binho

APPLICATION NOTE

Programming STM32 microcontrollers over I2C with the Binho Pulsar

AN0002

Document information

Keywords	AN0002, Pulsar, Supernova, I2C, STM32, STM32H5, STM32H503, system bootloader, in-system programming, firmware update
Abstract	The system bootloader programmed into the ROM of STM32 microcontrollers offers an I2C interface on most families, allowing internal flash memory to be erased, programmed and verified over two wires with no debug probe attached. This application note describes that interface, the connections required between a Binho Pulsar USB host adapter and the target, and the procedure for programming a device. The same procedure and the same utility work with a Binho Supernova. A complete reference utility written in Python accompanies the document. Measured results are given for an STM32H503 on a NUCLEO-H503RB board.

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

REVISION 1.0
29 AUGUST 2026

1 Introduction

The system bootloader programmed into the ROM of every STM32 microcontroller provides a means of programming internal flash memory over a serial interface, without a debug probe and without any user code present on the device. The interfaces offered by the bootloader vary by device and are listed per part number in AN2606.

I2C appears on that list for most STM32 families, which makes it the broadest of the in-system programming options. Where a board already routes an I2C bus to reach sensors, EEPROMs or a power-management device, the same two wires can carry a firmware update, and no dedicated programming header is required. Where a product is already manufactured in volume, an I2C bus is usually present somewhere on the board whether or not it was planned for programming.

This note is the I2C counterpart to AN0001, which covers the I3C interface on the STM32H5 series. The command set is nearly the same on both. What differs is how the target reports status and how it says it is busy, and those differences shape the host code more than the command list does.

1.1 Scope

This document describes:

- the structure of the I2C system bootloader protocol, including the mechanism used to report command status and the mechanism used to signal that the target is busy
- the connections required between a Binho Pulsar and an STM32 target
- the procedure for identifying, erasing, programming and verifying a device
- a method for confirming from outside the programming tool that the target is running the newly programmed image
- the accompanying reference utility, which drives either a Pulsar or a Supernova

It does not cover programming over UART, USB, SPI, CAN or I3C; option-byte programming; readout protection, which the ROM bootloader on the tested part does not offer over this interface; or production programming fixtures. For I3C on the STM32H5 series, see AN0001.

1.2 Applicable devices

The I2C system bootloader is not specific to one family. The pins it uses and the address it answers on differ per part, and both are given in AN2606 for every device that offers the interface. Neither is guessable, and neither is consistent across families.

Table 1. Applicable devices

Family	Example part	Bootloader I2C interface	Basis
STM32H5	STM32H503	I2C2, `PB3` / `PB4`, address `0x67`	Tested for this document
STM32H5	STM32H563	Per AN2606	AN2606, not tested
STM32G0, G4	—	Per AN2606	AN2606, not tested
STM32L4, L5	—	Per AN2606	AN2606, not tested
STM32U5	—	Per AN2606	AN2606, not tested

Only the STM32H503 row was exercised on hardware. The other rows record that AN2606 lists an I2C interface for those families; the pins, the address and the supported command list must be read from AN2606 for the part in hand before any of this document's commands are issued.

1.3 Related documents

Table 2. Related documents

Document	Title
AN2606	STM32 microcontroller system memory boot mode
AN4221	I2C protocol used in the STM32 bootloader
RM0492	STM32H503 reference manual
AN0001	Programming STM32 microcontrollers over I3C with the Binho Supernova

2 System overview

2.1 Roles on the bus

The host adapter is the I2C controller. The target, held in system bootloader mode, is an I2C target at a fixed address given in AN2606. On the STM32H503 that address is `0x67`, stated in AN2606 as `0b1100111x`, where `x` is the read/write bit.

The address is fixed in ROM and cannot be changed, which has one consequence worth planning for: two STM32s of the same part number cannot be held in the bootloader on one bus at the same time. This is the practical difference from the I3C interface of AN0001, where dynamic address assignment gives each target a distinct address at run time and several targets can share a bus.

