length 最大 1024 → 最大 1033 バイト）

field	内容
header	固定 `0xAA55`（DATA パケット識別子）
seq_id	シーケンス番号（0–65535）。本モジュールから返るときは直前に受信した DATA の seq_id を反映する
length	payload バイト数。**0 から 1024 までの範囲**
payload	コマンドごとに定義されるバイト列
crc16	CCITT 多項式 `0x1021`、初期値 `0xFFFF`。対象は header + seq_id + length + payload
footer	固定 `0x0D`

4.4 最大ペイロードサイズ

1 個の DATA パケットで運べる payload は `1024` バイトまでです。これを超える単一の入出力は **転送バッファ**（§ 5.6）を経由して分割します。

4.5 CRC 計算

CRC	多項式	初期値	対象範囲
CRC-8（CONTROL 用）	0x07	0x00	header(2) + command(2) + data(4) = 8 バイト
CRC-16 CCITT（DATA 用）	0x1021	0xFFFF	header(2) + seq_id(2) + length(2) + payload(N)

> ⚠ **CAUTION / 注意**
> 形式検証（header / 長さ / footer）または CRC 照合のいずれかで弾かれた場合、本モジュールは **NAK（`0x02`）を返します**。この経路の NAK の `data` 欄には §5.2 のエラーコードは入らず、本モジュール内部の seq_counter が格納されます。ホストは `command = NAK` の検出のみでパケットレベルエラーを識別し、同じパケットを再送してください。

5

CHAPTER 05

アプリケーション層プロトコル

コマンド個別の話に入る前に、すべてのコマンドに共通する規約を定義します。エンディアン、エラーコード、6 種類の実行パターン、タイムアウト、並行実行制約、転送バッファを扱います。

5 アプリケーション層プロトコル

5.1 エンディアン

すべての多バイトフィールドは **little-endian** で送受信します。例: `data = 0x12345678` は線上では `[0x78, 0x56, 0x34, 0x12]` の順に流れます（offset 0 が LSB）。

ビット記法は LSB を 0 とします。本書では CONTROL.data の構造を以下のように記述します。

記法	意味
`data[7:0]`	byte 0（= `data & 0xFF`）
`data[15:8]`	byte 1
`data[23:16]`	byte 2
`data[31:24]`	byte 3

例: パラメータが「instance 番号（8 bit）と offset（16 bit）」のとき、`data[7:0] = instance`、`data[23:8] = offset` と表記します。

5.2 エラーコード

NAK（command 欄に `0x02`）が返る場面は 2 段階に分かれます。**(1) パケット形式検証で弾かれる場合**（header / 長さ / footer / CRC の不正。§4.5）— この経路の NAK には下表のエラーコードは入らず、内部 seq_counter が入ります。**(2) コマンド処理段階で拒否される場合**（パラメータ範囲外・未初期化・セッション未開始など）— この経路の NAK の `data[7:0]` に、以下のエラーコードが格納されます。

値	名前	意味
`0x00`	OK	成功（ACK で使用、NAK では使用しない）

値	名前	意味
0x01	INVALID_COMMAND	未定義のコマンドコード、または範囲予約のみで未実装
0x02	INVALID_STATE	現在の状態で実行不可（例: DATA 待ち中に別の CONTROL）
0x03	INVALID_PARAM	パラメータが不正（例: インスタンス番号範囲外、サイズ不一致）
0x04	HARDWARE_BUSY	対象ハードウェアが busy
0x05	OUT_OF_RANGE	アドレス、オフセット、サイズが許容範囲外
0x06	NOT_INITIALIZED	対象インスタンスまたはペリフェラルが未初期化
0x07	OPERATION_FAILED	ハードウェア操作の失敗
0x08	SESSION_ACTIVE	セッションが既に開始されている
0x09	NO_SESSION	セッション未開始のまま、セッション内のみ有効なコマンドを送られた
0x0A	TIMEOUT	操作タイムアウト（将来用）
0x0B	INCOMPLETE_DATA	転送バッファに必要バイト数が揃っていない

5.3 実行パターン

コマンドによって、ホストと本モジュールの間で行き来するパケットの数や順序は異なります。本仕様ではこのやりとりの形を 6 種類に分類し、**Pattern A〜F** と呼びます。各コマンドは必ずいずれかひとつのパターンに属します。

Pattern	通信フロー（`M→D` = ホスト→本モジュール、`D→M` = 本モジュール→ホスト）	例
A	`M→D CONTROL(cmd)` · `D→M ACK(0)`	AI_RESET, BUFFER_CLEAR, FFT_INIT
B	`M→D CONTROL(cmd)` · `D→M ACK(status32)`	AI_GET_LOSS, SYS_GET_UPTIME
B+DATA	`M→D CONTROL(cmd)` · `D→M DATA(payload)`	FLASH_READ_DATA, BUFFER_READ

C	`M→D START` · `D→M ACK`
	`M→D GET_BUSY` · `D→M ACK(busy)`（繰返し）
	`M→D GET_RESULT` · `D→M DATA(result)`
		AI_TRAIN_START, FFT_START

D	`M→D START` · `D→M ACK(hw_status)` `M→D GET_STATUS` · `D→M ACK(hw_status)`（繰返し）	FLASH_ERASE_*
E	`M→D CONTROL(cmd,meta)` · `D→M ACK(0)`（中間） `M→D DATA(payload)` · `D→M ACK/NAK`（最終）	AI_INIT, BUFFER_WRITE, FLASH_WRITE_*
F	`M→D SESSION_BEGIN` · `D→M ACK` …操作群… `M→D SESSION_END` · `D→M ACK`	FLASH_SESSION_BEGIN/END

