binho

APPLICATION NOTE

Single-port I2C/SMBus/I3C operation for NVMe-MI on Binho Supernova

AN0014

Document information

Keywords	AN0014, Supernova, I3C, I2C, SMBus, MCTP, clock stretching, legacy I2C, tTIMEOUT, NVMe-MI, ARP, Hot-Join, PIC18F16Q20, pyCosmicSDK
Abstract	Many SMBus and MCTP devices hold SCL low while they prepare a response. The Supernova's I3C engine cannot wait for that, so on its I3C port such a target used to lose bits. Adapter firmware 4.6.0 and pyCosmicSDK 1.6.0 add a mode in which legacy I2C transfers on the I3C port wait for the target, up to the SMBus 35 ms timeout. This note explains the mode, supplies a script that reads and writes a stretching target with it, and supplies the firmware of the reference target used to measure it: 3600 of 3600 reads correct at 100 kHz and 400 kHz over three runs, with the target stretching from 0 to 30 ms, and every stretched read correct up to the mode's ceiling of about 750 kHz. The script also runs the SMBus to I3C Basic detection flow of the PCI-SIG out-of-band management specification as far as the adapter can.

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

REVISION 1.0
6 NOVEMBER 2026

1 Introduction

An I2C target may hold SCL low after it acknowledges its address, to gain time before it answers. The controller must wait until SCL is released. The I2C specification calls this clock stretching, and SMBus allows it for up to 35 ms (tTIMEOUT). SMBus and MCTP devices use it routinely: a management controller that is fetching a value holds the clock until the value is ready.

The Supernova's I3C port can address legacy I2C targets on the same bus as I3C devices. Its I3C engine, however, keeps clocking while a target holds SCL low, so a stretching target misses bits and the transfer returns wrong data or no acknowledge. Adapter firmware 4.6.0 and pyCosmicSDK 1.6.0 add a mode that fixes this for legacy I2C transfers on the I3C port.

1.1 Scope

This document covers:

- what the clock-stretching mode changes, and what it refuses while it is on
- the SCL rates it runs at and how close they come to the request
- a script that reads and writes a stretching target through the mode
- a reference target that stretches for a configurable time, and its firmware
- measured results against that target
- the SMBus to I3C Basic detection flow used for NVMe-MI, and which of its steps the adapter performs

It does not cover SMBus framing beyond the PEC check of an ARP Get UDID (block read and write, quick command), MCTP or NVMe-MI messages, or the Supernova's separate I2C port.

1.2 Applicable devices

Table 1. Applicable devices

Role	Part	Identifier	Basis
Controller	Binho Supernova	hardware revision C, firmware 4.6.1, I3C HV port	measured, this document
Controller	Binho Supernova	firmware 4.5.1 or earlier	refuses the mode with `FW_UNSUPPORTED_COMMAND`
Controller	Binho Pulsar	any	not applicable: the Pulsar has no I3C port
Reference target	Microchip PIC18F16Q20 Curiosity Nano	firmware `stretch_target_pic18q20.c`	measured, this document

1.3 Related documents

Table 2. Related documents

Document	Title
UM10204	I2C-bus specification and user manual, Rev. 7, NXP Semiconductors
SMBus 3.2	System Management Bus Specification, version 3.2
PCI-SIG ECN	PCI-SIG Engineering Change Notice: Chapter 12. Architectural Out-of-Band Management
AN0015	Setting the I2C SCL duty cycle and an exact bus frequency with the Binho Supernova and Pulsar

2 System overview

With the mode off, `legacy_i2c_read` and `legacy_i2c_write` on the I3C port are carried out by the I3C engine, which drives SCL on its own schedule. With the mode on, the adapter's firmware clocks those transfers in software and checks SCL after every phase it releases. When a target holds SCL low, the firmware waits.