2.2 Status reporting

Every command is a write followed by a one-byte read. The target answers with `0x79` for ACK or `0x1F` for NACK. This is the request-and-response shape most readers will expect, and it is the main way the I2C interface is easier to drive than the I3C one, where status arrives

asynchronously as an in-band interrupt.

Table 3. Status values returned by the bootloader

Value	Meaning	Observed
`0x79`	ACK, command accepted	Yes
`0x1F`	NACK, command rejected	Yes
`0x76`	BUSY	No, never transmitted

ST defines a BUSY byte, `0x76`. On the tested part it is never transmitted. Every rejection observed came back as `0x1F`, and a busy target does not answer with a status byte at all. Host code written to wait for `0x76` waits forever.

2.3 How the target says it is busy

This is the one mechanism that shapes the host code, and it is worth understanding before writing any.

An erase or a program takes far longer than a bus transaction. The bootloader offers two ways of covering that gap, and the choice of which to use is the host's:

- **Plain commands** hold SCL low, which is ordinary I2C clock stretching, until the operation completes.
- **No-stretch commands**, a separate set of opcodes, do not stretch. A busy target instead **stops acknowledging its own address**, and the host polls by retrying the transaction until the address is acknowledged again.

A NACKed address means nothing was delivered, so retrying is always safe.

Which set the utility uses depends on the adapter. A Pulsar on its dedicated I2C port tolerates clock stretching, so the plain commands are used and nothing further is needed. A Supernova driving the bus from its I3C port does not tolerate it, so the utility switches to the no-stretch variants automatically on that path; Section 5.7 explains why. `--no-stretch` forces them on any adapter.

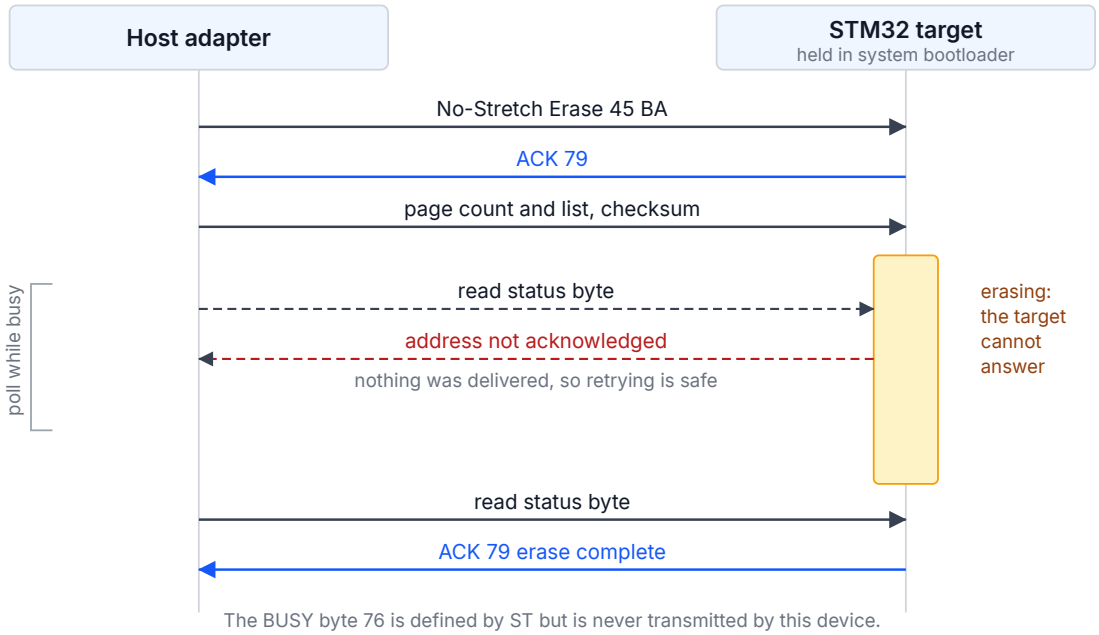

Figure 1. Erase using the no-stretch commands. The target reports that it is busy by not acknowledging its address, rather than by returning a status byte.