PATTERN E の中間 ACK について

中間 ACK は、ホストが大きな DATA を送出する前に、本モジュールが CONTROL の内容を受理しパラメータ検証を済ませたことを確認する応答です。これにより、検証で弾かれるはずの DATA をムダに送出することを避けます。**Pattern B+DATA や C の GET_RESULT には中間 ACK はありません。**ホストは CONTROL 送出後、直接 DATA 受信状態に入ります。

5.4 タイムアウト

用途	推奨値	適用
CONTROL 応答待ち	2000 ms	Pattern A / B / D / F の ACK
DATA 応答待ち	1000 ms	Pattern B+DATA, C の GET_RESULT
Pattern E 最終 ACK 待ち	400 ms	DATA 送出後の最終 ACK

Flash 操作のハードウェア所要時間（Pattern D の GET_STATUS ループの目安）

操作	最大時間
Sector erase（program flash, 2 KB）	~50 ms
Sector erase（data flash, 256 B）	~6 ms
Block erase（program flash, 32 KB）	~3 s
Block erase（data flash, 8 KB）	~400 ms
Word write（program flash, 4 B）	< 1 ms
Byte write（data flash, 1 B）	< 1 ms

NOTE / 注記
ブロック消去のような長時間操作は、本モジュール側で Flash 消去ハードウェアと CPU が並行動作する設計となっており、消去中も本モジュールは内部ウォッチドッグを自走でリフレッシュします。ホスト側は任意の間隔で `GET_STATUS` をポーリングしてかまいません（ホスト自身のタイムアウト設計には十分な余裕を持たせること）。

5.5 並行実行制約

関係	同時実行	補足
AI 演算 ↔ AI 演算	不可	インスタンス番号が異なっても同時実行は不可
AI 演算 ↔ FFT 演算	不可	同一アクセラレータを共有
AI/FFT ↔ Flash	可	別ハードウェア
任意の操作 ↔ 転送バッファ占有操作	不可	転送バッファは AI/FFT/重み I/O で共有（§ 5.6）

`AI_GET_BUSY`（0x25）と `FFT_GET_BUSY`（0x32）は **同一の busy 状態** を返します。直近に開始した演算（種別を問わない）の完了確認には、どちらのコマンドを用いてもかまいません。

5.6 転送バッファ

DATA パケット 1 個の上限 1024 バイトを超える入出力のために、本モジュールは内部に **2048 バイトの転送バッファ** を備えています。

ホスト → 本モジュール（大きな入力）

1. `BUFFER_CLEAR` でリセット
2. `BUFFER_WRITE` で段階的に書込み
3. 本来のコマンドを **DATA なし** で CONTROL のみ発行
4. 本モジュールはバッファ内容を入力に使用

本モジュール → ホスト（大きな出力）

1. 結果取得コマンドを発行
2. 1024 B 超なら結果をバッファに書き、ACK の `data` に総バイト数を返す
3. `BUFFER_READ` を必要回数発行して回収

> ⚠ **CAUTION / 注意**
> 転送バッファは AI / FFT / 重み I/O で **共有** されています。複数の操作で同時に占有することはできません。実行完了後、本モジュールは自身の操作で使用した領域をクリアします。

6

CHAPTER 06 · COMMAND REFERENCE

コマンド一覧

本モジュールが提供する個別コマンドを、機能カテゴリごとに記述します。各コマンドは Pattern・目的・CONTROL.data・DATA payload・応答・エラー・備考の項目を持ちます。

6 コマンド一覧

各コマンドの記述は次の項目を持ちます。`Pattern` = §5.3 の 6 種類のいずれか、`目的` / `CONTROL.data` / `DATA payload` / `応答` / `エラー` / `備考`。

6.1 Buffer (0x0A–0x0C)

転送バッファ（§5.6）に対する操作。大きな入出力をするための前準備・事後処理として、AI/FFT/重み I/O コマンドと組み合わせて使います。バッファ単体での用途は想定していません。

0x0A CMD_BUFFER_CLEAR — Pattern A

目的	転送バッファの書込みトラッキング（高水位線）を 0 にリセットする
CONTROL.data	`0x00000000（reserved）`
応答	`ACK, data = 0`
備考	新規のチャンク転送を開始する直前に必ず呼び出すこと

0x0B CMD_BUFFER_WRITE — Pattern E

目的	バッファの任意 offset に最大 1024 バイトのチャンクを書き込む
CONTROL.data	`data[15:0] = offset（0–2047）／ data[31:16] = 0（reserved）`
DATA payload	書き込むバイト列（`length` は 1–1024）
応答	成功: 中間 ACK(0) → 最終 ACK(0) ／ 失敗: 最終 NAK
エラー	offset + length > 2048 — `OUT_OF_RANGE`

0x0C	CMD_BUFFER_READ	Pattern B+DATA
目的	バッファの任意 offset から指定 length 分を読み出す	
CONTROL.data	`data[15:0] = offset` ／ `data[31:16] = length`（内部で 1024 に clamp）	
応答	成功: DATA(payload[length]) を 1 パケットで直送 ／ 失敗: NAK	
エラー	offset + length が書込済み範囲を超える	`OUT_OF_RANGE`

6.2 Flash (0x10–0x1E)

内蔵不揮発メモリへの永続化機能。典型用途は AI モデル重みの保存、アプリ設定の保存、データロギングです。

Flash メモリの構成

領域	アドレス範囲	サイズ	書込単位 / 用途
Program flash	0x10000000–0x1003FFFF	256 KB	ワード（4 バイト）単位。比較的大きなデータ
Data flash	0x18000000–0x18001FFF	8 KB	バイト（1 バイト）単位。細かいデータ

