

# BHA260AB

Ultra-low power, high performance,  
programmable Smart Sensor with  
integrated Accelerometer

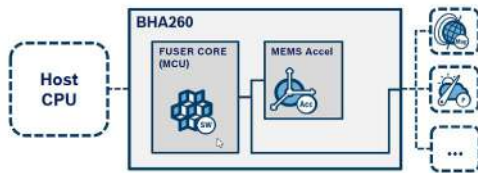


## BHA260AB Datasheet

Document revision	1.7
Document release date	November 2020
Document number	BST-BHA260AB-DS000-07
Technical reference code(s)	0 273 141 392
Notes	Data and descriptions in this document are subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product appearance.

## BHA260AB

### Ultra-low power, high performance, programmable Smart Sensor with integrated Accelerometer and Gyroscope



#### General description

The BHA260AB is an ultra-low power, customizable smart sensor consisting of Bosch Sensortec's new, programmable 32-bit microcontroller (Fuser2), a state-of-the-art 3-Axis Accelerometer and a powerful Event-driven Software Framework containing pre-installed sensor fusion and other sensor data processing software within a small 22 pad LGA package.

The Fuser2 Core can be configured to operate at 20 MHz (Long Run mode) or 50 MHz (Turbo mode). It can boot from a wide variety of hosts, ranging from a small Cortex-M0™ MCU up to multicore application processors.

In combination with its wide connectivity and extendibility, the BHA260AB is a versatile and ideal solution when it comes to running always-on sensor data processing algorithms at ultra-low power consumption.

#### Hardware Functions

- ▶ Integrated CPU Core
  - ARC EM4 CPU (up to 3.6 CoreMark/MHz)
  - Long Run mode: 950 µA @ 20 MHz running CoreMark
  - Turbo mode: 2.8 mA @ 50 MHz running CoreMark
  - Floating Point Unit (FPU)
  - Memory Protection Unit (MPU)
  - 4-channel micro DMA Controller
  - 2-way associative Cache Controller
  - ARCV2 16/32 bit instruction set
- ▶ Memories
  - 256 kByte on-chip SRAM
  - 144 kByte on-chip ROM preloaded with software

- ▶ Connectivity
  - Host interface configurable as SPI or I2C
  - 2 master interfaces (1x selectable SPI/I2C master and 1x I2C master)
  - Up to 16 GPIOs
  - Fast I/O operations:
    - SPI and GPIOs up to 50 MHz
    - I2C up to 3.4 MHz
- ▶ Integrated sensor (3-DoF Accel)
  - 16-bit 3-axis accelerometer
- ▶ Other functions
  - Hardware reset, brown-out detector and core watchdog
  - On-chip voltage regulator
  - On-chip system (20, 50 MHz) and timer (128 kHz) oscillators
  - 2-wire compact JTAG interface and hardware debug support with action points
  - Up to 4 timers and counters, 12 event channels with hardware support for time synchronization
- ▶ Single Power Supply (1.8V)
- ▶ Package
  - LGA, 22 pads, 2.7 mm x 2.6 mm x 0.78 mm

#### Software Features

- ▶ Open sensor development platform
- ▶ Integrated Event-driven Software Framework and OpenRTOS™ with virtual sensor stack
- ▶ Integrated BSX sensor fusion software for reliable 3D orientation, activity recognition, and more
- ▶ Powerful SDK for easy customization with support for
  - Metaware C Compiler for ARC
  - GNU C Compiler for ARC

#### Target applications and devices

- 24/7 always-on sensor data processing at ultra-low power consumption
- 3D orientation, power management and wake-up control, step counting, position tracking, activity recognition, pose and head tracking, context awareness
- Wrist-mounted, Hearables, Eyewear and other wearable devices
- Smartphones and other mobile communication devices
- AR/VR/MR reality headset and controller devices

# Table of Contents

Table of Contents .....	3
List of Figures.....	9
List of Tables.....	10
<b>GENERAL DESCRIPTION.....</b>	<b>13</b>
<b>1 Overview.....</b>	<b>13</b>
<b>2 Hardware Features .....</b>	<b>14</b>
<b>2.1 CPU Core (Fuser2).....</b>	<b>14</b>
<b>2.2 System Control blocks .....</b>	<b>14</b>
2.2.1 Reset options .....	14
2.2.2 Oscillators & Clocks .....	14
<b>2.3 Host Interface .....</b>	<b>14</b>
<b>2.4 Memory Subsystem .....</b>	<b>15</b>
2.4.1 On-Chip SRAM .....	15
2.4.2 ROM with integrated Software.....	15
2.4.3 OTP.....	15
<b>2.5 Peripheral Subsystem .....</b>	<b>15</b>
2.5.1 Secondary Master Interface 1.....	15
2.5.2 Secondary Master Interface 2.....	15
2.5.3 Secondary Master Interface 3.....	15
2.5.4 GPIOs .....	16
2.5.5 Universal Timers & Counter .....	16
2.5.6 Interrupt Sources.....	16
2.5.7 Event Subsystem .....	16
<b>2.6 Debug Interface.....</b>	<b>16</b>
<b>2.7 Integrated physical Sensors.....</b>	<b>17</b>
<b>3 Pad connections and description.....</b>	<b>18</b>
<b>3.1 Pad description .....</b>	<b>18</b>
<b>3.2 Setup of Multifunction pins .....</b>	<b>19</b>
<b>4 System Configuration .....</b>	<b>22</b>
<b>4.1 Connection Diagram.....</b>	<b>22</b>
<b>4.2 Block Diagram .....</b>	<b>23</b>
<b>4.3 Physical Primary Host Interface.....</b>	<b>23</b>
4.3.1 Host interrupt.....	23
4.3.2 Operation in I2C mode .....	24
4.3.3 Operation in SPI mode .....	25
4.3.4 Burst mode operation .....	27
4.3.4.1 Burst mode operation on DMA enabled registers.....	28
4.3.4.2 Burst mode operation on regular registers .....	28
<b>4.4 Host Data Interface .....</b>	<b>28</b>

4.4.1	Host Data Access.....	28
4.4.2	Host Command Protocol .....	30
<b>5</b>	<b>Device Initialization and Start-up.....</b>	<b>32</b>
<b>5.1</b>	<b>Power On and Reset.....</b>	<b>32</b>
<b>5.2</b>	<b>BHA260AB Initialization and Boot mode .....</b>	<b>32</b>
5.2.1	BHA260AB Initialization.....	32
5.2.2	Secure Boot Mode.....	34
<b>5.3</b>	<b>Device Operation .....</b>	<b>35</b>
<b>5.4</b>	<b>Processor Execution Modes .....</b>	<b>36</b>
<b>5.5</b>	<b>Power States and Run Levels .....</b>	<b>36</b>
<b>5.6</b>	<b>Debug and Post Mortem Support .....</b>	<b>38</b>
<b>6</b>	<b>Device Configuration and Operation .....</b>	<b>39</b>
<b>6.1</b>	<b>Overview of the Event-driven Software Framework and Virtual Sensor Stack... 39</b>	
<b>6.2</b>	<b>Using Virtual sensors .....</b>	<b>39</b>
<b>6.3</b>	<b>Using FIFOs and FIFO Events .....</b>	<b>40</b>
6.3.1	FIFOs .....	41
6.3.2	FIFO Events .....	41
<b>6.4</b>	<b>Using the Parameter Interface.....</b>	<b>42</b>
<b>6.5</b>	<b>Current consumption in operation .....</b>	<b>42</b>
	<b>FUNCTIONAL DESCRIPTION .....</b>	<b>44</b>
<b>7</b>	<b>Integrated Product Software .....</b>	<b>44</b>
<b>7.1</b>	<b>Event-driven Software Framework and Virtual sensor stack.....</b>	<b>44</b>
<b>7.2</b>	<b>Hardware abstraction layer .....</b>	<b>44</b>
<b>7.3</b>	<b>BSX Sensor Fusion Lib .....</b>	<b>44</b>
<b>7.4</b>	<b>OpenRTOS multithreading real-time kernel .....</b>	<b>45</b>
<b>7.5</b>	<b>Virtual Sensor Stack.....</b>	<b>45</b>
<b>7.6</b>	<b>Other available Software modules.....</b>	<b>46</b>
7.6.1	Standard C library (libc) and standard math (libm).....	46
7.6.2	Libraries for Digital signature .....	46
<b>8</b>	<b>Software Development Kit (SDK).....</b>	<b>47</b>
<b>9</b>	<b>Memory Configuration and Memory Map .....</b>	<b>48</b>
<b>9.1</b>	<b>On-Chip Memories .....</b>	<b>49</b>
<b>9.2</b>	<b>Peripheral Space .....</b>	<b>49</b>
<b>10</b>	<b>Pin and Interface Configuration.....</b>	<b>50</b>
<b>10.1</b>	<b>Configuration of Master Interfaces .....</b>	<b>50</b>
<b>10.2</b>	<b>General-Purpose Inputs/Outputs .....</b>	<b>50</b>
<b>11</b>	<b>Event and Interrupt Configuration .....</b>	<b>52</b>
	<b>REFERENCE PART.....</b>	<b>55</b>

<b>12 Host Interface Register Map</b> .....	<b>55</b>
<b>12.1 Register Description</b> .....	<b>56</b>
12.1.1 Host Channel 0 - Command Input (0x00) .....	56
12.1.2 Host Channel 1 - Wake-up FIFO Output (0x01).....	57
12.1.3 Host Channel 2 - Non-Wake-up FIFO Output (0x02) .....	57
12.1.4 Host Channel 3 - Status and Debug FIFO Output (0x03).....	57
12.1.5 Reserved (0x04).....	57
12.1.6 Chip Control (0x05) .....	57
12.1.7 Host Interface Control (0x06).....	58
12.1.8 Host Interrupt Control (0x07) .....	59
12.1.9 WGP1 – WGP3 - General Purpose Host Writeable (0x08-0x13) .....	61
12.1.10 Reset Request (0x14) .....	61
12.1.11 Timestamp Event Request (0x15) .....	61
12.1.12 Host Control (0x16) .....	62
12.1.13 Host Status (0x17).....	63
12.1.14 Host Channel CRC (0x18-0x1B).....	64
12.1.15 Fuser2 Product Identifier (0x1C).....	64
12.1.16 Fuser2 Revision Identifier (0x1D) .....	64
12.1.17 ROM Version (0x1E-0x1F) .....	64
12.1.18 Kernel Version (0x20-0x21).....	64
12.1.19 User Version (0x22-0x23).....	64
12.1.20 Feature Status (0x24).....	65
12.1.21 Boot Status (0x25).....	65
12.1.22 Host Interrupt Timestamp (0x26-0x2A).....	66
12.1.23 Chip ID (0x2B).....	66
12.1.24 Interrupt Status (0x2D) .....	66
12.1.25 Error Value (0x2E).....	68
12.1.26 Error Aux, Debug Value, Debug State (0x2F-0x31) .....	71
12.1.27 RGP5 – RGP7 - General-Purpose Host Readable (0x32-0x3D) .....	71
<b>13 Host Interface Commands</b> .....	<b>72</b>
<b>13.1 Bootloader Commands (incl. bootloader status codes)</b> .....	<b>73</b>
13.1.1 Download Post Mortem Data (0x0001) .....	73
13.1.2 Upload to Program RAM (0x0002) .....	75
13.1.3 Boot Program RAM (0x0003) .....	75
13.1.4 Raise Host Interface Speed (0x0017).....	75
<b>13.2 Main Firmware Commands (incl. main firmware status codes)</b> .....	<b>76</b>
13.2.1 Set Sensor Data Injection Mode (0x0007) .....	76
13.2.2 Inject Sensor Data (0x0008).....	77
13.2.3 FIFO Flush (0x0009) .....	78
13.2.4 Soft Pass-Through (0x000A) .....	79
13.2.5 Request Sensor Self-Test (0x000B) .....	82
13.2.6 Request Sensor Fast Offset Compensation (0x000C) .....	83
13.2.7 Configure Sensor (0x000D).....	84
13.2.8 Change Sensor Dynamic Range (0x000E) .....	85
13.2.9 Control FIFO Format (0x0015) .....	85

<b>13.3 Parameter Interface .....</b>	<b>86</b>
13.3.1 Reading and writing Parameters .....	86
13.3.2 System Parameters .....	87
13.3.2.1 Meta Event Control (0x0101, 0x0102).....	88
13.3.2.2 FIFO Control (0x0103).....	88
13.3.2.3 Firmware Version (0x0104).....	89
13.3.2.4 Timestamps (0x0105) .....	90
13.3.2.5 Virtual Sensors Present (0x011F) .....	90
13.3.2.6 Physical Sensors Present (0x0120) .....	91
13.3.2.7 Physical Sensor Information (0x0121 – 0x0160).....	92
13.3.3 BSX Algorithm Parameters.....	94
13.3.3.1 Reading and writing the BSX State Exchange Structure .....	94
13.3.3.2 BSX Version (0x027E).....	95
13.3.4 Virtual Sensor Information Parameters (0x0301 – 0x0395) .....	95
13.3.5 Virtual Sensor Configuration Parameters (0x0501 – 0x0595) .....	96
13.3.6 Physical Sensor Control Parameters (0x0E00 – 0x0EFF) .....	97
<b>13.4 Command Error Response.....</b>	<b>98</b>
<b>14 FIFO Data Formats .....</b>	<b>99</b>
<b>14.1 Wake-up and Non-Wake-up FIFO .....</b>	<b>99</b>
<b>14.2 Status and Debug FIFO.....</b>	<b>100</b>
14.2.1 Synchronous mode .....	100
14.2.2 Asynchronous mode.....	100
<b>15 FIFO Data Types and Format.....</b>	<b>101</b>
<b>15.1 Format of virtual sensor events .....</b>	<b>103</b>
15.1.1 Format “Quaternion+” .....	103
15.1.2 Format “Euler” .....	103
15.1.3 Format “3D Vector” .....	104
15.1.4 Format “Activity” .....	104
15.1.5 Format of Scalar data .....	105
15.1.6 Format of sensors without payload .....	105
<b>15.2 Retrieving timestamps of virtual sensor events.....</b>	<b>105</b>
<b>15.3 Format of Meta Events .....</b>	<b>106</b>
15.3.1 Meta Event: Flush Complete .....	107
15.3.2 Meta Event: Sample Rate Changed .....	107
15.3.3 Meta Event: Power Mode Changed .....	108
15.3.4 Meta Event: System Error .....	108
15.3.5 Meta Event: Algorithm Events .....	108
15.3.6 Meta Event: Sensor Status.....	108
15.3.7 Meta Event: Sensor Error .....	108
15.3.8 Meta Event: FIFO Overflow .....	108
15.3.9 Meta Event: Dynamic Range Changed.....	109
15.3.10 Meta Event: FIFO Watermark.....	109
15.3.11 Meta Event: Initialized .....	109
15.3.12 Meta Event: Transfer Cause.....	109

15.3.13 Meta Event: Software Framework .....	110
15.3.14 Meta Event: Reset.....	110
15.3.15 Meta Event: Spacer.....	110
<b>15.4 Debug Data.....</b>	<b>110</b>
<b>16 Reading FIFO Data .....</b>	<b>112</b>
16.1 Host Interrupt Behavior .....	112
16.2 FIFO Overflow Handling .....	114
16.3 Application Processor Suspend Mode .....	115
16.4 Loss of sync recovery .....	115
<b>17 Error detection and recovery .....</b>	<b>116</b>
<b>18 Physical and Electrical Specifications.....</b>	<b>118</b>
18.1 Absolute Maximum Ratings .....	118
18.2 Operating Conditions.....	118
18.3 Electrical characteristics .....	118
18.4 Physical Characteristics and Measurement Performance .....	120
18.5 Timing Characteristics.....	121
18.5.1 Fuser2 Power-Up and Power-Down Timing Characteristics .....	121
18.5.2 Host I2C Interface Timing.....	122
18.5.3 Host SPI Interface Timing.....	124
18.5.4 Master Interface I2C Timing .....	125
18.5.5 Master Interface SPI Timing .....	125
<b>19 Mechanical Specifications.....</b>	<b>127</b>
19.1 Outline Dimensions.....	127
19.2 Device Marking .....	127
19.3 Sensing axes and axes remapping .....	127
19.4 PCB Footprint recommendation.....	130
<b>20 Packaging Specifications (Tape &amp; Reel).....</b>	<b>131</b>
20.1 Orientation within the tape .....	132
20.2 Multiple sourcing.....	132
<b>21 Handling, Soldering and Environmental Guidelines.....</b>	<b>133</b>
21.1 Handling instructions .....	133
21.2 Soldering guidelines .....	133
21.3 Environmental Safety.....	134
<b>LEGAL DISCLAIMER.....</b>	<b>136</b>
<b>22 Engineering Samples .....</b>	<b>136</b>
<b>23 Product use .....</b>	<b>137</b>
<b>24 Application examples and hints .....</b>	<b>138</b>
<b>25 References .....</b>	<b>139</b>

**DOCUMENT HISTORY AND MODIFICATION ..... 140**  
**26 Document History..... 140**



## List of Figures

Figure 1: Pad connections.....	18
Figure 2: Pinout BHA260AB.....	19
Figure 3: Connection diagram sketch for BHA260AB.....	22
Figure 4: Block diagram of BHA260AB.....	23
Figure 5: I2C Write Transaction.....	25
Figure 6: I2C Read Transaction – Combined Format (with Repeat START) .....	25
Figure 7: I2C Read Transaction - Split Format (1) .....	25
Figure 8: I2C Read Transaction - Split Format (2) .....	25
Figure 9: SPI write transaction in 4-Wire and 3-Wire mode .....	27
Figure 10: SPI read transaction in 4-wire mode.....	27
Figure 11: SPI read transaction in 3-wire mode.....	27
Figure 12: Overview host data interface .....	29
Figure 13: BHA260AB secure boot concept (incl. signing and verification of FW images) ...	35
Figure 14: Transitions of the power states.....	38
Figure 15: Structure of the BHA260AB Event-driven Software Framework.....	44
Figure 16: Fuser2 memory map .....	48
Figure 17: Overview configuration options of master interface ports.....	50
Figure 18: Active High Level Host Interrupt Example .....	113
Figure 19: Active Low Edge Triggered Host Interrupt Example.....	114
Figure 22: Host I2C interface timing .....	122
Figure 23: Write Transaction on 4-Wire Host SPI.....	124
Figure 24: Read Transaction on 4-Wire Host SPI.....	124
Figure 25: Read Transaction on 3-Wire Host SPI.....	124
Figure 26: Master Interface SPI Clock Timing .....	125
Figure 27: Master Interface SPI Timing .....	125
Figure 30: Outline dimensions.....	127
Figure 31: Sensing Axes .....	128
Figure 32: Placement Options with Respect to Device Orientation .....	128
Figure 33: Example component placement.....	129
Figure 34: Footprint recommendation <sup>1)</sup> .....	130
Figure 36: Part orientation within tape .....	132
Figure 37: Soldering profile for BHA260AB .....	134

## List of Tables

Table 1: Pad connections .....	18
Table 2: Pad functions and configuration options .....	19
Table 3: Pad functions and init conditions .....	21
Table 4: Typical vales for external circuit components.....	22
Table 5: I2C device address selection options.....	24
Table 6: Overview DMA channels .....	28
Table 7: Basic overview of the BHA260AB host interface registers .....	29
Table 8: Structure of Command Packet.....	30
Table 9: Structure of a Status Packet .....	31
Table 10: Available reset sources.....	32
Table 12: Relevant registers for host boot mode .....	33
Table 13: Available power states in Fuser2 Core MCU.....	37
Table 14: Power consumption vs. operating mode .....	43
Table 15: Configuration options of master interface ports.....	50
Table 16: MCU Interrupts (Sources and Mapping).....	52
Table 17: Host interface register map.....	55
Table 18: Chip Control Register (0x05) .....	57
Table 19: Host Interface Control Register (0x06).....	58
Table 20: Host Interrupt Control Register (0x07) .....	60
Table 21: Reset Request Register (0x14).....	61
Table 22: Timestamp Event Request Register (0x15) .....	62
Table 23 Timestamp Event Status Packet.....	62
Table 24: Host Control Register (0x16) .....	62
Table 25: Host Status Register (0x17).....	63
Table 26: Feature Status Register (0x24).....	65
Table 27: Boot Status Register (0x25).....	65
Table 28: Interrupt Status Register (0x2D) .....	67
Table 29: Error Value Register (0x2E).....	68
Table 30: Debug State Register values (0x31) .....	71
Table 31: Overview BHA260AB Host Interface Commands.....	72
Table 32: Download Post Mortem Data - Command.....	73
Table 33: Download Post Mortem Data - Response .....	73
Table 34: Diagnostic bit field .....	74
Table 35: Upload to Program RAM – Command.....	75
Table 36: Boot Program RAM – Command .....	75
Table 39: Raise Host Interface Speed - Command.....	75
Table 40: Raise Host Interface Speed - Response .....	76
Table 44: Set Sensor Data Injection Mode – Command.....	76
Table 45: Set Sensor Data Injection Mode – Response.....	77
Table 46: Inject Sensor Data – Command.....	77
Table 47: FIFO Flush – Command .....	78
Table 48: Soft Pass-Through – Command.....	79
Table 49: Sensor Driver Mode description.....	79
Table 50: Arbitrary Device Mode description .....	80
Table 51: CMD config description.....	81

Table 52: Soft pass through - Response .....	81
Table 53: Request Sensor Self Test - Command.....	82
Table 54: Sensor self-test result - Response .....	82
Table 55: Request Sensor Fast Offset Compensation - Command .....	83
Table 56: Sensor fast offset compensation - Response.....	83
Table 57: Configure Sensor - Command .....	84
Table 58: Change Sensor Dynamic Range - Command .....	85
Table 59: Control FIFO Format - Command .....	86
Table 60: Parameter Read Format .....	86
Table 61: Parameter Groups .....	87
Table 62: System Parameter Overview .....	87
Table 63: Meta Event Control .....	88
Table 64: FIFO Control.....	89
Table 65: Firmware Version .....	89
Table 66: Timestamps .....	90
Table 67: Virtual Sensor Present.....	91
Table 68: Physical Sensors Present.....	92
Table 69: Physical Sensor Information .....	92
Table 70: Orientation Matrix Write Format.....	93
Table 71: Algorithm Parameters .....	94
Table 72: BSX State Exchange Structure .....	95
Table 73: BSX Version .....	95
Table 74: Virtual Sensor Information Structure .....	95
Table 75: Virtual Sensor Configuration Structure.....	96
Table 76: Physical Sensor Control Parameters .....	97
Table 77: Command Error Response .....	98
Table 78: FIFO Data Format .....	99
Table 79: Overview of FIFO Event IDs .....	101
Table 80: Virtual Sensor Event Format “Quaternion+” .....	103
Table 81: Virtual Sensor Event Format “Euler” .....	103
Table 82: Virtual Sensor Event Format “3D Vector” .....	104
Table 83: Virtual Sensor Event Format “Activity” .....	104
Table 84: Bitmap of activities.....	104
Table 85: Virtual Sensor Event Format for scalar data.....	105
Table 86: Virtual Sensor Event Format with no payload .....	105
Table 87: Overview of Meta Events .....	106
Table 88: Sensor Status Values .....	108
Table 89: List of Reset Causes.....	110
Table 90: Debug Data Format .....	110
Table 91: Flags in Debug Data Format.....	111
Table 92: Data Chronology after FIFO Overflow.....	115
Table 101: Absolute maximum ratings.....	118
Table 102: Operating conditions .....	118
Table 103: Electrical characteristics .....	118
Table 104: Operating conditions accelerometer .....	120
Table 105: Output signal accelerometer .....	120
Table 108: Fuser2 Power-Up and Power-Down Timing Characteristics.....	121

Table 109: Host I2C interface timing.....123  
Table 110: Host SPI Interface Timing Parameters in Long-Run (LR) and Turbo mode.....124  
Table 111: Master Interface SPI Timing Parameters for Long-Run (LR) and Turbo Mode ..125  
Table 113: BHA260AB Device Marking.....127  
Table 115: Document History .....140

# GENERAL DESCRIPTION

## 1 Overview

The BHA260AB belongs to the BHA260 family of ultra-low power programmable smart sensors. The BHA260AB integrates the Fuser2 core processor, which is based on the 32-Bit ARC™ EM4™ floating point RISC processor, an integrated Acceleration sensor (3DoF Accel) and a powerful Event-driven Software Framework specifically designed for signal data processing and comes with pre-installed sensor fusion and other sensor data processing algorithms.

