Host Interface Protocol Specification

Introduction

Starting from Android KitKat, there has been a transition of sensor processing from Application Processor (AP) to dedicated low-power microcontroller that manages all the sensors and wakes up the AP only if there is need for it. These so called sensor-hubs can be simple data buffering engines for the sensors or capable of running complex algorithms using raw sensor data to provide 9-axis fusion based rotation vectors, step detection/count and always on context-aware capabilities. As the capabilities of sensor-hubs increases, so does the complexity of the communication between the hub and the host processor.

This document introduces a capable communication protocol between a sensor-hub and host processor (AP). It allows for passing simple sensor reading to complex algorithm results to the AP using any underlying physical link layer communication such as I2C, SPI, UART, SLIMBus, etc. The packet format is meant to be self-describing so that the receiver will know how to decode the packets easily. It must however be kept in mind that an implementation may not use or support all the capabilities of this packet protocol. The supported packet types and format specific information must be known to the embedded sensor-hub and host driver/ software that handle the packet protocol. Thus this protocol (unlike higher level USB protocols such as HID) is not meant to be plug-and-play since the hub-host interaction is tightly coupled to a mutually agreed upon subset of this specification that also depends on the sensors present in the system and the capabilities of the hub itself.

Example Scenario

The following figure shows a sensor-hub connected to the AP via SPI interface. In addition to the communication bus there is also an interrupt output from the sensor-hub to AP for the purpose of waking up the AP when sensor-hub has an event or data to report. We will use this example setup to demonstrate some of the use case scenarios for the packet protocol.

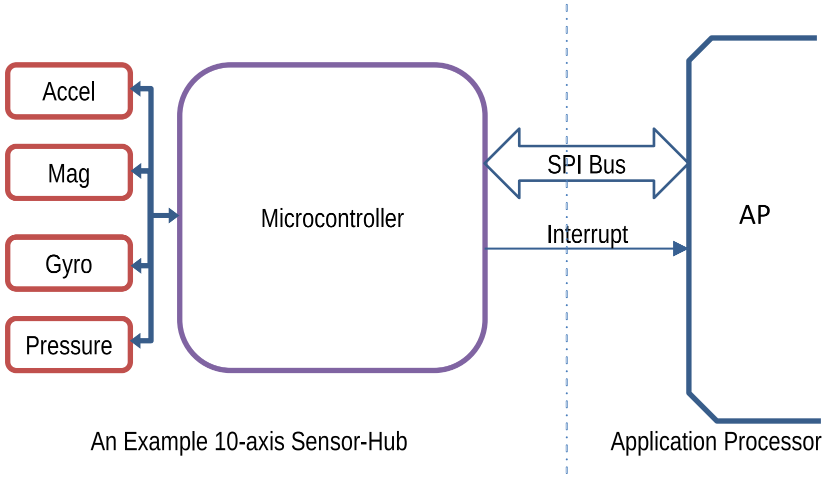


Figure 1

Sensor Types and Sensor Sub-types

OSP introduces the concept of sensor sub-types for vendor/application defined sensors in addition to the Android supported sensor types. Android allows defining manufacturers private sensor types. The sub-type introduced here adds flexibility to these private sensor types by allowing related sensor results or events to be defined as sub-type for the base sensor or result. The need for it becomes clear once we understand the need for making this distinction. In our example setup, the sensor-hub may provide Context Aware information for user posture in the private (vendor defined) sensor type of CONTEXT_POSTURE while the sub-type would define WALKING, STANDING, SITTING, etc. Similarly, in situations where the sensor-hub can only provide partial results that are not suitable for publishing to Android, it may use the private types and sub-types for sensors to convey this information to the higher algorithm running on AP.

Host Interface Protocol Packets

This protocol defines 5 basic packet types:

1. Sensor Data Packet
2. Sensor Control Read Request Packet
3. Sensor Control Write Request Packet
4. Sensor Control Response Packet
5. Sensor Test Data Packet (mostly identical to Sensor Data Packet)

These packets are described in details below.

Sensor Data Packet

Sensor Data Packet carries the sensor values reported by the sensor hub to the host. It can be a minimum of 5 bytes and maximum of 269 bytes depending on the various options as described below. Note that it is possible to have a packet which has 0 data payload. Example of such a packet is the time-sync packet where the (private) sensor type is specified to be SYSTEM_REAL_TIME_CLOCK and is used to synchronize between hub and host time or notify time counter rollover event.

Control Byte[4]	Sensor ID Byte[4]	Attribute Byte[4]	Time Stamp (4/8 bytes)[1]	Data Payload (0-N bytes)[2]	Check Sum (0/2 bytes)[3]

Notes:

1. Time Stamp is always present and can be either 32-bit or 64-bit depending on the **TF** bit setting.
2. Data payload depends on **Data Size** selection and **Sensor Type** specified.
3. Check Sum is optional and if desired, the **CS** bit must be set.
4. First 3 bytes are always present in the Sensor Data Packet and specify the expected size of the other fields in addition to sensor attributes. These bytes are detailed below.

Control Byte						Sensor Identifier Byte		Attribute Byte			
Ver	Pkt ID	CS	DF	TF	A/P	Meta Data	Sensor Type	Sensor Sub Type	F	Data Sz	TSz

	Byte 1: Control Byte
Bit 7 Ver	**Version** 0 = Current Version 1 = For future extension
Bit 6:4 Pkt ID	**Packet Identifier** **000 = Sensor Data Packet** 001 = Sensor Control Read Request 010 = Sensor Control Write Request 011 = Sensor Control Response 100 = Sensor Test Data Packet 101-111 = Unused – for later use
Bit 3 CS	**Check Sum** 0 = No Check sum used in the packet 1 = 16-bit CRC Checksum used
Bit 2 DF	**Data Format** 0 = Raw counts 1 = Application defined units (typically Android specific) for the sensor type specified.
Bit 1 TF	**Time Format** 0 = Raw counts of time stamp base on the platform 1 = Fixed Point format specified in seconds
Bit 0 A/P	**Android Sensor Type / Private Sensor Type** 0 = Sensor Type as per Android enumeration 1 = Vendor/User defined Sensor Type enumeration
	Byte 2: Sensor Identifier Byte
Bit 7:6 Meta Data	**Meta Data** Sensor specific meta-data. E.g. SENSOR_STATUS_ACCURACY_xxx defined for Orientation Sensor in Android.
Bit 5:0 Sensor Type	**Sensor Type** **A/P = 0:** 0-63 = Unique enumerator for sensors (including results derived from physical sensors) supported & defined by Android **A/P = 1:** Any vendor and/or application specific sensors/results types that is not meant to be published directly to Android SDK.
	Byte 3: Attribute Byte
Bit 7:4 Sub Type	**Sensor Sub-type** **A/P = 1:** 0-15 = Vendor or Application specific sensor dependent values. Can be used to identify a sub-set of result/types that are related to the base sensor type. E.g. IN_POCKET can be a subtype for CONTEXT_CARRY and ROLLOVER_EVENT can be a subtype for SYSTEM_REAL_TIME_CLOCK **A/P = 0:** Unused
Bit 3 F	**Flush Status** 0 – Don't care 1 – Flush Complete Event for the sensor
Bit 2:1 Data Sz	**Data Size** 00 = 8-bit 01 = 16-bit 10 = 32-bit 11 = 64-bit
Bit 0 TF	**Time Stamp Size** 0 = 32-bit Time Stamp used in the packet 1 = 64-bit Time Stamp in the packet