消去粒度

書込前に対象領域の消去が必要です。消去は以下の粒度で行えます。書込アドレスは粒度に応じたアラインメントを満たす必要があります。

単位	Program flash	Data flash
Sector（細かい消去）	2 KB	256 B
Block（粗い消去）	32 KB	8 KB

Flash セッション

Flash 操作は **セッション** の中でのみ実行できます。ホストは `FLASH_SESSION_BEGIN` でセッションを開始し（対象が Program/Data flash かのモードを指定）、必要な操作を済ませたら `FLASH_SESSION_END` で閉じます。

> **CAUTION / 注意**
> セッション外で erase / write / read コマンドを発行すると `NAK with NO_SESSION` が返ります。セッション中はアクセスコードを本モジュールが内部保持するため、ホストが意識する必要はありません。

— 初期化・消去コマンド

0x10 CMD_FLASH_INIT — Pattern A

目的	Flash ペリフェラルの初期化
CONTROL.data	0 (reserved)
応答	ACK(0)
備考	通常 FLASH_SESSION_BEGIN が代行する。個別呼出しは不要

0x11 CMD_FLASH_OPEN — Pattern B

目的	Flash のオープン
CONTROL.data	0 (reserved)
応答	ACK, data[7:0] = ハードウェアステータス (0 = OK, 0xFF = NG)
エラー	ハードウェア操作失敗 OPERATION_FAILED

0x12 CMD_FLASH_CLOSE — Pattern B

目的	Flash のクローズ
CONTROL.data	0 (reserved)
応答	ACK, data[7:0] = ハードウェアステータス
エラー	ハードウェア操作失敗 OPERATION_FAILED

0x13 CMD_FLASH_ERASE_SECTOR_PROG

Pattern D

目的	Program flash の 2 KB sector を消去（要件: セッション mode=1 開始済）
CONTROL.data	消去対象アドレス（32 bit。本モジュールが sector 境界に丸める）
応答	ACK, data[7:0]: `0x00` 消去開始成功（続けて GET_STATUS で完了確認）／ `0x01` busy ／ `0xFF` NG
エラー	NO_SESSION / HARDWARE_BUSY / OPERATION_FAILED —
HW 所要時間	~50 ms

0x14 CMD_FLASH_ERASE_SECTOR_DATA

Pattern D

目的	Data flash の 256 B sector を消去（要件: セッション開始済）
CONTROL.data	消去対象アドレス（sector 境界に丸める）
応答	0x13 と同形式
エラー	NO_SESSION / HARDWARE_BUSY / OPERATION_FAILED —
HW 所要時間	~6 ms

0x15 CMD_FLASH_ERASE_BLOCK_PROG

Pattern D

目的	Program flash の 32 KB block を消去（要件: セッション mode=1 開始済）
CONTROL.data	消去対象アドレス（block 境界に丸める）
応答	0x13 と同形式
HW 所要時間	最大 ~3 秒
備考	消去中も本モジュールは内部ウォッチドッグを自走でリフレッシュするため、ホスト側で短い間隔のポーリングを強制されることはない（§ 5.4 参照）

0x16 CMD_FLASH_ERASE_BLOCK_DATA

Pattern D

目的	Data flash の 8 KB block を消去（要件: セッション開始済）
CONTROL.data	消去対象アドレス（block 境界に丸める）
応答	0x13 と同形式
HW 所要時間	~400 ms

0x17 CMD_FLASH_GET_STATUS

Pattern D poll (B)

目的	Flash ハードウェアのステータスを取得（erase / write の完了確認に使用）。要件なし（セッション外でも呼出可）
CONTROL.data	0 (reserved)
応答 data[7:0]	0x00 アクセス可能（完了またはアイドル） 0x01 Data flash erasing ／ 0x02 Data flash writing 0x04 Program flash erasing ／ 0x08 Program flash writing
エラー	なし

0x18 CMD_FLASH_READ_DATA

Pattern B+DATA

目的	Data flash から **常に 1024 バイト** 読み出す（要件: セッション開始済）
CONTROL.data	読出開始アドレス（0x18000000–0x18001FFF、終端 + 1023 まで収まること）
応答	DATA(payload[1024]) を 1 パケットで返す（中間 ACK 無し）
エラー	NO_SESSION / OUT_OF_RANGE

0x19 CMD_FLASH_READ_PROG

Pattern B+DATA

目的	Program flash から **常に 1024 バイト** 読み出す（要件: セッション開始済）
CONTROL.data	読出開始アドレス（0x10000000–0x1003FFFF、終端 + 1023 まで収まること）
応答	DATA(payload[1024]) を 1 パケットで返す（中間 ACK 無し）
エラー	NO_SESSION / OUT_OF_RANGE

0x1A CMD_FLASH_WRITE_DATA_BYTE

Pattern E

目的	Data flash に複数バイトを書き込む。アドレス addr+0, addr+1, … の順に payload の各バイトを書く（要件: セッション開始済、対象 sector が事前に消去済）
CONTROL.data	書込開始アドレス
DATA payload	書き込むバイト列（任意長 1–1024）
応答	中間 ACK(0) → 最終 ACK, data[7:0] = 0（全 OK）または NAK with OPERATION_FAILED
エラー	NO_SESSION / INVALID_PARAM / OPERATION_FAILED —
備考	各バイト書込後に読み返しで検証。検証失敗時は途中で中断される（それまでに書き込まれたバイトは残る）

