binho

APPLICATION NOTE

Interfacing STMicroelectronics MEMS sensors over I3C with the Binho Supernova

AN0005

Document information

Keywords	AN0005, Supernova, I3C Target Board, I3C, MIPI I3C, STMicroelectronics, LSM6DSV, LPS22DF, STEVAL-MKI239AA, STEVAL-MKI224V1A, IBI, in-band interrupt, ENTDAA, CCC, BCR, DCR, MEMS
Abstract	The current generation of STMicroelectronics MEMS sensors offers a MIPI I3C interface alongside I2C and SPI. This application note shows how to bring one up from a Binho Supernova USB host adapter and the Binho I3C Target Board, and how to exercise the I3C features the sensor implements, without writing any firmware. It covers enumeration and dynamic address assignment, identifying a part from the bus rather than a lookup table, register access, streaming decoded sensor data, in-band interrupts with no interrupt pin connected, motion-triggered interrupts, and group addressing. Worked examples use the LSM6DSV six-axis IMU and the LPS22DF pressure sensor, and a reference Python utility that produced every output in this document is supplied.

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

REVISION 1.0
1 SEPTEMBER 2026

1 Introduction

The current generation of STMicroelectronics MEMS sensors offers a MIPI I3C interface alongside I2C and SPI, on the same two pins. I3C keeps that two-wire footprint and adds a substantial feature set on top of it.

Table 1. What I3C adds to a two-wire sensor interface

Feature	What it gives a design
Dynamic address assignment	The controller hands out addresses at run time, so static address collisions stop being a board-design constraint and identical parts can share a bus
Identification from the bus	Every target reports a 48-bit provisional identifier, a device class and its bus capabilities, so a controller can enumerate a bus it was not built for
Common command codes	A standard command set, broadcast and direct, for enumeration, identification, configuration, interrupt control and reset, the same on every compliant target
In-band interrupts	The sensor interrupts the host on the data line, carrying a byte that says why, so no interrupt pin has to be routed or allocated
Group addressing	One write reaches several targets at once, addressed as a group
Higher signalling rates	Push-pull clocking, to 12.5 MHz in single data rate mode, against the open-drain limits of I2C
Lower power	Push-pull signalling avoids the static pull-up current I2C spends continuously, and activity states let a controller tell targets what bus latency to expect so they can idle harder
Hot-join	A target powered up after the bus is running can ask to be enumerated
Timing control	The controller can have targets timestamp their interrupts against a shared reference
Standard reset and recovery	A defined target reset and defined error detection and recovery, rather than per-vendor sequences
I2C compatibility	Legacy I2C devices can share the bus

A target implements the subset of that a sensor needs, and the subset differs between parts. This note works through two STMicroelectronics parts from a Binho Supernova USB host adapter and the Binho I3C Target Board: enumerating them, identifying them from the bus, reading and writing registers, streaming decoded measurements, getting interrupts over the data line with no interrupt pin connected, triggering one by tapping the board, and establishing which of the features above each target actually implements.

Nothing here requires a microcontroller, firmware, or a driver port. Every step is a command from a host PC.

1.1 Scope

By the end of this document a reader with the hardware in front of them will have enumerated a sensor onto an I3C bus, identified it from its own registers, read and written registers, streamed decoded measurements, received in-band interrupts at the sensor's configured output data rate with no interrupt pin connected, triggered an interrupt by tapping the board, and established what the target implements from observable effects.

The two parts worked through are the LSM6DSV and the LPS22DF. Both are exercised on hardware and every output in this document is a real capture.

1.2 Applicable devices

Table 2. Devices covered

Device	Function	Evaluation board	Exercised on hardware
LSM6DSV	6-axis IMU, accelerometer and gyroscope	STEVAL-MKI239AA	Yes
LPS22DF	Absolute pressure sensor	STEVAL-MKI224V1A	Yes

The method applies to any STMicroelectronics MEMS part with an I3C interface on a DIL24 adapter board. Adding one to the supplied utility is a single table entry, described in Section 11.