Sensor Control Read/Write Request Packet

Sensor Control Request packets are typically sent from Host to Hub and are meant for specifying sensor behavior or attributes such as enable, data rate, bias or offset settings etc. They can also be used to query the existing values of such parameters if applicable.

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
Ver	Pkt ID	CS[1]	DF	A/P	Meta Data	Sensor Type	Sensor Sub Type	Req. Seq. Number

Attribute Byte 2		
Parameter ID	Data Payload (0-N bytes)	Check Sum[1]

Notes:

1. Check Sum is optional and if desired, the CS bit must be set.

	Byte 1: Control Byte
Bit 7 Ver	**Version** 0 = Current Version 1 = For future extension
Bit 6:4 Pkt ID	**Packet Identifier** 001 = Sensor Control Read Request 010 = Sensor Control Write Request
Bit 3 CS	**Check Sum** 0 = No Check sum used in the packet 1 = 16-bit CRC Checksum used
Bit 2:1 DF	**Data Format** 00 = Integer (Raw counts) 01 = Fixed Point (in Application defined units) 10 = Single Precision Floating Point (in Application defined units) 11 = Double Precision Floating Point (in Application defined units)
Bit 0 A/P	**Android Sensor Type / Private Sensor Type** 0 = Sensor Type as per Android enumeration 1 = Vendor/User defined Sensor Type enumeration
	Byte 2: Sensor Identifier Byte
Bit 7:6 Meta Data	**Meta Data** Sensor specific meta-data. When unused, set to 0
Bit 5:0 Sensor Type	**Sensor Type** **A/P = 0:** 0-63 = Unique enumerator for sensors (including results derived from physical sensors) supported & defined by Android **A/P = 1:** Any vendor and/or application specific sensors/results types that is not meant to be published directly to Android SDK.
	Byte 3: Attribute Byte 1
Bit 7:4 Sub Type	**Sensor Sub-type** **A/P = 1:** 0-15 = Vendor or Application specific sensor dependent values. Can be used to identify a sub-set of result/types that are related to the base sensor type. E.g. IN_POCKET can be a subtype for CONTEXT_CARRY. **A/P = 0:** Unused
Bit 3:0 Req. Seq. Number	**Request Sequence Number** Sequential integer value set by host handler that allows it to distinguish between multiple responses. Hub handler should use the request sequence number in its response packet for that request. A sequence number of 0 is used to suppress response.

	Byte 4: Attribute Byte 2
Bit 7:0 Parameter ID	**Parameter Identifier** Enumerated parameter identifier for sensor attributes that can be read and/or written such as range, enable/disable, data rate, bias, filter settings, etc.

Sensor Control Response Packet

Sensor Control Response packets are typically sent from sensor hub to host in response to a request packet. The hub handler must ensure that the sequence number of a response matches with the corresponding request.

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
Ver	Pkt ID	CS[1]	DF	A/P	Meta Data	Sensor Type	Sensor Sub Type	Resp. Seq. Number

Attribute Byte 2		
Parameter ID	Data Payload (0-N bytes)	Check Sum[1]

Notes:

1. Check Sum is optional and if desired, the CS bit must be set.

	Byte 1: Control Byte
Bit 7 Ver	**Version** 0 = Current Version 1 = For future extension
Bit 6:4 Pkt ID	**Packet Identifier** **011 = Sensor Control Response**
Bit 3 CS	**Check Sum** 0 = No Check sum used in the packet 1 = 16-bit CRC Checksum used
Bit 2:1 DF	**Data Format** 00 = Integer (Raw counts) 01 = Fixed Point (in Application defined units) 10 = Single Precision Floating Point (in Application defined units) 11 = Double Precision Floating Point (in Application defined units)
Bit 0 A/P	**Android Sensor Type / Private Sensor Type** 0 = Sensor Type as per Android enumeration 1 = Vendor/User defined Sensor Type enumeration
	Byte 2: Sensor Identifier Byte
Bit 7:6 Meta Data	**Meta Data** Sensor specific meta-data. When unused, set to 0
Bit 5:0 Sensor Type	**Sensor Type** **A/P = 0:** 0-63 = Unique enumerator for sensors (including results derived from physical sensors) supported & defined by Android **A/P = 1:** Any vendor and/or application specific sensors/results types that is not meant to be published directly to Android SDK.

	Byte 3: Attribute Byte 1
Bit 7:4 Sub Type	**Sensor Sub-type** **A/P = 1:** 0-15 = Vendor or Application specific sensor dependent values. Can be used to identify a sub-set of result/types that are related to the base sensor type. E.g. IN_POCKET can be a subtype for CONTEXT_CARRY. **A/P = 0:** Unused
Bit 3:0 Resp. Seq. Number	**Response Sequence Number** Integer value provided in the request packet from host that allows it to distinguish between multiple responses. Hub handler should use the request sequence number in its response packet from the corresponding request. Response sequence number cannot be 0
	Byte 4: Attribute Byte 2
Bit 7:0 Parameter ID	**Parameter Identifier** Enumerated parameter identifier for sensor attributes that can be read and/or written such as range, enable/disable, data rate, bias, filter settings, etc.

Sensor Test Data Packet

Sensor Test Data packet is used to feed externally captured or simulated data into the sensor hub for the purpose of testing and validation of the algorithms running on it. This format is similar to the Sensor Data packet format with a different packet id.