0x1B CMD_FLASH_WRITE_PROG_WORD

Pattern E

目的	Program flash に **4 バイト 1 ワード** を書き込む（要件: セッション開始済、対象 sector が事前に消去済、addr が 4 バイトアライン）
CONTROL.data	書込アドレス
DATA payload	4 バイト（LE uint32）。length < 4 の場合は INVALID_PARAM
応答	中間 ACK(0) → 最終 ACK, data[7:0] = 0 または NAK with OPERATION_FAILED
エラー	NO_SESSION / INVALID_PARAM / OPERATION_FAILED —

0x1C CMD_FLASH_SESSION_BEGIN

Pattern F

目的	Flash セッションを開始し、以降の erase / write / read を受け付け可能な状態にする
CONTROL.data	data[7:0] = mode（0 = data flash, 1 = program flash）
応答	ACK, data[7:0] = 0 または NAK（SESSION_ACTIVE / INVALID_PARAM (mode > 1) / OPERATION_FAILED）
備考	セッション中は erase / write をコマンドだけで実行できる

0x1D CMD_FLASH_SESSION_END

Pattern F

目的	Flash セッションを終了（self-program 無効化 + close を実施）
CONTROL.data	0（reserved）
応答	ACK, data[7:0] = 0 または NAK with NO_SESSION

0x1E CMD_FLASH_CONTROL_SELFPROG

Pattern B

目的	Self-program の有効/無効フラグを直接設定（通常は SESSION_BEGIN/END が代行）
CONTROL.data	self-program フラグ値（ハードウェアで規定された値）
応答	ACK, data[7:0] = ハードウェアステータス
エラー	ハードウェア操作失敗 OPERATION_FAILED

6.3 AI/ML (0x20–0x2D)

本モジュールに搭載された小規模ニューラルネットワークを使ったオンデバイス学習（On-Device Learning, ODL）および推論機能を提供します。

– ニューラルネットワークモデルの構造

本モジュールは **入力層 → 隠れ層 → 出力層** という 3 層構造の小型ネットワークを内部に持ちます。ホストは初期化時に、入力次元数・隠れ層ユニット数・出力次元数・学習率・活性化関数・損失関数・初期化シード・その他スケーリング係数を指定して 1 つのネットワークを構成します。上限値は構成バリエーション（構成 A / B）によって異なります（§ 1.3）。

– インスタンス

本モジュールは独立したネットワークを複数同時に保持できます。それぞれを **インスタンス** と呼び、0 始まりの番号で識別します。インスタンスごとに別々のパラメータと学習状態を持てますが、演算ハードウェアは共有されているため、複数のインスタンスを **同時に** 実行することはできません（§ 5.5）。

– 数値表現: bfloat16

AI 機能で扱う数値（入力・出力・教師値・重み・loss）はすべて **bfloat16** 形式です。bfloat16 は 16 bit 浮動小数点形式の一種で、内訳は 1 sign + 8 exponent + 7 mantissa。線上では 2 バイト little-endian で送ります。

`0.01`	≈ `0x3C23`（典型的な学習率）
`1.0`	= `0x3F80`

– 学習方式（オンデバイス学習）

本モジュールの学習はサンプル単位のオンラインアルゴリズムです。ホストは「1 件の入力ベクトル + 1 件の教師値ベクトル」を 1 回の `AI_TRAIN_START` で投入し、本モジュールはその場で重みを更新します（バッチ学習ではありません）。推論のみ行う場合は `AI_PREDICT_START`（教師値なし）を使います。

— ODL_Parameters 構造体（18 バイト, packed, LE）

`AI_INIT` の DATA payload はこの構造体です。値はすべて little-endian。

off	size	field	型	説明
0	2	inputSize	uint16	入力次元数
2	2	hiddenSize	uint16	隠れ層ユニット数
4	2	outputSize	uint16	出力次元数
6	2	forgettingFactor	bfloat16	学習率 / 忘却係数（典型 0.01 = 0x3C23）
8	1	activationFunction	uint8	0 = Linear, 1 = Sigmoid, 2 = ReLU
9	1	lossFunction	uint8	0 = MAE, 1 = MSE
10	2	seed	uint16	重み初期化乱数シード
12	2	scaleAlpha	bfloat16	重み α スケール
14	2	scaleGamma	bfloat16	重み γ スケール（ESN モード用）
16	2	leakRate	bfloat16	リークレート（ESN モード用）

scaleGamma と leakRate は Echo State Network 系のモデル設定用です。通常の前向きネットワークとして使う場合は意味を持ちません（デフォルト値で良い）。

重み配列（β, P）

配列	サイズ計算式	用途
β (beta)	hiddenSize × outputSize × 2	出力層の重み
P	hiddenSize × hiddenSize × 2	共分散行列

ホストは `AI_SET_WEIGHT_*` で外部からセット、`AI_GET_WEIGHT_*` で取り出せます。典型用途は「学習済み重みを Flash に保存し次回起動時に復元」「学習済みモジュールから別モジュールへ重みを転送」。

サイズ制約（構成バリエーション依存）

インスタンス数・`inputSize`・`hiddenSize` の上限は構成バリエーション（§1.3）に依存します。各構成の具体的な上限値は **Solist-AI データシート**を参照してください。許容上限を超える値を渡すと `OUT_OF_RANGE` が返ります。

— 初期化・リセット

0x20 CMD_AI_INIT

Pattern E (A+DATA)