The BHA260AB provides a variety of interfaces to connect sensors and other peripheral devices like GNSS receiver, Wi-Fi and Bluetooth radios. This makes the BHA260AB a full sensor subsystem and computing platform for always-on sensor data processing algorithms at lowest power consumption.

The BHA260AB offers 2 secondary high speed master interfaces with I2C and/or SPI capability and up to 12 GPIOs which can be configured in a flexible way (e.g. as Chip Select or Interrupt lines).

The BHA260AB is intended to be used as coprocessor offloading the main CPU from any sensor data processing related tasks, like sensor fusion, data batching, position tracking, activity recognition and gesture detection with high precision and low latency while significantly reducing the overall system power consumption. The BHA260AB supports a wide variety of host CPU devices, ranging from a small MCU up to multicore application processors. The BHA260AB communicates with the Host CPU through its primary high speed I2C or SPI interface.

The BHA260AB can be used as co-processor in different applications:

- **As a high end Smart Sensor with integrated features:**

The BHA260AB can be used as a ready to use solution without the need of further programming. The BHA260AB typically comes with more than 20 integrated virtual sensors depending on the firmware loaded. These are independently configurable sensor data processing algorithms that offer software programmed features like device orientation in many different formats, gesture recognition algorithms, smart dynamic offset calibration, activity recognition and step counting. Bosch Sensortec offers Firmware versions for a variety of use cases (e.g. wearables, hearables, etc.)

- **As an open programmable Smart Sensor:**

In addition to the integrated algorithms (here called “Virtual Sensors”), the BHA260AB functionality can be customized and extended by the customer. Bosch Sensortec offers a Software Development Kit (SDK) that enables the development of additional algorithms (virtual sensors) for the BHA260AB.

In order to make it easier to develop efficient sensor data processing algorithms, a sophisticated Event-driven Software Framework, provides powerful sensor management services including power management, FIFO management, event and time synchronization.

## 2 Hardware Features

### 2.1 CPU Core (Fuser2)

- 32-bit Synopsys DesignWare™ ARC™ EM4 CPU
- ARCV2 16/32-bit RISC instruction set architecture
- CPU provides up to 1.6 DMIPS/MHz, 3.6 CoreMark/MHz
- Operating frequency: 20 MHz (Long Run mode) and 50 MHz (Turbo mode)
- Maximum CPU power consumption in Long Run mode: 950  $\mu$ A (47.5  $\mu$ A/MHz)
- Maximum CPU power consumption in Turbo mode: 2.8 mA (56  $\mu$ A/MHz)
- Harvard architecture with closely coupled memories for instructions (ICCM) and data (DCCM)
- Single-precision FPU with IEEE 754 compliance
- Memory protection unit (MPU)
- 4-channel micro-DMA controller
- Timer and core watchdog
- Action-point support for debugging
- Hardware support for fast CRC32 calculation

### 2.2 System Control blocks

- Brown Out reset
- Voltage regulator for digital core
- On chip system oscillator: 20 MHz, 50 MHz
- On chip timer oscillator: 128 kHz

#### 2.2.1 Reset options

Available reset sources:

- Power On Reset
- Reset Pad
- Host Reset Command
- Core Watchdog

#### 2.2.2 Oscillators & Clocks

Available clock domains <sup>1)</sup>:

- Host interface clock (up to 50 MHz in SPI mode)
- System clock (20 MHz or 50 MHz) provided by internal system oscillator
- Timer clock (64 kHz derived from 128 kHz internal timer oscillator)
- External clock (from GPIO for the universal timer; up to 1 MHz)
- JTAG debug clock (up to system clock / 8)

Note:

1) These clock domains are asynchronous to each other

### 2.3 Host Interface

- Pin-configurable as either I2C (up to 3.4 MHz) or SPI (up to 50 MHz) slave interface
- I2C slave interface monitored by I2C watchdog

- Configurable (latched/non-latched, push-pull or open drain) host interrupt output
- Dual DMA enabled command I/O FIFO for efficient command and status handling
- Dual data FIFO sensor event batching

## 2.4 Memory Subsystem

### 2.4.1 On-Chip SRAM

256 kByte of SRAM in total, organized 9 individual RAM banks

- 16 kByte dedicated to ICCM space
- 16 kByte dedicated to DCCM space
- 7x32 kByte RAM banks in a RAM pool, configurable to either ICCM space, DCCM space or powered off

### 2.4.2 ROM with integrated Software

144 kByte of program ROM in fast-access ICCM space, containing:

- Bootloader
- BSX sensor fusion library
- Functions of standard C and math library
- OpenRTOS kernel
- SHA256 and ECDSA (NIST FIPS PUB 186-4) digital signature libraries

### 2.4.3 OTP

- 128 Byte OTP memory for Bosch internal factory calibration and secure boot key storage (not customer programmable)

## 2.5 Peripheral Subsystem

### 2.5.1 Secondary Master Interface 1

- Master Interface 1 (M1) configured as SPI for connection with integrated Accelerometer.
- Integrated Acceleration sensor attached to M1, default mode is SPI at 10 MHz
- Master Interface 1 is solely used for internal connection of integrated Accelerometer and not available for connections to external devices

### 2.5.2 Secondary Master Interface 2

- Master Interface 2 (M2) configurable as I2C (up to 1 MHz) or SPI (up to 50 MHz) for connection of additional physical sensors to the BHA260AB (see Configuration of Master Interfaces)
- Various chip selects can be associated to M2 in SPI mode
- For more Information about the Master Interface Configuration, see Section 10.1 *Configuration of Master Interfaces*

### 2.5.3 Secondary Master Interface 3

- Master Interface 3 (M3) configurable as I2C (up to 1 MHz) for connection of additional physical sensors to the BHA260AB (see Configuration of Master Interfaces)
- For more Information about the Master Interface Configuration, see Section 10.1 *Configuration of Master Interfaces*

## 2.5.4 GPIOs

- 12 software configurable GPIOs available for use as CS lines, Interrupt lines or other purposes (see General-Purpose Inputs/Outputs).
- Supported configurations:
  - Input: internal configurable Pull-Ups, High-Z, selectable as event interrupt source
  - Output: Push/Pull, High-Z, Open Drain, selectable drive strength
- For more Information about the GPIOs Configuration, see Section 10.2 *Event and Interrupt Configuration*

## 2.5.5 Universal Timers & Counter

- One 32-bit real-time counter incremented by the Timer Clock running at 64 kHz (used for time stamp generation)
- One 32-bit interval timer for generation of periodic interrupts, software counters, etc. It is driven by the Timer clock running at 64 kHz
- One 16-bit universal timer with input-capture and output-compare capability
- Software API to access/program all timers & counter

## 2.5.6 Interrupt Sources

Various sources can generate interrupt requests to the MCU in the BHA260AB (Fuser2 Core) for which interrupt handlers can be registered.

For some interrupt sources, default handlers are already registered to ensure proper operation of the Event-driven Software Framework.

Interrupt requests can originate from:

- Host interface
  - Host writes or read into specific host interface registers
  - Host interface detects end of data transmission
  - Buffer overflow or underflow during data transmission
- I2C and SPI master interfaces
  - Master interface signals end of a data transmission
  - GPIO events

For more detailed information about the interrupt configuration, see section 11 *Event and Interrupt Configuration*.

## 2.5.7 Event Subsystem

- Advanced event subsystem supporting effort free time synchronization between external events and internally handled (sensor) data
  - Up to 10 event channels, which can be mapped to a variety of GPIO pins
  - Any GPIO, associated to an event channel, can serve as external interrupt source
  - Event triggered hardware logic for capture and storage of real-time counter value for automatic time synchronization between external events and internal time base
- For more detailed information about the interrupt configuration, see section 11 *Event and Interrupt Configuration*.

## 2.6 Debug Interface

- 2-wire JTAG interface



- Supports direct read/write of CPU data and aux registers as well as ICCM/DCCM memory locations
- Hardware breakpoint and single step execution
- Supported with Metaware Debugger for ARC and OpenOCD for GCC

## 2.7 Integrated physical Sensors

- Low power, low noise 16 bit triaxial accelerometer
- Supports data rates up to 1600 Hz
- Built in power management unit and enhanced interrupt engine (for Fuser2 wake-up)
- Auxiliary master interface
  - For direct magnetometer attachment (upgrade to 6DoF Compass module )

### 3 Pad connections and description

#### 3.1 Pad description

Figure 1: Pad connections

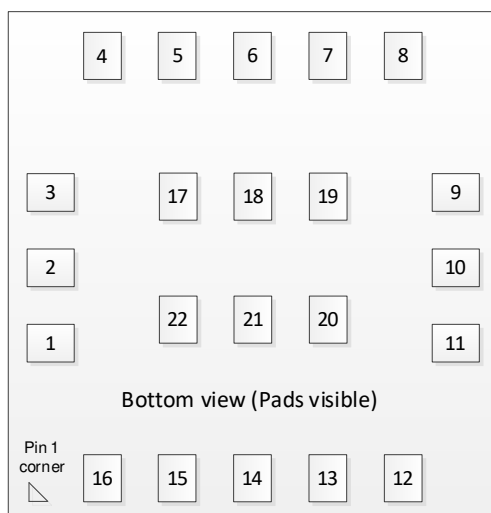


Table 1: Pad connections

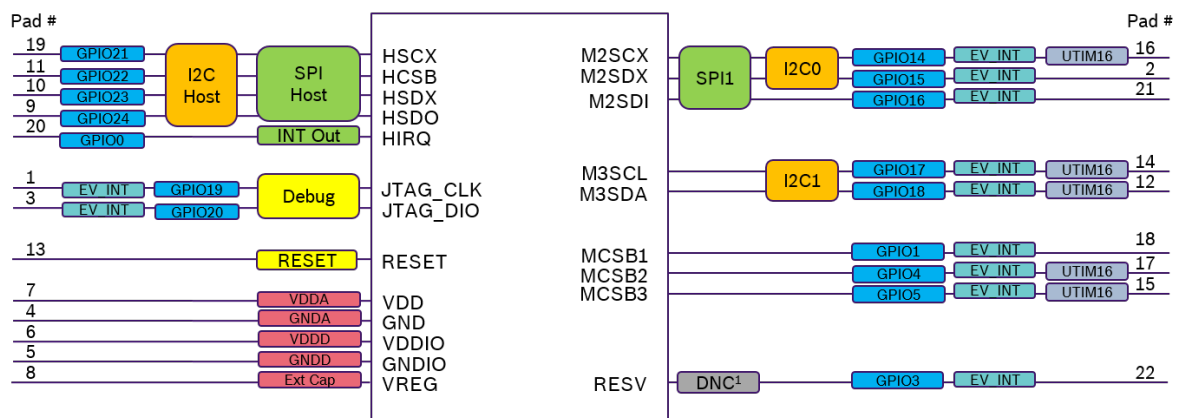
Pad #	Pad Name	Function Group	Reset Value	Description
1	JTAG_CLK	Debug	Input, Pull-Up	Debug Clock
2	M2SDX	Master Interface 2	Input, Pull-Up	M2: SPI MOSI / I2C SDA
3	JTAG_DIO	Debug	Input, Pull-Up	Debug Data
4	GND	Ground	-	Analog Sensor Ground
5	GNDIO	Ground	-	Digital IO and Fuser Ground
6	VDDIO	Supply	-	Digital IO and Fuser Supply
7	VDD	Supply	-	Analog Sensor Supply
8	VREG	Supply	-	Voltage regulator output
9	HSDO	Host Interface	Input, Pull-Down	Host Interface: SPI MISO / I2C address select
10	HSDX	Host Interface	Input, Pull-Up	Host Interface: SPI MOSI, I2C SDA
11	HCSB	Host Interface	Input, Pull-Up	Host Interface: SPI Chip select / Protocol Select
12	M3SDA	Master Interface 3	Input, Pull-Up	M3 I2C SDA
13	RESETN	System Control	Input, Pull-Up	Reset input, active low
14	M3SCL	Master Interface 3	Input, Pull-Up	M3 I2C SCL
15	MCSB3	GPIO	Input, Pull-Up	SPI Chip Select 3
16	M2SCX	Master Interface 2	Input, Pull-Up	M2: SPI SCK / SCL

17	MCSB2	GPIO	Input, Pull-Up	SPI Chip Select 2
18	MCSB1	GPIO	Input, Pull-Up	SPI Chip Select 1
19	HSCX	Host Interface	Input, Pull-Up	Host Interface: SPI SCK / I2C SCL
20	HIRQ	Host Interface	Input, Pull-Up	Host Interface Interrupt Output
21	M2SDI	Master Interface 2	Input, Pull-Up	M2: SPI MISO / unused
22	RESV	Reserved	Input, Pull-Up	Do Not Connect (internal Accel interrupt 2)

### 3.2 Setup of Multifunction pins

Almost all pads of the device can have multiple functions assigned. Next to their primary function (Host Interface, Master Interfaces, Debug, OIS and Aux Interface), most pads can also serve as General Purpose IOs (GPIOs), as Event Interrupt Inputs, or as timer/capture inputs/outputs. The following diagram depicts the multiple assignments of the pads.

Figure 2: Pinout BHA260AB



**Note:**

- 1) It is recommended not to use the RESV pad, since it is used for internal connections between the sensor and Fuser2. This pad is marked therefore as DNC (Do not connect) for function 1.
- 2) The BHA260AB host interface is required to boot the device. After successful boot, the associated pads can be used as GPIOs, if the host connection is not needed anymore

The selection of the function of each pad depends on the boot state of the device and the configuration within the firmware. The following table describes the available functions of each pad and how they are configured.

Table 2: Pad functions and configuration options

Pad #	Pad Name	Primary Function Group	Function 1	Function 2	Function 3	Function 4		Function 5	
			SPI etc		GPIO	Event Interrupt <sup>3)</sup>		Universal Timer	
						Config 1	Config 2	Config 1	Config 2
13	RESETN	System Control	Fuser2 Reset, active low						
19	HSCX		Host SPI SCK	Host I2C SCL	GPIO21 <sup>1)</sup>				

11	HCSB	Host Interface <sup>1)</sup>	Host SPI chip sel	Protocol sel: keep high for I2C	GPIO22 <sup>1)</sup>				
10	HSDX		Host SPI MOSI	Host I2C SDA	GPIO23 <sup>1)</sup>				
9	HSDO		Host SPI MISO	Host I2C adr0	GPIO24 <sup>1)</sup>				
20	HIRQ		Host interrupt <sup>2)</sup>	Host interrupt <sup>2)</sup>	GPIO0				
16	M2SCX	Master Interface 2	SPI SCK	I2C SCL	GPIO14		EVINT4		
2	M2SDX		SPI MOSI	I2C SDA	GPIO15		EVINT7		
21	M2SDI		SPI MISO		GPIO16	EVINT11		IN	
14	M3SCL	Master Interface 3	I2C SCL		GPIO17		EVINT8		OUT
12	M3SDA		I2C SDA		GPIO18		EVINT9		IN
18	MCSB1	Chip Selects/ Event Interrupts <sup>2)</sup>	SPI chip select 1		GPIO1		EVINT1		
17	MCSB2		SPI chip select 2		GPIO4	EVINT2	EVINT3		STRT
15	MCSB3		SPI chip select 3		GPIO5	EVINT3	EVINT2	STRT	CAPT
22	RESV		(Integrated sensor IRQ 2)		GPIO3	EVINT1			
1	JTAG_CLK	Debug <sup>1)</sup>	JTAG clock		GPIO19		EVINT10		
3	JTAG_DIO		JTAG data inout		GPIO20		EVINT11		

**Notes:**

- 1) Function is locked after initial selection. This is not valid for HIRQ.
- 2) Function 1 is defined by firmware. In hardware these pins are standard GPIOs.
- 3) GPIO must be configured for correct direction – event interrupts are inputs

**Table 3: Pad functions and init conditions**

Pad #	Pad Name	Primary Function Group	Function after (refers to functions in Table 2)		
			Reset	Bootloader completed	Firmware load
13	RESETN	System Control	1	1	1
19	HSCX	Host Interface <sup>1)</sup>	2	1 or 2: defined by HCSB	1 or 2: defined by HCSB
11	HCSB				
10	HSDX				
9	HSDO				
20	HIRQ		3		
16	M2SCX	Master Interface 2	3	3	defined by FW: sif_cfg=0: 1 sif_cfg=1: 2 sif_Cfg=2: 2
2	M2SDX				
21	M2SDI				
14	M3SCL	Master Interface 3	3	3	3 or 1 Defined by FW
12	M3SDA				
18	MCSB1	Chip Selects / Event Interrupts <sup>2)</sup>	1	1	Defined by FW
17	MCSB2				
15	MCSB3				
22	RESV				
1	JTAG_CLK	Debug <sup>1)</sup>	3	1 or 3: Defined by product type	1 or 3: Defined by FW
3	JTAG_DIO				

**Notes:**

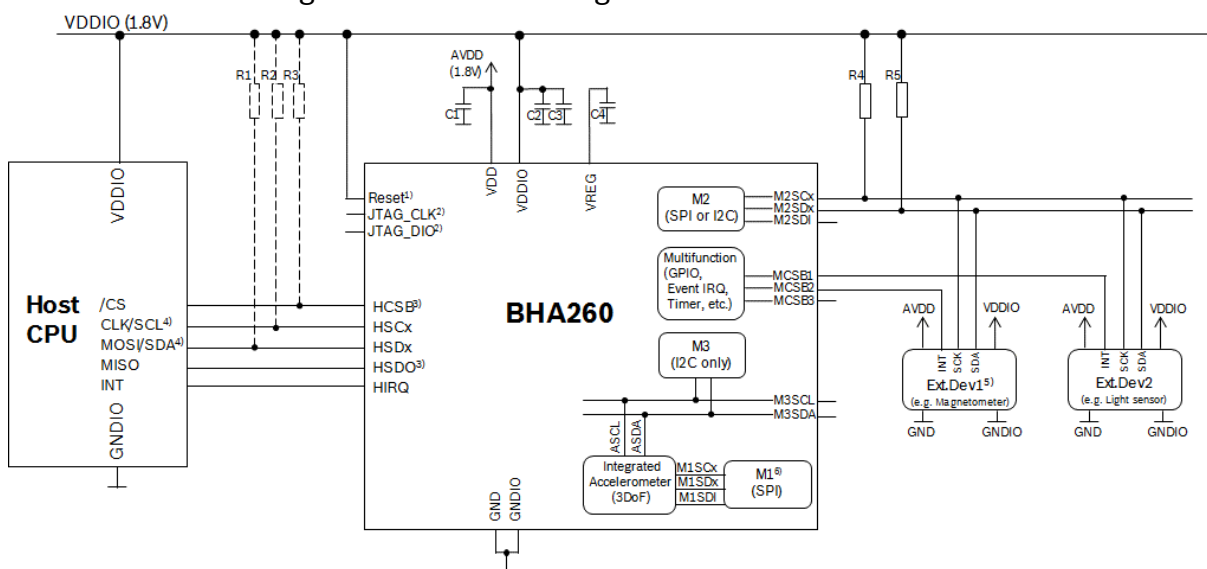
- 1) Function is locked after initial selection. This is not valid for HIRQ.
- 2) Function 1 is defined by firmware. In hardware these pins are standard GPIOs.

## 4 System Configuration

### 4.1 Connection Diagram

Figure 3 shows a possible connection of BHA260AB in SPI host protocol mode, with one external I2C sensor bus. For using I2C host protocol instead of SPI, the dashed pull-up resistors have to be used, HSDO and HCSB line need to be connected according to note 3).

Figure 3: Connection diagram sketch for BHA260AB



Notes:

- 1) Recommendation: Connect to VDDIO, if not needed
- 2) Leave open, if not needed
- 3) When using host interface in I2C mode: connect R1...R3 as indicated; do not apply host connection to HCSB and HSDO; HSDO can be used for device address selection, if needed
- 4) Host signal name depends on SPI/I2C configuration
- 5) Example configuration shows one external I2C bus on M2. Different configurations, e.g. using M2 in SPI mode, are also possible
- 6) M1 bus not externally accessible, only used for internal connection of accelerometer

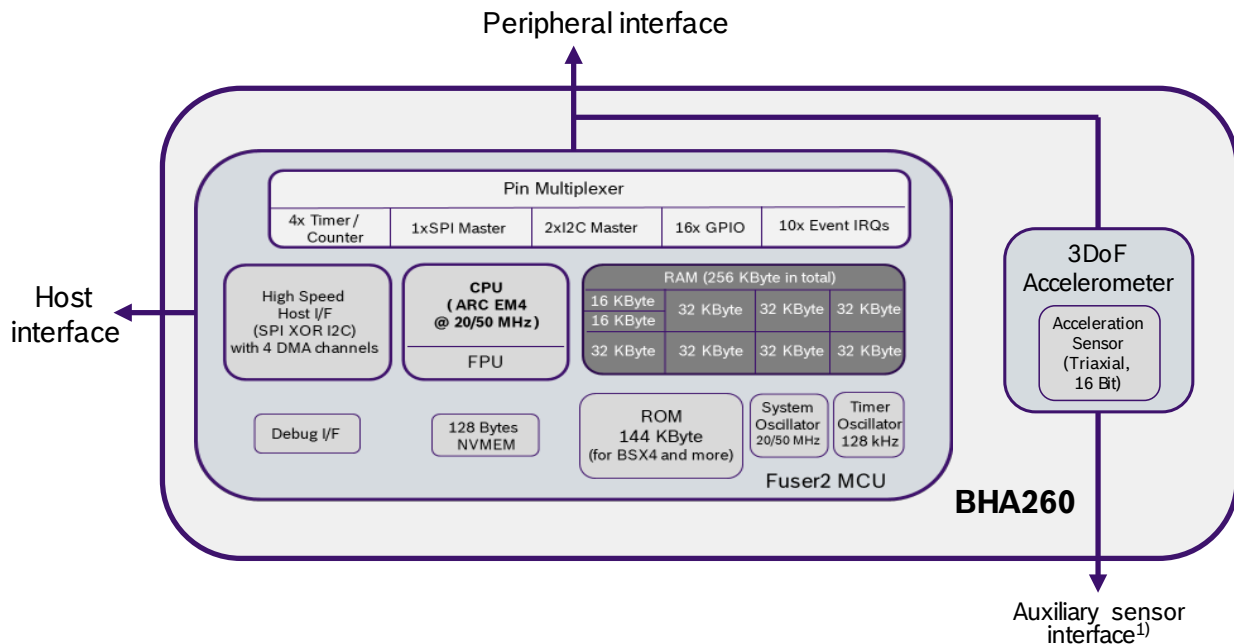
Table 4: Typical values for external circuit components

Component	Value	Remarks
R1	4.7 kΩ	Pull-Up resistor for SDA, only for host interface in I2C mode
R2	4.7 kΩ	Pull-Up resistor for SCL, only for host interface in I2C mode
R3	0 Ω	Only for host interface in I2C mode (connect to VDDIO)
R4	4.7 kΩ	Pull-Up resistor for M3SCL, only for M3 interface configured in I2C mode
R5	4.7 kΩ	Pull-Up resistor for M3SDA, only for M3 interface configured in I2C mode
C1	100 nF	Bypass capacitor AVDD to GND
C2	100 nF	Bypass capacitor VDDIO to GNDIO
C3	1 μF	Bypass capacitor VDDIO to GNDIO

C4	220 nF	Regulator capacitor VREG
----	--------	--------------------------

## 4.2 Block Diagram

Figure 4: Block diagram of BHA260AB



## 4.3 Physical Primary Host Interface

The BHA260AB provides a high-speed data interface and a single interrupt line (the Host Interrupt Signal HIRQ) as main interface to the host processor.

The host interface can be operated either in I2C or SPI mode. The default protocol after power-up or reset is I2C. Pulling the HCSB (Host Chip Select) line to low at any point in time switches the host interface into SPI mode, where it remains until reset or a new power-on cycle.

In both interface modes, all data is treated MSbit first.

The host interface provides access to the host register map of BHA260AB, which provides all necessary functionality to operate the device.

### 4.3.1 Host interrupt

The interrupt line is implemented based on one of the device's GPIO pins (GPIO0) and therefore as flexible in its configuration options (e.g. active high, active low, open-drain or push pull, level or pulse, with selectable drive-strength) as any other GPIO pin.

When using a default firmware image, the interrupt line is configured to be active high, push pull, level, low drive strength.

*Note:*

*It is also possible to operate the host interface in polling mode and re-use the interrupt line as GPIO pad.*

### 4.3.2 Operation in I2C mode

In I2C mode, the host interface is implemented as an I2C slave interface as described in the I2C bus specification created from NXP (see Reference 1 in Section 26) and implements data transfer rate up to 3.4 Mbit/s in high-speed mode.