2.4 Host adapter options

Two configurations are covered by this document, and both were verified on hardware.

Table 4. Host adapter configurations covered

Adapter	Port	Status
Binho Pulsar	Dedicated I2C port	Verified, used for all measurements
Binho Supernova	I3C port, driven in I2C mode	Verified functionally

The Supernova drives a legacy I2C target from its I3C port, which supplies the bus pull-ups. The utility selects this with `--i2c-port i3c`. Programming from the Supernova's dedicated I2C port is outside the scope of this revision.

3 Hardware setup

3.1 Required equipment

Table 5. Required equipment

Item	Used here
Host adapter	Binho Pulsar, firmware 4.4.0
Target board	NUCLEO-H503RB
Wiring	Five wires: three for the bus, two for reset control
Host	Python 3.12

A Binho Supernova substitutes for the Pulsar throughout, with the connection differences noted in Section 3.3.

3.2 Connections

Table 6. Connections between the Pulsar and the target

Pulsar	Target	Required
SCL	`PB3`	Yes
SDA	`PB4`	Yes
GND	GND	Yes
GPIO 1	`NRST`	Optional, for automatic bootloader entry
GPIO 2	`BOOT0`	Optional, for automatic bootloader entry

`PB3` and `PB4` are the pins AN2606 assigns to the bootloader's I2C interface on this part. They are not the pins a reader would guess: on many STM32 boards the pins labelled for I2C in the user manual belong to a different peripheral instance, and `PB3` and `PB4` carry `JTDO`/`TRACESWO` and `NJTRST` in their reset default. Confirm against AN2606 for the part in hand rather than against the board silkscreen.

Without the two GPIO wires, the utility prompts for bootloader entry to be performed by hand. With them, the whole cycle runs unattended, which is what the procedure in Section 6 assumes.

3.3 Connecting a Supernova instead

Wire SCL, SDA and GND to the Supernova's **I3C port**, and add `--i2c-port i3c` to every command. The GPIO connections are unchanged, and the procedure of Section 6 applies as written.

Two differences are worth knowing before choosing this path.

The I3C controller offers a fixed set of legacy I2C rates rather than an arbitrary clock, so `--i2c-speed` snaps to the nearest of them. The utility reports the rate it actually programmed, and says so when that differs from the one requested.

Sessions opened reliably at 100 kHz on the configuration described here, and did not open at all at 400 kHz or 1 MHz, where the target either stopped acknowledging its address or rejected the first command. The measurements of Section 9 were taken with a Pulsar on its dedicated I2C port, where all three rates worked, and they should not be assumed to carry over to this one. Where bus speed matters, use the dedicated port.

3.4 Bus considerations

The bus must have pull-up resistors. The Pulsar's dedicated I2C port and the Supernova's I3C port both supply their own, selectable in software; the utility requests 1 kΩ by default and `--pullup` changes it. A bus with no pull-ups cannot reach its idle-high state, and a controller on such a bus reports that it could not generate a start condition rather than reporting a missing target.

> **CAUTION** Set the bus voltage to match the target before connecting. The utility sets 3.3 V, which suits a NUCLEO-H503RB. Driving 3.3 V into a target running at a lower supply can damage it.

One further consideration applies only when the Supernova drives the bus from its I3C port. An I3C controller is not obliged to tolerate a target that holds SCL, because I3C targets never stretch the clock, and it may abort the transfer rather than wait. The reference utility handles this automatically; Section 5.7 describes the mechanism, and no user action is required.

4 Software setup

4.1 Installing the host package

The utility talks to the adapter through its vendor package. Install the one matching the adapter in use:

```
pip install binhopulsar        # for a Pulsar
pip install binhosupernova     # for a Supernova
```

Versions used for this document: `binhopulsar` 1.2.0, `binhosupernova` 4.2.0, Python 3.12. No other package is required; the utility parses Intel HEX itself.