目的	AI インスタンスを初期化する（パラメータ設定とリセット）
CONTROL.data	`data[7:0] = instance`（0 または 1）
DATA payload	18 バイト ODL_Parameters（§6.3）
応答	中間 ACK(0) → 最終 ACK, data = 0 または NAK
エラー	instance 範囲外 — INVALID_PARAM length < 18 — INVALID_PARAM inputSize = 0 または上限超 — OUT_OF_RANGE hiddenSize = 0 または上限超 — OUT_OF_RANGE outputSize = 0 — INVALID_PARAM
備考	成功後は本モジュールが inputSize / hiddenSize / outputSize を保持。 GET_RESULT および GET_WEIGHT_*_SIZE の応答サイズはこれらから自動計算される

0x21 CMD_AI_RESET

Pattern A

目的	AI 学習状態を初期値にリセットする（パラメータは保持される）
CONTROL.data	`data[7:0] = instance`
応答	ACK(0) または NAK with INVALID_PARAM

0x22 CMD_AI_CLEAR_RAM

Pattern A

目的	すべての AI インスタンスをリセットし、内部 RAM の AI 関連領域を消去する
CONTROL.data	0（reserved）
応答	ACK(0)
エラー	なし

— 学習・推論

0x23 CMD_AI_TRAIN_START

Pattern C · 2 mode

目的	AI インスタンスで 1 サンプル分の学習を実行（推論 → loss 算出 → 重み更新）
CONTROL.data	`data[7:0] = instance` ／ `data[31:16] = payload_size`（**0** = バッファモード、**非 0** = 直接モード）
DATA payload（直接モード）	`inputSize×2 + outputSize×2` バイト。`[0 .. inputSize*2-1]` = x[]（入力）、後続 = t[]（教師値）。いずれも bfloat16 LE 配列
バッファモード	① BUFFER_CLEAR ② BUFFER_WRITE で x[] ‖ t[] を連結して書込（1024 B 超は複数回）③ AI_TRAIN_START を payload_size = 0 で送る
応答	直接: 中間 ACK(0) → 最終 ACK, data = 0 ／ バッファ: 最終 ACK, data = 0
エラー	instance 範囲外 / サイズ不一致 — INVALID_PARAM INIT 未実行 — NOT_INITIALIZED バッファに必要バイト未満 — INCOMPLETE_DATA
完了タイミング	最終 ACK は学習完了後に返る。GET_BUSY を経由せず直接 GET_RESULT / GET_LOSS を発行してよい

0x24 CMD_AI_PREDICT_START

Pattern C · 2 mode

目的	AI インスタンスで推論を開始する（教師値なし、重み更新も行わない）
CONTROL.data	`data[7:0] = instance` ／ `data[31:16] = payload_size`（0 = バッファモード、非 0 = 直接モード）
DATA payload（直接モード）	`inputSize×2` バイト = x[]（bfloat16）
応答	直接: 中間 ACK(0) → 最終 ACK, data = 0 ／ バッファ: 最終 ACK, data = 0
エラー	TRAIN_START と同じ
完了タイミング	演算は非同期で進行する。GET_BUSY で完了を確認してから GET_RESULT を発行すること

— 完了確認・結果・重み I/O

0x25 CMD_AI_GET_BUSY

Pattern C (POLL)

目的	AI / FFT 演算の完了確認
CONTROL.data	`0 (reserved)`
応答	`ACK, data[7:0] = busy_flag (0 = 完了, 1 = 実行中)`
備考	AI と FFT で **共通の busy 状態** を返す（§5.5）

0x26 CMD_AI_GET_RESULT

Pattern C (GET_RESULT)

目的	直近の推論 / 学習の出力 y[] を取得する
CONTROL.data	`data[7:0] = instance`
応答	DATA(payload[outputSize×2]) を 1 パケットで返す
エラー	instance 範囲外 — INVALID_PARAM INIT 未実行 — NOT_INITIALIZED outputSize × 2 > 512 — OUT_OF_RANGE
備考	応答サイズは outputSize（INIT 時の値）から計算。受信 DATA.length にも実サイズが入る

0x27 CMD_AI_GET_LOSS

Pattern B

目的	直近の学習で計算された loss 値を取得する
CONTROL.data	`data[7:0] = instance`
応答	`ACK, data[15:0] = loss (bfloat16)`
エラー	INVALID_PARAM / NOT_INITIALIZED —

0x28 CMD_AI_SET_WEIGHT_BETA

Pattern E · 2 mode

目的	β 重み配列を書き込む
CONTROL.data	`data[7:0] = instance` ／ `data[23:8] = offset`（バイト単位、書込開始位置）
DATA payload	（直接モード）bfloat16 配列のバイト列（最大 1024）。バッファモードは BUFFER_CLEAR + BUFFER_WRITE 後 CONTROL のみ送る
応答	直接: 中間 ACK(0) → 最終 ACK, data = 0 ／ バッファ: ACK, data = 0
備考	1024 バイト超の書込みはバッファモードを使用すること

0x2A CMD_AI_SET_WEIGHT_P

Pattern E · 2 mode

目的	共分散行列 P を書き込む
CONTROL.data	`data[7:0] = instance` ／ `data[23:8] = offset`（バイト単位、書込開始位置）
DATA payload	（直接モード）bfloat16 配列のバイト列（最大 1024）。バッファモードは BUFFER_CLEAR + BUFFER_WRITE 後 CONTROL のみ送る（0x28 と同一構造）
応答	直接: 中間 ACK(0) → 最終 ACK, data = 0 ／ バッファ: ACK, data = 0
備考	P の総バイト数は hiddenSize × hiddenSize × 2。1024 バイト超はバッファモードを使用すること