The I2C bus consists of two wires, HSCX (Host Serial Clock) and HSDX (Host Serial Data). For connections to the host, both bus lines must be externally connected to a positive supply voltage (VDDIO) via pull-up resistors or current source.

A data transfer is always initiated from the host. The BHA260AB can operate as either a transmitter or receiver only, if a valid device address has been received from the host. The valid device address can be selected by adjusting the state on the HSDO pin as:

Table 5: I2C device address selection options

HSDO	I2C device address
HIGH	0x29
LOW	0x28

All I2C transactions start with the host sending a START bit followed by the slave device address (7-bit only) and the read-write indication bit, where a logical '0' (LOW) defines the transaction as a write. If the received slave device address matches the defined device address, the BHA260AB is selected (i.e. ready for communication) and responds with an ACK (driving HSDX low). In case on a non-matching device address, the BHA260AB is not selected and responds with a NAK (leaving HSDX high).

After the ACK, the host must provide the host interface register address next and so the very first host transaction must be a write operation. The MSbit of the register address is ignored. The BHA260AB always responds with an ACK (even for reserved registers). If the host wishes to write into the selected internal register, it has to append the write data to the register address within the same transaction.

The host can read the selected internal register in one of two ways:

- in the combined format (i.e. issue a repeat-START bit, slave address and read indication)
- in a separate transaction (i.e. issue a STOP bit and start a new read transaction with START bit, slave device address and read indication).

During multiple writes, the BHA260AB acknowledges all data bytes with an ACK. During reads, the host responds with an ACK to all data bytes except the last one, which is acknowledged with NAK. Write and read transactions are illustrated in the figures below. All communication is MSbit first. The I2C host interface does not stretch the clock.



Figure 5: I2C Write Transaction

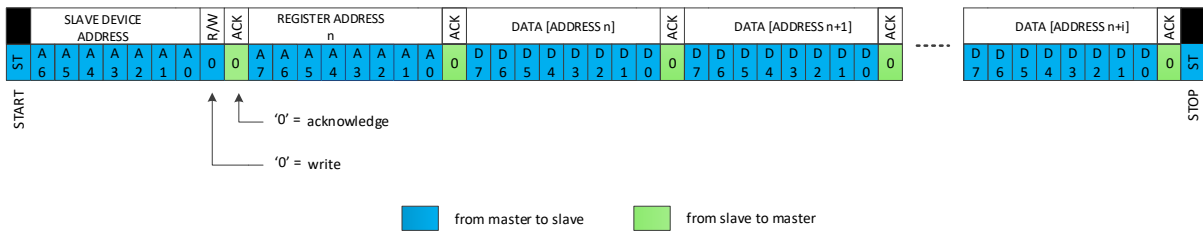


Figure 6: I2C Read Transaction – Combined Format (with Repeat START)

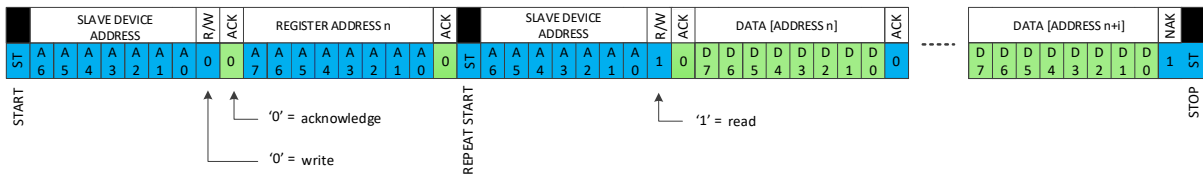


Figure 7: I2C Read Transaction - Split Format (1)

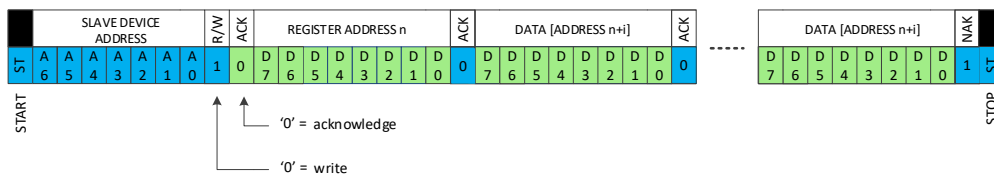
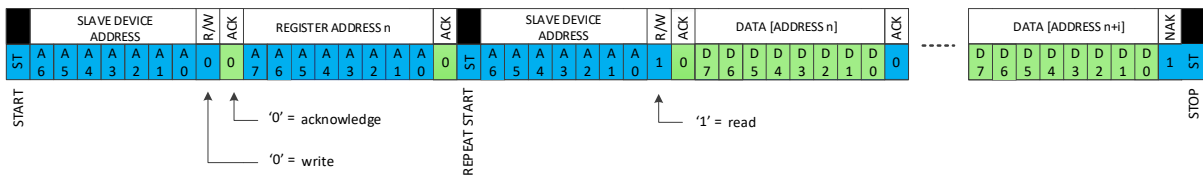


Figure 8: I2C Read Transaction - Split Format (2)



An I2C watchdog is available inside BHA260AB to monitor the activity of the I2C protocol engine. It can be enabled and configured via the Host Control register in Section 12.1.12.

When the I2C watchdog is enabled, if a transaction is in progress but the I2C protocol engine does not carry out any transaction activity for a configured amount of time (either 1ms or 50ms), the transaction is considered to be stalled and the I2C protocol engine aborts the stalled transaction by assuming idle state. Conditions that may cause a stalled transaction include:

The I2C master is reset during a transaction and the clock stops while the slave was transmitting a '0'.

The slave is transmitting data; Due to spurious clocks, it gets out-of-sync with the master and waits for an acknowledgement that never arrives.

### 4.3.3 Operation in SPI mode

In SPI mode, the host interface is implemented as an SPI slave interface and implements data transfer rate up to 20 Mbit/s (in Long Run mode) or up to 50 Mbit/s (in Turbo mode).

By default, the SPI interface consists of four wires (4-wire SPI protocol), an active-low chip select HCSB (Host Chip Select), a clock line HSCX (Host Serial Clock), a data input line HSDX (Host Serial Data) and a data output line HSDO (Host Serial Data Output).

The SPI interface can be configured to operate also only with three wires (3-wire SPI protocol), where the HSDX line is used as bidirectional data line while the HSDO line is omitted. In order to enable this mode, the respective bit in the Host Control register needs to be set.

A SPI transaction starts with the host driving the HCSB to logical '0' (LOW). The host first sends the internal register address to be accessed, where the first bit (MSbit A7) indicates whether the access is intended to be a read or write access (with A7='0' indicating a write and A7='1' indicating a read) and the following 7 bits specify the actual address of the register.

In case of a write access the host has to follow the address byte with the data byte on the HSDX line within the same sequence.

In case of a read access, the BHA260AB provides data with each additional clock pulse on HSCX either on the HSDO line (4-wire SPI) or on the HSDX (3-wire SPI) depending on the SPI mode setting in the Host Control register.

If the sequence is interrupted, by pulling HCSB to HIGH, the next SPI transaction, starting with a new HCSB falling to LOW, will require a new register address transmission, as described above, before any data can be read or written successfully.

Two CPOL/CPHA configurations are supported: '0/0' and '1/1'

- With CPOL=CPHA=0 HSCX has to be LOW, when a sequence is initiated with a falling edge of HCSN.
- With CPOL=CPHA =1 HSCX has to be HIGH, when a sequence is initiated with a falling edge of HCSN.

In both supported configurations, data is presented on the falling edge of the clock and sampled on the rising edge.

The next three figures illustrate the protocol. The corresponding timing parameters are listed in Section 19.5.1.

Figure 9: SPI write transaction in 4-Wire and 3-Wire mode

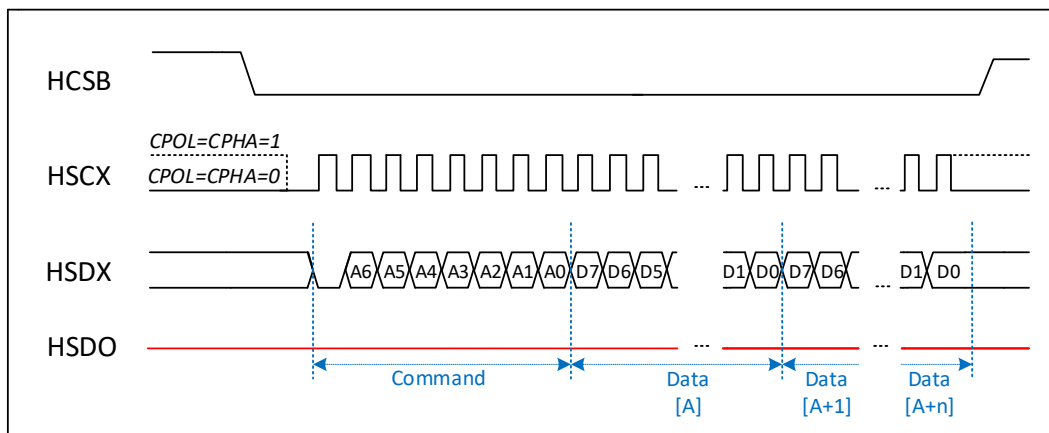


Figure 10: SPI read transaction in 4-wire mode

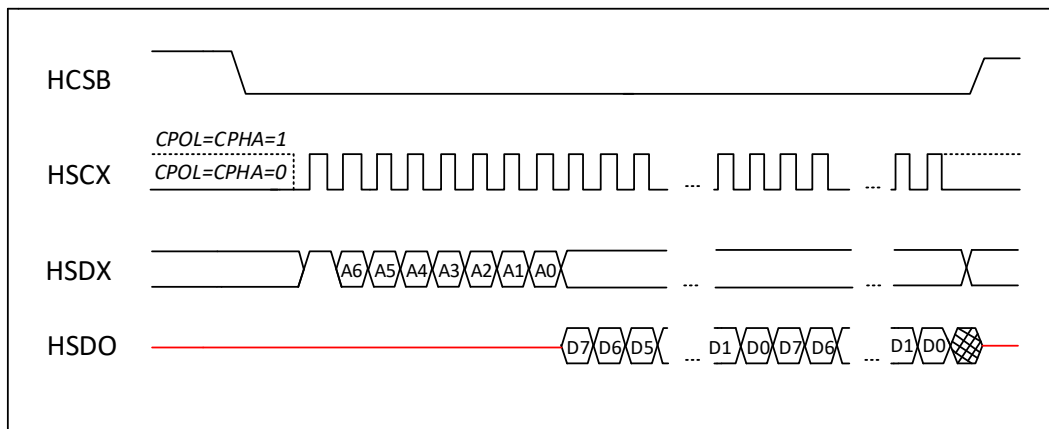
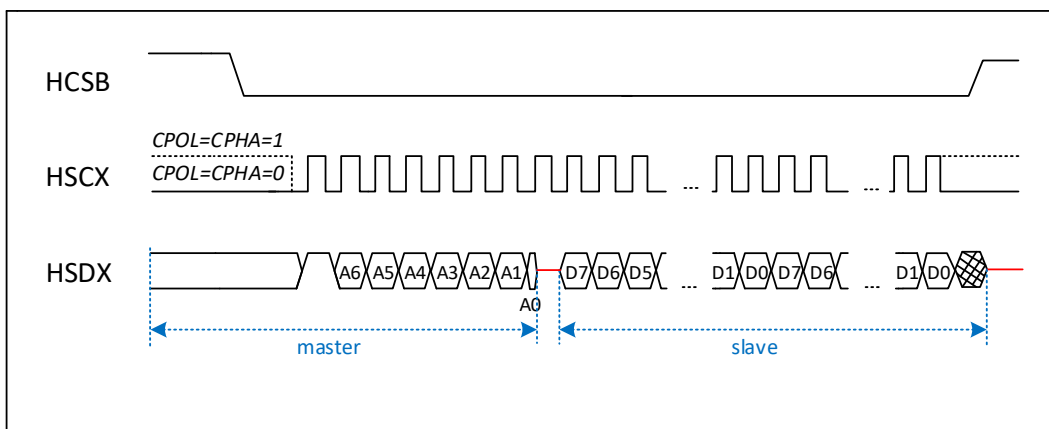


Figure 11: SPI read transaction in 3-wire mode



### 4.3.4 Burst mode operation

A data transmission sequence can cover the read/write transmission of multiple data bytes. In this burst mode operation, the transmission of multiple bytes is supported by two different features of the BHA260AB.

#### 4.3.4.1 Burst mode operation on DMA enabled registers

The register addresses 0x00:0x03 are linked to four individual DMA engines (one per each address), which allow for the efficient transfer of bigger data portions between the BHA260AB and the host, by directly mapping these register addresses to defined memory regions of the BHA260AB internal closely coupled memories (ICCM and/or DCCM).

The DMA engine on address 0x00 is designed to enable write operations into the BHA260AB, while the engines on the other three addresses are designed to enable read operations from the BHA260AB device.

The usage of the four channels is described below.

Table 6: Overview DMA channels

Register Address	DMA Channel	Operation Mode	Function
0x00	0	Write	Send Commands to the BHA260AB with variable length (including firmware upload)
0x01	1	Read	FIFO for reading Wake-Up sensor data
0x02	2	Read	FIFO for reading Non-Wake-Up sensor data
0x03	3	Read	FIFO for reading Status and Debug information

Reading more bytes from a channel than available results in zeros being read.

Writing more bytes than expected is considered as an overflow and discards the whole transmitted data within this session.

#### 4.3.4.2 Burst mode operation on regular registers

Burst mode operations to all other registers are supported by auto-incrementing the register address so that consecutive registers can be read or written within one sequence.

Reading or writing to reserved registers can cause unpredictable effects and thus shall be avoided.

## 4.4 Host Data Interface

### 4.4.1 Host Data Access

The BHA260AB can be controlled entirely by the host through reading and writing its host interface (HIF) registers. This can be for example, booting the device, resetting the device, enabling/disabling virtual sensors, changing parameters at runtime.

These registers are summarized in Table 7 and described in detail in Section 12. *Host Interface Register Map*.

Figure 12: Overview host data interface

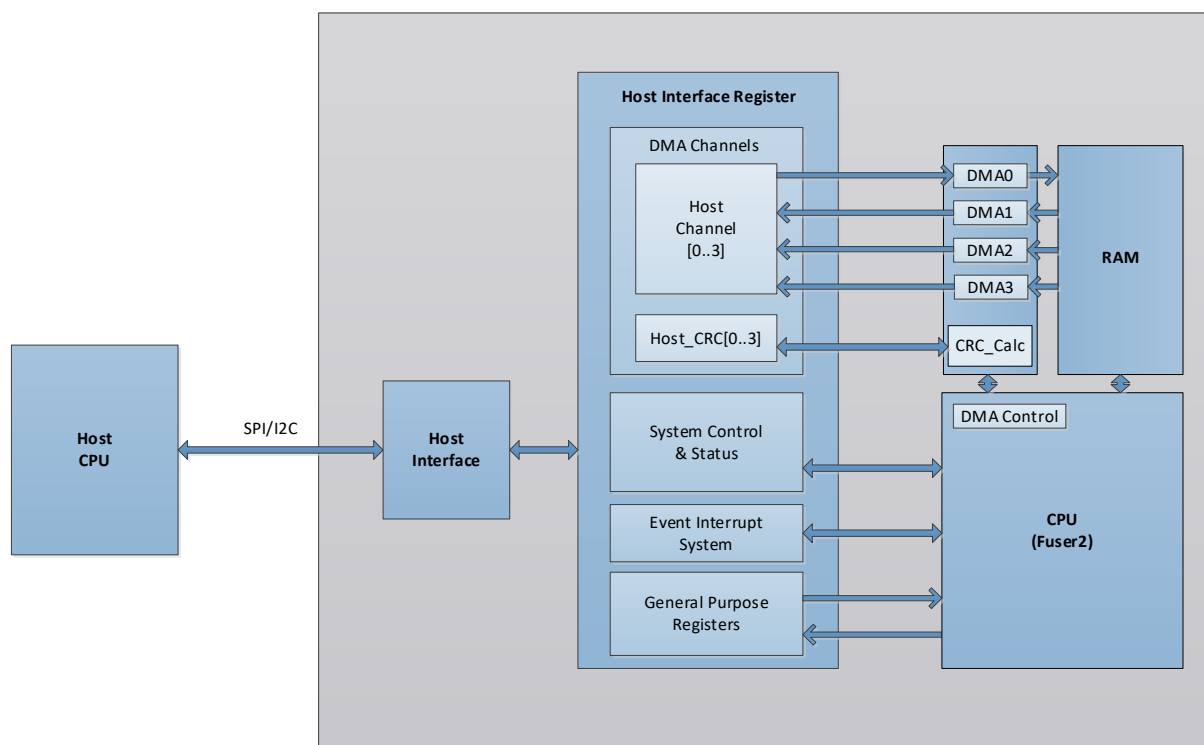


Table 7: Basic overview of the BHA260AB host interface registers

Section	Host Register Name	Address	Main Function
DMA Channels	Host_Channel[0]	0x00	Input for the Command interface
	Host_Channel[1]	0x01	Output for wake-up virtual sensor data
	Host_Channel[2]	0x02	Output for non-wake-up virtual sensor data
	Host_Channel[3]	0x03	Output for the Command interface & Debug
	Host_CRC[0..3]	0x18 – 0x1B	Calculated CRC for last channel used
System Control & Status	Chip Control	0x05	Basic Chip Control (debug, turbo-mode)
	Host Interface Control	0x06	Host interface & DMA Control
	Host Interrupt Control	0x07	Enable & Disable Interrupts
	Reset Request	0x14	Host controlled Software reset
	Host Control	0x16	SPI & I2C mode settings
	Host Status	0x17	Host interface status information
	Fuser2_Product_ID	0x1C	Fuser2 Product specific number
	Fuser2_Revision_ID	0x1D	Fuser2 Revision specific number
	ROM Version[0..1]	0x1E – 0x1F	16bit ROM image revision
	Kernel Version[0..1]	0x20 – 0x21	16bit kernel image revision
	User Version[0..1]	0x22 – 0x22	16bit user image revision
	Feature Status	0x24	Feature description
Boot Status	0x25	Boot status description	
Chip ID	0x2B	Product specific number	
Interrupt Status	0x2D	Interrupt status info	
Internal Debug Registers	0x2E-0x31	Debug & Post mortem infos	

Event Interrupt System	Timestamp Event Request	0x15	Host controlled software event interrupt request
	Host Interrupt Timestamp[0..4]	0x26-0x2A	Timestamp of last host interrupt (can be configured as software event interrupt response via Host Interface Control Register)
General Purpose Registers	Host_Gen_Purpose	0x08 – 0x13	General Purpose Read/Write register
	Proc_Gen_Purpose	0x32 – 0x3D	General Purpose Read only register
Reserved	Reserved Registers Reserved Reserved	0x04, 0x2C 0x3E – 0xFF	Do not use

#### 4.4.2 Host Command Protocol

When it comes to operate and configure the BHA260AB, the main communication mechanism between the host and the BHA260AB firmware is the Host Command Protocol. For example, it is used to initially upload a firmware to the device or to configure and enable / disable virtual sensors.

The Host Command Protocol is realized by sending command packets to the *Host Channel 0 - Command Input (0x00)* and reading back the status packets out of the *Host Channel 3 - Status and Debug FIFO Output (0x03)*, if any are available for that specific command.

There are several different commands, which serve different purposes, all are summarized and explained in Table 31 in Section 13. *Host Interface Commands*.

The overall structure of a Command Packet is always the same: 2 bytes command + 2 bytes length + (optionally) 4 bytes or more parameters or data. This is shown in the *Table 8: Structure of Command Packet*.

The structure for the Status Packet is shown in *Table 9: Structure of a Status Packet*.

The size of a command packet must be a multiple of 4 bytes. If the command length does not end on a 4-byte boundary, the command must be padded out to the 4-byte boundary with zeroes. These padded bytes must be included in the Length field of the command structure.

Table 8: Structure of Command Packet

Field Name	Byte Offset	Description
Command ID	0x00-0x01	2 byte command, LSB first
Length	0x02-0x03	Command content's length in bytes, LSB first
Contents	0x04.. Length + 0x03	Optional parameters or data

In response to certain commands, a status packet of the following format will be placed into the output channel 3 (Status and Debug FIFO):

Table 9: Structure of a Status Packet

Field Name	Byte Offset	Description
Status Code	0x00-0x01	2 byte status, LSB first
Length	0x02-0x03	Status Contents length in bytes, LSB first
Contents	0x04.. Length + 0x03	Optional parameters or data

A list of all possible status packets are given in the Table 31 in *Section 13. Host Interface Commands*. All status packets' size will be a multiple of 4 bytes.

## 5 Device Initialization and Start-up

### 5.1 Power On and Reset

The BHA260AB has four possible reset sources described in the following table:

Table 10: Available reset sources

Reset	Source	Suppres sible	Reset is effective on
Power-On Reset	Analog Power Management Unit	No	All Fuser2 blocks
External Reset	External pad RESETN, active low	No	All Fuser2 blocks
Host Reset	Host write access to register Reset Request (0x14)	No	All Fuser2 blocks except 2-wire JTAG debug interface
Core Watchdog Reset	Core watchdog	Yes <sup>1)</sup>	All Fuser2 blocks except 2-wire JTAG debug interface

Note:

1) Either “Core Watchdog disabled” or “JTAG enabled” suppresses the Core Watchdog reset.

Due to an integrated power-on reset controller with brown-out detector, it is possible to reset the BHA260AB without an external reset circuitry. The External Reset mechanism can be used if the host requires to trigger of a hard reset without relying on the operation of the host interface. For instance, a dedicated GPIO of the host can be connected to the RESETN pad. Optionally, a button with a pull-up circuitry maybe used. If not used, the RESETN pad should be connected to VDDIO.

After Reset, the bootloader in ROM is executed.

### 5.2 BHA260AB Initialization and Boot mode

Before the BHA260AB can operate as a smart sensor, an initialization procedure has to be completed. During this initialization, the appropriate FW image will be loaded to the BHA260AB.

An external Host processor will trigger and control the initialization procedure and will load the FW image to the BHA260AB by writing the appropriate file over the primary host interface.

The BHA260AB requires a firmware to operate. It is the responsibility of the integrated bootloader in the ROM to make this firmware available to the processor and to verify it.

#### 5.2.1 BHA260AB Initialization

An external Host processor will trigger and control the initialization procedure, including loading the appropriate FW image to the BHA260AB.

After initialization the boot loader will set the Host Interface Ready bit in the Boot Status register to 1, in order to indicate that it is ready to receive a firmware upload via the host interface.

The sequence below should be used to load the firmware and start the device:

1. Reset the Fuser2 Core with a Host Reset as specified in *Table 10: Available reset sources*.



2. Optional: Write the register *Host Control (0x16)* to select 3-wire SPI mode and/or I2C watchdog mode and time out.
3. Write the register *Host Interrupt Control (0x07)* to configure the type of host interrupt signaling desired (see *Section 12.1.8* for details).
4. Optional: Write the *Chip Control (0x05)* register to set the CPU Turbo Disable bit as appropriate (0 = fastest firmware verification, highest power consumption, running at 50MHz; 1 = slower , lower power consumption, running at 20MHz). See *Section 12.1.6* for details.
5. Poll the Boot Status register or wait for T\_boot\_bl\_host time (see Table 108 in Section 19.5.1) until the Host Interface Ready bit is set.
6. Optional: Read the registers: *Fuser2 Product Identifier (0x1C)*, *Fuser2 Revision Identifier (0x1D)*, and *ROM Version (0x1E-0x1F)* to identify the revision of Fuser2 and select the proper Firmware image to load.
7. Load the Firmware image to the BHA260AB by issuing the Bootloader Command “*Upload to Program RAM (0x0002)*” to the register *Host Channel 0 - Command Input (0x00)* using multiple of 4 Bytes during each write transaction.  
 Note: When uploading the firmware using multiple write transactions, only the first one shall include the Upload to Program RAM (0x0002) command. Subsequent transactions shall include firmware data only. The initial command includes the total overall length of the firmware image in words (bytes divided by 4), regardless of whether the entire firmware image will be written with one or more write transactions. Each Write to the Command Interface register shall contain one or more 4 byte packets.
8. Poll the register *Boot Status (0x25)* until the *Firmware Verify Done* bit is set or the *Firmware Verify Error* bit is set (in which case, the error shall be handled, e.g. by raising an exception or retrying). Under typical conditions, the first poll of boot status will already indicate a completion of the verification, since the verification is started in parallel to the loading.
9. Issue the Command “*Boot Program RAM (0x0003)*” to *Host Channel 0 - Command Input (0x00)* to start execution of the firmware. In the register *Boot Status (0x25)*, the *Host Interface Ready* bit will clear at the start of the boot process.
10. Wait for firmware boot time, T\_boot\_fw\_host, (see *Table 108* in *Section 19.5.1*) or poll the *Boot Status (0x25)* register until the *Host Interface Ready* bit is set again. The “*Initialized*” Meta Events will be then inserted in the Wake-Up and Non-Wake-Up FIFOs, and the host interrupt pin will be asserted.
11. Clear the first interrupt by reading the content of the FIFO.