1.3 Related documents

Table 3. Related documents

Document	Subject
LSM6DSV datasheet	Register map and electrical characteristics
AN5922	LSM6DSV application note, including its interrupt generation
LPS22DF datasheet, DS13316	Register map, and its published CCC values
MIPI I3C Specification v1.1	Common command codes, in-band interrupts, dynamic addressing
Binho I3C Target Board product page	Sockets, jumpers, and supported adapter boards

2 Required equipment and software

2.1 Hardware

Table 4. Required equipment

Item
Binho Supernova USB host adapter
Binho I3C Target Board
STEVAL-MKI239AA or STEVAL-MKI224V1A, or another DIL24 MEMS adapter board
USB-C cable

The evaluation board drops into the labelled DIL24 socket. Nothing else is connected for any procedure in this document.

2.2 Connections

Table 5. Signals used

Signal	Purpose
SCL	I3C clock, adapter to target
SDA	I3C data, bidirectional
3.3 V	Target supply, from the adapter
GND	Return

Two wires and a supply. **No interrupt pin, reset line or strapping resistor is connected for anything in this note**, including the interrupt sections: on I3C the sensor signals the host on SDA.

The bus runs at 3.3 V throughout. The target board provides the pull-ups.

2.3 Software

```
pip install binhosupernova
```

Python 3.12 and the `binhosupernova` package are the only dependencies. Unpack the assets archive that accompanies this note and confirm the setup with one command:

```
python i3c_mems.py scan
```

3 First contact: enumerate the bus

One command puts the sensor on the bus with an address the host assigned:

```
python i3c_mems.py scan
```

```
1 target(s) on the bus at 3300 mV, PUSH_PULL_5_MHZ_50_DC, OPEN_DRAIN_1_MHZ

  dynamic address 0x08
    PID  02 08 00 70 12 0B   device id 0x0070
    BCR  0x07   DCR 0x44
    HDR  SDR only   IBI capable
    profile  lsm6dsv
```

Two common command codes did that. **RSTDAA** clears any dynamic address already assigned, so the bus starts from a known state. **ENTDAA** then runs dynamic address assignment: the controller broadcasts, every target arbitrates using its 48-bit provisional identifier, and each one is handed an address in turn. With one part present that address is `0x08`.

No address was configured anywhere. The part's own static address was not used and does not need to be known.

Everything after the address in that output came from the target rather than from a lookup table, which is the first thing that feels different from I2C. On I2C a host writes an address it picked in advance and hopes something answers. Here the device introduces itself.

4 Identify the part from the bus

```
python i3c_mems.py identify --device lsm6dsv
```

4.1 What the identity registers carry

Three registers do the introducing, and all three are read with a command rather than from the register file.

The provisional identifier (PID) is 48 bits and carries a MIPI member ID and a part number:

Table 6. Provisional identifier, per part

Field	LSM6DSV	LPS22DF
PID	`02 08 00 70 12 0B`	`02 08 00 B4 10 0B`
MIPI member id (47:33)	`0x0104`	`0x0104`
Device id (31:16)	`0x0070`	`0x00B4`
Instance id (15:12)	`0b0001`	`0b0001`

The member ID is the same for both, because both are STMicroelectronics parts. The device ID carries the same value the part reports in its `WHO_AM_I` register, so a controller that has never met the part can identify it before touching the register file.

The device characteristics register (DCR) says what class of device it is:

Table 7. Device characteristics register

Part	DCR	Meaning
LSM6DSV	`0x44`	6-axis IMU
LPS22DF	`0x62`	Pressure sensor

DCR identifies the class, not the part. `0x62` is what the MIPI registry assigns to a pressure sensor, and any compliant pressure sensor reports it. A controller reading `0x62` knows it is talking to a pressure sensor and needs the PID to learn which one.

The bus characteristics register (BCR) says what the part can do on the bus. Both parts here report `0x07`:

```
BCR 0x07
  device role                     I3C target
  advanced capabilities (bit 5)   no, so SDR only
  virtual target support          no
  offline capable                 yes
  IBI mandatory payload           yes
  IBI capable                     yes
  max data speed limit            yes
```

Two useful facts land there before anything is configured. The part can raise in-band interrupts and will send a payload when it does, which Section 7 uses. And bit 5 is clear, so the part is single-data-rate only, which answers the high-data-rate question from the target itself in one command rather than from a datasheet table.