Control Byte[4]	Sensor ID Byte[4]	Attribute Byte[4]	Time Stamp (4/8 bytes)[1]	Data Payload (0-256 bytes)[2]	Check Sum (0/2 bytes)[3]

Notes:

1. Time Stamp is always present and can be 32-bit or 64-bit depending on the **TF** bit setting.
2. Data Payload depends on **Data Size** selection and **Sensor Type** specified.
3. Check Sum is optional and if desired, the **CS** bit must be set.
4. First 3 bytes are always present in the Sensor Data Packet and specify the expected size of the other fields in addition to sensor attributes. These bytes are detailed below.

Control Byte						Sensor Identifier Byte		Attribute Byte			
Ver	Pkt ID	CS	DF	TF	A/P	Meta Data	Sensor Type	Sensor Sub Type	Un	Data Sz	TSz

	Byte 1: Control Byte
Bit 7 Ver	**Version** 0 = Current Version 1 = For future extension
Bit 6:4 Pkt ID	**Packet Identifier** **100 = Sensor Data Packet**
Bit 3 CS	**Check Sum** 0 = No Check sum used in the packet 1 = 16-bit CRC Checksum used
Bit 2 DF	**Data Format** 0 = Raw counts 1 = Application defined units (typically Android) for the sensor type specified.
Bit 1 TF	**Time Format** 0 = Raw counts of time stamp base on the platform 1 = Fixed Point format specified in seconds
Bit 0 A/P	**Android Sensor Type / Private Sensor Type** 0 = Sensor Type as per Android enumeration 1 = Vendor/User defined Sensor Type enumeration

	Byte 2: Sensor Identifier Byte
Bit 7:6 Meta Data	**Meta Data** Sensor specific meta-data. E.g. SENSOR_STATUS_ACCURACY_xxx defined for Orientation Sensor in Android.
Bit 5:0 Sensor Type	**Sensor Type** **A/P = 0:** 0-63 = Unique enumerator for sensors (including results derived from physical sensors) supported & defined by Android **A/P = 1:** Any vendor and/or application specific sensors/results types that is not meant to be published directly to Android SDK.
	Byte 3: Attribute Byte
Bit 7:4 Sub Type	**Sensor Sub-type** **A/P = 1:** 0-15 = Vendor or Application specific sensor dependent values. Can be used to identify a sub-set of result/types that are related to the base sensor type. E.g. IN_POCKET can be a subtype for CONTEXT_CARRY. **A/P = 0:** Unused
Bit 3 Un	**Unused**
Bit 2:1 Data Sz	**Data Size** 00 = 8-bit 01 = 16-bit 10 = 32-bit 11 = 64-bit
Bit 0 TSz	**Time Stamp Size** 0 = 32-bit Time Stamp used in the packet 1 = 64-bit Time Stamp in the packet

Byte Ordering of Data

Proposed byte ordering for the data payload and time stamps is Big Endian which calls for MSB to be sent out first. The reasoning for this choice is explained below.

Most platforms running embedded fusion or Android use ARM processors that have alignment requirements for its data types (i.e. 16-bit data must start at even address, 32-bit data must start at addresses whose first two bits are 0 and so on.). Since the packet format is a byte stream the alignment for the data payload cannot be guaranteed and so keeping the byte ordering as Little Endian to allow for structure mapping does not help in any manner. It might actually cause bugs to creep in if the implementation is not carefully done.

CRC Calculation

CRC-16 is calculated over the length of the packet keeping the 2-bytes of CRC-16 placeholder at 0x00 value. The Calculated CRC value is then inserted in its place. CRC polynomial is the following:

- CRC-16-CCITT $(x^{16} + x^{12} + x^5 + 1)$

Parameters

The following table is the list of parameters used for the Request and Response commands

ID (Hex)	Name	Data Type	Access	Description {Packet Size in bytes without checksum}
0x00	Error Code in Data	Int-32	R[1]	Data payload contains error code (int32_t). Only applicable for response packet in which non-zero sequence number was specified.{4+4}
0x01	Enable	Bool	W[2]	Enable/Subscribe request for sensor. {4+1}
0x02	Batch	Uint-64[2]	W	▪ [0] Sensor sampling period ▪ [1] Maximum report latency as requested by host. {4+16}
0x03	Flush	-	W	Flush FIFO request from host. {4}
0x04	Range & Resolution	FixP-32[2]	R	▪ [0] Max range of the sensor. Same units as reported values. {4+4} ▪ [1] Sensor resolution. Same units as reported values. {4+8}
0x05	Power	FixP-32	R	Power cost of enabling the sensor (mA). This includes the average power consumed by the associated software. {4+4}
0x06	Min-Max Delay	Int-32[2]	R	▪ [0] Min Delay: Sampling period corresponding to the fastest rate supported by the (continuous) sensor (µs). For on-change and special reporting mode sensors this is always 0. For one-shot sensors, it is -1. ▪ [1] Max Delay: For continuous and on-change sensors, the sampling period, in microseconds, corresponding to the slowest rate the sensor supports (µs). For special and one-shot sensors it is reported as 0. {4+8}
0x07	FIFO Reserved Event & Max Event Count	Uint-32[2]	R	▪ [0] Reserved Event Count: The number of events reserved for this sensor in the hardware FIFO. If there is a dedicated FIFO for this sensor, then this parameter is the size of this dedicated FIFO. For shared-FIFO scheme this value is 0. ▪ [1] Max Event Count: The maximum number of events that could be stored in the FIFOs for this sensor. This value is used to estimate how quickly the FIFO will get full when registering to the sensor at a specific rate, supposing no other sensors are activated. {4+8}
0x08	Axis Mapping	Int-8[3]	RW[3]	Axis mapping from device frame to Android body frame. {4+3}
0x09	Conversion Offset	Int-32[3]	RW	Raw data offset corresponding to zero measurement value. {4+12}
0x0A	Conversion Scale	FixP-32[3]	RW	Raw value to unit value conversion factor for each sensor axis. {4+12}
0x0B	Sensor Noise	FixP-32[3]	RW	Sensor noise based on Power Spectral Density test results for the hardware platform (in dB/rootHz). {4+12}
0x0C	Timestamp Offset	FixP-32	RW	Time (in seconds) from sensor signal acquisition to time stamp capture (including any filter propagation delay). {4+4}
0x0D	On Time & Wake Time	Uint-32[2]	RW	**[0] On Time:** Time (in microseconds) for the sensor to power on and provide a valid sample. [1] **Wake Time:** Time (in microseconds) for the sensor to exit Sleep mode and provide a valid sample. {4+8}
0x0E	HPF & LPF Cutoff	Uint-16[2]	RW	▪ **[0]** High pass filter 3dB cutoff in Hz. ▪ **[1]** Low pass filter 3dB cutoff in Hz. {4+4}

[1] Read Only Parameter: Can be queried by host. Attempt to write will be ignored or returned with error code in the response packet if the request has a valid sequence number.