The BHA260AB is now ready to receive the majority of host interface commands, like enabling virtual sensors or setting parameters.

For a firmware upload, the following registers are relevant:

Table 11: Relevant registers for host boot mode

Register Name Register	Address	Register Value
Command Upload Stream (Host_Channel[0])	Host Channel 0 - Command	Register to write firmware upload command and other boot commands

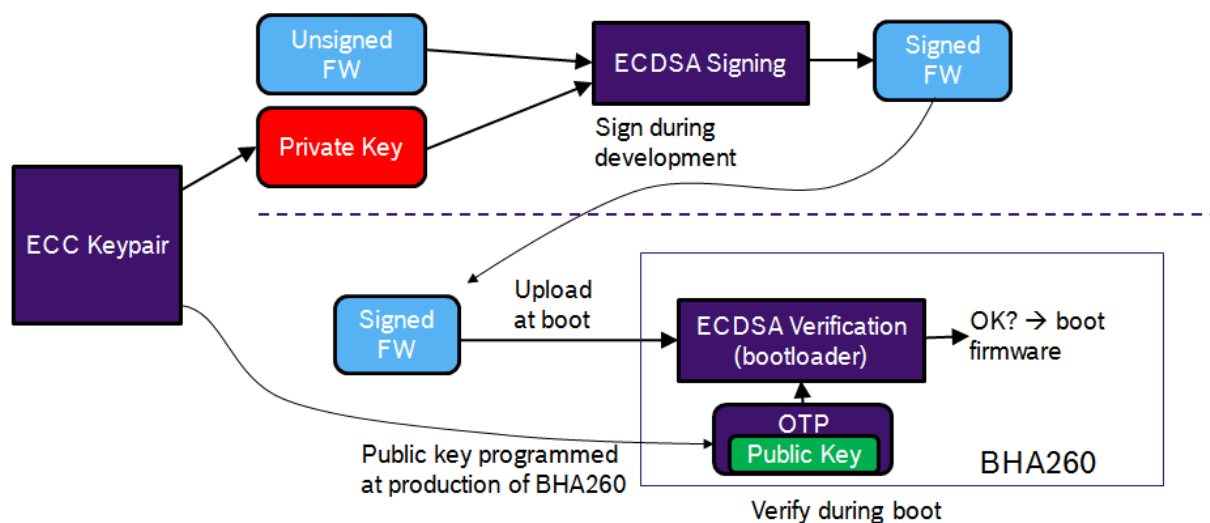
	Input (0x00)	
Chip Control	Chip Control (0x05)	1 – CPU Turbo Disable
Host Interrupt Control	Host Interrupt Control (0x07)	Configure type of host interrupt signaling desired
Reset Request	Reset Request (0x14)	1 – Reset MCU (Fuser2 Core)
Host Control	Host Control (0x16)	Configure 3 wire SPI or I2C watchdog
Fuser2 Product Identifier	Fuser2 Product Identifier (0x1C)	0x89
Fuser2 Revision Identifier	Fuser2 Revision Identifier (0x1D)	0x02 or 0x03
ROM Version	ROM Version (0x1E- 0x1F)	5166
Boot Status	Boot Status (0x25)	Boot status bits to help the host initialize the MCU at the correct times, and learn the results of various operations that may fail, such as image verification.

## 5.2.2 Secure Boot Mode

The BHA260AB uses a secure boot mode concept that verifies the authenticity of a firmware image before it is executed. With this, it is possible to protect the firmware executed on the BHA260AB against tampering and hacking. The secure boot concept applies to both host and stand-alone boot modes.

The BHA260AB uses the Elliptic Curve Digital Signature Algorithm (ECDSA) for determining the authenticity of a firmware. After compilation, the firmware is signed by usage of a secret private key. The matching public key is stored in the one-time programmable memory of BHA260AB during production. The bootloader verifies the firmware image with the ECDSA algorithm using the public key stored in the device. The firmware will be executed if the verification was successful.

Figure 13: BHA260AB secure boot concept (incl. signing and verification of FW images)



The firmware image consists of two sections, a kernel payload section and a user payload section. The content of the kernel payload section comprises of all algorithms, libraries and configuration items that cannot be modified. With the Software Development Kit (SDK), the kernel payload is provided as a signed binary and is authenticated to run on the BHA260AB. This concept allows for only authorized firmware images running on a particular device.

The user payload section allows for any kind of custom configuration and extension of the existing functionality. By default, the user payload section of the firmware is signed with a default key provided within the SDK.

If a customer specific key is required, Bosch Sensortec can provide a dedicated SDK for this, which uses a separate key pair (generated and owned by the customer) for signing the user mode section. Please contact your Bosch Sensortec representative for details.

### 5.3 Device Operation

Once the BHA260AB is operational and the firmware has booted, all functionality including the virtual sensor suite is available. The BHA260AB indicates its readiness by inserting an “Initialized” meta-event into each of its two FIFOs. It is recommended that the host wait for these events before attempting to query or configure the sensors or other features.

If an incorrect firmware is executed (for example, it is built for a different set of sensors), the FIFO will instead contain one or more Sensor Errors or Error meta-events.

In the nominal case, the host is now free to query which virtual sensors are present by reading the parameter “*Virtual Sensors Present (0x011F)*”, read the Sensor Information parameters, configure the algorithms by changing the Algorithm parameters, and/or configure the sensors to start measuring using the Sensor Configuration parameters.

The host may also use the *Meta Event Control (0x0101, 0x0102)* parameter to configure which meta-events appear in the FIFOs, such as the FIFO Overflow, FIFO Watermark, etc. It can specify whether certain meta-events can cause an immediate host interrupt or are batched to be processed at a later point of time.

Finally, the host may configure the optional FIFO Watermark thresholds using the *FIFO Control (0x0103)* parameter. This allows the host to be informed that either one or both of the FIFOs have reached a level at which the host shall read its contents to avoid data in the FIFO from being overwritten. This is especially useful when the Application Processor is asleep.

## 5.4 Processor Execution Modes

As previously introduced, the ARC EM4 CPU of the BHA260AB supports kernel and user execution mode.

- **Kernel Mode**  
Kernel mode is reserved for low level drivers, the RTOS, the Event-driven Software Framework, the BSX library, and other key internal functions, as well as interrupt handlers.
- **User Mode**  
User mode is used for user-provided enhancements (hooks), physical and virtual sensors. The MPU will block access to memory mapped registers with defined exceptions (e.g., universal timer and GPIO).

Elevation from user to kernel mode is implemented by a trap function. It will check that escalation to kernel mode is permitted from the calling function.

## 5.5 Power States and Run Levels

The various power states and run levels of BHA260AB are defined independently for the Fuser2 Core MCU and the integrated acceleration sensor.

During operation they are also handled independently, specifically, the physical driver of the acceleration sensor in the Fuser2 firmware controls the power modes of the acceleration sensor. As an example, it is possible that the Fuser2 remains in sleep mode, while the acceleration sensor remains active, waiting for movement in order to wake-up the Fuser2 core.

The Fuser2 operates in the following power states: Power-Up, Active, Light sleep, Regular sleep, and Deep sleep. In the states Active and Light Sleep, the Fuser2 can run at two clock speeds: Long Run mode (20 MHz core frequency) and Turbo mode (50 MHz Core Frequency). For the transition between the two Run Levels, there is a further intermittent Power State called Ramp.

The details of the power states and Run Levels are described in the Table 13: Available power states in Fuser2 Core MCU shown below.

The power states and run levels are controlled by the Event-driven Software Framework internally and therefore there is no need for a user to interact with them directly.

Table 12: Available power states in Fuser2 Core MCU

Power state	Name	Timer Oscillator	System Oscillator at Run Level	Description
PWRUP	Power-Up	Off	Off (powering up)	The analog subsystem powers up with the timer oscillator disabled. All digital modules (CPU, memories, peripherals ...) are in reset. When the power levels are sound and the system oscillator is stable, the device transitions to ACTV state, provided no other reset is asserted.
ACTV	Active	Unchanged	On	The processor is active. Upon entry from PWRUP or DSLP, the timer oscillator is disabled and activation is under firmware control. The firmware may issue a sleep instruction for transition into any of the sleep modes (LSLP, SLP, DSLP). In the absence of chip activity (e.g. peripherals inactive), a transition to SLP/DSLP is carried out immediately. Otherwise a transition to LSLP is carried out.
LSLP	Light Sleep	Unchanged	On	CPU enters sleep mode. System oscillator is enabled. A valid interrupt wakes the CPU and the device transitions to ACTV. Otherwise: if light sleep request is active, the device remains in this state. If a regular sleep or deep sleep request is active, the device remains in this state until all core activity ceases. It then transitions to SLP / DSLP.
SLP	Regular Sleep	Unchanged	Off	The system oscillator is disabled. On a valid interrupt or host interface DMA request, the system oscillator is re-enabled. When the system oscillator is stable, the device transitions to ACTV if a valid interrupt is active. Otherwise the system transitions to LSLP
DSLP	Deep Sleep	Off	Off	Same as SLP. Additionally, the timer oscillator is disabled, allowing a lower current consumption.
RAMP	Ramping	Unchanged	Off (transitioning)	This state is entered whenever the run level is switched from Long Run to Turbo or vice versa. The analog subsystem adjusts the regulator voltage and system oscillator to the requested target. While the system oscillator is changing, no system clock is provided to the digital modules. The timer oscillator clock remains unaffected. When voltage and system oscillator are stable, the device transitions back to ACTV.

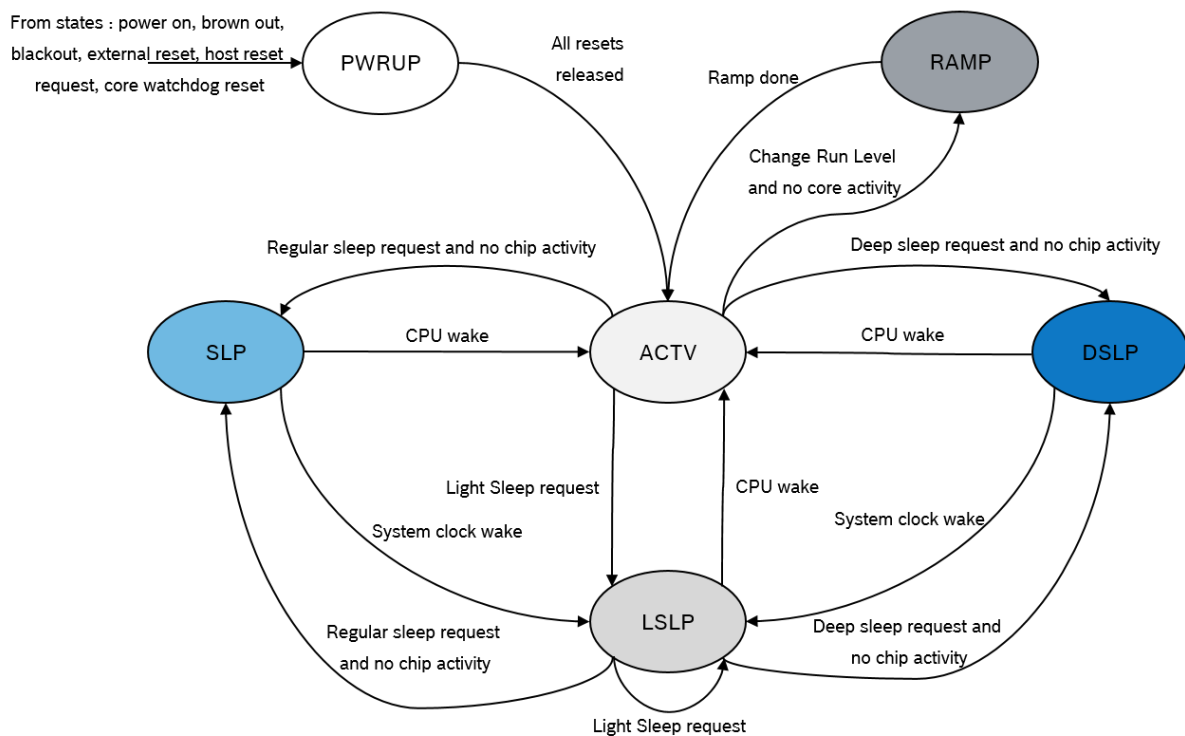


Figure 14: Transitions of the power states

## 5.6 Debug and Post Mortem Support

The BHA260AB includes several features for debugging and post-mortem support:

- A 2-wire (compact) cJTAG (IEEE1149.7) interface debug port for the embedded ARC EM4 CPU.
- A debug printf() function for textual output through the data FIFOs
- Several dedicated and general-purpose host registers for monitoring the status of the system
- A post-mortem analysis capability, filling dedicated data structures in RAM in case of a watchdog reset or a critical exception, for the purpose of analysis after reset.

The details of these features are described in the *BHI260-BHA260 Programmer's Manual* and *Application Notes* on the usage and installation of the debugger software that can be found in *Section: 26 Reference*.

The JTAG interface is enabled or disabled during boot, depending on the configuration setting in the firmware. If the JTAG is enabled, it cannot be disabled anymore during runtime (without issuing a reset), in order to avoid that the firmware disables the debug capability by accident.

The pads of the JTAG interface (JTAG\_CLK, JTAG\_DIO) can be used as general purpose IOs in case the debugging is not enabled.

## 6 Device Configuration and Operation

### 6.1 Overview of the Event-driven Software Framework and Virtual Sensor Stack

The Event-driven Software Framework consists of different software layers that abstracts typical functions in a smart sensor such as power management, sensor data acquisition, sensor data synchronization, sensor data processing, and features a complete Virtual sensor stack ready to use. The core component of the virtual sensor stack are Virtual Sensors. They are the main providers for data out of BHA260AB via the FIFOs.

A Virtual sensor typically consumes data from one or more Virtual or Physical sensors and either sends the processed data to the FIFO or another Virtual sensor. A Physical sensor driver communicates with an external physical sensor connected to the BHA260AB (or one of the physical sensors integrated inside the BHA260AB) to retrieve data and send it to a Virtual Sensor.

There are different types of virtual sensors, depending on the way the data is generated and written into the FIFO.

- **Continuous:** Data frames are written synchronously into the FIFO by the Virtual sensor at the configured sample rate.
- **On-Change:** Data frames are written when there is a change in value of the virtual sensor. The data frames have a maximum sample frequency as the one configured.
- **One-Shot:** Only one data frame is written in the FIFO following a trigger to enable the sensor.

The fundamental parts of the Event-driven Software Framework are explained in the following sections.

### 6.2 Using Virtual sensors

Virtual sensors are part of the firmware image, and hence there is a correlation between the number of Virtual sensors compiled into the firmware image and the size of this firmware, ensuring better control of the RAM memory. The remaining memory space then can be used as a FIFO for data batching.

Each Virtual sensor (e.g. Gravity Sensor) may come in two types defined by two distinct Sensor IDs. These types are wake-up virtual sensor and non-wake-up virtual sensor and their data is placed in one of the two corresponding output FIFOs (i.e. wake-up FIFO and non-wake-up FIFO).

To configure a virtual sensor, the “Configure Sensor” host command needs to be used, see Section 13.2.7. It takes three parameters to configure the behavior of the virtual sensor:

- Sensor ID
- Sample Rate
- Latency

The **Sensor ID** is a unique identifier, which defines a specific Virtual sensor. This Sensor ID is defined as an 8 bit unsigned integer value.

A list of all Virtual sensors and their corresponding sensor IDs can be found in Section 15.

The **sample rate** or output data rate (ODR) defines the desired frequency, at which the BHA260AB should provide the Virtual sensor's data. Sample Rate is defined as a 32-bit float value in Hz.

A Virtual sensor is enabled by setting its sample rate parameter to a value higher than 0 Hz. Once enabled, the Event-driven Framework will operate the virtual sensor by providing it data and transferring the processed data to the corresponding FIFO. To disable a Virtual sensor, the sample rate has to be set to 0 Hz.

If the configured sample rate parameter is supported by the virtual sensor, this value will be selected as the actual sample rate. In the case that the configured sample rate is not supported, the actual sample rate will be selected by the Event-driven Software Framework and will be in the range of 90%...210% of the requested data rate as explained in Section 7.3.

The **Latency** defines the maximum time that the BHA260AB allows a Virtual sensor event to remain in the FIFO, before the host is notified via the host interrupt.

The latency parameter is defined as 24 bit unsigned integer value in milliseconds.

With the latency set to 0 ms the host will instantly receive an interrupt after a new sample is stored in the FIFO. This behavior is similar to that of a Data Ready Interrupt.

The latency feature enables the batching of data for a period of time, during which the host may sleep, while the BHA260AB continues collecting and processing sensor data, e.g. setting the latency parameter to its maximum value allows to keep the host into sleep mode for ~4.6 h. This "offloading" of the sensor processing or "batching" of sensor data, allows a significant reduction of the system power consumption in real world applications.

Whether a virtual sensor is available in the setup is determined by the presence (or absence) of the physical sensors required to implement these virtual sensors and whether the virtual sensor implementation has been integrated in the firmware configuration at the time of build.

It is possible to check at runtime which virtual sensors are supported by the current setup and firmware by means of the "*Virtual Sensors Present (0x011F)*" parameter, see Section 13.3.2.5.

Additionally, the "*Virtual Sensor Information Parameters (0x0301 – 0x0395)*" and "*Virtual Sensor Configuration Parameters (0x0501 – 0x0595)*" parameters give access to the details and capabilities as well as the current configuration of each virtual sensor.

Please refer to Section 13.3.4 and Section 13.3.5 for details.

### 6.3 Using FIFOs and FIFO Events

The concept of FIFOs and Events is fundamental to the proper operation of the BHA260AB.

All virtual sensor data and associated timing information is sent to a FIFO. The data is structured in a specific package format and referred to as a Virtual Sensor Event. In this context Virtual Sensor Events are not only single occurrences like a double-tap, but also any virtual sensor data like an accelerometer data sample. In addition to Virtual Sensor Events, the FIFOs also contain other kind of data, as described in the following sections.



### 6.3.1 FIFOs

The BHA260AB has three output FIFOs: the Wake-Up FIFO, the Non-Wake-Up FIFO and the Status and Debug FIFO.

The first two FIFOs contain “FIFO Events”, which is mainly data from the enabled virtual sensors, together with certain associated FIFO Events like timestamps.

By default, the Status and Debug FIFO contains responses to host commands. Alternatively, by setting the “Async Status Channel” bit in the Host Interface Control register, the output of additional debug and asynchronous status messages can be enabled like debug text strings.

The size of the Wake-Up and Non-Wake-Up FIFOs can be configured at build time, while the Status and Debug FIFO has a fixed size of 512 bytes.

Before the host can read data from any of the FIFOs, it needs to wait for the BHA260AB to request a transfer by raising a host interrupt request.

The behavior of the host interrupt request can be configured by setting the virtual sensors ODR, the virtual sensors latency, the FIFO watermark settings and the ‘AP suspended’ state of the ‘Host Interface Control’ register.

The host can also initiate an immediate transfer without waiting for an interrupt. In this case, the ‘FIFO Flush’ command with the appropriate parameter should be used, see *Section 13.2.3 FIFO Flush (0x0009)* for details.

A transfer request from the BHA260AB to the host is indicated via the HIRQ pin (Host Interrupt Signal) and the Interrupt Status register.

By reading the *Interrupt Status (0x2D)* register, the host can find out, which of the FIFOs contains data. Next, the host shall read out all data related to the transfer request from the respective Host Output Channels [1...3]. The amount of data in each FIFO is given by the first two bytes read from the FIFO (16bit value, LSB first). The data in the FIFO are structured in a specific format that allows to parse and distinguish the contained information, even when read as a single byte stream ( see *Section 14 FIFO Data Formats*).

### 6.3.2 FIFO Events

The term “FIFO Event” relates to any data packet that can be read from the Wake-Up and Non-Wake-Up FIFO. Several types of FIFO events exist:

- Virtual Sensor Events
- Timestamp Events
- Meta Events
- Debug Data Events

Any FIFO event starts with a 1-byte FIFO Event ID followed by zero or more payload bytes. The amount of payload bytes depends on the FIFO Event ID and is fixed per FIFO Event ID. By this a parser can separate the different events from the FIFO byte streams. A complete list of FIFO Event IDs are given in *Table 79: Overview of FIFO Event IDs*.

**Virtual Sensor Events** are the data output samples from any of the virtual sensors. They contain the sample data as described in *Section 15.1 Format of virtual sensor events*. The big majority of FIFO Events are Virtual Sensor Events.

**Timestamp Events** declare a certain point of time, which is then valid for all following events, until a new timestamp event occurs. Different absolute and delta timestamp events exist, in order to optimize data load. They can be disabled for use cases which do not require timing information of events. See details in Section 15.2 *Retrieving timestamps of virtual sensor events*.

**Meta Events** are part of the Event-driven Software Framework and are used by the BHA260AB to notify the host about the occurrence of situations that might be of interest to the host but are not regular sensor data and they can be individually enabled or disabled. Examples for Meta Events are the Initialization of the FIFOs, confirmation for a change of ODR or updates of the calibration status of virtual sensors. They are placed in the different FIFOs by the Fuser2 Core and read by the host like any other FIFO Event. Together with the timestamps, the host can exactly reconstruct the time a FIFO Meta Event occurred. A complete list of all Meta Events are given in Section 15.3 *Format of Meta Events*. It also describes in which one of the FIFOs the different Meta Events are placed.

**Debug Data Events** are used to communicate any kind of debug information from the firmware to the host, e.g. by using printf() calls in the firmware. See details in Section 15.4 *Debug Data*.

## 6.4 Using the Parameter Interface

The Parameter Interface is integrated as a subsystem of the generic command interface, i.e. reading or writing of parameters is handled by sending appropriate commands to the BHA260AB following the command protocol described in Section 4.4.2 *Host Command Protocol*.

The intention of the Parameter Interface is to modify the configuration of the BHA260AB during runtime. The parameter space of the BHA260AB is very wide and covers several parameter pages to allow a versatile and flexible configuration of the device behavior.

Parameter related commands are grouped into the following sections:

- System Parameters
- BSX Algorithm Parameters
- Virtual Sensor Information Parameters
- Virtual Sensor Configuration Parameters
- Virtual Sensor Control Parameters
- Customer Defined Parameters

A complete list including descriptions of all available parameters is given in Section 13.3 *Parameter Interface*.

## 6.5 Current consumption in operation

The table below lists example current consumption numbers of the BHA260AB device for typical virtual sensors. The BSX fusion library can be configured to run in the default High Performance mode or in a Low Power mode optimized for power-constraint applications, like e.g. wearables.

Table 13: Power consumption vs. operating mode

Virtual Sensor activated	ODR [Hz]	BHA260AB current [mA typical] <sup>1</sup>	
		Long run	Turbo
Game Rotation Vector (6DoF, Accelerometer and Gyroscope) <sup>3</sup>	25	0.230	0.259
	400	0.475	0.570
	800	-	0.956
Rotation Vector (9DoF, Accelerometer, Gyroscope and Magnetometer) <sup>2,3</sup>	25	0.262	0.298
	400	0.586	0.749
	800	-	1.115
Geomagnetic Rotation vector (6DoF, Accelerometer and Magnetometer) <sup>2</sup>	25	0.184	0.207
	200	0.364	0.458
Step detector	n.a.	0.076	0.093
Step Counter	n.a.	0.076	0.087
Tilt	n.a.	0.033	0.046
Significant Motion	n.a.	0.037	0.045
Glance gesture	n.a.	0.073	0.089
Pickup Gesture	n.a.	0.072	0.083
Activity Recognition	n.a.	0.077	0.093
Wakeup Gesture	n.a.	0.234	0.265

**Note:**

- 1) Current consumption may vary depending on firmware version and sensor related configurations.
- 2) Requires an external magnetometer. Power consumption of the magnetometer is not included.
- 3) Requires an external gyroscope. Power consumption of the gyroscope is not included.