4.2 Confirming with WHO_AM_I

Both parts also carry a conventional identity register, which is worth reading as a check that register access is framed correctly:

Table 8. Identity register

Part	Register	Value
LSM6DSV	`WHO_AM_I` at `0x0F`	`0x70`
LPS22DF	`WHO_AM_I` at `0x0F`	`0xB4`

The LPS22DF datasheet is unusually helpful here: table 15 publishes the exact bytes each of the identity commands returns. Measured against it, all eight agree, which makes it a good way to confirm a new setup end to end.

Table 9. LPS22DF, published values against measured

Command	Datasheet	Measured
GETPID	`02 08 00 B4 10 0B`	`02 08 00 B4 10 0B`
GETBCR	`07`	`07`
GETDCR	`62`	`62`
GETSTATUS	`00 00`	`00 00`
GETMXDS	`08 60`	`08 60`
GETCAPS	`00 11 18 00`	`00 11 18 00`
GETMWL	`00 08`	`00 08`
GETMRL	`00 10 05`	`00 10 05`

5 Read and write registers

Register access is a private transfer: send the register address, then read the bytes.

```
python i3c_mems.py read  --device lsm6dsv 0x0F
python i3c_mems.py write --device lps22df 0x10 0x18
```

The framing is the same on both parts:

Table 10. Register access framing

Property	LSM6DSV	LPS22DF
Register address width	8 bits	8 bits
Data width	8 bits	8 bits
Dummy bytes before a read payload	0	0

> NOTE **Set `CTRL3.IF_INC` on the LSM6DSV before any multi-byte read.** With that bit clear the register address does not advance, so a six-byte read of the output registers returns the first register six times. All three axes then read identically, which looks like a decoding fault and is not. It also means the output registers are never fully read, so data-ready never clears and a data-ready interrupt fires once and stops.

The leading `0x7E` broadcast header is optional on these parts. A read issued with it and without it returns the same value.

6 Stream sensor data

6.1 LSM6DSV, accelerometer

```
python i3c_mems.py stream --device lsm6dsv --seconds 5
```

```
  x    -0.029 g   y     0.014 g   z     1.011 g   magnitude     1.012 g
  x    -0.029 g   y     0.013 g   z     1.010 g   magnitude     1.010 g
  x    -0.028 g   y     0.013 g   z     1.011 g   magnitude     1.011 g
```

Six bytes from `OUTX_L_A` at `0x28`, three signed 16-bit axes, at 16384 counts per g on the ±2 g full scale set in `CTRL8`.

At rest the magnitude is 1 g, which is a free check that the whole chain is right: framing, byte order, sign handling and scale. Tilt the board and the axes redistribute while the magnitude stays put.

6.2 LPS22DF, pressure and temperature

```
python i3c_mems.py stream --device lps22df --seconds 5
```

```
  pressure  1006.297 hPa   temperature    22.520 C
  pressure  1006.295 hPa   temperature    22.520 C
  pressure  1006.369 hPa   temperature    22.520 C
```

Pressure is 24 bits at 4096 counts per hPa from `PRESS_OUT_XL` at `0x28`; temperature is 16 bits at 100 counts per °C from `TEMP_OUT_L` at `0x2B`. About 1006 hPa and 22 °C is right for an indoor bench.

The observable for this part needs no instrument: breathe on it or lift it, and the pressure moves about 12 Pa per metre of altitude.

6.3 Output data rate

Table 11. Output data rate selection

Part	Register	Field	Values used here
LSM6DSV	`CTRL1` at `0x10`	bits 3:0	`0x03` 15 Hz, `0x04` 30 Hz, `0x05` 60 Hz, `0x06` 120 Hz
LPS22DF	`CTRL_REG1` at `0x10`	bits 6:3	`0011` 10 Hz, `0111` 100 Hz

The rate matters in Section 7, because the interrupt rate follows it.

7 Interrupts over the bus, with no interrupt pin

In-band interrupts are the feature that changes a sensor design most visibly. The part tells the host new data is ready over the same two wires that carry the data, and the host learns why before reading anything, so no interrupt pin is routed, allocated or brought out to a connector.