Four facts shape how the mode is used:

- **It is a mode, not a per-transfer option.** `enable_i2c_clock_stretching(frequency_hz)` turns it on for every later legacy I2C transfer; `disable_i2c_clock_stretching()` turns it off. `initialize` and `configure` also turn it off.
- **It excludes I3C traffic.** While it is on, SDR, HDR and CCC calls, `init_bus` and `reset_bus` raise `I3cError` with status `FW_I3C_I2C_CLOCK_STRETCHING_ACTIVE`. Turn it off before talking to an I3C device.
- **It enforces the SMBus timeout.** A stretch longer than 35 ms ends the transfer with `FW_BUS_TIMEOUT`, and the bus is released.
- **The rate is set by a software clock.** The call accepts 10 kHz to 1 MHz and returns the rate it computed: 97975 Hz for 100 kHz and 391644 Hz for 400 kHz, within 1 % of the wire. Above 400 kHz the returned rate and the wire differ by up to 7 %, and a 600 kHz request runs at 615 kHz (section 9.2). The clock tops out at about 750 kHz, so 800 kHz and 1 MHz requests both run there.

3 Hardware setup

3.1 Required equipment

Table 3. Required equipment

Item
Binho Supernova, firmware 4.6.0 or later
USB-C cable
An I2C or SMBus target that stretches SCL, or the reference target of section 3.4
Pull-up resistors on SCL and SDA, if the target board has none

3.2 Connections

Table 4. Connections between the Supernova I3C HV port and the target

Supernova I3C HV port	Target	Required
SCL	SCL	required
SDA	SDA	required
GND	GND	required
VTARG	target I/O supply	optional, if the target is powered from the adapter

3.3 Bus considerations

The I3C port has no pull-up on SCL. An I2C bus needs one on both lines. On our bench it came from a Binho Pulsar's I2C port set to 1 kOhm and left idle; a pair of 1 kOhm to 2.2 kOhm resistors to the bus voltage does the same job. Without a pull-up on SCL the reported rate drops sharply, because the firmware waits for every rising edge.

> **Caution:** the I3C HV port selects its voltage from `set_voltage` : 1200 mV to 3300 mV. Set it to the target's I/O voltage before connecting a target that is not 3.3 V tolerant.
>
> **Caution:** only one controller on the bus. If a second adapter supplies the pull-up, do not start transfers from it.

3.4 The reference target

To measure the mode we needed a target whose stretch we could set. `stretch_target_pic18q20.c`, in the assets archive, turns a PIC18F16Q20 Curiosity Nano into one. Its I2C peripheral holds SCL after every address match for a programmable time, which is how a busy SMBus device behaves.

Table 5. Reference target protocol, address 0x2A

Transfer	Effect
write `0xF0`, low byte, high byte	stretch that many microseconds after every later address match; 0 turns stretching off
read	returns `0xA5` for every byte
any other write	accepted and discarded

SCL and SDA are on RB6 and RB5. Build it with MPLAB XC8 and program it with MPLAB X or the Curiosity Nano's drag-and-drop programmer.

4 Software setup

Install pyCosmicSDK (latest release):

```
pip install --upgrade pycosmicsdk
```

The script `smbus_stretch.py` is in the assets archive. It needs nothing else.

5 The clock-stretching mode

5.1 Using it

The calls in isolation:

```
import pycosmicsdk as p

dev = p.Device.open(model=p.DeviceModel.SUPERNOVA)
i3c = dev.i3c()
i3c.set_voltage(voltage_mv=3300)
i3c.bring_up(
    push_pull_rate=p.I3cPushPullRate.PUSH_PULL_1_MHZ_DC_40,
    open_drain_rate=p.I3cOpenDrainRate.OPEN_DRAIN_100_KHZ,
    i2c_open_drain_rate=p.I2cOpenDrainRate.STANDARD_MODE,
)
print(i3c.enable_i2c_clock_stretching(100_000))
# I2cClockStretching(enabled=True, frequency_hz=97975)
data = i3c.legacy_i2c_read(0x2A, 16)
i3c.disable_i2c_clock_stretching()
```

A register read, the SMBus Read Byte or Read Word pattern, passes the register as `subaddress`: `i3c.legacy_i2c_read(0x0B, 2, subaddress=bytes([0x09]))`.