4.2 Obtaining the reference utility

`stm32_flash.py` is supplied in the assets archive that accompanies this document:

```
https://cdn.binho.io/application-notes/AN0002/AN0002-assets.zip
```

The same files, together with the Markdown source of this document, are published at `https://github.com/binhollc/application-notes`. Section 12 lists the archive contents.

The archive expands to an `assets` directory holding the utility, and a `firmware` subdirectory holding the verification images of Section 7. Every command in this document is shown as it would be run from inside that `assets` directory.

The utility is a single file with no imports beyond the standard library and one adapter package, so it can be copied anywhere and run directly. It detects which adapter is attached, but both adapters offer more than one bus, so `--bus i2c` names the one in use.

5 Bootloader protocol

5.1 Entering system bootloader mode

`BOOT0` is held high while `NRST` is pulsed low and released. The device starts in system memory rather than in the application.

`NRST` is released by switching the host pin to an input rather than by driving it high, so that the on-board reset circuit is not fought. The utility does this.

To return to the application, `BOOT0` is driven low and `NRST` pulsed again. Where those pins are not wired, the Go command jumps to the programmed image without a reset.

5.2 Session start

There is none. Unlike the I3C interface, which requires a synchronisation byte to open a session, the I2C interface begins with the first command. Nothing is sent before it.

5.3 Command framing

Every command is a two-byte write of the opcode and its complement, followed by a one-byte read of the status:

```
write 0x67  [op, op ^ 0xFF]
read  0x67  1 byte              -> 0x79 ACK, or 0x1F NACK
```

Commands taking an address send four bytes big-endian followed by a one-byte XOR checksum of those four bytes, and take another ACK before their payload.

5.4 Reading the Get response

Get returns one frame of `2 + N` bytes: a count, a protocol version, then `N` opcodes. The host cannot know `N` in advance, and reading past the end of the frame times the bus out.

The utility issues Get twice. The first pass reads two bytes to learn `N`; the second reissues Get and reads the whole frame. Reading short is harmless, because the target abandons the remainder of the frame and moves on to its closing ACK. Reading long is not.

5.5 Supported commands

Table 7. Commands supported over I2C on the STM32H503

Opcode	Command
`0x00`	Get
`0x01`	Get Version
`0x02`	Get ID
`0x11`	Read Memory
`0x21`	Go
`0x31`	Write Memory
`0x44`	Extended Erase
`0x50`	Special Command
`0x63`	Write Protect
`0x73`	Write Unprotect
`0x32`	No-Stretch Write Memory
`0x45`	No-Stretch Erase
`0x64`	No-Stretch Write Protect
`0x74`	No-Stretch Write Unprotect
`0xA1`	Checksum

Fifteen commands, five more than the same part offers over I3C. Note what is absent: Readout Protect (`0x82`) and Readout Unprotect (`0x92`). ST's Open Bootloader reference implementation includes them, but the ROM bootloader on this part does not offer them over I2C, and the ROM is the authority. The reference implementation and the ROM are different programs, and where they disagree it is the device that is right.

5.6 Erase

Extended Erase takes a page count followed by a list of page numbers, or a special value.

Table 8. Extended Erase special values, and what the STM32H503 accepts

Value	Meaning	Pulsar, dedicated I2C port	Supernova, via its I3C port
`0xFFFF`	Mass erase	Accepted, 0.05 s	Rejected
`0xFFFE`	Bank 1 erase	Accepted	Accepted, clears all 128 KB in 0.02 s
`0xFFFD`	Bank 2 erase	Accepted	Rejected

The difference between those two columns is the point. The bootloader is the same program talking to the same flash in both cases; only the controller changed. A Pulsar on its dedicated I2C port gets everything the bootloader offers, while the same commands sent from a Supernova's I3C port in legacy mode are refused.