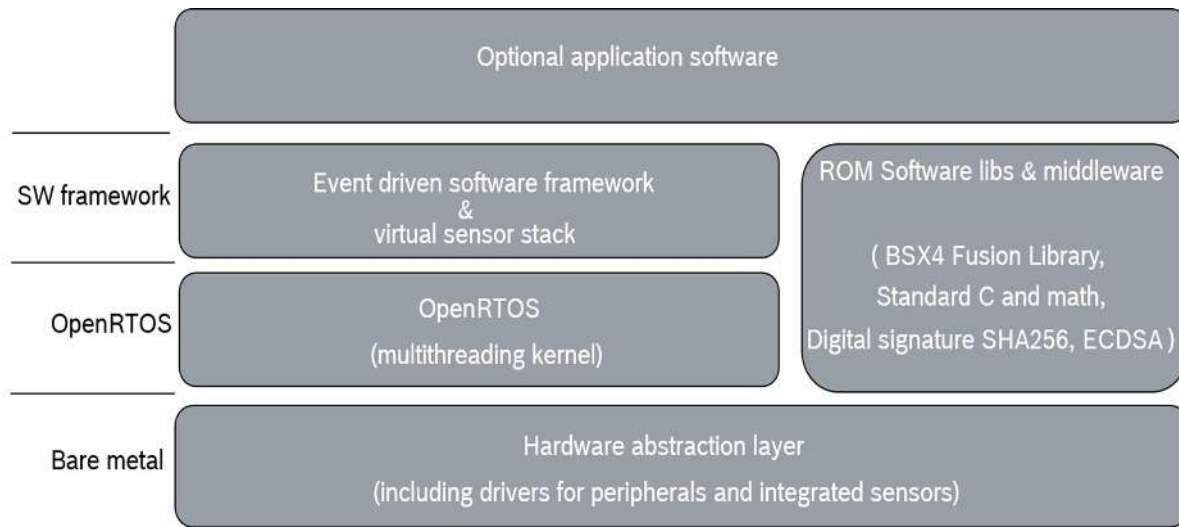
# FUNCTIONAL DESCRIPTION

## 7 Integrated Product Software

### 7.1 Event-driven Software Framework and Virtual sensor stack

Figure 15 shows the structure of the Event-driven Software Framework running inside the BHA260AB.

Figure 15: Structure of the BHA260AB Event-driven Software Framework



Using the BHA260AB's SDK, additional customer specific application software can be added. Specifically, new in-sensor data processing algorithms can be added as new custom virtual sensors. Also, if additional external sensors are connected over one of the secondary interfaces, the corresponding new physical sensor data can be made available to the BHA260AB by integrating a custom driver.

### 7.2 Hardware abstraction layer

The hardware abstraction layer provides a low level API, with support functions to access peripherals like SPI and I2C master interfaces, Timer and Event subsystem. This is typically only used in a Physical driver.

This layer is used by the Event-driven Software Framework and therefore there is typically no need for a user to interact with it directly within the context of a Virtual Sensor Driver.

### 7.3 BSX Sensor Fusion Lib

The BSX sensor fusion library can be used to provide corrected and fused sensor data based off the raw sensor data received from the connected sensors. Virtual sensor data can be identical to the raw physical sensor data, offset calibrated data or certain preprocessed data via BSX algorithms like orientation, gesture recognition algorithms, step counter, etc. For a more comprehensive description of the virtual sensors integrated in the BHA260AB see *Table 79: Overview of FIFO Event IDs*.

The virtual sensor stack computes the sensor data with a maximum Output Data Rate (ODR) of up to 800 Hz and stores them in the FIFOs. The host CPU can access the FIFO over the host interface.

Based on the max ODR of 800 Hz, additional ODRs are available with a step size of 0.5x down to 1.56 Hz.

Down to an ODR of 100 Hz, the sensor fusion software will accordingly reduce the data rate of the fusion computation and data sampled from the required physical sensors. Below an ODR of 100 Hz, the data provided is sub sampled from a 100 Hz signal to avoid loss in signal quality.

If the virtual sensor is set to a supported ODR, this will be the actual ODR used for processing and writing data in the appropriate FIFO. In case that the requested ODR is not supported, the actual output data rate will be in the range of 90%...210% of the requested data rate.

If multiple virtual sensors with different output data rates are requested by the host, the internal physical and virtual sensor data rates will be selected by the Event-driven Software Framework such that all the requested output data rates can be fulfilled.

*Note:*

*1) Specific configurations can be created, supporting e.g. higher ODRs*

## 7.4 OpenRTOS multithreading real-time kernel

The integrated OpenRTOS kernel is the commercial version of the well-established FreeRTOS operating system for embedded devices. For more details, refer to Section 26, References 2: <https://www.highintegritysystems.com/openrtos/>

It is used by the Event-driven Software Framework internally and therefore there is typically no need for a user to interact with it directly.

## 7.5 Virtual Sensor Stack

The core of the functionality delivered by the BHA260AB as a smart sensor is the Virtual Sensor Stack. The BHA260AB provides many algorithms that implement sensor data processing tasks that run inside the BHA260AB MCU (Fuser2 Core), providing pre-processed data to the Host processor to improve algorithm reliability and performance as well as improving system power consumption. These algorithms implemented as software modules in Fuser2 Core firmware are called virtual sensors. The set of all integrated algorithms in the BHA260AB is the virtual sensor stack.

The BHA260AB supports all Virtual Sensors as defined in the Android CDD and beyond. Originally based on the Android specification virtual sensors are independent of another. So each virtual sensor has its own sample rate, type, report latency, and trigger mode.

Each virtual sensor software module may access a physical sensor via its sensor driver, or it may use the output data from other software modules (e.g. the BSX library) as an input in order to derive processed data.

The virtual sensors generate virtual sensor events, which may be continuous (e.g. samples at a configured data rate) or single events (on-change, one-shot, or special; e.g. when a step or significant motion has been detected). These virtual sensor events are represented as data

packets of a specific virtual sensor data type and are placed into one of the output FIFOs of BHA260AB , either *Host Channel 1 - Wake-up* FIFO Output (0x01) and/or *Host Channel 2 - Non-Wake-up* FIFO Output (0x02).

Each virtual sensor has a fixed ID and can be configured as a wake-up or non-wake-up sensor. For most virtual sensor types, both a wake-up and a non-wake-up ID exists, each can be also configured with an independent sample rate and report latency values.

For the definition and details of all virtual sensors supported in the SDK, see Section 15 *FIFO Data Types and Format*.

In addition to the existing Virtual sensors, additional features (algorithms) can be implemented and added as new virtual sensors by creating and uploading a new firmware to the BHA260AB . In this way, the functionality of the BHA260AB can be extended while still using the benefits of the available Event-driven Software Framework. For details and technical support please refer to corresponding application notes in Section 26 *References* or contact our regional offices, distributors and sales representatives.

## 7.6 Other available Software modules

### 7.6.1 Standard C library (libc) and standard math (libm)

As part of the ROM libraries the most used functions of the standard C library (libc) and standard math library (libm) are included in the ROM image. Additional functions are linked on demand into the RAM firmware image.

### 7.6.2 Libraries for Digital signature

As part of the ROM libraries a SHA256 library and an ECDSA160 library are included in the ROM image. These SHA and ECDSA algorithms are used by the bootloader for firmware verification. They can also be used within a customized firmware image if needed.

## 8 Software Development Kit (SDK)

A Software Development Kit (SDK) is available for users who want to create customized firmware, e.g. for running their own sensor data processing algorithms with low latency and at ultra-low power consumption on the BHA260AB.

In addition to the SDK, a C compiler toolchain for the ARC processor is required.

Supported Toolchains are:

- Metaware C Compiler for ARC
- GNU C Compiler for ARC

For details and technical support please refer to corresponding application notes Reference 3 and Reference 7 in Section 26 *References*.

## 9 Memory Configuration and Memory Map

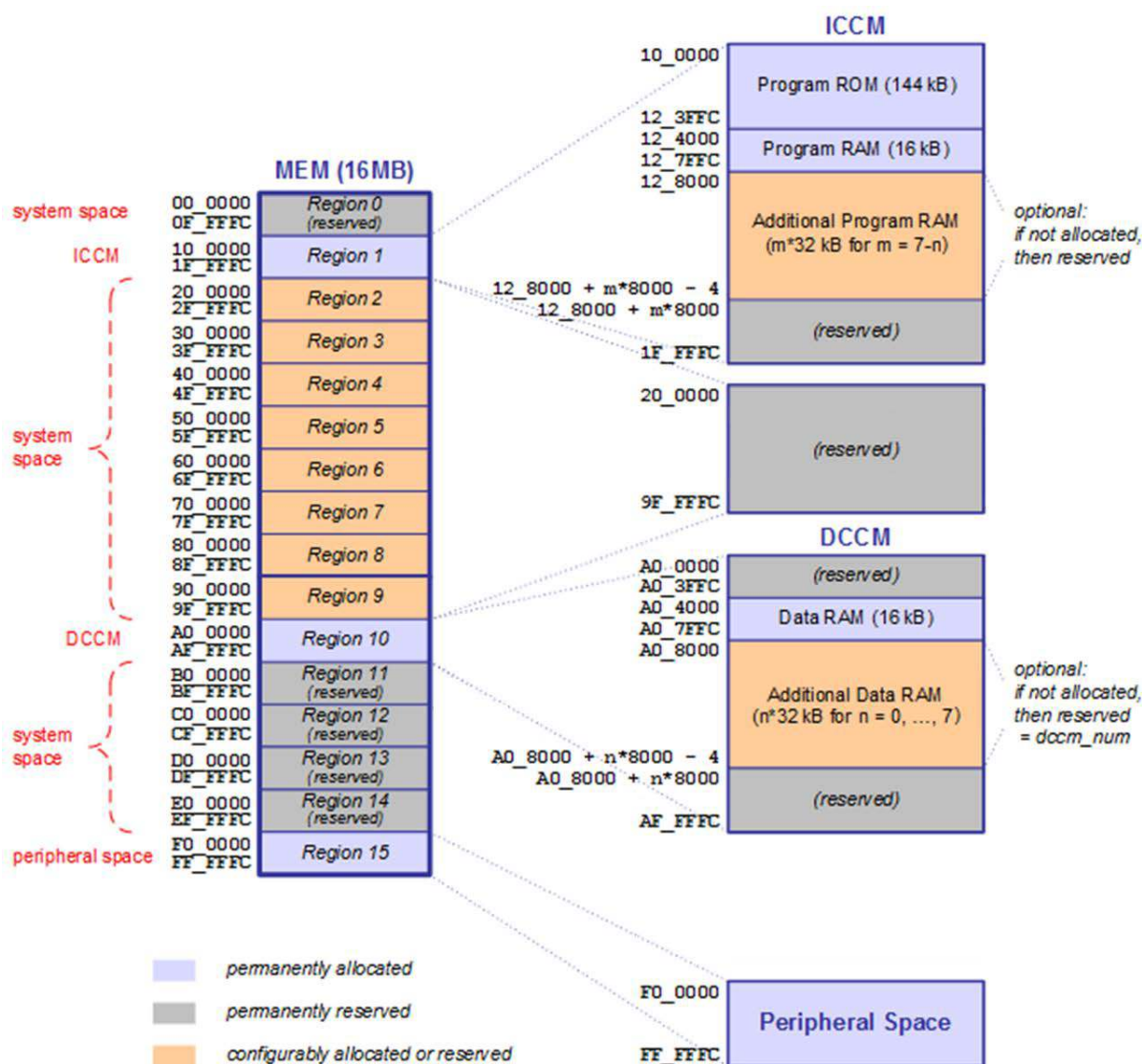
The memory space addressable by the ARC EM4 CPU in the Fuser2 Core MCU has a total size of 16 Mbytes. It is divided equally into 16 regions, each with a size of 1 MByte. The following types of memories are allocated in the memory map:

- On-chip ROM
- On-chip SRAM

Further, the peripheral IO is also allocated in the memory map (Memory-mapped IO).

The details of the memory map are shown in the diagram below and described in the following sections.

Figure 16: Fuser2 memory map





## 9.1 On-Chip Memories

The ARC EM4 CPU in the Fuser2 Core MCU allocates separate memory areas for program code fetch and data load/store due to the nature of its Harvard architecture. The on-chip RAM and ROM are closely coupled to the CPU, allowing for the maximum access speed. These closely coupled memory areas are called ICCM (Instruction Closely Coupled Memory for instruction code) and DCCM (Data Closely Coupled Memory for data).

The on-chip ROM of 144 KBytes is permanently allocated to the ICCM area, since its main purpose is code storage. It is mask-programmed by Bosch Sensortec and contains the boot loader (executed after reset) and various libraries. See Section 7 *Integrated Product Software* for more information.

The total on-chip RAM of 256 KBytes is laid out on several memory banks and configurable in a very flexible manner.

Two 16 Kbytes memory banks are permanently enabled and fixed allocated to the ICCM and DCCM area respectively, to serve the absolute minimum needs of the ARC EM4 CPU for operation.

The remaining 224 KBytes form a pool of 7 RAM banks with a size of 32 KByte each, which can be enabled or disabled according to the requirements of the firmware, allowing for the optimization of current consumption (see Block diagram in Section 4.2). Each of the configurable 32 Kbytes memory banks can have one of the following states:

- Powered-off
- Allocated to the ICCM area
- Allocated to the DCCM area

The allocation of configurable memory banks is performed by the bootloader depending on the requirements of the firmware. It is ensured that only the minimum of required RAM banks for instructions and data are allocated and powered on automatically without the need for the programmer to take care for this. The amount of allocated RAM banks for ICCM and DCCM are reported separately during compilation of the firmware.

## 9.2 Peripheral Space

The Fuser2 core uses memory-mapped IO to control its peripherals. Therefore, all registers controlling the hardware are mapped into the peripheral space of the memory map. However, the programmer does not typically have to modify these registers, since API functions are available for all peripheral IO related operations.

## 10 Pin and Interface Configuration

### 10.1 Configuration of Master Interfaces

The BHA260AB has 3 secondary master interface ports, named M1 to M3, see Figure 17.

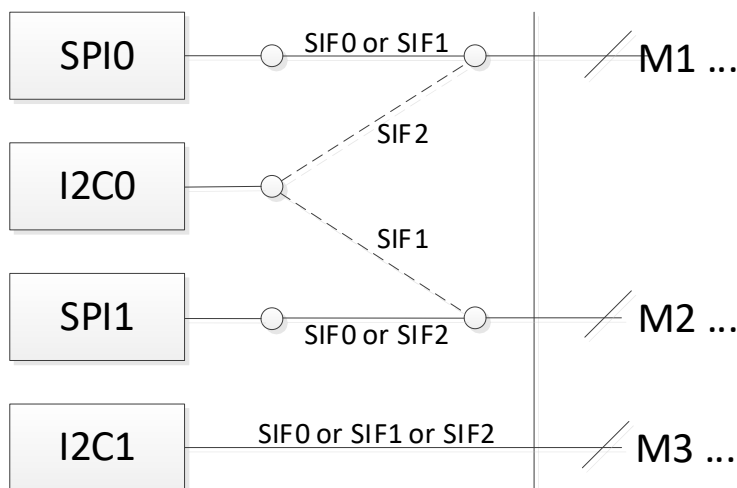
Inside the BHA260AB two SPI (SPI0, SPI1) and two I2C (I2C0, I2C1) master interface controllers are implemented, which can be mapped in various ways to these 3 master interface ports.

The possible configurations of the master interface ports M1, M2 and M3 (mapping combinations to the internal interface controllers SIF0, SIF1, SIF2 ) are listed in the table below:

Table 14: Configuration options of master interface ports

Configuration	Master interface port		
	M1	M2	M3
SIF0	SPI0	SPI1	I2C1
SIF1	SPI0	I2C0	I2C1
SIF2	I2C0	SPI1	I2C1

Figure 17: Overview configuration options of master interface ports



**Note:** M1 is not exposed externally in the BHA260AB

The master interface connection routing is configured by a setting in the board configuration file which is part of SDK and details of which can be found in the *BHI260-BHA260 Programmer's Manual Programmer's Manual*.

### 10.2 General-Purpose Inputs/Outputs

Besides their functional purpose (like e.g. host or master interface), almost all pins of the BHA260AB can also be configured as General Purpose Inputs/Output (GPIO). The following characteristics of a GPIO pin can be configured:

- Input or output
- Drive strength as output, see *Table 103: Electrical characteristics* in *Section 19.3 Electrical characteristics* for low and high driver strength characteristics
- Tri-state
- Pull-up, pull-down (all GPIOs have pull-up functionality, except the GPIO24, which is a pull-down), see *Table 103: Electrical characteristics* in *Section 19.3* for resistor values

Before the firmware is loaded, the state of the GPIOs is according to the column “Reset value” in *Table 1: Pad connections* in *Section 3.1 Pad description*, i.e. most pins are configured as inputs with pull-up enabled.

The settings in the board configuration file define:

- The default configuration of all GPIOs after the firmware has started execution
- Which of the GPIOs are used as SPI chip select or interrupt lines for attached physical sensors

These settings are active after a firmware has been loaded.

If a GPIO is not used as SPI chip select or interrupt input by the Event-driven Software Framework, it can be controlled by low-level API functions provided by the Hardware Abstraction Layer of Software Development Kit. See the *BHI260-BHA260 Programmer's Manual* in Reference 3 from *Section 26* for more information.

## 11 Event and Interrupt Configuration

The BHA260AB provides various sources, which can generate Fuser2 Core interrupts. In this section, the Fuser2 Core interrupt mechanisms are described.

The interrupt handlers are managed by the Event-driven Software Framework and therefore there is typically no need for a user to interact with them directly.

However, for cases where a user needs to extend the system, e.g. by connecting an additional external device (like a sensor) or a signal (like a camera shutter interrupt for time synchronization), additional interrupt handlers need to be installed.

The SDK provides a method for this in form of the physical sensor driver, which integrates such extensions into the overall framework in a smooth way.

For details, please refer to corresponding *BHI260-BHA260 Programmer's Manual* as specified in Section 26 Reference or contact our regional offices, distributors and sales representatives.

Table 16 provides an overview of available interrupt sources, where *interrupt input* describes the signal fed into the interrupt controller and *interrupt output* describes the mapped output of the interrupt controller.

Table 15: MCU Interrupts (Sources and Mapping)

Block	Interrupt Input	Interrupt Output	Map	Event Int	Description
Computational Core	Timer: internal	--	irq16	--	CPU internal Timer0 has triggered
	Watchdog: internal	--	irq18	--	Watchdog timer has triggered
	The DMA interrupts bypass the interrupt controller and connect directly to the processor interrupt inputs.	dma0_done_int	irq17	--	DMA transfer is completed
		dma1_done_int	irq19	--	
		dma2_done_int	irq20	--	
		dma3_done_int	irq21	--	
		dma0_err_int dma1_err_int dma2_err_int dma3_err_int	irq34	--	Error occurred during DMA transfer
				--	
--					
Host Interface	hif_comm_irq	hif_comm_int	irq22	no	Host ends a communication thread (I2C mode: Stop bit received; SPI mode: HSCB de-asserted)
	host_ev_irq	host_ev_int	irq30	yes	Host controlled software interrupt request generation. Is triggered when host writes '1' to bit 0 in the Timestamp Event Request register

	hif_read_irq[0:15]	hif_read_int	irq35	no	Host read request on one of the following registers: Proc_GenPurpose3_B0 Proc_GenPurpose3_B1 Proc_GenPurpose3_B2 Proc_GenPurpose3_B3 Proc_GenPurpose4_B0 ... Proc_GenPurpose6_B3
	hif_write_irq[0:15]	hif_write_int	irq36	no	Host write request on one of the following registers: Host_GenPurpose0_B0 Host_GenPurpose0_B1 Host_GenPurpose0_B2 Host_GenPurpose0_B3 Host_GenPurpose1_B0 ... Host_GenPurpose3_B3
	hif_buf0_oflw_irq	hif_buf_err_int	irq32	no	DMA channel 0 buffer overflow
	hif_buf1_uflw_irq			no	DMA channel 1 buffer underflow
	hif_buf2_uflw_irq			no	DMA channel 2 buffer underflow
	hif_buf3_uflw_irq			no	DMA channel 3 buffer underflow
SPI Master lface 0	spim0_irq	spim0_int	irq23	no	Transaction on master interface is completed
SPI Master lface 1	spim1_irq	spim1_int	irq24	no	
I2C Master lface 0	i2cm0_irq	i2cm0_int	irq25	no	
I2C Master lface 1	i2cm1_irq	i2cm1_int	irq26	no	
Real Time Counter	rtc_irq	rtc_int	irq27	no	RTC counter overflow
Interval Timer	itmr_irq	itmr_int	irq28	no	Interval timer reaches the programmed limit
Universal Timer	utmr_irq	utmr_int	irq29	no	Depending on programmed setup of the universal timer one of the conditions has occurred: <ul style="list-style-type: none"> <li>• Universal timer reaches programmed limit</li> <li>• Compare function triggers</li> </ul>

					• - Input capture event occurs
PAD	gpio_ev_irq0	gpio_ev_int	irq31	yes	Event Channel triggers according to programmed setup
	gpio_ev_irq1			yes	
	gpio_ev_irq2			yes	
	gpio_ev_irq3			yes	
	gpio_ev_irq4			yes	
	gpio_ev_irq5			yes	
	gpio_ev_irq6			yes	
	gpio_ev_irq7			yes	
	gpio_ev_irq8			yes	
	gpio_ev_irq9			yes	
	gpio_ev_irq10			yes	
	gpio_ev_irq11			yes	
SW	sw_irq[0:3]	sw_int[0]	irq37	--	Software interrupt triggered
		sw_int[1]	irq38	--	
		sw_int[2]	irq39	--	
		sw_int[3]	irq40	--	

# REFERENCE PART

## 12 Host Interface Register Map

This is the detailed description of the host interface register map introduced in Section 4.4 *Host Data Interface*.

All interaction between the host and the BHA260AB is through the register map, as summarized below. Each host register can be written from either the host or the Fuser2 Core side:

- Registers declared as “read-write” can be read and updated by the host.
- Registers declared as “read-only” can be read by the host, but can only be updated by the BHA260AB MCU (Fuser2 core).
- Registers declared as “write-only” can be updated by the host.

The register map is valid for both SPI and I2C host interface protocol.

Table 16: Host interface register map

Register Address	Register Name	Description	Host Access
0x00	Host Channel 0 - Command Input	The host can write command packets to this DMA Channel (input)	write-only
0x01	Host Channel 1 - Wake-up FIFO Output	The host can read the virtual sensor data generated and stored in the wake-up FIFO through this DMA Channel (output)	read-only
0x02	Host Channel 2 - Non-Wake-up FIFO Output	The host can read the virtual sensor data generated and stored in the Non-wake-up FIFO through this DMA Channel (output)	read-only
0x03	Host Channel 3 - Status and Debug FIFO Output	The host can read Status and Debug packets from FIFO DMA Channel (output)	read-only
0x04	Reserved	Do not use	-
0x05	Chip Control	Control fundamental behavior of Fuser2 Core, like the firmware upload speed or clearing the error and debug registers	read-write
0x06	Host Interface Control	Control various actions regarding the host interface	read-write
0x07	Host Interrupt Control	Control characteristics of host interrupt, mask interrupt sources	read-write
0x08-0x13	WGP1 – WGP3 - General Purpose Host Writeable	General-purpose registers for communication from the host to the Fuser2 Core. While the host can read/write these registers, the BHA260AB ARC CPU has read access only.	read-write
0x14	Reset Request	Host-controlled reset of Fuser2 Core	read-write
0x15	Timestamp Event Request	The host can trigger an event interrupt (with timestamp response)	read-write
0x16	Host Control	Control the SPI mode and the I2C watchdog behavior of the BHA260AB	read-write
0x17	Host Status	Check the power status of the chip, the selected host protocol, and DMA channels	read-only

0x18-0x1B	Host Channel CRC (32 bits)	Get CRC of the last used DMA channel	read-only
0x1C	Fuser2 Product Identifier	This register identifies the Fuser2 core. It reads 0x89 for BHA260AB	read-only
0x1D	Fuser2 Revision Identifier	Hardware revision of the Fuser2 core. It reads 0x02 or 0x03 for the BHA260AB	read-only
0x1E-0x1F	ROM Version	ROM Image revision	read-only
0x20-0x21	Kernel Version	Firmware Kernel image revision	read-only
0x22-0x23	User Version	Firmware User image revision	read-only
0x24	Feature Status	Get status of various firmware-related features	read-only
0x25	Boot Status	Get status of boot loader	read-only
0x26-0x2A	Host Interrupt Timestamp (40 bits)	Get timestamp of last host interrupt or Timestamp Event Request	read-only
0x2B	Chip ID	Product specific identifier. For BHA260AB it reads 0x74 or 0xF4	read-only
0x2C	Reserved	do not use	-
0x2D	Interrupt Status	Get current status of interrupt sources	read-only
0x2E	Error Value	Firmware-related error code	read-only
0x2F	Error Aux	Auxiliary information for firmware related error	read-only
0x30	Debug Value	Firmware-related debug value	read-only
0x31	Debug State	Firmware-related debug state	read-only
0x32-0x3D	(RGP5-RGP7) - General-Purpose Host Readable	General-purpose registers for communication from the Fuser2 Core to the host: while the BHA260AB ARC processor can read/write these registers, the host has read access only.	read-only
0x3E-0x7F	Reserved	do not use	-

## 12.1 Register Description

### 12.1.1 Host Channel 0 - Command Input (0x00)

The host should write one or more command packets with different size to this channel. This register address allows write-only access.