5.2 Status codes

Table 6. Status codes specific to the mode

Status	Raised by	Meaning
`FW_UNSUPPORTED_COMMAND`	enable, disable	firmware 4.5.1 or earlier
`FW_INVALID_PARAMETER`	enable	rate outside 10 kHz to 1 MHz
`FW_INTERFACE_NOT_INITIALIZED`	enable, disable	the I3C controller was not brought up
`FW_I3C_I2C_CLOCK_STRETCHING_ACTIVE`	SDR, HDR, CCC, `init_bus` , `reset_bus`	the mode is on; turn it off first
`FW_BUS_TIMEOUT`	legacy read or write	the target held SCL longer than 35 ms

5.3 The SMBus timeout

During firmware validation, with the reference target set to stretch 40 ms, 100 reads returned 50 `FW_BUS_TIMEOUT` and 50 `FW_I3C_NACK_ADDRESS`, the longest in 46 ms, and the adapter stayed responsive. The address NACKs come from the target, which misses the next address while it is still counting out its 40 ms.

6 Procedure

1. Connect the target and the pull-ups as in section 3.2. Power the target.
2. Read from the target. The first line shows the firmware and the rate in effect:

```
$ python smbus_stretch.py read 0x2A 16
FW-4.6.1  I3C port 3300 mV  stretching on, SCL 97975 Hz
read  0x2A: a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5
```

3. Write to it:

```
$ python smbus_stretch.py write 0x2A 0x01 0x02 0x03
FW-4.6.1  I3C port 3300 mV  stretching on, SCL 97975 Hz
write 0x2A: 01 02 03
```

4. With the reference target, sweep the stretch. For each duration the script sets the target's stretch, reads 16 bytes 100 times, and checks every byte:

```
$ python smbus_stretch.py sweep
FW-4.6.1  I3C port 3300 mV  stretching on, SCL 97975 Hz
  stretch  reads   ok  median ms  result
      0 us    100  100       6.02  all bytes correct
     50 us    100  100       6.02  all bytes correct
    500 us    100  100       7.20  all bytes correct
   2000 us    100  100       8.13  all bytes correct
  10000 us    100  100      18.01  all bytes correct
  30000 us    100  100      38.99  all bytes correct
```

5. Repeat at 400 kHz:

```
$ python smbus_stretch.py --khz 400 sweep
FW-4.6.1  I3C port 3300 mV  stretching on, SCL 391644 Hz
  stretch  reads   ok  median ms  result
      0 us    100  100       5.10  all bytes correct
     50 us    100  100       5.04  all bytes correct
    500 us    100  100       6.11  all bytes correct
   2000 us    100  100       8.25  all bytes correct
  10000 us    100  100      16.90  all bytes correct
  30000 us    100  100      37.77  all bytes correct
```

The median time per read grows with the stretch, and by about the stretch: the adapter waited each time. The script sets the target's stretch back to 0 before it exits.

7 The SMBus to I3C Basic detection flow

NVMe-MI devices that move from SMBus to I3C Basic share one pair of pins, so the host has to find out which protocol each device speaks before it changes the bus. The flow the host follows is defined in the PCI-SIG Engineering Change Notice, Chapter 12, Architectural Out-of-Band Management. The `handoff` command of `smbus_stretch.py` runs it on the Supernova I3C port. Table 7 lists the steps and who performs each.