That is the second time this document has had to make that distinction, and for the same underlying reason as Section 5.7: an I3C controller driving a legacy bus is not a drop-in substitute for an I2C controller, and the ways it differs show up as the target apparently rejecting things it accepts perfectly well from another host.

Where a mass erase is refused, the bank one value covers the same ground on a single-bank part. The reference utility tries mass erase first and falls back, reporting that it has done so. On a multi-bank part that fallback would clear only the first bank, which is why it says what it did rather than reporting a mass erase it did not perform.

For a page list, the count field is **the number of pages minus one**, not the number of pages. Erasing a single page 0 sends `00 00` for the count and `00 00` for the page. Sending the plain count erases one page more than intended.

The two interfaces on this part do not agree: the I3C interface takes the count itself where I2C takes the count minus one. Both encodings were established by writing a distinct pattern into two adjacent pages, erasing the first alone, and confirming which pages came back erased. AN0001 covers the I3C case.

How many pages one command may carry depends on the controller in the same way. A Pulsar on its dedicated I2C port has lists of four pages accepted; a Supernova driving the bus from its I3C port has anything beyond **two pages** rejected, while one or two are accepted and clear exactly the pages named. Nothing found in AN4221 or AN2606 states such a limit; it was established by trying successive lengths and confirming coverage page by page.

The reference utility therefore splits a page list only where the controller in use requires it, so a 128 KB erase from a Supernova's I3C port becomes eight commands taking about 0.19 s, and stays a single command from a Pulsar. A host that sends one long list on the constrained path will have it rejected, and the failure looks like a malformed request rather than an oversized one.

A flash page on the STM32H503 is 8 KB. The utility's `--page-size` option sets this for `--erase pages`.

5.7 Clock stretching, and why the utility waits

Immediately after a command, and before it settles into NACKing its address, the target holds SCL low briefly. A conventional I2C controller rides this out, and a Pulsar on its dedicated I2C port needs nothing further.

An I3C controller in legacy I2C mode is not obliged to. I3C targets never stretch the clock, so an I3C peripheral may enforce a bus timeout and abort the transfer instead of waiting. On a Supernova driving the bus from its I3C port, this appeared as a bus timeout on the status read following the erase page-list write, in roughly a quarter of attempts. Tracing every transfer

with the interval since the previous one showed the failing read was always the one issued with no gap at all, which is what identifies the cause as the stretch window rather than the bus.

The remedy is not a longer timeout. Waiting about 2 ms before the status read skips the stretch window entirely, so the target presents a clean NACK, which the ordinary busy-poll already handles. With that settle in place the timeout does not arise at all.

Table 9. Effect of the settle on a Supernova driving I2C from its I3C port

Strategy	Cycles completed	Timeouts absorbed
No settle	9 of 12	—
2 ms settle before status reads	12 of 12	0
2 ms settle plus retry on timeout	12 of 12	0

The third row is the evidence that the settle addresses the cause rather than masking it: with it in place, the retry never fires. The utility applies the settle automatically and only on that path, so no option controls it.

6 Programming procedure

6.1 Identifying the target

The `info` command resets the target into the bootloader, initializes the bus, and reports what responded:

```
python stm32_flash.py --bus i2c info

  Adapter: pulsar  fw 4.4.0
  Bus: I2C
  Resetting target into bootloader (NRST=GPIO1, BOOT0=GPIO2)
  I2C: 100 kHz, target 0x67
  Bootloader session established

  Protocol version : 2.0
  Device ID        : 0x474  (STM32H503)
  Commands         : 15
      0x00  Get
      ...
      0xA1  Checksum
```

The fifteen supported commands are listed in full between those lines; they are the ones given in Table 7. The adapter's serial number is printed on the first line and is omitted here.

This is the recommended first step on any new board, as it confirms wiring, bus configuration and bootloader entry before any destructive operation is attempted.