The overall structure of a Command Packet is always the same: 2 bytes command + 2 bytes length + (optionally) 4 bytes or more parameters or data. This is shown in the *Table 8: Structure of Command Packet*.

The host commands and their responses are described in Section 4.4.2 *Host Command Protocol*.

A list of available commands is described in Section 13 *Host Interface Commands*.



### 12.1.2 Host Channel 1 - Wake-up FIFO Output (0x01)

When the host receives a host interrupt from BHA260AB, it should read the first two bytes of this channel to determine the total transfer size, then read the remainder one or more transfers until the total number have been read. If there is no data available for the wake-up FIFO, the first two bytes will read 0. If there is data available, this data will be the virtual sensor data generated and stored in the wake-up FIFO since the last wake-up FIFO read access (or wake-up FIFO flush operation). This register address allows read-only access.

### 12.1.3 Host Channel 2 - Non-Wake-up FIFO Output (0x02)

When the host receives a host interrupt from BHA260AB, it should read the first two bytes of this channel to determine the total transfer size, then read the remainder one or more transfers until the total number have been read. If there is no data available for the non-wake-up FIFO, the first two bytes will read 0. If there is data available, this data will be the virtual sensor data generated and stored in the non-wake-up FIFO since the last non-wake-up FIFO read access (or non-wake-up FIFO flush operation). This register address allows read-only access.

### 12.1.4 Host Channel 3 - Status and Debug FIFO Output (0x03)

When the host receives a host interrupt from BHA260AB, it should read the first two bytes of this channel to determine the total transfer size, then read the remainder in one or more transfers until the total number have been read. If there is no status or debug data available, the first two bytes will read 0.

The general structure of host commands and their responses are described in Section 4.4.2 *Host Command Protocol*. A list of available commands is described in Section 13 *Host Interface Commands*.

This Status and Debug FIFO has two different operational modes, Synchronous Mode and Asynchronous Mode. For more details see the description of the bit 7 in register *Host Interface Control (0x06)* in Section 12.1.7. See also the Sections 14.2.1 *Synchronous mode* and 14.2.2 *Asynchronous mode*. This register address allows read-only access.

### 12.1.5 Reserved (0x04)

Do not use this register.

### 12.1.6 Chip Control (0x05)

This register provides bits that control fundamental behavior of the chip, like the speed during firmware image upload or clearing the error and debug registers. This register address allows read-write access.

Table 17: Chip Control Register (0x05)

Bit Number	Register Value	Description
------------	----------------	-------------

0	CPU Turbo Disable	When written with a 1, the Fuser2 Core is not allowed to run at its maximum speed during firmware image upload and signature verification. By default, if not modified by the host, the bootloader will attempt to run at the maximum CPU speed and the current drawn by the BHA260AB may peak up to ~3 mA during firmware upload. After booting the firmware, the run level is defined by the setting compiled into the firmware.
1	Clear Error Regs	When written with a 1, the error and debug registers (address 0x2E-0x31) are cleared. See Sections 12.1.25 and 12.1.26.
2-7	Reserved	

### 12.1.7 Host Interface Control (0x06)

This register provides bits that control fundamental behavior of the chip and allows the host to rapidly request specific actions that affect the state of the Fuser2 Core host interface. This include Aborting the transfer of one of the FIFO Channels (0 to 3), informing of AP suspend mode to the BHA260AB, controlling what is written to the Host Interrupt Timestamp register and controlling the operating mode of the Status and Debug FIFO.

This register address allows read-write access.

Table 18: Host Interface Control Register (0x06)

Bit	Identifier	Description
0	Abort Transfer on Channel 0	Abort transfer indicates that the host does not intend to complete a transfer with a channel; all pending data in the 32 bytes channel buffer for that channel is cleared, as well as any partial sensor sample that remains, and the interrupt pending bit for that channel in the Interrupt Status register is de-asserted. Since the data is not discarded, the BHA260AB will soon request another transfer. Abort Transfer simply stops the transfer of the specified channel for this data transfer session. In order to discard data entirely, use the FIFO Discard command <sup>1)</sup> <sup>2) 3)</sup> .
1	Abort Transfer on Channel 1	
2	Abort Transfer on Channel 2	
3	Abort Transfer on Channel 3	
4	AP is Suspended	The AP can inform about its power status to the BHA260AB just before going into a suspend (sleep) power mode or just after going to awake (normal) power mode. This affects whether the Fuser2 issue a host interrupt: <ul style="list-style-type: none"> <li>• When 0 (AP signals that it is awake mode): any wake-up or non-wake-up virtual sensor event may trigger a host interrupt if so configured.</li> <li>• When 1 (AP signals that it is in suspend mode): only wake-up virtual sensor events may trigger a host interrupt and wake the host.</li> </ul>
5	Reserved	

6	Timestamp_Event_Request (via Host Interrupt Timestamp registers)	<p>Controls which timestamp is written to the Host Interrupt Timestamp register:</p> <ul style="list-style-type: none"> <li>• When 0: Timestamp of the last Host Interrupt generated by the BHA260AB</li> <li>• When 1: The timestamp at the time of the last Host Timestamp Request register write access. In this case, the Host Event Request Timestamp will not be output as a status packet.</li> </ul> <p>Reset value is 0. Both timestamps are always available via the Timestamps parameter (see Section 13.3.2.4).</p>
7	Async Status Channel	<p>This bit controls the operating mode of the Status and Debug FIFO (Output Channel 3):</p> <ul style="list-style-type: none"> <li>• When 0: Synchronous mode, see Section 14.2.1</li> <li>• When 1: Asynchronous mode, see Section 14.2.2</li> </ul>

**Notes:**

- 1) The host interrupt signal and host interrupt asserted bit in the Interrupt Status register remain asserted and set until all FIFOs with pending transfers have been dealt with – either by aborting them using this register, or by reading their pending data out.
- 2) The Abort Transfer bits do not auto-clear. It is up to the host to set these four bits correctly every time it writes this register. It should clear these bits no sooner than 2 ms after asserting them.
- 3) Aborting a transfer causes data loss, if the current transaction is < 32 bytes, or if there is only 32 bytes left in the current FIFO block. In those cases those 32 bytes will be lost.

### 12.1.8 Host Interrupt Control (0x07)

This register allows the host to configure the driving mode for the host interrupt (interrupt generated by BHA260AB to signal an interrupt event to the host), as well as to specify for which reasons it would like to be interrupted. This register provides bits that control fundamental behavior of the chip. This register address allows read-write access.

Upon boot, i.e. before the main firmware initialization is complete, the BHA260AB will not assert the host interrupt, unless the reset was caused by the watch dog timer or fatal error; in that case, this register will contain the previous configuration information, so the bootloader will reuse the value of this register to determine how to assert the interrupt.

The bootloader does not issue any interrupts, since at this stage the device cannot know the interrupt configuration required by the host. By default, the Host Interrupt Signal HIRQ (Pad 10) is used within the SDK as an interrupt output, a firmware may use any other GPIO, or no interrupt at all. Hence the interrupt can be initialized only after loading the firmware, and then it is initialized according to the value of this register.

After the host has issued a *Boot Program RAM (0x0003)* command, The host must poll the *12.1.21 Boot Status (0x25)* register until the *Host Interface Ready bit* is asserted (Host is ready). At any time prior to or after this, the host may write the Host Interrupt Control register to configure the type of host interrupt signal desired – active high or low, level or pulse, and push-pull vs. open drain. Once configured by the host, subsequent host interrupts will be signaled this way.

The default value of this register after reset is 0. If the value of 0 is the desired configuration (active high, level, push-pull), the host does not need to write this register. Once the main firmware initialization is complete, it will assert the host interrupt using this mode.

Table 19: Host Interrupt Control Register (0x07)

Bit	Identifier	Description	
0	Wake-up FIFO Interrupt Mask	Setting this bit disable host interrupts due to the wake-up FIFO (Host_Channel_1), while clearing this bit (the default) allows it to be asserted whenever conditions warrant	
		<b>Value</b>	<b>Operation</b>
		0	Interrupt is active
		1	Interrupt is masked
1	Non-Wake-up FIFO Interrupt Mask	Setting this bit disable host interrupts due to the non-wake-up FIFO (Host_Channel_2), while clearing this bit (the default) allows it to be asserted whenever conditions warrant	
		<b>Value</b>	<b>Operation</b>
		0	Interrupt is active
		1	Interrupt is masked
2	Status Available Interrupt Mask	Disable host interrupts due to available output from synchronous Status Channel	
		<b>Value</b>	<b>Operation</b>
		0	Interrupt is active
		1	Interrupt is masked
3	Debug Available Interrupt Mask	Disable host interrupts due to available output from the asynchronous Status Channel	
		<b>Value</b>	<b>Operation</b>
		0	Interrupt is active
		1	Interrupt is masked
4	Fault Occurred Interrupt Mask	Disable host interrupts due to various faults.	
		<b>Value</b>	<b>Operation</b>
		0	Interrupt is active
		1	Interrupt is masked
5	Active Low Interrupt	Level of the host interrupt:	
		<b>Value</b>	<b>Operation</b>
		0	Active high
		1	Active low
6	Edge Host Interrupt	Host interrupt to drive a level- or edge sensitive host interrupt. 0 = level (e.g., asserted until FIFO empty), 1 = edge (pulse for 10 μs)	
		<b>Value</b>	<b>Operation</b>

		0	Level output
		1	Pulse output
7	Open Drain Interrupt	0 = push-pull, 1 = open drain (normally open drain is used with active low)	
		Value	Operation
		0	Push-pull
		1	Open-drain

### 12.1.9 WGP1 – WGP3 - General Purpose Host Writeable (0x08-0x13)

These registers are available for customer use in custom drivers or extensions. This range of register addresses allows read-write access. While the host can read/write these registers, the BHA260AB ARC CPU has read access only.

#### 12.1.10 Reset Request (0x14)

This register allows the Host to reset the Fuser2 Core with a register write access. This register address allows read-write access.

Table 20: Reset Request Register (0x14)

Bit Number	Field	Description	
0 <sup>1) 2)</sup>	Request Reset	Writing '1' causes a reset. This bit self-clears upon reset	
		Value	Operation
		0	No operation
		1	Initiate reset
1-7	Reserved		

Notes:

- 1) The host may change this bit at any time but only in a single-register transaction (i.e. bursting is not allowed).
- 2) The host shall wait  $T_{wait}$  (4  $\mu$ s) after issuing this reset request before resuming transactions since the reset may have to wake the device from sleep.

#### 12.1.11 Timestamp Event Request (0x15)

This register is used by the host to trigger a hardware timestamp of the exact time the register write transaction occurs. The host can query the timestamp value using the *Parameter Timestamps (0x0105)*.

Table 21: Timestamp Event Request Register (0x15)

Bit Number	Field	Value	Operation
0	Trigger Timestamp	0	No Operation
		1	Timestamp interrupt is triggered
1-7	Reserved		

If the Trigger Timestamp bit (Bit 0) of the *Timestamp Event Request* register **is not set**, register write transactions will not trigger any timestamp.

If the Trigger Timestamp bit (Bit 0) of the *Timestamp Event Request* register **is set**, the timestamp will be accessible to the Host in two different ways depending on the value of the *Timestamp\_Event\_Request* bit in the *Host Interface Control (0x06)* register:

- If the *Timestamp\_Event\_Request* bit in the *Host Interface Control (0x06)* register **is set**, the timestamp will instead be written to the Host Interrupt Timestamp (0x26-0x2A).
- If the *Timestamp\_Event\_Request* bit in the *Host Interface Control (0x06)* register **is NOT set**, a Timestamp Event Status Packet will be generated that can be read from the Status Channel. This Packet have the format explained in Table 23 Timestamp Event Status Packet

Table 22 Timestamp Event Status Packet

Field Name	Byte Offset	Description
Status Code	0x00-0x01	Timestamp Event Status Code = 0x000D
Length	0x02-0x03	Total number of bytes to follow = 8
Contents	0x04-0x08	40 bit timestamp
Reserved	0x09-0x0B	Reserved

### 12.1.12 Host Control (0x16)

This register is used by the host to control the SPI mode and the I2C watchdog behavior of the BHA260AB. This register address allows read-write access.

Table 23: Host Control Register (0x16)

Bit	Identifier	Description	
0	SPI Protocol	This bit has meaning only when the host interface is operated in SPI mode. It distinguishes between 4-wire and 3-wire SPI host protocol. <sup>1)</sup>	
		Value	Operation
		0.	4-wire SPI
		1	3-wire SPI

1	I2C Watchdog	This bit has meaning only when the host interface is operated in I2C mode. It controls whether the I2C watchdog timer is enabled. <sup>2)</sup>	
		Value	Operation
		0	Disable I2C Watchdog
		1	Enable I2C Watchdog
2	I2C Timeout	This bit has meaning only when the I2C watchdog is enabled. It set the duration of inactivity that triggers the watchdog. <sup>3)</sup>	
		Value	Operation
		0	1 ms timeout
		1	50 ms timeout
3-7	Reserved		

**Notes:**

1) The host may change this bit at any time but only in a single-register transaction (i.e. bursting is not allowed)

2) The host may change this bit at any time but only in a single-register transaction (i.e. bursting is not allowed).

Since the watchdog timer uses the timer oscillator as its time base, the function is not available in deep sleep and is available only after firmware has enabled the timer oscillator.

3) The same restrictions as for the field I2C Watchdog apply also to I2C Timeout.

**12.1.13 Host Status (0x17)**

This register provides a way for the host to determine the chip's power state and the selected host protocol (SPI or I2C; it will already be determined just by virtue of reading this register after reset). This register address allows read-only access.

Table 24: Host Status Register (0x17)

Bit	Identifier	Description	
0	Power State	This bit indicates whether the BHA260AB is active or sleeping.	
		Value	Operation
		0	active
		1	sleeping
1	Host Protocol	This bit indicates which protocol on the host interface was selected at chip start-up.	
		Value	Operation
		0	I2C
		1	SPI
2-3	Reserved		

4	Host Channel 0 Not Ready	These bits are for debugging only. The host normally does not need to check this first. These bits are set whenever firmware is actively loading bytes into the channel just prior to asserting the host IRQ. If the host reads a channel while the Not Ready bit is set, it will read 0s. This indicates that the host should try again (or wait for the Host Interrupt Signal HIRQ to assert, which indicates the data is ready, or wait for the Interrupt Status register to tell if the data is ready, if the host does not utilize the HIRQ signal).	
5	Host Channel 1 Not Ready		
6	Host Channel 2 Not Ready		
7	Host Channel 3 Not Ready		
		<b>Value</b>	<b>Operation</b>
		0	Channel is not ready
		1	Channel is ready

#### 12.1.14 Host Channel CRC (0x18-0x1B)

This 32 bit register holds the CRC calculated over the bytes transferred via the last channel being used. The bit-field initializes to 0xFFFFFFFF (initial CRC seed) every time a different channel is accessed. HOST\_CRC[0] holds the lower byte of the 32-bit CRC and HOST\_CRC[3] the upper byte. The CRC is calculated on a byte-by-byte basis. CRC calculation stops on an underflow error condition. The CRC polynomial follows the ISO-3309 / Ethernet / MPEG-2 standard:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$$

No CRC is calculated when reading from an empty buffer or when the channel is locked.

#### 12.1.15 Fuser2 Product Identifier (0x1C)

This register identifies the Fuser2 core. It reads 0x89 for BHA260AB.

#### 12.1.16 Fuser2 Revision Identifier (0x1D)

This register identifies the hardware revision of the Fuser2 core. It reads 0x02 or 0x03 for the BHA260AB.

#### 12.1.17 ROM Version (0x1E-0x1F)

This 16 bit register contains the build number corresponding to the ROM firmware image of the Fuser2. It reads 0x142E for BHA260AB.

#### 12.1.18 Kernel Version (0x20-0x21)

This 16 bit register contains the build number corresponding to the kernel portion of the firmware image. If none is present, this will read back 0.

#### 12.1.19 User Version (0x22-0x23)

This 16 bit register contains the build number corresponding to the user portion of the firmware image. If none is present, this will read back 0.



### 12.1.20 Feature Status (0x24)

Table 25: Feature Status Register (0x24)

Bit	Identifier	Description								
0	Reserved	This bit always reads 0								
1	OpenRTOS Framework	Indicates if the current firmware uses OpenRTOS instead of the legacy interrupt-based framework. Currently, this bit always reads as 1. The value is valid after the firmware has initialized								
		<table border="1"> <thead> <tr> <th>Value</th> <th>Operation</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>OpenRTOS not used</td> </tr> <tr> <td>1</td> <td>OpenRTOS used</td> </tr> </tbody> </table>	Value	Operation	0	OpenRTOS not used	1	OpenRTOS used		
		Value	Operation							
0	OpenRTOS not used									
1	OpenRTOS used									
2-4	Host Interface ID	This indicates the Android interface compatibility. The value depends on the release of the Software Development Kit. The value is valid after the firmware has initialized.								
		<table border="1"> <thead> <tr> <th>Value</th> <th>Operation</th> </tr> </thead> <tbody> <tr> <td>2</td> <td>Android M</td> </tr> <tr> <td>3</td> <td>Android N</td> </tr> <tr> <td>4</td> <td>Android O</td> </tr> </tbody> </table>	Value	Operation	2	Android M	3	Android N	4	Android O
		Value	Operation							
		2	Android M							
3	Android N									
4	Android O									
5-7	Algorithm Id	This indicates the version of the sensor fusion algorithms. BHA260AB uses BSX. The value is valid after the firmware has initialized.								
		<table border="1"> <thead> <tr> <th>Value</th> <th>Operation</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>none</td> </tr> <tr> <td>1</td> <td>BSX</td> </tr> </tbody> </table>	Value	Operation	0	none	1	BSX		
		Value	Operation							
0	none									
1	BSX									

### 12.1.21 Boot Status (0x25)

Table 26: Boot Status Register (0x25)

Bit	Identifier	Description
0-3	Reserved	These bits always read 0
4	Host Interface Ready	<p>After reset, the host should poll this register until the Host Interface Ready bit is set. Once set, the host interface is ready for the host to upload program RAM or do other operations. Until this, the host should not attempt to write commands.</p> <p>The Host Interface Ready bit will clear once an upload has completed and the host has issued a Boot RAM command. Once boot of the uploaded image completes, the Host Interface Ready bit will be set again. Until this, the host should not attempt to write commands during the window when Host Interface Ready is clear.</p>

		<b>Value</b>	<b>Operation</b>
		0	Not ready
		1	Ready
5	Firmware Verify Done	Indicates whether the verification of the firmware image loaded via the host interface is done.	
		<b>Value</b>	<b>Operation</b>
		0	Verification not done
		1	Verification done
6	Firmware Verify Error	Indicates whether the verification of the firmware image loaded via the host interface was successful.	
		<b>Value</b>	<b>Operation</b>
		0	Verification passed
		1	Verification failed
7	Firmware Idle	Indicates whether the firmware is running or halted.	
		<b>Value</b>	<b>Operation</b>
		0	Firmware running
		1	Firmware halted

See Section 5 for the usage of these bits.

### 12.1.22 Host Interrupt Timestamp (0x26-0x2A)

This area is written with the 40 bit timestamp of the host IRQ assertion or the Host Event Request, whichever the host has requested.

If the `Timestamp_Event_Request` bit in the *Host Interface Control (0x06)* register **is not set**, the timestamp written here will be the time of the HIRQ (Host Interrupt Signal) assertion. The update occurs immediately before the Interrupt Status register is updated and the HIRQ signal is asserted.

If the `Timestamp_Event_Request` bit in the *Host Interface Control (0x06)* register **is set**, actual Host Interrupt Timestamp will only be accessible via the Timestamps parameter.

### 12.1.23 Chip ID (0x2B)

This register outputs the value of the Bosch Sensortec product specific identifier. For BHA260AB it reads 0x74 or 0xF4.

### 12.1.24 Interrupt Status (0x2D)

This provides a way for a host that does not use the Host Interrupt Signal HIRQ, to determine when data is available in one of the three output channels: either Wake-up FIFO, Non-Wake-up FIFO or the Status and Debug FIFO.

*Note:*

The time at which the host interrupt was asserted can be queried via the Host Interrupt Timestamp Parameter or via the Host Interrupt Timestamp registers (if enabled); the later follow the Interrupt Status in the register address space, so it is possible for the host to read both items in one transaction.

The interrupt mask bits in the Host Interface Control register (one for each of the two FIFOs) will suppress only the Host Interrupt bit and Host IRQ signal. The bits of this register however will remain asserted, so that the host can still learn that there is a pending interrupt by reading this register. If any bits are set, there is a pending interrupt.

These bits are cleared when the host reads the Wake FIFO, Non-Wake FIFO, or Status channels, for just the channel that has been read from. The Host Interrupt Asserted bit and the HOST IRQ signal will remain asserted until all interrupt sources are retrieved by the host.

Table 27: Interrupt Status Register (0x2D)

Bit	Identifier	Description	
0	Host Interrupt Asserted	This bit reflects the state of the host interrupt GPIO pin.	
		Value	Operation
		0	Host interrupt not asserted
		1	Host interrupt asserted
1-2	Wake-up FIFO Status	The value of these 2 bits show if the data in the Wake-up FIFO fulfills the condition for generating an interrupt. The value will be 1 "Immediate", if a virtual sensor event has occurred which was configured with no latency; the value is 2 "Latency" if a sensor has generated data, and the latency period has expired; the value will be 3 "Watermark" if the watermark for the respective FIFO was reached.	
		Value	Operation
		0	No data in this FIFO at the time this Int Status was generated
		1	Immediate (sensor with 0 latency now has data)
		2	Latency (latency timed out for sensor with non-zero latency)
		3	Watermark (watermark reached in FIFO)
3-4	Non-Wake-up FIFO Status	The value of these 2 bits show if the data in the Non-Wake-up FIFO fulfills the condition for generating an interrupt. The value will be 1 "Immediate", if a virtual sensor event has occurred which was configured with no latency; the value is 2 "Latency" if a sensor has generated data, and the latency period has expired; the value will be 3 "Watermark" if the watermark for the respective FIFO was reached.	
		Value	Operation
		0	No data in this FIFO at the time this Int Status was generated
		1	Immediate (sensor with 0 latency now has data)

		2	Latency (latency timed out for sensor with non-zero latency)
		3	Watermark (watermark reached in FIFO)
5	Status	This bit is set if there is data in the Status and Debug FIFO as a result of a command, when the Status and Debug FIFO is in Synchronous Mode (Async bit in <i>Host Interface Control (0x06)</i> is clear).	
		Value	Operation
		0	No data in the Status and Debug FIFO
		1	Data in the Status and Debug FIFO
6	Debug	This bit is set if asynchronous command responses or debug data is currently stored in the Status and Debug FIFO. If the Async bit in <i>Host Interface Control (0x06)</i> is set (Async Mode), then even response packets as a result of a command will set this bit, not the Status bit.	
		Value	Operation
		0	No data in the Status and Debug FIFO
		1	Data in the Status and Debug FIFO
7	Reset or Fault	This bit is set if this interrupt was caused by a fatal error. The Reset bit will be set after reset, and cleared when the initialization sequence completes and the host starts the code RAM upload. A reset will normally <i>not</i> generate a hardware interrupt, as the host needs to configure the interrupt type required using the Host Interrupt Control register before it is safe for the Fuser2 Core to assert it. However, if the reset occurred due to a watchdog reset or fatal error, and memory records the previous Host Interrupt Control value, a hardware interrupt can be generated.	
		Value	Operation
		0	No data in the Status and Debug FIFO
		1	Data in the Status and Debug FIFO

### 12.1.25 Error Value (0x2E)

The Error Value register reports an internal error code. Some of this information is also reported in the Status Interface using the Error Meta Event.