[2] Write Only Parameter: Can only be written by host. Attempt to read will be ignored or returned with error code in response packet if the request has a valid sequence number.

[3] Read-Write Parameter: Host can set this parameter or query its current value. Parameter values are validated and if found invalid in a write (set) request, a response error code will be sent back if request has a valid sequence number, otherwise the invalid set request is quietly ignored.

0x0F	Sensor Name	Char[32]	R	Short human readable description/part number. Null terminated. Unused bytes must be zero. {4+32}
0x10	XYZ Offset from Origin	FixP-32[3]	RW	Offset (in meters) of the given sensor from a platform specific origin. {4+12}
0x11	Factory Cal – SKOR matrix	FixP-32[3][3]	RW	SKOR matrix as part of Linear Factory Calibration parameters. {4+36}
0x12	Factory Cal – Offset	FixP-32[3]	RW	Sensor offset as part of Linear Factory Calibration parameters. {4+12}
0x13	Non-linear Effects	FixP-16[4][3]	RW	Parameters as part of Non-linear Factory Calibration. {4+24}
0x14	Bias Stability	FixP-32[3]	RW	Measure of how badly the bias wanders with time. Sensor Units?{4+12}
0x15	Repeatability	FixP-32[3]	RW	Bias repeatability in sensor units. {4+12}
0x16	Temperature Coefficients	FixP-16[3][2]	RW	2 temperature coefficients for each axis. {4+12}
0x17	Shake Susceptibility	FixP-16[3]	RW	Susceptibility to shaking. {4+6}
0x18	Expected Norm	FixP-32	RW	Nominal magnitude of the sensor at STP when device is still. {4+4}
0x19	Version	Char[32]	R	MotionQ version string. Null terminated. Unused bytes must be zero. {4+32}
0x1A	Dynamic Cal - Scale	FixP-32[3]	RW[4]	Scale parameter (for each axis) for the sensor's runtime calibration. {4+12}
0x1B	Dynamic Cal - Skew	FixP-32[3]	RW	Skew parameter (for each axis) for the sensor's runtime calibration. {4+12}
0x1C	Dynamic Cal - Offset	FixP-32[3]	RW	Offset parameter (for each axis) for the sensor's runtime calibration. {4+12}
0x1D	Dynamic Cal - Rotation	FixP-32[3]	RW	Rotation parameter (for each axis) for the sensor's runtime calibration. {4+12}
0x1E	Dynamic Cal - Quality	FixP-32[3]	RW	Quality parameters for the sensor's runtime calibration. {4+12}
0x1F	Dynamic Cal – Calibration Source	Int-8	RW	Calibration source identifier (Default, Initial, Regular or Premium). {4+1}
0x20	Configuration Done	-	W	Used to indicate that Host is done with Configuration read/write of the device. This packet should always be sent after other configuration commands are used to setup the device. Certain configurations related to sensors cannot be modified after sending this parameter.
0x21-0xFF	Reserved for future	-	-	Reserved for future extension of parameter list

Data Types Reference:

- Int-<size> - Integer value of 'size' bits. E.g. Int-32 = 32-bit Integer.
- Uint-<size> - Unsigned integer of 'size' bits.
- Char – Character (1-byte) type
- FixP-<size> - Fixed point data format of 'size' bits. This is application specific and can be translated as NTPRECISE or NTEXTENDED type.

Notes:

- Certain configuration parameters such as calibration parameters & sensor settings are only allowed to be set during 'Standby' state of the device, while most can be queried during the 'Standby' or 'Idle' state of the device.

[4] Write is allowed only at device startup to inject previously stored calibration data.

- In Android a Batch request (that sets the sensor sampling period and FIFO buffering if max report latency > 0) is always expected to precede an Enable request.
- Parameters must be specified in their default format unless multiple formats for the same parameter type are supported on both host and hub. For example, the sampling period specified in the Batch request has default format of Uint-64 but can also be specified as Float or Double. In this case the hub must support multiple data formats for this to work.
- Max packet size (52-bytes) at this time is for parameter ID 0x18 (Non-linear Effects). Based on this information, it is expected that all configuration packets can be received by a hardware interface (I2C, SPI, etc.) having a FIFO/buffer/DMA of the aforementioned size without the possibility of data loss due to ISR latency.
- Factory Calibration parameters are derived from Dynamic calibration parameters as one time calibration as part of manufacturing process. It is the responsibility of the application layer running on AP to differentiate between factory calibration mode and regular calibration.

Example Sensor Control Packets

Calibrated Accelerometer Sensor Enable Request[5]

Case 1: Enable - No Response Desired

Request Format

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
0b	010b	0b	00b	0b	00b	000001b	0000b	0000b

Attribute Byte 2		
00000001b	0x01	<None>

Request Byte Stream

[0x20, 0x01, 0x00, 0x01, 0x01]

Case 2: Enable - Response Desired

Request Format

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
0b	010b	0b	00b	0b	00b	000001b	0000b	0001b

Attribute Byte 2		
00000001b	0x01	<None>

Request Byte Stream

[0x20, 0x01, 0x01, 0x01, 0x01]

[5] Note that an Enable request must always be called **after a Batch request** for the corresponding sensor.

Response Format

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
0b	011b	0b	00b	0b	00b	000001b	0000b	0001b

Attribute Byte 2		
00000000b	0x00000000h (return code 0 = Success)	<None>

Response Byte Stream

[0x30, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00]

Note: The response packet has parameter ID = 0x00 (Error Code in Data) and not 0x01 (Enable) as in the original request. The host should look for sequence number and sensor ID to match a response with the request sent.