6.2 Reading the existing contents

Flash contents should be archived before an erase. The following reads the whole 128 KB flash of an STM32H503RB:

```
python stm32_flash.py --bus i2c read backup.bin --address 0x08000000 --length 0x20000
```

6.3 Programming an image

The `flash` command performs the complete sequence: erase, program, read back and compare, then reset into the application. The verification images of Section 7 are used here so that the commands can be run exactly as printed.

A raw binary requires an explicit base address:

```
python stm32_flash.py --bus i2c flash firmware/an0002_image_a.bin --address 0x08000000
```

An Intel HEX file carries its own addresses, so the option is omitted. Running `make hex` in the `firmware` directory produces one:

```
python stm32_flash.py --bus i2c flash firmware/build/an0002_image_a.hex
```

By default the whole device is mass-erased. `--erase pages` erases only the pages the image covers, which is faster for a small image:

```
python stm32_flash.py --bus i2c flash firmware/an0002_image_a.bin --address 0x08000000 --
erase pages
```

To drive a Supernova from its I3C port instead, add `--i2c-port i3c`. Session options may be given before or after the subcommand:

```
python stm32_flash.py --bus i2c --i2c-port i3c flash firmware/an0002_image_a.bin --address
0x08000000
python stm32_flash.py --bus i2c flash firmware/an0002_image_a.bin --address 0x08000000 --
i2c-port i3c
```

6.4 Verification

Verification is performed by default. The programmed range is read back over the same interface and compared byte for byte with the source image. It is disabled with `--no-verify` where programming time is critical and the image is verified by other means.

6.5 Returning the device to the application

Where GPIO 1 and GPIO 2 are connected, the utility drives `BOOT0` low and pulses `NRST` after programming, so the device starts the newly programmed application. Where those pins are not connected, the Go command is issued instead. `--no-run` leaves the device in the bootloader.

7 Confirming the update on the target

The `flash` command reads the programmed range back and compares it byte for byte, which proves that the contents of flash match the image supplied. It does not prove that the device is executing the new firmware: a target left in the bootloader, or one whose reset never released, would pass that check and still not be running the application.

Two small firmware images are supplied with this document to close that gap. They are identical except for the letter they report and the rate at which they blink the user LED, so the change is visible from outside the programming tool.

7.1 The verification images

Table 10. Verification images supplied with this document

Image	File	Reports	LED
A	`firmware/an0002_image_a.bin`	`IMAGE A`	1 Hz
B	`firmware/an0002_image_b.bin`	`IMAGE B`	5 Hz

The images are 864 and 860 bytes and are linked to run from `0x08000000`. Source, a linker script and a makefile are supplied alongside them, so the images can be inspected and rebuilt rather than taken on trust.

Neither image uses `PB3` or `PB4`, so they can be loaded and observed without disturbing the programming connections.

7.2 Procedure

Open a serial terminal on the ST-LINK virtual COM port at 115200 8N1, then program Image A:

```
python stm32_flash.py --bus i2c flash firmware/an0002_image_a.bin --address 0x08000000
```

The utility reports each stage as it goes. The output below is abridged; the adapter and session lines of Section 6.1 appear first, and the exact times vary between runs:

```
...
Erasing 1 page
  done in 0.02 s

Writing
  segment at 0x08000000
  wrote 864 bytes in 0.11 s (7.7 KiB/s)

Verifying
  all 864 bytes match (0.12 s)

Resetting into application

Success.
```

The terminal shows the banner and then one heartbeat line per second:

```
========================================
  Binho  |  AN0002
  Programming STM32 microcontrollers
  over I2C with the Binho Pulsar
========================================
  Running: IMAGE A
  Kernel clock: 32 MHz
  LED blink: 1 Hz
========================================
  IMAGE A  alive, 1 s
  IMAGE A  alive, 2 s
```

Program Image B and the reported letter changes and the LED blinks five times faster. The heartbeat matters: a banner printed only at reset is missed if the terminal is opened afterwards, and the point of these images is to tell at a glance which one is running.