7.1 Three steps

Every in-band interrupt in this note is the same three steps, in this order:

1. **Enable the source in the sensor.** Data-ready, or a motion feature.
2. **Route it to the in-band interrupt.** Which register does this is per part, and is where most of the difficulty lives.
3. **Send ENEC with its ENINT bit.** Until this arrives the bus stays silent no matter what the sensor is doing.

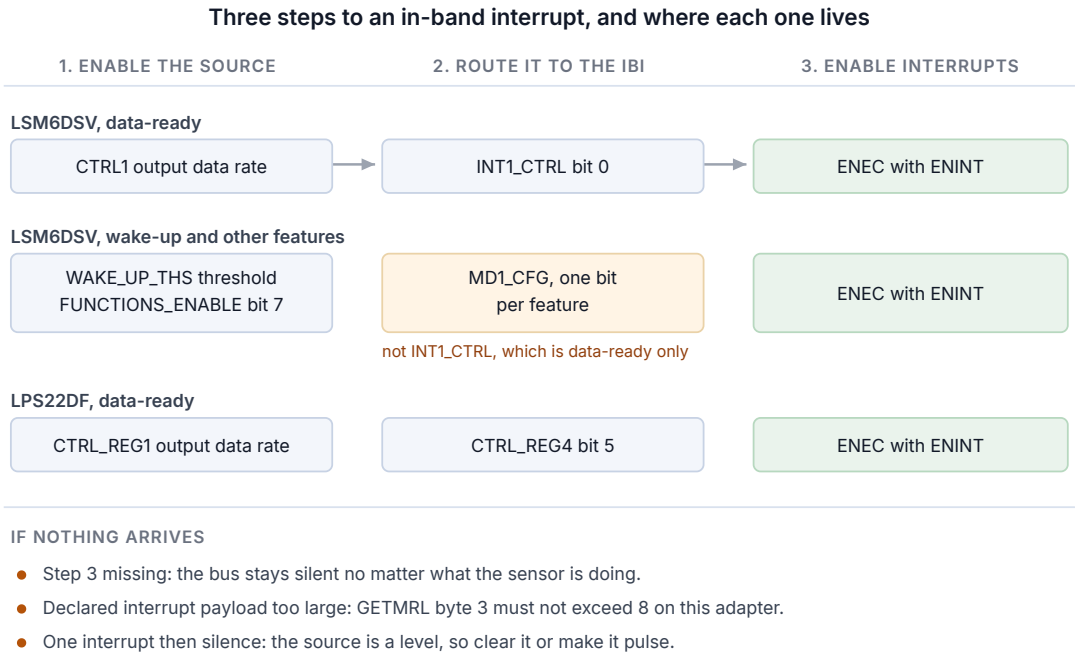

Figure 1. Three steps to an in-band interrupt, and which register carries each one on each part.

```
python i3c_mems.py ibi --device lsm6dsv --seconds 3
```

```
lsm6dsv at 0x08: interrupt routed to the I3C IBI, data-ready through INT1_CTRL, cleared by
reading the sample
  IBI from 0x08  MDB 02  extra 00 00 00 05 04 00 00  data ready=True
  IBI from 0x08  MDB 02  extra 00 00 00 05 04 00 00  data ready=True

  183 interrupts in 3.00 s = 61.00/s
  configured output data rate 60 Hz (+1.7%)
```

Sixty-one per second against a configured 60 Hz, with nothing connected to either interrupt pin.

Table 12. Interrupt rate against configured output data rate

Part	Configured	Measured	Window
LSM6DSV	15 Hz	15.67 per second	3.0 s
LSM6DSV	30 Hz	32.00 per second	3.0 s
LSM6DSV	60 Hz	61.00 per second	3.0 s
LSM6DSV	120 Hz	121.67 per second	3.0 s
LPS22DF	10 Hz	9.62 per second	3.0 s

> **NOTE** the rate is comparable with the configured rate. Individual arrival times are not, because the host transport delivers interrupts in batches. No jitter or latency figure is given in this document for that reason.