Case 3: Disable[6] – No Response Desired

Request Format

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
0b	010b	0b	00b	0b	00b	000001b	0000b	0000b

Attribute Byte 2		
00000001b	0x00	<None>

Request Byte Stream

[0x20, 0x01, 0x00, 0x01, 0x00]

Calibrated Accelerometer Sensor Batch Request

Batch – No Response Desired

- Sampling Period = 50000000 (50ms) = 0x2FAF080
- Max Report Latency = 0

Request Format

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
0b	010b	0b	00b	0b	00b	000001b	0000b	0000b

Attribute Byte 2		
00000010b	0x0000000002FAF080h, 0x0000000000000000h	<None>

Request Byte Stream

[0x20, 0x01, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x02, 0xFA, 0xF0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

[6] A Disable request for a sensor has the effect of resetting any batching parameters associated with the sensor when it was enabled.

Sensor Flush Request for Calibrated Magnetometer

Case 1: Flush Request – No Response Desired

Request Format

Control Byte					Sensor Identifier Byte		Attribute Byte 1	
0b	010b	0b	00b	0b	00b	000010b	0000b	0000b

Attribute Byte 2		
00000011b	0x01	<None>

Request Byte Stream

[0x20, 0x02, 0x00, 0x03, 0x01]

Case 2: Flush Complete Event (from Device to Host)

Data Packet Format[7]

Control Byte						Sensor Identifier Byte		Attribute Byte			
0	000	0	1	1	0	00	000010	0000	1	10	1

Data Packet Bytes

[0x06, 0x02, 0x0D]

Note: Device will send the Flush Complete Event at the end of sending all sensor data accumulated on the batch FIFO. While it's possible that the FIFO may have sensor data for sensor types other than Calibrated Magnetometer that gets sent to the host, only the Calibrated Magnetometer data will contain the Flush Complete Event packet.

[7] Note that the data packet does not contain any data since it is only used to indicate flush complete event.

Device-Host Handshake and Transport (SPI)

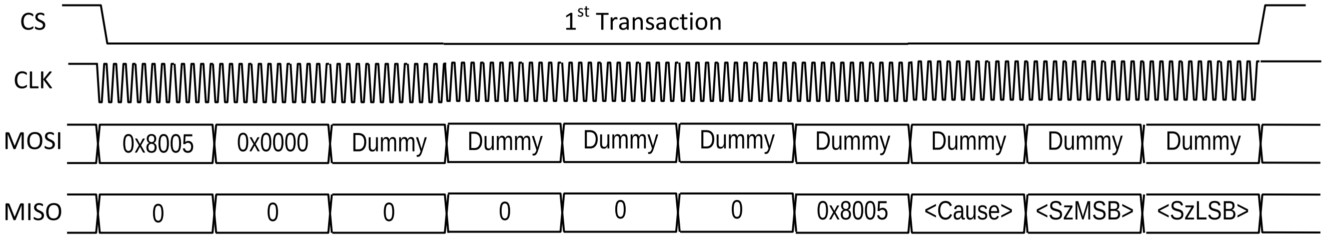
This section presents a proposal for low level transport scheme using SPI interface having 16-bit transfer size. This is not part of the Host Interface (HIF) Protocol but presents a deterministic and tested way of moving HIF packets between the device and host. The user is free to tweak and modify this scheme per their own platform and implementation.

Get Cause Command

For the purpose of sensor data transfer in deterministic manner, the following command sequence is proposed. The command is called “Get-Cause” and it has the following behavior:

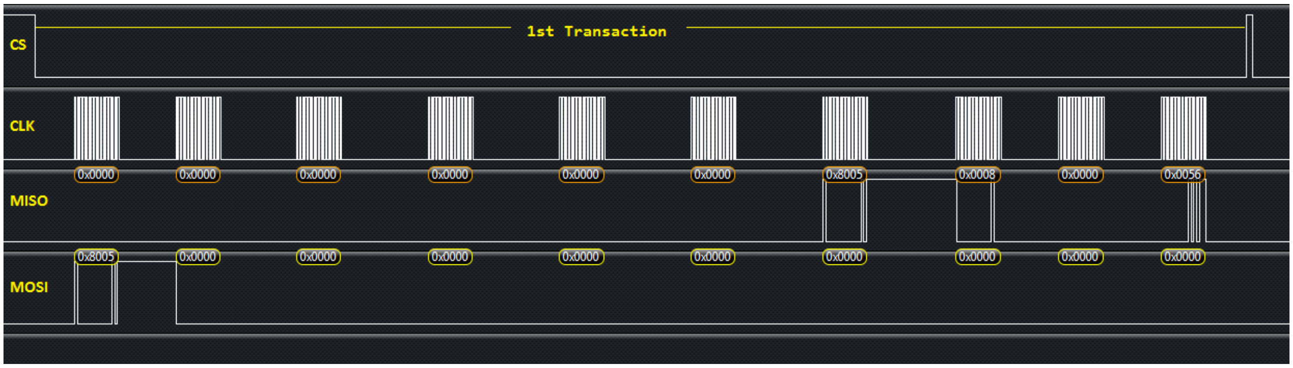
1. [Device] – Sensor data is ready[8]. Assert INT_EVT to host
2. [Host] – INT_EVT ISR invoked. Host issues Get Cause command (0x8005, 0x0000) and keeps sending dummy data after that waiting to read the response within the same SPI transaction (1st transaction in the SPI waveform diagram below).
3. [Device] – SPI transaction by host invokes the SPI ISR. The command is interpreted within the ISR and response values are loaded onto the SPI transmit FIFO. There are two possible scenarios in this case:
    a. For sensor data event, the cause will indicate that it is a sensor data ready event. The 32-bit size (in bytes) of the sensor packet ready to be sent will follow the ‘Cause’ value as shown in the waveform.
    b. For any other event (e.g. Voice Sense event), the ‘Cause’ will contain the appropriate event code and the 32-bit size field can have implementation specific interpretation.
4. [Host] – Host reads the response to Get Cause command followed by the 3 16-bit values. It then ends the SPI transaction.
5. [Host] – If the size field is non-zero, host will immediately issue a second transaction to read out the packet. This time host will shift out (((Size + 3) /4) * 2) 16-bit dummy words to read the packet from the device. To keep the timings tight and deterministic, it is expected that the packet readout will be done from the Host driver’s interrupt thread (in Linux).
   If the size field is zero for a sensor data event, the host does nothing further. If it is any other event, the host will handle the event in application specific manner.

SPI Waveform Diagram (1st Transaction)



[8] It is assumed here that the current state of MotionQ allows the data to be sent to host.