0x29 CMD_AI_GET_WEIGHT_BETA

B+DATA / Buffer 自動

目的	β 重み配列を読み出す
CONTROL.data	`data[7:0] = instance` ／ `data[23:8] = offset`（将来用、本リリースは `0` 推奨）
応答 total_size = hiddenSize×outputSize×2	total_size > 2048 → NAK with OUT_OF_RANGE ／ ≤ 1024（Direct）→ DATA を 1 パケット直送 ／ 1024 < … ≤ 2048（Buffer）→ ACK, data = total_size（BUFFER_READ で回収）
エラー	INVALID_PARAM / NOT_INITIALIZED / OUT_OF_RANGE —
備考	事前に GET_WEIGHT_BETA_SIZE (0x2C) で total_size を取得すると Direct / Buffer のいずれが返るか判定できる

0x2B CMD_AI_GET_WEIGHT_P

B+DATA / Buffer 自動

目的	共分散行列 P を読み出す
CONTROL.data	`data[7:0] = instance` ／ `data[23:8] = offset`（将来用、本リリースは `0` 推奨）
応答 total_size = hiddenSize×hiddenSize×2	total_size > 2048 → NAK with OUT_OF_RANGE ／ ≤ 1024（Direct）→ DATA を 1 パケット直送 ／ 1024 < … ≤ 2048（Buffer）→ ACK, data = total_size（BUFFER_READ で回収）
エラー	INVALID_PARAM / NOT_INITIALIZED / OUT_OF_RANGE —
備考	事前に GET_WEIGHT_P_SIZE (0x2D) で total_size を取得すると Direct / Buffer のいずれが返るか判定できる

0x2C CMD_AI_GET_WEIGHT_BETA_SIZE

Pattern B

目的	β 重みの総バイト数を取得する
CONTROL.data	`data[7:0] = instance`
応答	ACK, data = total_size（uint32, バイト数）= hiddenSize × outputSize × 2
エラー	INVALID_PARAM / NOT_INITIALIZED —

0x2D CMD_AI_GET_WEIGHT_P_SIZE

Pattern B

目的	共分散行列 P の総バイト数を取得する
CONTROL.data	`data[7:0] = instance`
応答	ACK, data = total_size（uint32, バイト数）= hiddenSize × hiddenSize × 2
エラー	INVALID_PARAM / NOT_INITIALIZED —

6.4 FFT (0x30–0x33)

ホストが入力サンプル列を送ると、本モジュールがハードウェア FFT 演算を実行し結果サンプル列を返します。典型用途は音声・振動など時系列信号の周波数解析です。AI と FFT は同じ演算リソースを使うため同時実行できません（§ 5.5）。

項目	形式
入力 / 出力サンプル	int16_t（線上では 2 バイト LE）
サンプル数（FFT 点数）	1〜1024
窓関数	矩形（なし）または Hann
呼出フロー	FFT_INIT（1 回）→（FFT_START → FFT_GET_BUSY → FFT_GET_RESULT）を各回

0x30 CMD_FFT_INIT

Pattern A

目的	FFT を初期化する（サイズと窓関数を設定）。同じ設定で繰り返す場合は再初期化不要
CONTROL.data	`data[15:0] = size (1–1024) ／ data[23:16] = window (0 = なし, 1 = Hann) ／ data[31:24] = 0`
応答	`ACK(0)`
エラー	size = 0 または > 1024 ／ window > 1 — `INVALID_PARAM`

0x31 CMD_FFT_START

Pattern C · 2 mode

目的	FFT 演算を開始する
CONTROL.data	`data[15:0] = size`（INIT で指定したサイズと一致必須）／ `data[31:16] = 0`
DATA payload	（直接モード）size×2 バイト（int16_t 配列）。バッファモードは BUFFER_CLEAR + BUFFER_WRITE 後 CONTROL のみ送る
応答	直接: 中間 ACK(0) → 最終 ACK, data = 0 ／ バッファ: 最終 ACK, data = 0
エラー	INIT 未実行 — NOT_INITIALIZED バッファ不足 — INCOMPLETE_DATA サイズ不一致 — INVALID_PARAM

0x32	CMD_FFT_GET_BUSY	Pattern C (POLL)
目的	FFT / AI 演算の完了確認	
CONTROL.data	0 (reserved)	
応答	ACK, data[7:0] = busy_flag (0 = 完了, 1 = 実行中)	
備考	AI_GET_BUSY と **同一の busy 状態** を返す（§5.5）	

0x33	CMD_FFT_GET_RESULT	C / Buffer 自動
目的	FFT 結果を取得する	
CONTROL.data	data[15:0] = size（INIT で指定したサイズと一致必須）	
応答 result_size = size×2	> 2048 → NAK with OUT_OF_RANGE ／ ≤ 1024（Direct）→ DATA を 1 パケット直送 ／ 1024 < … ≤ 2048（Buffer）→ ACK, data = result_size（BUFFER_READ で回収）	
エラー	NOT_INITIALIZED / INVALID_PARAM / OUT_OF_RANGE	—

6.5 System (0xD0–0xD3)

本モジュール自身の状態を問い合わせるコマンド。本リリースで完全に実装されているのは `SYS_GET_CAPABILITY` と `SYS_GET_UPTIME` で、`SYS_GET_VERSION` / `SYS_GET_STATUS` は将来仕様の枠組みのみが提供されています。

0xD0	CMD_SYS_GET_VERSION	本リリース未提供
目的	ファームウェアバージョン文字列を取得する（Pattern B+DATA、将来仕様）	
本リリース応答	NAK with NOT_INITIALIZED	
将来仕様	DATA(payload = ASCIIZ バージョン文字列) を返す予定	