7.2 The mandatory data byte

Each interrupt carries a mandatory data byte saying what fired, so a host can decide whether to read the sensor at all:

Table 13. Mandatory data byte observed

Part	Source	Mandatory byte	Full payload
LSM6DSV	Data-ready	`0x02`	`02 00 00 00 05 04 00 00`
LSM6DSV	Wake-up	`0x04`	`04 00 00 02 05 04 00 00`
LPS22DF	Data-ready, pulsed	`0x02`	`02 00 00 33 00`
LPS22DF	Data-ready, level	`0x02`	`02 00 00 03 00`

On both parts the payload carries more than the mandatory byte. On the LSM6DSV byte 3 is `ALL_INT_SRC`, which is `0x02` for wake-up and `0x00` for data-ready, so the host can tell a motion event from a new sample without a register read. On the LPS22DF byte 3 is the `STATUS` register: the two payloads above differ because in the pulsed case nothing reads the sample, so the overrun bits set and the byte reads `0x33`, while in the level case the host reads the sample after each interrupt and it stays `0x03`.

7.3 Routing, per part

LSM6DSV. Data-ready is routed by `INT1_CTRL` at `0x0D`, bit 0 `int1_drdy_xl`. Every other interrupt function is different: it must be enabled globally by `FUNCTIONS_ENABLE` at `0x50` bit 7, and routed through `MD1_CFG` at `0x5E`, not `INT1_CTRL`.

Table 14. LSM6DSV interrupt routing

Source	Enable	Route
Accelerometer data-ready	`CTRL1` output data rate	`INT1_CTRL` bit 0
Wake-up, tap, 6D, free-fall, activity	`FUNCTIONS_ENABLE` bit 7	`MD1_CFG`, one bit per feature

`CTRL5` bit 0 `INT_EN_I3C` is not part of this. It activates the physical interrupt pins while I3C is in use, and has no effect on the in-band interrupt.

LPS22DF. Data-ready is `CTRL_REG4` at `0x13`, bit 5. Enabling it alone delivers one interrupt and then silence, because data-ready is a level that stays asserted until the sample is read. Two fixes work and agree to the sample:

Table 15. LPS22DF data-ready, level against pulsed

Configuration	Interrupts in 3 s	Rate
`DRDY` only, sample not read	1	0.33/s
`DRDY`, sample read after each interrupt	29	9.67/s
`DRDY` with `DRDY_PLS`, bit 6	29	9.62/s

> NOTE **One interrupt and then nothing is the most common way an in-band interrupt setup fails, and it is not a bus problem.** Either the condition is still asserted and the host has to clear it, or the source can be made to pulse instead.

7.4 Tap the board and watch the interrupt arrive

The demonstration worth running. Wake-up detection on the LSM6DSV turns physical motion into a bus event:

```
python i3c_mems.py wake --device lsm6dsv --seconds 25
```

```
lsm6dsv at 0x08: wake-up armed at threshold 2, routed to the I3C in-band interrupt

  TAP OR TILT THE BOARD. Listening for 25 s.

  t=  9.67s  MDB 04  payload says WU   axes X
  t=  9.91s  MDB 04  payload says WU   axes X,Y,Z
  t= 10.36s  MDB 04  payload says WU   axes X,Y
  t= 11.05s  MDB 04  payload says WU   axes Y
  t= 13.17s  MDB 04  payload says WU   axes Z

  80 wake-up interrupt(s) in 90.2 s
  peak deviation from 1 g during the window: 0.428 g
```

The threshold is in units of full scale over 64, so at ±2 g one count is about 31 mg. Two counts sits above the noise of a still board and below a deliberate tap: over a 35 second window with the board untouched, this configuration produced nothing.

Which feature fired comes from the payload and needs no register access. The per-axis bits live in `WAKE_UP_SRC` at `0x45`, and with the interrupt latched and re-triggering they can be cleared before the host reads them, so a blank axis field is normal and does not mean the interrupt was spurious.

7.5 Interrupt payload size

One setup step is silent when it is wrong, and presents as no interrupts at all.

A target declares the maximum interrupt payload it will send in the third byte of `GETMRL`, and the controller has its own limit. **The Supernova delivers interrupt payloads of up to eight bytes.** The LSM6DSV declares ten and pads its payload to whatever it has declared, so until it is asked for eight or fewer, none of its interrupts are delivered.