Sample Run Waveform (1st Transaction)

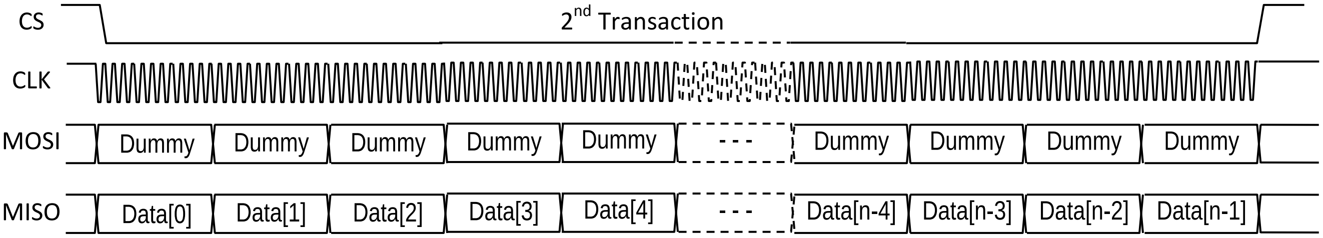


Notes on Sample Run (1st Transaction):

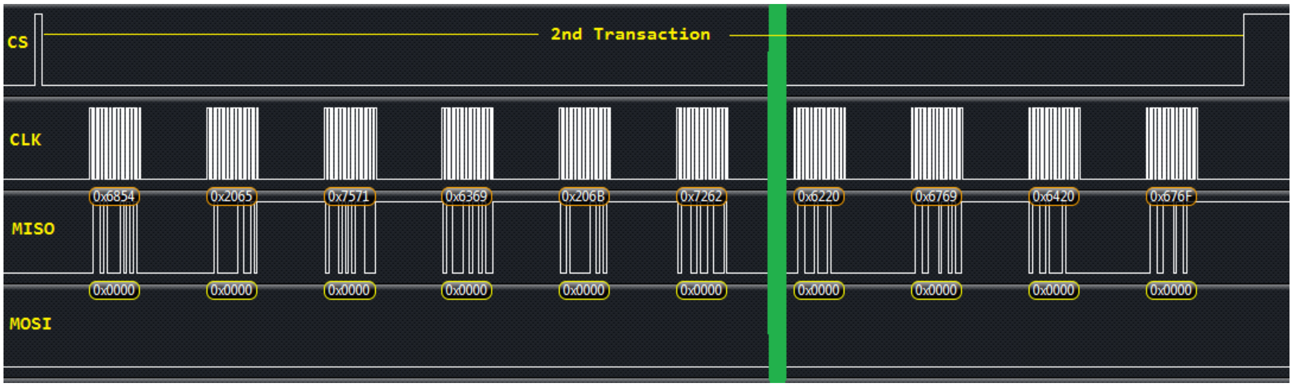
1. This test run was done using Audience's MQ100-EVM2 connected to an STM32 Discovery F3 board (Android host emulator) over SPI interface.
2. In the above screen shot the signals are as follows (from top to bottom): CS, CLK, MISO, MOSI.
3. The three SPI clock rates used for testing (4.5MHz, 9MHz & 18MHz) worked without any issues.
4. At 18MHz the 1st transaction for Get Cause took **37.6µs** to complete
5. At all clock rates only 4-dummy words needed to be shifted out by the host before the command response showed up.
6. As shown, the 'Cause' value is 0x0008 and the 'Size' value is 0x00000056 (86 bytes).

SPI Waveform Diagram (2nd Transaction)

This transaction will happen only for sensor data packet as specified by the corresponding 'Cause' value. The size 'n' in this case is the nearest 4-byte multiple of the 'Size' value received in the 1st Transaction.



Sample Run Waveform (2nd Transaction)



Notes on Sample Run (2nd Transaction)

1. In the above screen shot the signals are as follows (from top to bottom): CS, CLK, MISO, MOSI.
2. At 18MHz the 2nd transaction for Get Cause took **132µs** to complete transfer of (((86 + 3) /4) * 2) = 44 words (88 bytes).
3. The 2nd transaction immediately followed the 1st transaction as indicated by the narrow pulse (0.2µs) on the CS signal on the left.

Configuration Command

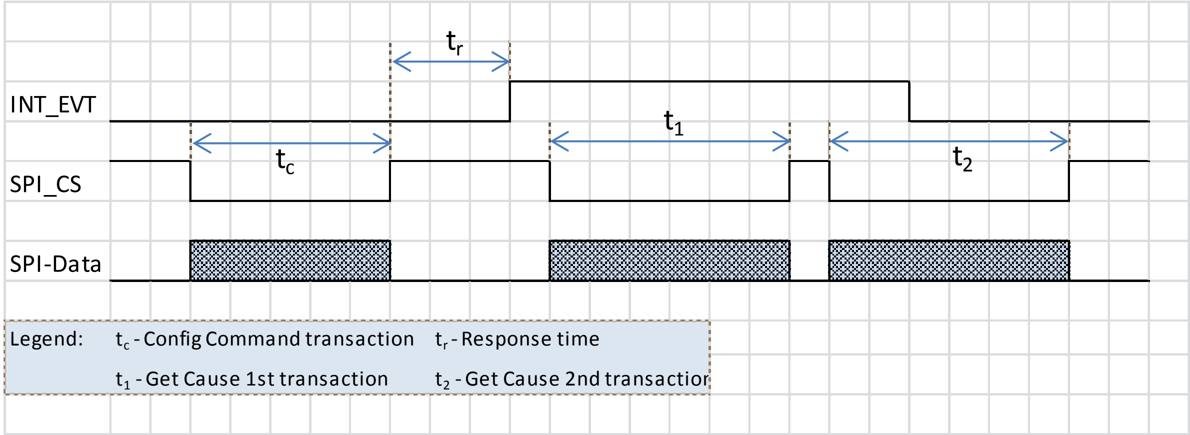
This command is used for sending Sensor Control Request packets that provide means to read or write parameters defined for motion algorithms and sensors (see Parameters table). This command will have the following behavior sequence:

1. [Host] – Higher layer sends Sensor Control Request packet to the driver. Host driver issues Config command (0x8006) with packet size and the packet [0x8006, Size-in-bytes, packet+padding[9]] within a single SPI transfer (see SPI waveform below).
2. [Device] – Config command is received and processed by the firmware. The SensorHub handler will generate the appropriate response to the Sensor Control Request, packetize it and keep it ready in a separate Sensor Control Request queue[10]. It will then assert the INT_EVT to host. In case a packet error is detected (such as unsupported packet fields, bad CRC or malformed packet) a special response packet is generated as described below in 4.a. This step constitutes the response time 't_r' in the timeline waveform below.
3. [Host] – INT_EVT ISR invoked. Host issues Get Cause[11] command (0x8005, 0x0000) and keeps sending dummy data after that waiting to read the response within the same SPI transaction (1st transaction in the SPI waveform diagram below).
4. [Device] – SPI transaction by host invokes the SPI ISR. The command is interpreted within the ISR and response values are loaded onto the SPI transmit FIFO. The 'Cause' value will indicate that this is a Config Command response. For the 'Size' value that follow the 'Cause' there are two possible scenarios:
    a. In situations where the packet could not be reliably parsed due to incorrect formatting, unsupported fields, corruption or CRC failure, the device will send a special response packet with sequence number and sensor ID field set to 0 and parameter ID of "Error Code in Data". The data field will contain the corresponding error code (signed 32-bit).
    b. In situations where the packet was parsed there are two possibilities:
        i. A response was desired by having non-zero sequence number: A non-zero 'Size' value indicates that a response packet is ready to be read.
        ii. If sequence number in the request was 0, INT_EVT is not asserted and Config Command response is not generated. The exception to this is scenario 4.a described above.

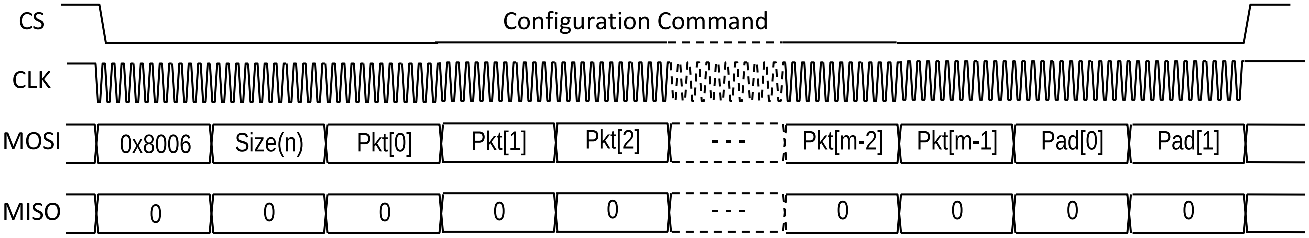
[9] Padding bytes maybe needed to align the packet to 4-bytes boundary. The 'Size' field of the command will provide the actual size of the packet so that the handler will be able to ignore the padding bytes.
[10] The firmware state must allow for the particular Sensor Control Request packet to be handled.
[11] Note that the sequence of Get Cause is identical to the sensor data transfer, thus simplifying the handler implementation.

5. [Host] – Host reads the response to Get Cause command followed by the 3 16-bit values. It then ends the SPI transaction.
6. [Host] – If the size field is non-zero, host will immediately issue a second transaction to read out the packet. This time host will shift out (m = ((n + 3) /4) * 2) 16-bit dummy words to read the packet from the device. To keep the timings tight and deterministic, it is expected that the packet readout will be done from the Host driver's interrupt thread (in Linux).

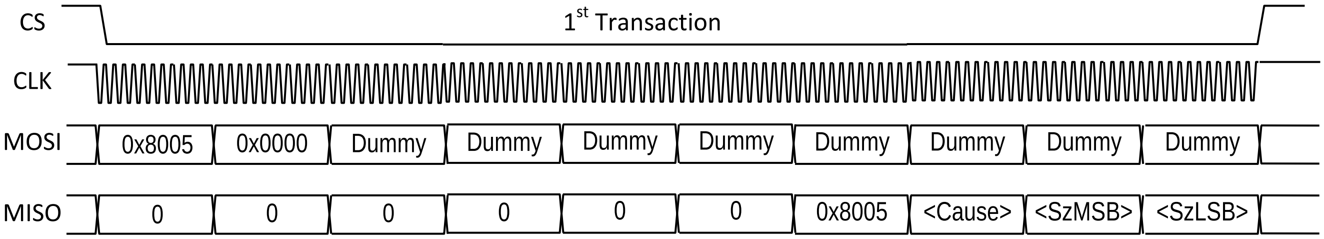
SPI Waveform – Configuration Command Timeline


t_r
INT_EVT
t_1
t_c
t_2
SPI_CS
SPI-Data
Legend:
t_c - Config Command transaction
t_r - Response time
t_1 - Get Cause 1st transaction
t_2 - Get Cause 2nd transactior

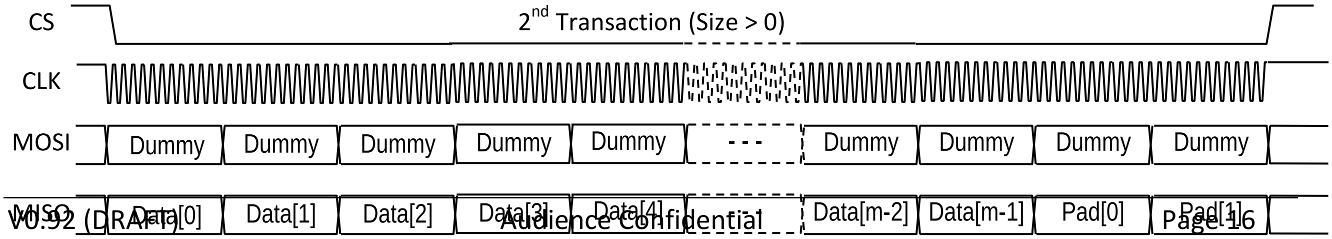
SPI Waveform: Configuration Command



SPI Waveform: Get Cause – 1st Transaction



SPI Waveform: Get Cause – 2nd Transaction



Cause and Error Codes

Cause Values	Description
0x0000	Reserved
0x0001	Sensor Data Ready
0x0002	Magnetometer calibration data updated
0x0003	Accelerometer calibration data updated
0x0004	Gyroscope calibration data updated
0x0005	Temperature change detected (used for oscillator calibration)
0x0006	Configuration Command response
0x0007	Unrecoverable error in MotionQ
0x0008-0F	Reserved for MotionQ events
0x0010-1F	Reserved for VoiceQ events
0x0020 –	User/Application defined

Error Values	Description
0	Success
-1	Command failed
-2	Command not supported in current state of FW
-3	Malformed or corrupt packet
-4	Packet checksum failed
-5	Unsupported packet format
-6	[TBD]
-7	[TBD]
-8	[TBD]
-9	[TBD]

Device-Host Handshake and Transport (I^{2}C)

This section presents a proposal for low level transport scheme using I^{2}C interface having 7-bit address and 8-bit transfer size. This is not part of the Host Interface (HIF) Protocol but presents a deterministic and tested way of moving HIF packets between the device and host. The user is free to tweak and modify this scheme per their own platform and implementation.