Error recovery strategies are described in Section 17

Table 28: Error Value Register (0x2E)

Error Register Values	Description (* indicates can be issued from bootloader)	Error Category
0x00	*No Error	
0x10	*Firmware Expected Version Mismatch	Fatal
0x11	*Firmware Upload Failed: Bad Header CRC	Fatal

0x12	*Firmware Upload Failed: SHA Hash Mismatch	Fatal
0x13	*Firmware Upload Failed: Bad Image CRC	Fatal
0x14	*Firmware Upload Failed: ECDSA Signature Verification Failed	Fatal
0x15	*Firmware Upload Failed: Bad Public Key CRC	Fatal
0x16	*Firmware Upload Failed: Signed Firmware Required	Fatal
0x17	*Firmware Upload Failed: FW Header Missing	Fatal
0x19	*Unexpected Watchdog Reset	Fatal
0x1A	ROM Version Mismatch	Fatal
0x1B	*Fatal Firmware Error	Fatal
0x1C	Chained Firmware Error: Next Payload Not Found	Fatal
0x1D	Chained Firmware Error: Payload Not Valid	Fatal
0x1E	Chained Firmware Error: Payload Entries Invalid	Fatal
0x1F	*Bootloader Error: OTP CRC Invalid	Fatal
0x20	Firmware Init Failed	Hardware
0x21	Sensor Init Failed: Unexpected Device ID	Hardware
0x22	Sensor Init Failed: No Response from Device	Programming
0x23	Sensor Init Failed: Unknown	Programming
0x24	Sensor Error: No Valid Data	Programming
0x25	Slow Sample Rate	Temporary
0x26	Data Overflow (saturated sensor data)	Fatal
0x27	Stack Overflow	Fatal
0x28	Insufficient Free RAM	Fatal
0x29	Sensor Init Failed: Driver Parsing Error	Fatal
0x2A	Too Many RAM Banks Required	Programming
0x2B	Invalid Event Specified	Programming
0x2C	More than 32 On Change	Programming
0x2D	Firmware Too Large	Fatal
0x2F	Invalid RAM Banks	Fatal
0x30	Math Error	Fatal
0x40	Memory Error	Fatal
0x41	SWI3 Error	Fatal
0x42	SWI4 Error	Fatal
0x43	Illegal Instruction Error	Fatal
0x44	*Unhandled Interrupt Error / Exception / Postmortem Available	Fatal
0x45	Invalid Memory Access	Fatal

0x50	Algorithm Error: BSX Init	Programming
0x51	Algorithm Error: BSX Do Step	Programming
0x52	Algorithm Error: Update Sub	Programming
0x53	Algorithm Error: Get Sub	Programming
0x54	Algorithm Error: Get Phys	Programming
0x55	Algorithm Error: Unsupported Phys Rate	Programming
0x56	Algorithm Error: Cannot find BSX Driver	Programming
0x60	Sensor Self-Test Failure	Hardware
0x61	Sensor Self-Test X Axis Failure	Hardware
0x62	Sensor Self-Test Y Axis Failure	Hardware
0x64	Sensor Self-Test Z Axis Failure	Hardware
0x65	FOC Failure	Hardware
0x66	Sensor Busy	Hardware
0x6F	Self-Test or FOC Test Unsupported	Programming
0x72	No Host Interrupt Set	Fatal
0x73	Event ID Passed to Host Interface Has No Known Size	Programming
0x75	Host Download Channel Underflow (Host Read Too Fast)	Temporary
0x76	Host Upload Channel Overflow (Host Wrote Too Fast)	Temporary
0x77	Host Download Channel Empty	Temporary
0x78	DMA Error	Hardware
0x79	Corrupted Input Block Chain	Programming
0x7A	Corrupted Output Block Chain	Programming
0x7B	Buffer Block Manager Error	Programming
0x7C	Input Channel Not Word Aligned	Temporary
0x7D	Too Many Flush Events	Temporary
0x7E	Unknown Host Channel Error	Hardware
0x81	Decimation Too Large	Programming
0x90	Master SPI/I2C Queue Overflow	Fatal
0x91	SPI/I2C Callback Error	Fatal
0xA0	Timer Scheduling Error	Fatal
0xB0	Invalid GPIO for Host IRQ	Fatal
0xB1	Error Sending Initialized Meta Events	Fatal
0xC0	*Command Error	Temporary
0xC1	*Command Too Long	Temporary
0xC2	*Command Buffer Overflow	Temporary

0xD0	User Mode Error: Sys Call Invalid	Fatal
0xD1	User Mode Error: Trap Invalid	Fatal
0xE1	Firmware Upload Failed: Firmware header corrupt	Fatal
0xE2	Sensor Data Injection: Invalid input stream	Programming

### 12.1.26 Error Aux, Debug Value, Debug State (0x2F-0x31)

The Error Aux, Debug Value, and Debug State registers are for Bosch Sensortec internal troubleshooting.

Table 29: Debug State Register values (0x31)

Debug State Register Values	Description
0x00-0x0F	Bootloader
0x10-0x1F	Firmware initialization stage 1
0x90-0x9F	Firmware initialization stage 2
0x20-0x2F	Outerloop (main routine, non-RTOS)
0x30-0x7F	Reserved
0x80-0x8F	Deinitialization (teardown on fatal failure)
0xA0-0xAF	Fatal failure
0xB0-0xBF	Inside RTOS task (0xBF = idle task = sleeping)

### 12.1.27 RGP5 – RGP7 - General-Purpose Host Readable (0x32-0x3D)

General-purpose registers for communication from the Fuser2 Core to the host. While the BHA260AB ARC processor can read/write these registers, the host has read access only.

These registers are available for customer use in custom drivers or extensions.

## 13 Host Interface Commands

The general structure of host commands and their responses are described in Section 4.4.2 *Host Command Protocol*.

This section describes the details of all available commands and their responses. The following table provides the overview of the existing commands and their availability in the bootloader and the Event-driven Software Framework.

Table 30: Overview BHA260AB Host Interface Commands

Command Id	Available in		Command	
	Bootloader	Framework	Description	Response Packet on Status Channel
0x0001	✓		Download Post Mortem Data	Crash Dump Status Packet
0x0002	✓		Upload to Program RAM	- none -
0x0003	✓		Boot Program RAM	- none -
0x0007		✓	Set Sensor Data Injection Mode	Injected Sensor Configuration Request Status Packet
0x0008		✓	Inject Sensor Data	- none -
0x0009		✓	FIFO Flush	- none -
0x000A		✓	Soft Pass-Through	Soft Pass-Through Results Status Packet
0x000B		✓	Request Sensor Self-Test	Sensor Self-Test Results Status Packet
0x000C		✓	Request Sensor Fast Offset Compensation	Sensor FOC Results Status Packet
0x000D		✓	Configure Sensor	- none -
0x000E		✓	Change Sensor Dynamic Range	- none -
0x000F		✓	Set Change Sensitivity	- none -
0x0010		✓	Debug Test	- none -
0x0011		✓	DUT Continue	DUT Test Status Packet
0x0012		✓	DUT Start Test	DUT Test Status Packet
0x0015		✓	Control FIFO Format	- none -
0x0017	✓		Raise Host Interface Speed	Raise Host Interface Speed Status Packet
0x0100 ... 0x0FFF		✓	Set Parameter (see Section 13.3)	Set Parameter Status Packet
0x1000 ... 0x1FFF		✓	Get Parameter (see Section 13.3)	Get parameter Status Packet
All others			Reserved	

Note:



Some commands do not return a status packet; however, a garbled or invalid command will receive a command error status packet. Bootloader commands can fail, so it is required that the host checks for and handles errors on any command.

## 13.1 Bootloader Commands (incl. bootloader status codes)

### 13.1.1 Download Post Mortem Data (0x0001)

This command requests that the device places all available data relevant to a watchdog or exception-related reset in the Status Interface for the host to read out. There are no additional fields.

Table 31: Download Post Mortem Data - Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Download Post Mortem Data Command = 0x0001
Length	0x02-0x03	Total number of bytes to follow = 0

A Crash Dump status packet will be returned in the status channel. It contains useful information as possible, such as stack trace, register values, key data structures, etc.

Table 32: Download Post Mortem Data - Response

Field Name	Byte Offset	Description
Status Code	0x00-0x01	Crash Dump Status Code = 0x0003
Length	0x02-0x03	Total number of bytes to follow
Context	0x04-0x83	R0-R26; GP, FP, SP, ILink, R30, Blink
Valid	0x84	0 = no info; 1 = crash occurred
Flags	0x85	1 = Host Interrupt Control valid 2 = Stack info valid 4 = Stack contents included 8 = Do not autoexec following reset
Stack Size	0x86-0x87	Number of bytes in ROM/Kernel stack
Stack Start	0x88-0x8B	RAM address of start of ROM/Kernel stack
ERET	0x8C-0x8F	See Reference 4 in Section 26
ERBTA	0x90-0x93	See Reference 4 in Section 26
ERSTATUS	0x94-0x97	See Reference 4 in Section 26
ECR	0x98-0x9B	See Reference 4 in Section 26
EFA	0x9C-0x9F	See Reference 4 in Section 26
Diagnostic	0xA0-0xA3	See Table 34
Icause	0xA4-0xA7	See Reference 4 in Section 26
Debug Value	0xA8	Host register "Debug Value"

Debug State	0xA9	Host register "Debug State"
Error Reason	0xAA	Host register "Error Value"
Interrupt State	0xAB	Host register "Error Aux"
Host Interrupt Control	0xAC	Host register "Host Interrupt Control"
Reset Reason	0xAD	0 = POR 1 = External 2 = Host 4 = Watchdog Timeout (or Fatal Exception – see ECR register below)
Host Reset Count	0xAE	Internal use
Error Report	0xAF	Exception cause 0x19 = Watchdog reset 0x1B = Fatal error 0x44 = Unhandled interrupt
MPU_ECR	0xB0-0xB3	See Reference 4 in Section 26
User SP	0xB4-0xB7	User stack pointer, see Reference 4 in Section 26
Reserved	0xB8-0xCC	Reserved bytes
Stack CRC	0xC4-0xC7	CRC of bytes 0xCC-Stack Size + 0xCB
CRC	0xC8-0xCB	CRC of bytes 0x04-0x0C7
Stack dump	0xCC... Stack Size + 0xCB	ROM/Kernel stack dump

Table 33: Diagnostic bit field

Bits	Name	Description
0-3	pm_state	This bit-field reflects the current power state of the BHA260AB (see Section 5.5). 0x0, 0x1: PWRUP 0x2, 0x3: ACTV 0x4, 0x5: LSLP 0x8: SLP 0x9: DSLP 0x6: RAMP 0x7, 0xA, 0xB: sleep transitions Others: reserved
4	rom_acc_err	When '1', this bit indicates that a write access was attempted to a ROM location.
5	iccm_acc_err	When '1', this bit indicates that a reserved location in the ICCM memory region has been accessed.
6	dccm_acc_err	When '1', this bit indicates that a reserved location in the DCCM memory region has been accessed.
7	Reserved-	
8-10	rst_src	This bit-field relays the source of a reset. It is cleared on a POR; otherwise, the encoding is:

		[0]=1: external reset happened [1]=1: host command reset happened [2]=1: watchdog reset happened
11-16	Reserved-	

### 13.1.2 Upload to Program RAM (0x0002)

The ROM bootloader will configure the input channel and its DMA interface to allow for upload of a firmware image to program RAM. The data stream will include:

Table 34: Upload to Program RAM – Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Upload to Program RAM Command = 0x0002
Length	0x02-0x03	Total number of 32 bit words to follow
Firmware Image	0x04.. Length + 0x03	Header + executable image

If the firmware image is found to be invalid (bad ECDSA signature or mismatched CRC), a System Error Status Packet will result. Once the image is verified, the host must issue a *Boot Program RAM (0x0003)* command. Once initialization is complete, a *Meta Event: Initialized* will be placed in both the Wake-up and Non-Wake-up FIFOs.

### 13.1.3 Boot Program RAM (0x0003)

This command is required to start a firmware image in RAM. However, this command will be ignored if the ECDSA or CRC verification failed.

Table 35: Boot Program RAM – Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Boot Program RAM Command = 0x0003
Length	0x02-0x03	Total number of bytes to follow = 0

### 13.1.4 Raise Host Interface Speed (0x0017)

This command is required to be sent by the host before a firmware upload at an SPI clock rate higher than 20MHz can be performed. It will switch BHA260AB from the power-saving Long Run mode into Turbo mode. To return to Long Run mode a reset has to be performed. However when the firmware is loaded, the power mode is defined by the firmware.

Table 36: Raise Host Interface Speed - Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Raise Host Interface Speed Command = 0x0017
Payload <sup>1)</sup>	0x02–0x07	0x02, 0x00, 0x80, 0x00, 0x00, 0x00

Note:

1) This parameter has a special protocol and doesn't match the regular command format.

The Raise Host Interface Speed status packet is sent upon successful reception of the command. If an unexpected status packet is available, the command must be resent until the correct response is received.

Table 37: Raise Host Interface Speed - Response

Field Name	Byte Offset	Description
Status Code	0x00-0x01	0x0F, 0x00
Length	0x02-0x03	0x04, 0x00
Command	0x04-0x05	0x17, 0x00
Status	0x06-0x07	0x00, 0x00

## 13.2 Main Firmware Commands (incl. main firmware status codes)

### 13.2.1 Set Sensor Data Injection Mode (0x0007)

The purpose of the Sensor Data Injection mode is to support debugging and firmware-in-the-loop testing. In this mode, the sensor input data to the algorithms executed in the BHA260AB firmware is provided by the host, instead of the physical sensors. A dedicated firmware version is required for this, which replaces the standard physical sensor drivers with data injection drivers. This can be compiled using the SDK.

This command gives the host the ability to enable either real time operation using injected sensor data (which must arrive fast enough to keep the BSX sensor fusion running at the proper rate), or step-by-step operation.

When in either injection mode, the internal requirements by the BSX sensor fusion library regarding the on/off state and required rates for each physical sensor will be passed back to the host through the Status Interface. The host must use this information to provide simulated sensor data at the proper rates (timestamp deltas).

In order for sensor data injection to work, the firmware image must be built with special physical sensor drivers which help with sensor data injection, rather than normal physical sensor drivers.

Table 38: Set Sensor Data Injection Mode – Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Set Sensor Data Injection Mode Command= 0x0007
Length	0x02-0x03	Total number of bytes to follow = 4
Injection Mode	0x04	Normal run mode: 0 = normal run mode Injection mode: 1 = real time injection 2 = step-by-step

Reserved	0x05-0x07	Reserved
----------	-----------	----------

The Injected Sensor Configuration Request Status Packet will be issued whenever the BSX fusion library requires a change to a simulated physical sensor, such as turning it on or off or changing its (simulated) rate. A requested Sample Rate of 0 means the host should stop injecting sensor data for this simulated physical sensor.

The host must ensure the timestamps for each incoming simulated physical sensor sample are increasing by the required period (1 divided by the rate).

Table 39: Set Sensor Data Injection Mode – Response

Field Name	Byte Offset	Description
Status Code	0x00-0x01	Injected Sensor Configuration Request Status Code = 0x0004
Length	0x02-0x03	Total number of bytes to follow = 8
Sample Rate	0x04-0x07	32 bit floating point number, in Hz
Sensor ID	0x08	Physical sensor ID
Reserved	0x09-0x0B	Reserved

### 13.2.2 Inject Sensor Data (0x0008)

Injected sensor data contents must follow the command outlined below:

Table 40: Inject Sensor Data – Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Inject Sensor Data Command = 0x0008
Length	0x02-0x03	Total number of bytes to follow, maximum of 124 (pad to multiple of 4)
Data	0x04.. Length + 0x03	One or more sets of: [Sensor ID, Sensor Data, Timestamp]

The format of the injected data follows the output format for virtual sensor events described in Section 15.1. This includes the full and delta timestamps, which allows to optimize the injected data stream by including delta or full timestamps only when necessary. Due to this format it's basically possible to replay previously recorded output data.

Sensor data injection must be enabled first using the Set Sensor Data Injection Mode Command. That allows the host to control whether real-time or step-by-step injection is to be performed.

The firmware must be built with special sensor data injection drivers instead of normal physical sensor drivers. These drivers intercept on/off and rate requests from the BSX fusion library and pass them to the host as Sensor Data Injection Request Status Interface packets. These drivers receive the injected sensor data and pass it to the rest of the system, including BSX.

The upper bounds for injected sensor data rates is determined by host bus utilization, CPU utilization, RAM available for storing the injected data, and number of injected samples per host injection. Injecting more per transaction will be more efficient than injecting single sensor samples per host transaction.

The host must ensure the timestamps for each incoming simulated physical sensor sample are increasing by the required period (1 divided by the rate).

### 13.2.3 FIFO Flush (0x0009)

With this command the host can request BHA260AB at any time to initiate a data transfer of the content in one or more FIFOs from BHA260AB to the host. The BHA260AB will assert the Host Interrupt Signal HIRQ, if data is available.

Furthermore, the FIFO Flush command can also be used to discard rather than transfer the data content of any of the FIFOs.

Table 41: FIFO Flush – Command

Field Name	Byte Offset	Description	
Command	0x00-0x01	FIFO Flush Command = 0x0009	
Length	0x02-0x03	Total number of bytes to follow = 4	
Flush Value	0x04	Value	Operation
		0xFF	Flushes (sends) all data of wake-up and non-wake-up FIFOs
		0xFE	Clears out (discards, not sent to host) all FIFOs
		0xFD	Flushes (sends) the wake-up FIFO only
		0xFC	Flushes (sends) the non-wake-up FIFO only
		0xFB	Clears out (discards) the wake-up FIFO
		0xFA	Clears out (discards) the non-wake-up FIFO
		0xF9	Clears out (discards) the debug/status FIFO
Reserved	0x05-0x07	Reserved	

FIFO Flush is an optional mechanism. In the standard FIFO read mechanism, the host does not use this register. Instead, then the watermark interrupt, sensor latency interrupt, wake-up and/or non-wake-up FIFO interrupts are used, what determine when the host interrupt occurs without the need of a FIFO Flush command. More details about FIFO reading can be found in Chapter: 16 Reading FIFO Data.

The type of FIFO Flush request determines also to which FIFOs the Flush Complete Meta Event will be appended to. The 0xFC (Flush Non-Wake-up) and 0xFD (Flush Wake-up) place the message in the respective FIFO. The 0xFF (Flush All) request will result in a Flush Complete Meta Event in both FIFOs.

If a FIFO Flush command is sent with a discard flush type (0xF9, 0xFA, 0xFB, or 0xFE), then the current FIFO transfer is canceled and data is lost for the specified FIFO(s).

### 13.2.4 Soft Pass-Through (0x000A)

This command can be used during normal operation to read and write registers on devices attached to the BHA260AB secondary master I2C or SPI busses. It has two selectable operating modes, the Sensor Driver Mode, using firmware built-in physical device drivers, and the Arbitrary Device Mode, allowing access to external devices without the need for a physical device driver by configuring the transfer parameters. The selection between the two depends on the value of the Master Bus field in Table 49 or

Table 50 respectively.

Table 42: Soft Pass-Through – Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Soft Pass-Through Command = 0x000A
Length	0x02-0x03	Total number of bytes to follow, (pad to multiple of 4) <sup>1)</sup>
Mode	0x04-0x05	Select Sensor Driver Mode or Arbitrary Device Mode. Bit fields defined in Table 49 or Table 50 below respectively
Transfer Rate	0x06-0x07	SPI clock rate in kHz <sup>2)</sup>
Cmd Config / Physical Sensor ID	0x08	Arbitrary Device Mode: Bit fields defined in Table 51 below (ignored for I2C) Sensor Driver Mode: Physical Sensor ID, see list in Section 13.3.2.6
Slave Address	0x09	7 bit I2C slave address or GPIO pin for chip select
Num Bytes	0x0A	Number of bytes to read/write (up to 244)
Reg	0x0B	Register address to read from/write to
(Write Data)	0x0C..Num Bytes + 0x0B	If a write command, the data to write

**Notes:**

- 1) The host must write commands in multiples of 4 bytes. If the total number of bytes to follow is not a multiple of 4, the host should write additional bytes to make it so, and also include them in the Length field. The Num Bytes field indicates the actual number of bytes to read or write.
- 2) Transfer Rate specifies the SPI clock rate in kHz (16bit unsigned int, LSB first). For I2C master busses, this value is ignored and the rate configured by the firmware image is used instead. The available clock speeds are derived from the current system oscillator (depending on Turbo / Long-Run mode) by dividing by the following values: 2, 3, 4, 6, 8, 16, 32. If a value is selected, that does not follow this rule, the firmware will select the highest possible value below the selected rate.

Table 43: Sensor Driver Mode description

Field Name	Bit Offset	Description
Direction	0	0 = read, 1 = write
Transfer Type	1	0 = single burst, 1 = multiple separate single transfers

Delay Control	2	0 = none (fastest), 1 = delay between bytes
Master Bus <sup>1)</sup>	3-4	0 = Sensor Driver Mode is used 1, 2, 3 = Arbitrary Device Mode is used. 1 = M1, 2 = M2, 3 = M3; See Table 50 for the rest of the Fields.
Reserved	5-7	Reserved
Delay Value	8-13	0 – 3: Invalid 4+: Multiples of 50 $\mu$ s (ignored if Delay Control = 0)
Reserved	14-15	Reserved

Table 44: Arbitrary Device Mode description

Field Name	Bit Offset	Description
Direction	0	0 = read, 1 = write
Transfer Type	1	0 = single burst, 1 = multiple separate single transfers
Delay Control	2	0 = none (fastest), 1 = delay between bytes
Master Bus <sup>1)</sup>	3-4	0 = Sensor Driver Mode is used. See Table 49 for the rest of the Fields 1 = M1, 2 = M2, 3 = M3; Arbitrary Device Mode is used. See the rest of this table
SPI Mode	5	0 = 4 wire, 1 = 3 wire (ignored if I2C)
CPOL	6	SPI clock polarity (ignored if I2C)
CPHA	7	SPI clock phase (ignored if I2C)
Delay Value	8-13	0 – 3: Invalid 4+: Multiples of 50 $\mu$ s (ignored if Delay Control = 0)
CS Level	14	Chip Select level (ignored if I2C): 0 = CS active low 1 = CS active high
LSbit First	15	Least significant bit first (ignored if I2C)

**Note:**

1) It is not possible to use a Master Bus that has not been previously enabled and initialized by the firmware image.

The value of this field selects between 'Sensor Driver Mode' (value 0) and 'Arbitrary Device Mode' (values 1-3)

The *Mode* Field in the *Soft Pass-Through* Command specifies how the transfer should occur. This includes direction: read or write, single burst or multiple single transfers, and fast or with a delay between bytes. The delay can be specified using 6 bits in multiples of 50  $\mu$ s (with 0 meaning no delay and a minimum of 4, meaning 200  $\mu$ s). SPI Mode specifies 3 or 4 wire SPI, if no device sensor driver has already configured the mode, as well as CPOL, CPHA, CS Level, and LSbit First. Master Bus Numbers 1-3 correspond to the hardware master bus number (M1 – M3), 0 for using a sensor descriptor of the firmware image.



The *Command Config* Field specifies how the firmware should format the command byte for SPI transactions. The position of the read bit indicates which bit in the command byte determines whether the command is a read or a write. This is usually either bit 0 or bit 7. The polarity of the read bit indicates how to interpret the read bit. A value of 0 means that a value of 0 in the read bit position is a read command, and a value of 1 means that a value of 1 in the read bit position is a read command. The address shift indicates how many bits to left shift the register by when including it in the SPI command byte. If the read bit position is 7, the address shift will typically be 0, while if the read bit position is 0, the address shift will typically be 1.

Table 45: CMD config description

Field Name	Bit Offset	Description
Address Shift	0-3	Number of bits to shift register address in command
Polarity of Read Bit	4	0 = active low, 1 = active high
Position of Read Bit	5-7	Bit number in command byte containing the read bit

The *Slave Address* Field in the *Soft Pass-Through* Command should be the 7 bit I2C slave address if the master bus is I2C, or the GPIO pin for the chip select if the master bus is SPI.

The *Reg* Field indicates which register on the connected device to read from or write to. When communicating with a SPI device in read mode, this register will be written first in a write transaction, then a second transaction will be performed to read the output data. In I2C mode, it will always be written with a write transaction followed by a RESTART condition and a read transaction.

The (*Write Data*) Field should be provided only if the Mode indicates this is a write transaction.