Table 7. Detection flow steps

Step	Flow	Performed by
1	Broadcast DISEC with DISHJ to 7Eh at the SMBus voltage; ACK means an I3C Basic device is present	script. Only the 7Eh header is open-drain; the CCC byte and data are push-pull at the I3C rate. A NACK is reported as `FW_I3C_NACK_ADDRESS`
2	SMBus device discovery at static addresses and at 61h (ARP Get UDID)	script, with the clock-stretching mode on. The script checks the PEC
3	An SMBus-only device was found: reset the SMBus interface and stay in SMBus mode	script: `hold_scl_low` holds SCL low for T2wrst (`--t2wrst-us`). A device with SMRST# can be reset from a GPIO instead
4	DISEC with DISHJ again, then SMBus pull-ups off and I3C Basic pull-ups on	script for DISEC. The adapter's SDA pull-up is set with `set_sda_pull_up` (`--sda-pull-up`) for the SMBus phase and follows the I3C engine after it. SCL has no pull-up on the adapter: the external one is switched by hand
5	Wait Tsmb2i3c, move to the I3C Basic voltage, broadcast ENEC with ENHJ	script. `--t-smb2i3c-ms` and `--i3c-voltage-mv` come from the specification and the device
6	The device Hot-Joins	firmware: it assigns the dynamic address itself and the script prints it

The clock-stretching mode refuses every CCC (section 5.2), so the script turns it on for step 2 only and off again before step 4.

To see the flow on a Saleae analyzer, `capture_flow.py` starts a Logic 2 capture through its automation server, runs `handoff` with the arguments that follow `--`, and saves the capture with Logic 2's I2C analyzer and the `flow_labels` extension, which names each step on its own row. `--step-delay-s` spaces the steps apart so each one can be found on the timeline.

> **Note:** only the SMBus-only branch has been run, 6 of 6 identical on firmware 4.6.1 on the bench of section 9.1, one of them after an adapter reset: step 1 NACKed, step 2 found the reference target at 0x2A and no ARP device, and the SMBus reset held SCL low for 35000 to 35777 us when asked for 35000 us. Steps 3 to 6 need a device that ACKs 7Eh and Hot-Joins, and the ARP and PEC path needs an ARP-capable device; neither was on the bench. The PEC routine is checked against the CRC-8/SMBUS check value only.

```
$ python smbus_stretch.py --khz 400 handoff --address 0x2A 0x2B --sda-pull-up ON --
t2wrst-us 35000
FW-4.6.1  I3C port 3300 mV (SMBus voltage)
1. DISEC(DISHJ) to 7Eh: NACK, SMBus only
2. SMBus discovery, stretching on, SCL 391644 Hz
  static 0x2A: ACK
  static 0x2B: FW_I3C_NACK_ADDRESS
  ARP 0x61 Get UDID: FW_I3C_NACK_ADDRESS (no ARP-capable device)
   SMBus mode stays.
   SMBus reset: SCL held low 35059 us (asked 35000 us)
```

8 Reference

Table 8. smbus_stretch.py options

Option	Default	Meaning
`--serial`	any	Supernova serial number
`--voltage-mv`	3300	I3C port voltage, 1200 to 3300 for the HV connector
`--khz`	100	SCL rate, 10 to 1000; runs at about 750 kHz at most

Table 9. smbus_stretch.py commands

Command	Arguments	Action
`read`	address, length, optional `--register`	read, optionally after writing one register byte
`write`	address, bytes	write the bytes
`sweep`	`--address`, `--count`, `--length`	reference target only: stretch 0, 50, 500, 2000, 10000 and 30000 us, `--count` reads each
`handoff`	`--address`, `--i3c-voltage-mv`, `--t-smb2i3c-ms`, `--hj-timeout-s`, `--no-smbus-only`, `--yes`, `--t2wrst-us`, `--sda-pull-up`, `--step-delay-s`, `--pulsar-pull-up`	the detection flow of section 7

Every command enables the mode before its transfers and disables it before exiting, so the I3C port is usable for I3C traffic afterwards.

9 Measured results

9.1 Test configuration

Table 10. Test configuration

Item	Value
Adapter	Binho Supernova, hardware revision C, firmware 4.6.1 (`FW-4.6.1`). The soak, the SMBus reset and the SDA pull-up measurements of section 9.2 are from 4.6.0 pre-release builds of the same feature
Host software	pyCosmicSDK 1.6.0 from PyPI, Linux
Bus	I3C HV port, 3.3 V, 1 kOhm pull-up on SCL and SDA from an idle Binho Pulsar, firmware 4.6.1
Analyzer	Acute TravelBus logic analyzer, 800 MS/s, threshold 1.6 V
Target	PIC18F16Q20 Curiosity Nano, `stretch_target_pic18q20.c`, address 0x2A

9.2 Results

The sweeps of section 6 read 1200 times, 16 bytes each, across both rates and six stretch durations. They were run three times, the third after an adapter reset: 3600 correct, no errors.