The sensor-hub device is the slave and the Android host is the I2C master. The slave device implements the following registers for communication and handshake.

Register Address	R/W	Description
0x00	R	Get-Cause Command: <Cause><Size-MSB><Size-LSB>
0x03	W	Configuration Command and Data: <Size-MSB><Size-LSB><Data-1><Data-2>…<Data-N>
0x48	R	Get-Cause Data: <Data-1><Data-2>…<Data-N>
0x7A	W	A write to this address forces soft reset of SensorHub.
0x7B	W	A write to this address toggles host suspend state variable maintained in SensorHub firmware. It can be used by Host to inform SensorHub of its imminent sleep/suspend mode before doing so. When Host resumes, it writes to this register again to inform SensorHub that it (host) is in active state.

In the I2C transaction diagrams below the following legends are used:

S	Start
P	Stop
Sr	Repeated Start
W	Write
N	Nack

SLA	Slave Address (7-bit)
RA	Register Address
R	Read
A	Ack

Get Cause Command

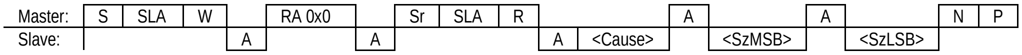
The command sequence is similar to SPI implementation. The main difference is that the command identifier in SPI is replaced by register access in I2C. Also the cause value supported on I2C is 1-byte and size field is 2 bytes. Behavior is as follows:

1. [Device] – Sensor data is ready[12]. Assert INT_EVT to host
2. [Host] – INT_EVT ISR invoked. Host issues an I2C register read at Get-Cause Command register (0x00) to read 3-bytes.
3. [Device] – The register address read is interpreted within the ISR and appropriate values are prepared to I2C transmit of the next 3 bytes. There are two possible scenarios in this case:

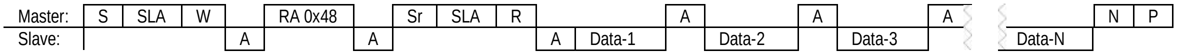
[12] It is assumed here that the current state of motion algorithms allows the data to be sent to host.

   a. For sensor data event, the cause will indicate that it is a sensor data ready event. The 16-bit size (in bytes) of the sensor packet ready to be sent will follow the 'Cause' value. (See I2C waveform below).
   b. For any other event (e.g. Voice Sense event), the 'Cause' will contain the appropriate event code and the 16-bit size field can have implementation specific interpretation.
4. [Host] – Host reads the 3 bytes from Get-Cause Command register and ends the I2C transaction.
5. [Host] – If the size field is non-zero, host will immediately issue a second transaction to read out the packet from the Get-Cause Data register. This time host will read the exact number of bytes that were specified in the Get-Cause Command's size field. If the size field is zero for a sensor data event, the host does nothing further. If it is any other event, the host will handle the event in application specific manner.

I2C Transaction Diagram – Get-Cause Command Register Read



I2C Transaction Diagram – Get-Cause Data Register Read



Configuration Command

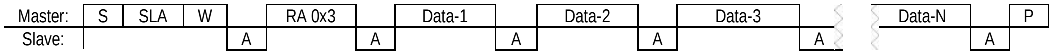
This command is used for sending Sensor Control Request packets that provide means to read or write parameters defined for motion algorithms and sensors (see Parameters table). This command will have the following behavior sequence:

1. [Host] – Higher layer sends Sensor Control Request packet to the driver. Host I2C driver issues write to Configuration Command register (0x03) with packet size and the packet data within a single I2C register write transaction (see waveform below).
2. [Device] – Configuration command is received and processed by the firmware. The SensorHub handler will generate the appropriate response to the Sensor Control Request, packetize it and keep it ready in a separate Sensor Control Request queue. It will then assert the INT_EVT to host. In case a packet error is detected (such as unsupported packet fields, bad CRC or malformed packet) a special response packet is generated as described below in 4.a.
3. [Host] – INT_EVT ISR invoked. Host issues an I2C register read at Get-Cause Command register (0x00) to read 3-bytes.
4. [Device] – The command is interpreted within the ISR and appropriate values are prepared to I2C transmit of the next 3 bytes. The 'Cause' value will indicate that this is a Config Command response. For the 'Size' value that follow the 'Cause' there are two possible scenarios:
   a. In situations where the packet could not be reliably parsed due to incorrect formatting, unsupported fields, corruption or CRC failure, the device will send a special response packet with sequence number and sensor ID field set to 0 and parameter ID of "Error Code in Data". The data field will contain the corresponding error code (signed 32-bit).

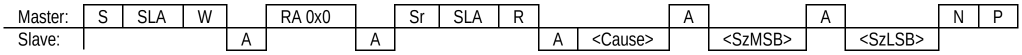
  b. In situations where the packet was parsed there are two possibilities:
     i. A response was desired by having non-zero sequence number: A non-zero ‘Size’ value indicates that a response packet is ready to be read.
     ii. If sequence number in the request was 0: INT_EVT is not asserted and Config Command response is not generated. The exception to this is scenario 4.a described above.

5. [Host] – If the size field is non-zero, host will immediately issue a second I2C transaction to read out the packet from the Get-Cause Data register. If the size field is zero, the host does nothing further. If it is any other ‘Cause’, the host will handle it in application specific manner.

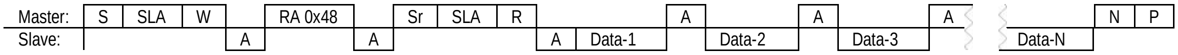
I2C Transaction Diagram – Configuration Write



I2C Transaction Diagram – Get-Cause Command Register Read



I2C Transaction Diagram – Get-Cause Data Register Read


Master:
S
SLA
W
RA 0x48
Sr
SLA
R
A
A
A
N
P
Slave:
A
A
A
Data-1
Data-2
Data-3
Data-N