Table 16. LSM6DSV declared interrupt payload against interrupts delivered

Declared payload	Interrupts in 2 s	Payload delivered
1	41	1 byte
4	41	4 bytes
8	42	8 bytes
9	0	none
10, the reset value	0	none

The supplied utility checks this and adjusts it, reporting what it did:

```
IBI payload negotiation   the target declared 10 byte(s), the adapter takes 8, so it was
                          asked for 8
```

Engaging I3C timing control with SETXTIME adds a three-byte timestamp ahead of the target's own payload, and it counts against the same budget. A part sitting at exactly eight bytes therefore goes quiet once timing control is on, until the declared payload is lowered by three.

> **NOTE** **If interrupts never arrive and every register says the source is firing, read the third byte of `GETMRL` first.**

8 Exercising the rest of the I3C feature set

The utility establishes what a target implements from observable effects rather than from result codes, because a target may acknowledge a command it does not implement. Each row below required an observed change: a value set and read back, an address moved and the old one confirmed to stop answering, interrupts counted against a configured rate.

```
python i3c_mems.py features --device lsm6dsv
```

Table 17. Measured common command code support

Command	LSM6DSV	LPS22DF	Evidence
ENTDAA	Supported	Supported	Target enumerated with the expected identifier
RSTDAA	Supported	Supported	Table cleared, old address then refuses
SETNEWDA	Supported	Supported	Address moved, old address refuses
GETPID, GETBCR, GETDCR	Supported	Supported	Returned the expected values
ENEC, DISEC	Supported	Supported	Interrupts start and stop, repeatably
SETMWL, GETMWL	Supported	Supported	Value tracks what was written
SETMRL, GETMRL	Supported	Supported	As above, including the interrupt payload byte
SETGRPA, RSTGRPA	Supported	Supported	A write through the group address lands only while assigned
SETXTIME, GETXTIME	Supported	Supported	A reported byte changed after the command
HDR modes	Not supported	Not supported	BCR bit 5 clear
Hot-join	Not observed	Not observed	No hot-join request at any point
ENTAS0 to ENTAS3	Undetermined	Undetermined	Needs a current measurement
RSTACT	Undetermined	Undetermined	No observable effect from the host
GETMXDS	Undetermined	Undetermined	Answers, no observable effect

Both parts report 15 supported, 4 not implemented and 8 undetermined, and each was measured repeatedly to confirm the counts are reproducible.

Group addressing is implemented on both of these parts, and it is worth its own note because it is a genuinely useful feature and because the obvious way to test it gives the wrong answer. A group address lets a controller address several targets at once, so it is a write destination: a read from it has no defined answer, and a target may refuse a group-address read while implementing the feature correctly. The test used here assigns a group address, writes a register through it, and reads that register back at the target's own dynamic address:

Table 18. Group addressing, LSM6DSV

Step	Write through group `0x20`	Register read back
Before SETGRPA	`I3C_NACK_ADDRESS`	unchanged
After SETGRPA	`SUCCESS`	changed
After RSTGRPA	`I3C_NACK_ADDRESS`	unchanged

A read at the group address is refused at every stage, including while group addressing is demonstrably working.

Activity states stay undetermined. ENTAS0 is acknowledged and ENTAS1 to ENTAS3 are refused on the LSM6DSV, but not on every attempt, and proving that a part entered a low-activity state needs a current measurement this setup cannot make. The utility reports how many of several attempts were refused rather than presenting one result code as a fact.

9 Bus rates

```
python i3c_mems.py rates --device lsm6dsv --iterations 100
```

Table 19. Highest error-free bus rate

Part	Push-pull	Open drain
LSM6DSV	12.5 MHz at 50% duty cycle	4.17 MHz
LPS22DF	12.5 MHz at 50% duty cycle	4.17 MHz

Both parts pass every rate the adapter offers. **Method:** 100 consecutive reads of the identity register must all return the expected value with no error, and a rate passes only if every iteration succeeds. This is a register read loop, not a throughput measurement, and no conclusion about sustained data rate should be drawn from it.

Both parts set BCR bit 0, declaring a maximum data speed limitation, and GETMXDS returns `08 60` on both. That limitation did not bite for this loop at any rate the adapter can produce.

10 Practical notes

Short list of the things that cost bench time.