Further results from the firmware validation of the same feature, on a pre-release build:

- Soak with random 1 to 64 byte reads and writes, the stretch changing every 100 transfers among 0, 50 us, 500 us, 2 ms and 10 ms: 0 bad reads in about 3400 stretched transfers per run at 100 kHz and 400 kHz, including a run after a power cycle.

SCL rate by request on firmware 4.6.1. Above 400 kHz, 20 reads of 16 bytes per stretch from 50 us to 30 ms:

Table 11. SCL rate by request

Request	Reported	On the wire	tLOW / tHIGH	Stretched reads
100 kHz	97975 Hz	98.2 kHz	5105 / 5075 ns	1500 of 1500
400 kHz	391644 Hz	395.3 kHz	1600 / 935 ns	1500 of 1500
600 kHz	583657 Hz	615.4 kHz	855 / 755 ns	100 of 100
800 kHz	746268 Hz	749.1 kHz	750 / 585 ns	100 of 100
1 MHz	746268 Hz	749.1 kHz	750 / 585 ns	100 of 100

On-the-wire values are medians over about 9000 SCL periods per rate. Every rate keeps tLOW and tHIGH above the UM10204 minimums for its mode. Above 400 kHz the returned rate does

not track the wire: a 500 kHz request reports 477707 Hz and runs at 467.3 kHz, 700 kHz reports 652173 Hz and runs at 696.9 kHz, and 600 kHz runs 2.6 % above the request. Measure the wire if an exact rate above 400 kHz matters.

With the reference target not stretching, reads above 400 kHz return 0xFF in the second byte, 0 of 100 at each rate, while every stretched read is correct. The adapter's SCL is the same in both cases; only the target's own clock stretching is off. We attribute it to the reference target's firmware not loading the second byte in time without stretching, not to the adapter.

SMBus reset (`hold_scl_low`), 5 holds per point, pre-release firmware `FW-4.5.1-b11a067e` , SCL low time on the Acute. Holds that span one of the analyzer's buffer boundaries read about 1.5 ms short and are left out:

Table 12. SCL hold for the SMBus reset

Requested	Measured on the wire	Reported by the adapter	Reads after the hold
1 ms	999.7 to 1678 us	1000 to 1677 us	5 of 5
10 ms	9987 to 11006 us	10000 to 10998 us	5 of 5
35 ms	34996 to 35941 us	35000 to 35937 us	5 of 5 per mode, and after a reset
100 ms	100013 to 100114 us	100000 us	5 of 5

A hold can run up to about 1 ms long, and up to 0.13 % short against the analyzer's clock, which is the tolerance of the adapter's internal oscillator; ask for T2wrst with that margin. 0 us and more than 1 s are refused with `FW_INVALID_PARAMETER` , and firmware 4.5.1 answers `FW_UNSUPPORTED_COMMAND` .

The SDA pull-up, measured as the delay from SCL rising to SDA crossing 1.6 V at each STOP with the Pulsar's pull-up off: 7813 ns with `OFF` , 5975 ns with `ON` and 5965 ns with `AUTO` . Reads were 10 of 10 in all three settings, with and without the clock-stretching mode.

These are what we measured on one bench, not a specification.