The transition from `IMAGE A` to `IMAGE B` is the end-to-end result: the image was erased, programmed and verified over I2C, and the device is running what was programmed.

> **IMPORTANT** Programming these images erases the application already on the device. Archive the existing contents first, as described in Section 6.2.

> **NOTE** The supplied binaries are built for the NUCLEO-H503RB. They drive `PA4` as USART3_TX and `PA5` as the user LED, which is how that board is wired. On any other board the serial output will not appear, because the pin and the routing to the virtual COM port differ. The images are small and the source is supplied, so retarget them by changing the pin and peripheral definitions at the top of `firmware/main.c` and rebuilding.

8 Reference utility

8.1 Commands

Table 11. Utility commands

Command	Action
`list`	List connected Binho adapters
`info`	Identify the target and report its capabilities
`flash`	Erase, program, verify and run an image
`read`	Read memory into a file
`erase`	Mass-erase user flash
`reset`	Reset the target via GPIO

8.2 Options

Session options describe the adapter, the bus and bootloader entry. They may be given either before or after the subcommand. The utility also accepts groups of I3C and SPI options, which have no effect on an I2C bus; they are listed in AN0001 and AN0003.

Table 12. Session options

Option	Default	Meaning
`--adapter`	auto	Force `pulsar` or `supernova`
`--bus`	auto	Force `i2c` or `i3c`
`--serial`	auto	Select an adapter by serial number
`--i2c-address`	`0x67`	Bootloader address, per AN2606
`--i2c-speed`	100000	Bus clock in Hz
`--i2c-bus`	`A`	Which I2C bus on a Pulsar
`--i2c-port`	`dedicated`	Which Supernova connector carries I2C
`--pullup`	1000	Bus pull-up value in ohms
`--nrst-pin`	1	GPIO driving `NRST`
`--boot0-pin`	2	GPIO driving `BOOT0`
`--manual`	off	Prompt for bootloader entry instead of using GPIO
`--reset-hold`	0.05	Seconds `NRST` is held low
`--boot-delay`	0.25	Seconds allowed for the bootloader to start
`--no-stretch`	off	Force the no-stretch command variants. See Section 2.3
`-v`, `-q`	off	Log every transfer, or suppress progress output

Table 13. Per-command options

Option	Applies to	Meaning
`--address`	`flash`, `read`	Base address for a raw binary
`--length`	`read`	Number of bytes to read
`--erase`	`flash`	`mass`, `pages` or `none`. Where a mass erase is rejected, `mass` falls back to a bank one erase and says so
`--page-size`	`flash`	Flash page size for `--erase pages`, default 8192
`--erase-timeout`	`flash`, `erase`	Seconds to allow for an erase
`--chunk-size`	`flash`, `read`	Bytes per transfer. Default 256, which is the most the bootloader accepts
`--no-verify`	`flash`	Skip the read-back comparison
`--no-run`	`flash`	Leave the device in the bootloader
`--run`	`info`, `read`, `erase`	Reset into the application afterwards
`--bootloader`	`reset`	Reset into the bootloader rather than the application

9 Measured results

9.1 Test configuration

Table 14. Test configuration

Item	Value
Target	STM32H503RB, NUCLEO-H503RB
Adapter	Binho Pulsar, hardware revision C, firmware 4.4.0
Host package	`binhopulsar` 1.2.0
Python	3.12
Bus voltage	3.3 V
Pull-ups	1 kΩ

9.2 Read throughput

A 128 KiB read was timed at three bus speeds. Session setup, which covers enumeration, reset into the bootloader and the opening handshake, was 0.81 s and did not vary with bus speed; it is reported separately so the transfer figures describe the bus alone.