0xD1 CMD_SYS_GET_STATUS

本リリース 常に 0

目的	システム状態ワードを取得する（Pattern B）
本リリース応答	ACK, data = 0x00000000
将来仕様 ビット定義	bit 0 = flash_busy ／ bit 1 = ai_fft_busy ／ bit 2 = reserved ／ bit 3 = adc_running ／ bit 4–31 = reserved

0xD2 CMD_SYS_GET_CAPABILITY

Pattern B

目的	本モジュールがビルド時に取り込んだ能力記述子（build-time capability）を取得する。`SYS_GET_STATUS`（動的な busy 状態）と異なり、稼働中は常に一定値を返す
CONTROL.data	0 (reserved)。本リリースでは中身を無視するが、将来のサブリクエスト用途に予約されているため、ホストは 0 を送信すること
応答 capability_word (uint32, LE)	bit[15:0] `input_max` = このビルドがサポートする ODL `inputSize` の最大値（`ODL_MAX_INPUTS`。ライブラリ変種により 256 または 512）／ bit[31:16] `features` = 将来の機能フラグ用予約（本リリースは 0 固定）
典型用途	マスタが起動時に一度発行し、以降のリクエスト構築（`SET_WEIGHT` 系ペイロード上限の検証など）に利用する
エラー	なし。ただし §4.2 の共通ルールに従い `layer_id ≠ 0x01` で受信した場合は NAK (INVALID_COMMAND)

0xD3 CMD_SYS_GET_UPTIME

Pattern B

目的	本モジュール起動からの経過時間を取得する
CONTROL.data	`0 (reserved)`
応答	`ACK, data = uptime (uint32)`
注意	返り値の単位はミリ秒に相当する近似値。内部クロック依存のため厳密なミリ秒からは乖離があり、絶対時刻ではなく **相対経過の単調増加カウンタ** として利用すること

A

APPENDIX

付録

本文を補完する参照情報をまとめます。全コマンドコードの一覧と、本モジュールを実利用するうえでの代表的な呼出シーケンスを掲載します。

A コマンドコード一覧

いずれも Application 層のコマンドです。ワイヤ上の `command`（16 bit）は `layer_id = 0x01` を上位バイト、下記 `Code` を下位バイトに置いた値になります（例: `FLASH_INIT` は `command = 0x0110`）。詳細は §4.2 を参照。

— Buffer / Flash

Code	名前	Pattern	提供状態
0x0A	BUFFER_CLEAR	A	提供
0x0B	BUFFER_WRITE	E	提供
0x0C	BUFFER_READ	B+DATA	提供
0x10	FLASH_INIT	A	提供
0x11	FLASH_OPEN	B	提供
0x12	FLASH_CLOSE	B	提供
0x13	FLASH_ERASE_SECTOR_PROG	D	提供
0x14	FLASH_ERASE_SECTOR_DATA	D	提供
0x15	FLASH_ERASE_BLOCK_PROG	D	提供
0x16	FLASH_ERASE_BLOCK_DATA	D	提供
0x17	FLASH_GET_STATUS	D poll	提供
0x18	FLASH_READ_DATA	B+DATA	提供

Code	名前	Pattern	提供状態
0x19	FLASH_READ_PROG	B+DATA	提供
0x1A	FLASH_WRITE_DATA_BYTE	E	提供
0x1B	FLASH_WRITE_PROG_WORD	E	提供
0x1C	FLASH_SESSION_BEGIN	F	提供
0x1D	FLASH_SESSION_END	F	提供
0x1E	FLASH_CONTROL_SELFPROG	B	提供

— AI/ML / FFT / System

Code	名前	Pattern	提供状態
0x20	AI_INIT	E	提供
0x21	AI_RESET	A	提供
0x22	AI_CLEAR_RAM	A	提供
0x23	AI_TRAIN_START	C	提供
0x24	AI_PREDICT_START	C	提供
0x25	AI_GET_BUSY	C poll	提供（AI/FFT 共用）
0x26	AI_GET_RESULT	C+DATA	提供
0x27	AI_GET_LOSS	B	提供
0x28	AI_SET_WEIGHT_BETA	E	提供
0x29	AI_GET_WEIGHT_BETA	B+DATA / Buffer	提供
0x2A	AI_SET_WEIGHT_P	E	提供
0x2B	AI_GET_WEIGHT_P	B+DATA / Buffer	提供
0x2C	AI_GET_WEIGHT_BETA_SIZE	B	提供

Code	名前	Pattern	提供状態
0x2D	AI_GET_WEIGHT_P_SIZE	B	提供

— FFT / System

Code	名前	Pattern	提供状態
0x30	FFT_INIT	A	提供
0x31	FFT_START	C	提供
0x32	FFT_GET_BUSY	C poll	提供
0x33	FFT_GET_RESULT	C+DATA / Buffer	提供
0xD0	SYS_GET_VERSION	B+DATA	本リリースでは未提供
0xD1	SYS_GET_STATUS	B	本リリースでは常に 0
0xD2	SYS_GET_CAPABILITY	B	提供
0xD3	SYS_GET_UPTIME	B	提供

B 典型呼出シーケンス

各 `CONTROL(NAME=0xNN, …)` の `0xNN` は読みやすさのため下位バイト（`code`）のみを表記しています。実際にワイヤに載る `command`（16 bit）は §4.2 の規則に従い `layer_id` を上位バイトに詰めた値です。すなわち PING/PONG など Transport 層は `command = 0x00NN`、Application 層（AI/Flash/FFT/System/Buffer）は `command = 0x01NN`。