10 Troubleshooting

Table 13. Troubleshooting

Symptom	Probable cause and action
`FW_UNSUPPORTED_COMMAND` on enable	Firmware 4.5.1 or earlier. Update the Supernova to 4.6.0 or later.
Reported rate far below the request	No pull-up on SCL. Add one (section 3.3).
`FW_I3C_I2C_CLOCK_STRETCHING_ACTIVE`	An I3C call was made with the mode on. Call `disable_i2c_clock_stretching` first.
`FW_BUS_TIMEOUT`	The target held SCL longer than 35 ms. Check the target; this is SMBus tTIMEOUT.
`FW_I3C_NACK_ADDRESS` right after reconfiguring the reference target	The target restarts its I2C peripheral to apply a new stretch. The script waits 10 ms for this.
Wrong data from a stretching target with the mode off	Expected: the I3C engine does not wait. Turn the mode on.
`FW_INVALID_PARAMETER` on enable above 400 kHz	Firmware before 4.6.0's 1 MHz limit. Update the Supernova.
`FW_I3C_I2C_NACK_BYTE` in `sweep` above 400 kHz	The reference target missed the reconfiguration write while it was not stretching: 1 in 24 sweeps on our bench. Run the sweep again.
`capture_flow.py` decodes wrong addresses and data on a Saleae Logic Pro	The digital threshold is at the device default. Pass `--threshold 3.3` for a 3.3 V bus: on our Logic Pro 8 it cut SDA edges seen with SCL high from 732 to 50 in the same sweep, all of them START and STOP.
`handoff` step 1 NACK with an I3C Basic device on the bus	The device is not in SMBus mode at the voltage set, or not powered. Check `--voltage-mv`.

11 Acronyms

Table 14. Acronyms

Term	Meaning
I2C	Inter-Integrated Circuit
I3C	MIPI Improved Inter-Integrated Circuit
SMBus	System Management Bus
MCTP	Management Component Transport Protocol
SCL	Serial clock line
SDA	Serial data line
SDR	Single data rate, the base I3C transfer mode
HDR	High data rate I3C transfer modes
CCC	I3C common command code
tTIMEOUT	SMBus clock-low timeout, 35 ms
ARP	SMBus Address Resolution Protocol
PEC	SMBus packet error code, a CRC-8
UDID	SMBus unique device identifier
DISEC, ENEC	I3C CCCs that disable and enable target events
Hot-Join	I3C event in which a target joins the bus and receives a dynamic address
NVMe-MI	NVM Express Management Interface
T2wrst, Tsmb2i3c	timing parameters of the PCI-SIG detection flow

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 `AN0014-assets.zip` contains `smbus_stretch.py`, `capture_flow.py`, the `flow_labels` Logic 2 extension, `test_flow_labels.py`, `stretch_target_pic18q20.c` and `LICENSE`.

13 Revision history

Table 15. Revision history

Revision	Date	Description
0.1	25 September 2026	Draft for internal review, measured on firmware 4.6.0 and pyCosmicSDK 1.6.0 pre-release builds.
0.2	28 September 2026	Retitled for NVMe-MI. Rates up to 1 MHz (746 kHz in effect). Section 7, the SMBus to I3C Basic detection flow, and the `handoff` command.
0.3	28 September 2026	`hold_scl_low` and `set_sda_pull_up` in the detection flow, with their measurements.
0.4	30 September 2026	Measured again on firmware 4.6.1 and pyCosmicSDK 1.6.0 from PyPI. SCL rate on the wire for every request; above 400 kHz the returned rate is approximate. `--serial` works with pyCosmicSDK 1.6.0.
1.0	6 November 2026	Published.

Legal information

Definitions

Draft

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

Disclaimers

Limited warranty and liability

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

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

Right to make changes

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

Suitability for use

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

Applications

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

Third-party products and specifications

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

Example code

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

Terms and conditions of commercial sale

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

Export control

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

Security

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

Trademarks

All referenced brands, product names, service names and trademarks are the property of their respective owners. **Binho** and the Binho logo are trademarks of Binho Inc. **PIC**, **MPLAB** and **Curiosity** are trademarks or registered trademarks of Microchip Technology Incorporated. **Acute** and **TravelBus** are trademarks or registered trademarks of Acute Technology Inc. **PCI-SIG** is a registered trademark of PCI-SIG. **NVMe** and **NVMe-MI** are trademarks of NVM Express, Inc.