Table 15. Measured 128 KiB read

Bus speed	Read phase	Throughput
100 kHz	17.47 s	7.3 KiB/s
400 kHz	7.59 s	16.9 KiB/s
1 MHz	5.44 s	23.5 KiB/s

All three reads returned byte-identical data.

9.3 Reliability

Table 16. Completed erase, program, verify and run cycles

Configuration	Cycles completed
Pulsar, dedicated I2C port, 100 kHz	29 of 29
Supernova, I3C port in I2C mode, 100 kHz	12 of 12

Cycles alternated between Image A and Image B, so each one erased, programmed and verified different contents from the cycle before it. The Supernova figure is with the settle of Section 5.7 applied; without it, 9 of 12 completed.

These figures describe one target and one host over a few dozen cycles. They are evidence that the procedure is repeatable, not a reliability specification.

10 Troubleshooting

Table 17. Troubleshooting

Symptom	Probable cause and action
Controller reports it could not generate a start condition	The bus has no pull-ups, or SDA or SCL is held low. Check that the port in use supplies pull-ups and that the wiring reaches the correct pins.
No response at the bootloader address	Wrong address or wrong pins. Confirm both against AN2606 for the part; they differ per device.
Target answers only in bootloader mode	Expected. The address is present only while the device is held in system memory boot mode.
Bus timeout on the status read after an erase	The controller does not tolerate the target holding SCL. Use a Pulsar on its dedicated I2C port, or a Supernova with `--i2c-port i3c`, where the utility applies the settle of Section 5.7.
Erase removes one page more than expected	The count field carries the page count minus one. See Section 5.6.
Verify fails after an apparently successful program	The erase did not cover the whole programmed range. Widen `--page-size` to the true page size for the part, or erase explicitly before programming.
Mass erase is rejected	Seen when driving I2C from a Supernova's I3C port, not from a Pulsar's dedicated port. A bank one erase covers the same ground; the utility falls back automatically. See Section 5.6.
An erase naming several pages is rejected	Same path: more than two pages in one command is rejected there and accepted from a dedicated I2C port. The utility splits the list where required. See Section 5.6.
Reads return data but the device does not run	The device was left in the bootloader. Check that `BOOT0` returns low and `NRST` is released, or omit `--no-run`.
Garbled serial output from the verification images	The terminal baud rate does not match. The images use 115200 8N1.

11 Acronyms

Table 18. Acronyms

Term	Meaning
ACK	Acknowledge
HSI	High-speed internal oscillator
I2C	Inter-Integrated Circuit
I3C	Improved Inter-Integrated Circuit
NACK	Not acknowledged
ROM	Read-only memory
SCL	Serial clock line
SDA	Serial data line
VCP	Virtual COM port

12 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.

The archive `AN0002-assets.zip` is published at `https://cdn.binho.io/application-notes/AN0002/AN0002-assets.zip` and contains the following.

Table 19. Contents of the assets archive

Path	Description
`AN0002.md`	This document in Markdown
`assets/stm32_flash.py`	Reference utility, I2C, I3C and SPI, Pulsar and Supernova
`assets/PROTOCOL.md`	Wire-level notes on the I2C bootloader protocol
`assets/test_hex_parser.py`	Tests for the Intel HEX parser
`assets/LICENSE`	MIT license
`assets/firmware/main.c`	Source for both verification images
`assets/firmware/link.ld`	Linker script
`assets/firmware/Makefile`	Builds both images
`assets/firmware/README.md`	Notes on the verification images
`assets/firmware/an0002_image_a.bin`	Prebuilt Image A
`assets/firmware/an0002_image_b.bin`	Prebuilt Image B

13 Revision history

Table 20. Revision history

Revision	Date	Description
1.0	29 August 2026	Initial release.

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. **Arm** and **Cortex** are registered trademarks of Arm Limited. **I3C** and **MIPI** are trademarks or registered trademarks of MIPI Alliance, Inc. **STM32**, **Nucleo** and **STMicroelectronics** are trademarks or registered trademarks of STMicroelectronics N.V.