Table 20. Practical notes

Part	Note
LSM6DSV	Set `CTRL3.IF_INC` before any multi-byte read, or every byte comes from the same register
LSM6DSV	Interrupt features other than data-ready need `FUNCTIONS_ENABLE` bit 7 and `MD1_CFG`, not `INT1_CTRL`
LSM6DSV	Declared interrupt payload is 10 bytes at reset; lower it to 8 or no interrupt is delivered
LPS22DF	Data-ready is a level. Read the sample after each interrupt, or set `DRDY_PLS`
LPS22DF	The static address is `0x5C`, or `0x5D` with SDO high, as its section 7.2 gives it
Both	The first read after enumeration may not be settled. Issue a dummy read, as the datasheets advise
Both	SETXTIME latches a mode that changes the interrupt payload. `SETXTIME 0xFF` turns it off

11 The utility

Table 21. Utility commands

Command	Purpose
`scan`	Enumerate the bus and report what answered
`identify`	Decode a target's identity registers
`read`, `write`	Read or write a register, with a read-back after a write
`stream`	Print decoded samples by polling
`ibi`	Route the sensor interrupt onto the bus and count what arrives
`wake`	Arm a motion interrupt and report each one that arrives
`features`	Establish what the target implements, from observable effects
`rates`	Find the highest error-free bus rate
`reset`, `power-cycle`	Return the part to a known state

Adding a device is one entry in the profile table: the register framing, the expected identity, the writes that start a data stream, the writes that route an interrupt, and a decoder for the interrupt payload.

12 Measured results

12.1 Test configuration

Table 22. Test configuration

Item	Value
Adapter	Binho Supernova, hardware revision B, firmware 4.4.0
SDK	`binhosupernova`, Python 3.12, Windows 11
Accessory	Binho I3C Target Board
Bus	3.3 V, push-pull 5 MHz at 50% duty cycle, open drain 1 MHz, fast-mode drive strength
Enumeration	RSTDAA followed by ENTDAA
Targets	LSM6DSV on STEVAL-MKI239AA, LPS22DF on STEVAL-MKI224V1A

12.2 Results

Identity. Every value in Section 4 is as tabulated. All eight of the LPS22DF's published CCC values match measurement byte for byte.

Interrupts. Rates as tabulated in Section 7.1, each within 2.3% of the configured output data rate. Motion-triggered interrupts on the LSM6DSV: 80 over a 90 second window, all within the period the board was being tapped, peak acceleration 0.428 g.

Support. 15 supported, 4 not implemented and 8 undetermined on both parts, reproducible across repeated runs.

Bus rates. Both parts pass every rate the adapter offers, by the method stated in Section 9.

Regression coverage. The commands shown in this document are checked against recorded output for both parts, 8 for the LPS22DF and 7 for the LSM6DSV, so a change in the utility that alters what a reader would see is caught.

13 Troubleshooting

Table 23. Troubleshooting

Symptom	Check
No target enumerates	Board seated in the correct socket; 3.3 V present; try `power-cycle`
No interrupts arrive at all	The declared interrupt payload, third byte of `GETMRL`. Then that ENEC was sent
One interrupt then silence	The source is a level and needs clearing, or can be set to pulse
All axes read the same value	`CTRL3.IF_INC` is clear on the LSM6DSV
Interrupt payload grew and interrupts stopped	Timing control is engaged; `SETXTIME 0xFF`
Identity reads wrongly once, then correctly	Expected after enumeration; issue a dummy read first
Enumeration fails after swapping boards	Power-cycle the target rail, then enumerate again

14 Acronyms

Table 24. Acronyms

Term	Meaning
BCR	Bus characteristics register
CCC	Common command code
DAA	Dynamic address assignment
DCR	Device characteristics register
ENTDAA	Enter dynamic address assignment
HDR	High data rate
IBI	In-band interrupt
MDB	Mandatory data byte
ODR	Output data rate
PID	Provisional identifier
SDR	Single data rate

15 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.

Table 25. Contents of the assets archive

File	Description
`AN0005.md`	Markdown source of this document
`assets/i3c_mems.py`	Reference utility
`assets/LICENSE`	MIT license covering the example code

16 Revision history

Table 26. Revision history

Revision	Date	Description
1.0	1 September 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. **I3C** and **MIPI** are trademarks or registered trademarks of MIPI Alliance, Inc. **STMicroelectronics** is a trademark or registered trademark of STMicroelectronics N.V.