B.1 AI 学習 1 サンプル（直接モード, inputSize=64, outputSize=3）

sequence · B.1

```
[初期化 — 1 回のみ]
M→D CONTROL(AI_INIT=0x20, data=0)
D→M ACK(0)                                ← 中間
M→D DATA(18 B ODL_Parameters)
D→M ACK(0)                                ← 最終

[学習 — サンプルごと]   payload_size = (64+3)*2 = 134
M→D CONTROL(AI_TRAIN_START=0x23, data=instance | payload_size<<16)
D→M ACK(0)                                ← 中間
M→D DATA([134 B = x[64] || t[3] in bfloat16 LE])
D→M ACK(0)                                ← 最終（学習完了）

[結果取得]
M→D CONTROL(AI_GET_RESULT=0x26, data=instance)
D→M DATA([6 B = y[3] in bfloat16])   ← 1 パケット直送
M→D CONTROL(AI_GET_LOSS=0x27, data=instance)
D→M ACK(data[15:0] = loss bfloat16)
```

B.2 FFT 1024 点（バッファモード）

sequence · B.2

```
[初期化 — 1 回のみ]
M→D CONTROL(FFT_INIT=0x30, data=1024 | window<<16)
D→M ACK(0)

[各回 — 入力 2048 B をバッファに書く]
M→D CONTROL(BUFFER_CLEAR=0x0A, 0)
D→M ACK(0)
M→D CONTROL(BUFFER_WRITE=0x0B, offset=0)
D→M ACK(0)                              ← 中間
M→D DATA([1024 B 入力前半])
D→M ACK(0)                              ← 最終
M→D CONTROL(BUFFER_WRITE=0x0B, offset=1024)
D→M ACK(0)                              ← 中間
M→D DATA([1024 B 入力後半])
D→M ACK(0)                              ← 最終

[FFT 実行 — バッファ内容を使用]
M→D CONTROL(FFT_START=0x31, size=1024)
D→M ACK(0)                              ← DATA 無し（バッファ使用）

[完了待ち]
loop:
  M→D CONTROL(FFT_GET_BUSY=0x32, 0)
  D→M ACK(data[7:0] = busy_flag)
```

sequence · B.2

```
  if busy_flag == 0: break


[結果取得 — result_size = 2048 のため Buffer Mode 自動]
M→D CONTROL(FFT_GET_RESULT=0x33, size=1024)
D→M ACK(data = 2048)                    ← Buffer Mode 通知
M→D CONTROL(BUFFER_READ=0x0C, offset=0 | 1024<<16)
D→M DATA([1024 B 結果前半])
M→D CONTROL(BUFFER_READ=0x0C, offset=1024 | 1024<<16)
D→M DATA([1024 B 結果後半])
```

B.3 Program Flash への 1 ワード書込

sequence · B.3

```
M→D CONTROL(FLASH_SESSION_BEGIN=0x1C, data=1)   ← mode=1: program flash
D→M ACK(0)
M→D CONTROL(FLASH_ERASE_SECTOR_PROG=0x13, addr=0x10010000)
D→M ACK(0)                              ← 消去開始成功

loop:                                   ← 完了待ち（約 50 ms）
  M→D CONTROL(FLASH_GET_STATUS=0x17, 0)
  D→M ACK(data[7:0] = hw_status)
  if hw_status == 0: break

M→D CONTROL(FLASH_WRITE_PROG_WORD=0x1B, addr=0x10010000)
D→M ACK(0)                              ← 中間
M→D DATA([4 B = 0xDEADBEEF in LE])
D→M ACK(data[7:0] = 0)                  ← 最終、書込成功

M→D CONTROL(FLASH_SESSION_END=0x1D, 0)
D→M ACK(0)
```

B.4 接続疎通確認（PING / PONG）

sequence · B.4

```
[PING/PONG は Transport 層 → command = 0x0005 / 0x0006 (layer_id = 0x00)]
M→D CONTROL(0x05 PING, data=0x12345678)
```

sequence · B.4

```
D→M CONTROL(0x06 PONG, data=0x12345678)    ← data はホストが送った値をそのまま返す
```

表記 / LEGEND

`M→D` = ホスト → 本モジュール、`D→M` = 本モジュール → ホスト。各シーケンスは代表的な呼び出し例であり、実際のパラメータ値・サイズは構成や用途に応じて変わります。

免責事項 / Disclaimer

本書に記載された情報は、発行時点のものであり、予告なく変更されることがあります。本書の内容については正確を期していますが、**当社は本書の記載内容に関して明示・黙示を問わずいかなる保証も行わず**、本書の使用または使用不能から生じるいかなる損害（逸失利益、事業の中断、データの損失等を含む）についても責任を負いません。本モジュールは評価・開発用途を目的としており、医療機器・航空宇宙・原子力など、人命に関わる用途や高い信頼性が要求される用途への使用を想定していません。

本モジュールの仕様、外観、付属品は改良のため予告なく変更される場合があります。お客様が本モジュールを組み込んだ最終製品の適合性・安全性については、お客様ご自身の責任において十分に評価・検証してください。

注記・特記事項 / Notes & Special Remarks

※1 本書中の数値（アドレス・サイズ・タイミング等）は、本インタフェース仕様を規定するものであり、ホスト側はこれらの記載に基づいて実装してください。本書に記載のない挙動には依存しないでください。

※2 Solist-AI™ の推論精度は学習データ・動作環境に依存し、特定の性能を保証するものではありません。

※3 ファームウェア更新により、将来コマンドの追加や非同期通知機能の提供など、機能が拡張される場合があります。

※4 輸出に際しては、関連する輸出関連法令を遵守してください。該当する場合は事前に許可が必要です。

著作権表示 / Copyright

本書の著作権は当社に帰属します。当社の事前の書面による許可なく、本書の全部または一部を複製・転載・改変・翻訳・再配布することを禁じます。