Once the command completes, a Soft Pass-Through status packet is returned in the Status Channel:

Table 46: Soft pass through - Response

Field Name	Byte Offset	Description
Status Code	0x00-0x01	Soft Pass-Through Results Status Code = 0x0005
Length	0x02-0x03	Total number of bytes to follow = 9 + Num Bytes (padded to multiple of 4)
Completion Status	0x04	1 = successful I2C transaction; (for SPI always 1) 2 = I2C NACK or I2C error (no equivalent for SPI)
Mode	0x05-0x06	Same as Soft Pass-Through command. Sensor Driver Mode or Arbitrary Device Mode. Bit fields defined in Table 49 or Table 50 respectively
Transfer Rate	0x07-0x08	SPI clock rate in kHz (same as requested, always 0 for I2C)
Cmd Config	0x09	Same as Soft Pass-Through command. Arbitrary Device Mode: Bit fields defined in Table 51 below (ignored for I2C) Sensor Driver Mode: Physical Sensor ID, see list in Section 13.3.2.6

Slave Address	0x0A	7 bit I2C slave address or GPIO pin for chip select
Num Bytes	0x0B	Number of bytes read/written
Reg	0x0C	Starting register for the read/write
(Read Data)	0x0D.. Num Bytes + 0x0C	If a read command, the data that was read

This returns a status for both I2C and SPI transfers, though SPI cannot detect that a slave device is not responding. It also returns the mode, transfer rate, cmd config, and slave address, to help the host distinguish between results for multiple commands requests. If the command was a read command, the read data is also returned. Returned data will be padded to make the status packet a multiple of 4 bytes. The actual number of bytes read will be indicated in the Num Bytes field.

### 13.2.5 Request Sensor Self-Test (0x000B)

This command requests that a specific physical sensor runs its self-test and reports the results.

Table 47: Request Sensor Self Test - Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Request Sensor Self Test Command = 0x000B
Length	0x02-0x03	Total number of bytes to follow = 4
Sensor ID	0x04	Sensor ID (only physical sensors)
Reserved	0x05-0x07	Reserved

Sensor ID must be a valid BSX physical input sensor ID. To perform a physical sensor Self Test, the sensor must be in inactive mode. If the specified physical sensor is currently active because a related virtual sensor is enabled, the command will be rejected by reporting an error in the Error Register (no status packet will be returned), see Section 12.1.25.

The synchronous Sensor Self-Test Results status packet will be returned once the self-test of this sensor is completed:

Table 48: Sensor self-test result - Response

Field Name	Byte Offset	Description
Status Code	0x00-0x01	Sensor Self-Test Result Status Code = 0x0006
Length	0x02-0x03	Total number of bytes to follow = 8
Sensor ID	0x04	Which sensor is reporting results
Test Status	0x05	0: Test Passed 1: X Axis Failed 2: Y Axis Failed 4: Z Axis Failed 7: Multiple Axis Failed or Single Test Failed (if testing of each axis cannot be done) 8: Unsupported: physical sensor driver does not

		implement self-test 9: No Device: physical sensor ID provided is invalid
X Offset	0x06-0x07	16 bit value, format defined by physical sensor driver
Y Offset	0x08-0x09	
Z Offset	0x0A-0x0B	

Sensor ID indicates which sensor is reporting test results.

X, Y, and Z Offset values are returned if available from the self-test results for this sensor, otherwise returned as 0.

- For the built-in Accel sensor, the offset values are returned in mg as 16bit signed values.
- For the built-in Gyro, no offset values are returned.
- For other external sensors, the value is defined by the design of the physical driver.

### 13.2.6 Request Sensor Fast Offset Compensation (0x000C)

This command requests that a specific physical sensor runs fast offset compensation and reports the results.

Table 49: Request Sensor Fast Offset Compensation - Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Request Sensor Fast Offset Compensation Command = 0x000C
Length	0x02-0x03	Total number of bytes to follow = 4
Sensor ID	0x04	Sensor ID
Reserved	0x05-0x07	Reserved

Sensor ID must be a valid BSX physical input sensor ID.

The following status packet returns the results of the Sensor FOC Request command.

Table 50: Sensor fast offset compensation - Response

Field Name	Byte Offset	Description
Status Code	0x00-0x01	Sensor Fast Offset Calibration Result Status Code = 0x0007
Length	0x02-0x03	Total number of bytes to follow = 8
Sensor ID	0x04	Which sensor is reporting results
FOC Status	0x05	See values below
X Offset	0x06-0x07	16 bit value in LSBs of the physical sensor
Y Offset	0x08-0x09	
Z Offset	0x0A-0x0B	

Sensor ID indicates which sensor is reporting test results.

The FOC status can be ERROR\_FOC\_FAIL (0x65), ERROR\_UNKNOWN (0x23), or SUCCESS (0).

### 13.2.7 Configure Sensor (0x000D)

This command allows the host to turn on or off a virtual sensor, as well as set its sample rate and latency.

Table 51: Configure Sensor - Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Configure Sensor Command = 0x000D
Length	0x02-0x03	Total number of bytes to follow = 8
Sensor ID	0x04	Sensor ID
Sample Rate	0x05-0x08	Floating point sample rate, in Hz
Latency	0x09-0x0B	Number of milliseconds (24 bits)

The *sensor ID* field can be any of the virtual sensors supported by the currently loaded firmware image – either non-wake-up or wake-up sensor.

The *sample rate* field is specified as a 32 bit float, allowing for a very wide range of rates. The host can turn off a sensor by setting the rate to 0.

The 24 bit *Latency* field is specified as 0 to request no latency or a non-zero number in milliseconds for the allowed delay before reporting this sensor's data to the host. This allows for latencies up to 4.66 hours.

Changes to the Sample Rate field take effect quickly, but not immediately. If the host wishes to know when the rate change is completed, it can enable and wait for the Sample Rate Changed Meta Event. If the host wishes to know when a sensor has been turned on or off, it can enable and monitor the Power Mode Changed Meta Event.

The actual rate is selected to be equal to or greater than the requested rate and twice that rate:

$$\text{Requested Rate} \leq \text{Actual Rate} < \text{Requested Rate} \times 2$$

The underlying BSX sensor fusion library supports the following rates:

1.5625 Hz, 3.125 Hz, 6.25 Hz, 12.5 Hz, 25 Hz, 50 Hz, 100 Hz, 200 Hz, 400 Hz, 800 Hz

So the requested rate will be modified to match the nearest value equal to or greater than the requested rate.

*Note:*

*The rates mentioned above are typical rates. Due to the accuracy of the rates of the underlying sensors, the actual rate may vary according to the tolerance of the sensors.*

Changes to the Max Report Latency take effect immediately. If a timer for the sensor using a different Max Report Latency is running, it will be updated. Due to timing it is possible for a change to be slightly too late to effect the current timer, but will affect subsequent samples. If Max Report Latency is set to 0 when it was previously not 0, then this will be treated the same as a flush request and result in immediate host transfer request.

### 13.2.8 Change Sensor Dynamic Range (0x000E)

This command allows the host to select a different dynamic range for a virtual sensor.

Table 52: Change Sensor Dynamic Range - Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Change Sensor Dynamic Range Command = 0x000E
Length	0x02-0x03	Total number of bytes to follow = 4
Sensor ID	0x04	Sensor ID
Dynamic Range	0x05-0x06	Dynamic Range
Reserved	0x07	Reserved

The sensor ID must correspond to a specific physical sensor's virtual sensor (either wake-up or non-wake-up). It must also be for a physical sensor supported by the currently loaded firmware image. Setting the dynamic range is not supported for any virtual sensor being derived from multiple physical sensors, like e.g. Rotation Vector.

The Dynamic Range field for the virtual Accelerometer, Gyroscope, and Magnetometer sensors will be able to control the actual dynamic range settings in the corresponding physical sensors. A value of 0 requests the default. The BSX fusion library will be informed of the request to change the dynamic range, and the corresponding scale factor for the sensor data outputs will change accordingly. The host may then read back the Physical Sensor Information to determine the actual current dynamic range, see Section 13.3.2.7.

The host should enable and watch for the Dynamic Range Changed Meta Event so it can apply the correct scale factor before and after the dynamic range change, if the change is made while the sensor is already enabled.

Reading back the parameter is especially important if the host sets a dynamic range for other virtual sensors that share the same underlying physical sensor. BHA260AB will automatically select the largest requested dynamic range of all virtual sensors that share that particular physical sensor. If the host does not specify a dynamic range for a specific virtual sensor (by setting it to 0), then only the virtual sensors with non-zero dynamic range requests from the host will be considered for selecting the actual dynamic range for the physical sensor.

The dynamic range will determine the scale factor for the sensor data, based on the data type (number of bits and signed/unsigned).

The units of measurement for dynamic range are the units commonly used for sensor ranges:

- Accelerometer: Earth g-s
- Gyroscope: degrees/second
- Magnetometer:  $\mu\text{T}$
- Others: Defined by physical sensor driver.

### 13.2.9 Control FIFO Format (0x0015)

This command lets the host change the format and amount of timestamps placed in the wake and non-wake FIFOs.

Table 53: Control FIFO Format - Command

Field Name	Byte Offset	Description
Command	0x00-0x01	Control FIFO Format Command = 0x0015
Length	0x02-0x03	Total number of bytes to follow = 4
FIFO Format Flags	0x04	Format bits: 0 = all bits off (default format) Bit 1 = disable delta timestamps between samples Bit 2 = disable full timestamp in FIFO header Bits 2-7 = reserved
Reserved	0x05-0x07	Reserved

With **FIFO Format Flags = 0x01**, delta timestamps between samples in the same FIFO transfer will be disabled, but the full timestamp in the FIFO header will still be output (see Table 78).

If the report latency of an enabled virtual sensor is set to 0, then usually (if the host is awake and responsive) each sample will be sent to the host in a single FIFO transfer with the full timestamp in the FIFO header, so it might appear to always contain timestamps.

With **FIFO Format Flags = 0x02**, the full timestamp in the FIFO header will be disabled, but there will still be delta timestamps between samples.

If you turn both sources of timestamps off with **FIFO Format Flags = 0x03**, then neither the FIFO header full timestamp nor the between sample delta timestamps will be output.

### 13.3 Parameter Interface

The Parameter Interface is used to allow configuration and query of the state of the system and the sensors. Writing and reading parameters follows the command structure of the Host Interface Commands as described in Section 13. They are distinguished from other kind of Host Interface Commands by the Command ID, which is in the range of 0x0100 to 0x0FFF for writing parameters and 0x1000 to 0x1FFF for reading parameters, see Table 31.

#### 13.3.1 Reading and writing Parameters

The parameters described in the following sections are either of read-only, write-only or read/write access. In a single transaction, a parameter can only be accessed as whole in either read or write mode. The way of specifying the access mode is uniform for all parameters and defined by the upper-most digit of the 4 digit hex Parameter ID. Setting it to '1' initiates read access, setting it to '0' initiates write access.

When initiating a read access, a packet of the following format has to be written to Input Channel 0:

Table 54: Parameter Read Format

Field Name	Byte Offset	Description
Command ID	0x00-0x01	Parameter Read = 0x1XXX, where XXX equals the Parameter ID to be read

Length	0x02-0x03	Total number of bytes to follow = 0
--------	-----------	-------------------------------------

The response data is then placed inside Output Channel 3 corresponding to the format specified for each parameter in the sections below and can be read from there.

To write a parameter the complete data structure, as defined below for each parameter, has to be written into Input Channel 0.

Within the parameter space, various groups of parameters exist:

Table 55: Parameter Groups

Parameter ID	Parameter Group
0x0100 – 0x01FF	System Parameters
0x0200 – 0x02FF	BSX Algorithm Parameters
0x0300 – 0x03FF	Virtual Sensor Information Parameters
0x0500 – 0x05FF	Virtual Sensor Configuration Parameters
0x0800 – 0x0DFF	Customer Defined
0x0E00 – 0x0EFF	Physical Sensor Control Parameters
Others	Reserved

### 13.3.2 System Parameters

These parameters control general system-wide features:

Table 56: System Parameter Overview

Parameter ID	Name	Parameter Read	Parameter Write
0x0101	Meta Event Control	Non-wake-up FIFO Meta Event Enables	Meta Event Enables
0x0102	Meta Event Control	Wake-up FIFO Meta Event Enables	Meta Event Enables
0x0103	FIFO Control	Wake-up and Non-wake-up FIFOs Watermark; FIFOs size	Watermark Configuration
0x0104	Firmware Version	Firmware Version	Not Used
0x0105	Timestamps	Timestamps	Not Used
0x0106	Framework Status	Internal Data; Watchdog	Not Used
0x011F	Virtual Sensors Present	Bitmap of Compiled-in Virtual Sensors	Not Used
0x0120	Physical Sensors Present	Bitmap of Compiled-in Physical Sensors	Not Used
0x0121 - 0x0160	Physical Sensor Information	Orientation Matrix etc.	Orientation Matrix

Others	Reserved	Not Used	Not Used
--------	----------	----------	----------

### 13.3.2.1 Meta Event Control (0x0101, 0x0102)

Meta Event Control Parameter 0x0101 is used for enabling and disabling non-wake-up FIFO Meta Events, while Meta Event Control Parameter 0x0102 is used for enabling and disabling wake-up FIFO Meta Events.

The 8 bytes in this writeable parameter is divided into 32 two-bit sections. Each section controls whether the corresponding Meta Event is enabled (so that it will appear in the output FIFO when it occurs), as well as whether the occurrence of this Meta Event will lead to an immediate host interrupt. See Section 15.3 *Format of Meta Events* for a list of Meta Events. Each specific Meta Event has a default enable state, also specified in that section.

The MSbit in each two-bit section is the event enable, and each LSbit is the event interrupt enable, as shown in the following table:

Table 57: Meta Event Control

Field Name	Byte Offset	Description								Access
Parameter ID	0x00 - 0x01	Meta Event Control: 0x0101 Non-Wake-up FIFO Meta Event Control 0x0102 Wake-up FIFO Meta Event Control								
Length	0x02 - 0x03	Number of bytes to follow = 8								
Bitmap		Bit 7: Event Enable	Bit 6: Int Enable	Bit 5: Event Enable	Bit 4: Int Enable	Bit 3: Event Enable	Bit 2: Int Enable	Bit 1: Event Enable	Bit 0: Int Enable	
	0x04	Meta Event 4		Meta Event 3		Meta Event 2		Meta Event 1		Read/Write
	0x05	Meta Event 8		Meta Event 7		Meta Event 6		Meta Event 5		Read/Write
	0x06	Meta Event 12		Meta Event 11		Meta Event 10		Meta Event 9		Read/Write
	0x07	Meta Event 16		Meta Event 15		Meta Event 14		Meta Event 13		Read/Write
	0x08	Meta Event 20 <sup>1)</sup>		Meta Event 19		Meta Event 18		Meta Event 17		Read/Write
	0x09	Meta Event 24		Meta Event 23		Meta Event 22		Meta Event 21		Read/Write
	0x0A	Meta Event 28		Meta Event 27		Meta Event 26		Meta Event 25		Read/Write
0x0B	Meta Event 32		Meta Event 31		Meta Event 30		Meta Event 29		Read/Write	

#### Notes

1) Although the Meta Event 20 "Spacer" is always enabled, it is reported as "disabled" in the meta Event Control parameter.

### 13.3.2.2 FIFO Control (0x0103)

This Parameter provides a mechanism for the host to set a target watermark level, which is the number of bytes the Wake-up FIFO buffer and/or the Non-Wake-up FIFO buffer may contain before they assert the host interrupt signal.

Set this value to 0 to disable this feature, or a non-zero value to set the watermark. Any value larger than the size of the FIFO will be treated the same as a value exactly equal to the size of the FIFO.



Data loss will likely occur with watermark values that are too high, since additional data may be written into the FIFO while the host responds on the interrupt signal. It is up to the customer to determine, in their application, based on the maximum host rate and host interrupt response time, to determine a safe maximum watermark level.

*Note:*

*A non-zero Watermark has no effect if all enabled virtual sensors are configured with a low maximum report latency, and the AP is active (the Host Interface Control register's AP Suspend bit is 0). In this case, host interrupts are generated based on the report latency requirements of the virtual sensors, and the FIFO level may never reach the watermark level. The Watermark is only useful when all enabled continuous output sensors are configured with non-zero latencies, or the host is suspended, and no wake-up events occur.*

The size of each FIFO can be retrieved by the host by reading this same Parameter it is returned in bytes 4-7 for the Wake-up FIFO and bytes 12-15 for the Non-Wake-up FIFO. This size is determined at compile time of the firmware image and can be configured in the board configuration file.

Table 58: FIFO Control

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	FIFO Control: 0x0103	-
Length	0x02 - 0x03	Number of bytes to follow = 16	-
Wake-up FIFO Watermark	0x04 - 0x07	Number of bytes in FIFO before interrupt is asserted	Read/Write
Wake-up FIFO Size	0x08 - 0x0B	FIFO Size in bytes	Read only
Non-Wake-up FIFO Watermark	0x0C - 0x0F	Number of bytes in FIFO before interrupt is asserted	Read/Write
Non-Wake-up FIFO Size	0x10 - 0x13	FIFO Size in bytes	Read only

### 13.3.2.3 Firmware Version (0x0104)

The firmware version register provides a host-readable version information.

The Custom Version Number can be set by the developer during compilation of the firmware in the board .cfg, see Reference 3 for details.

The Kernel hash 1, Kernel hash 2 and Customer Hash fields provide 6 bytes each for specifying a unique number to identify a build. This could be, for example, the most significant 6 bytes of a Git commit hash. While the Kernel hashes are defined by Bosch Sensortec and are fixed with every SDK release, the User Hash can be defined by the customer.

The Hash values are captured during the build of the firmware. If the SDK is located in a Git tree, the Git hash will be used. If the SDK is not located in git, the values of the file <SDK\_root>/version and <SDK\_root>/hash are used.

Table 59: Firmware Version

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Firmware Version: 0x0104	-

Length	0x02 - 0x03	Number of bytes to follow = 20	-
Custom Version Number	0x04 - 0x05		Read only
Kernel hash 1	0x06 - 0x0B		Read only
Kernel hash 2	0x0C - 0x11		Read only
User hash	0x12 - 0x17		Read only

#### 13.3.2.4 Timestamps (0x0105)

In order to provide a mechanism for the host to translate 40 bit sensor data timestamps to host-relative timestamps, this parameter may be read to determine the time at which the last host-interrupt was asserted, as well as the current system time.

Note:

The Host Interrupt Timestamp can be read more efficiently from the Host Interrupt Timestamp registers.

Table 60: Timestamps

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Timestamps: 0x0105	-
Length	0x02 - 0x03	Number of bytes to follow = 16	-
Host Interrupt Timestamp	0x04 - 0x08	Time (in units of 1/64000 seconds) latched in hardware that the last host interrupt was triggered (5 bytes)	Read only
Current Timestamp	0x09 - 0x0D	Time (in units of 1/64000 seconds) for current system time (5 bytes)	Read only
Timestamp Event	0x0E - 0x12	Time (in units of 1/64000 seconds) immediately upon receipt and processing of the Get Parameter command for this Parameter Number latched in hardware when host wrote to the Timestamp Event Request register (5 bytes)	Read only
Reserved	0x13	Reserved	Read only

#### 13.3.2.5 Virtual Sensors Present (0x011F)

This Parameter contains a read-only 256 bit bitmap, where a set bit indicates the corresponding virtual sensor is present in the system.

For example, if a virtual accelerometer, magnetometer, and humidity sensor were the only virtual sensors present, the bit map would have bits set for sensor ID **4 (accelerometer)**, **22 (magnetometer)** and **130 (humidity)**. In this example, the bit map would be:

Byte 0: 000**1** 0000 (binary; left most bit is bit 7, right most is bit 0)

Byte 1: 0000 0000 (left most is bit 15; right most is bit 8)

Byte 2: 0**1**00 0000

...

Byte 16: 0000 0100

...

Byte 31: 0000 0000

Table 61: Virtual Sensor Present

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Virtual Sensors Present: 0x011F	-
Length	0x02 - 0x03	Number of bytes to follow = 32	-
Virtual Sensors present	0x04 - 0x23	A 256-bit bitmap. In there, each virtual sensor is represented by one bit at the position of its ID, where '1' stands for present, and '0' stands for non-present.	Read only

### 13.3.2.6 Physical Sensors Present (0x0120)

This Parameter contains a read-only 64 bit bitmap, where a set bit indicates the corresponding physical sensor is present in the system. The sensor IDs in this parameter are BSX Input IDs, which are different from the BSX OUTPUT and WAKEUP IDs listed later. The following IDs are currently assigned:

- Accelerometer = 1
- Gyroscope = 3
- Magnetometer = 5
- Temperature Gyroscope = 7
- Anymotion = 9
- Pressure = 11
- Position = 13
- Humidity = 15
- Temperature = 17
- Gas Resistor = 19
- Physical Step Counter (e.g. Hardware Step counter) = 32
- Physical Step Detector = 33
- Physical Significant Motion = 34
- Physical Any Motion = 35
- External Camera Input = 36
- GPS = 48
- Light = 49
- Proximity = 50

For example, if a physical accelerometer, magnetometer, and humidity sensor were the only physical sensors present, the bit map would have bits set for sensor ID **1 (accelerometer)**, **5 (magnetometer)** and **15 (humidity)**. In this example, the bit map would be:

Byte 0: 0010 0010 (binary; left most bit is bit 7, right most is bit 0)

Byte 1: 1000 0000 (left most is bit 15; right most is bit 8)

Byte 2: 0000 0000

Byte 3: 0000 0000

Byte 4: 0000 0000

Byte 5: 0000 0000  
 Byte 6: 0000 0000  
 Byte 7: 0000 0000

Table 62: Physical Sensors Present

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Physical Sensors Present: 0x0120	-
Length	0x02 - 0x03	Number of bytes to follow = 8	-
Physical Sensors present	0x04 - 0x0B	A 64-bit bitmap. In there, each physical sensor is represented by one bit at the position of its ID, where '1' stands for present, and '0' stands for non-present.	Read only

### 13.3.2.7 Physical Sensor Information (0x0121 – 0x0160)

Each parameter in the range of 0x0121 to 0x0160 refers to a specific physical sensor ID. The parameter 0x0121 refers to Physical Sensor ID 0, while 0x0160 refers to Physical Sensor ID 63.

This structure is returned for any Parameter Numbers read when the corresponding physical sensor is present. If not present, this structure returns all 0s.

Table 63: Physical Sensor Information

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Physical Sensor Information: 0x0121 - 0x0160	-
Length	0x02 - 0x03	Number of bytes to follow = 20	-
Physical Sensor ID	0x04	Same as (Parameter Number - 0x0121)	Read only
Driver ID	0x05	Unique per driver / vendor / part number	Read only
Driver Version	0x06	Denotes notable change in behavior	Read only
Current Consumption	0x07	Estimated current consumption of the physical sensor. Scale factor is 0.1 mA	Read only
Current Dynamic Range	0x08 - 0x09	Current dynamic range of sensor in SI units	Read only
Flags	0x0A	Bit 0: IRQ enabled 0: disabled 1: enabled Bits 1-4: master interface used: 0: none 1: SPI0 2: I2C0 3: SPI1 4: I2C1 Bits 5-7: power mode 0: Sensor Not Present 1: Power Down	Read only

		2: Suspend 3: Self-Test 4: Interrupt Motion 5: One Shot 6: Low Power Active 7: Active	
Slave Address	0x0B	If I2C: 7 bit device address (MSbit = 0) If SPI: the GPIO pin used as chip select	Read only
GPIO Assignment	0x0C	GPIO pin used as data ready interrupt input	Read only
Current Rate	0x0D - 0x10	Current sample output rate of sensor in Hz, as 32-bit float value	Read only
Number of Axes	0x11	Number of Axes of the sensor (e.g., X/Y/Z = 3)	Read only
Orientation matrix	0x12 - 0x16	Orientation matrix of the sensor, 5 bytes, see description below	Read only
Reserved	0x17	-	-

The orientation matrix is provided for sensors that support this, such as 3 axis accelerometers, magnetometers, and gyroscopes. It is used to align the orientation of physical sensor axes to match the required ENU (east north up) orientation required by Android. The calculation is performed as:

$$|X\ Y\ Z| = |X_s\ Y_s\ Z_s| \cdot \begin{vmatrix} C_0 & C_1 & C_2 \\ C_3 & C_4 & C_5 \\ C_6 & C_7 & C_8 \end{vmatrix}$$

The orientation matrix output from this parameter is in the same order as the elements listed in the board configuration file used to generate a firmware image, described in Reference 3, where each matrix element is stored in successive nibbles.

For example, if the board configuration file contains:

```
#DriverID,Bus,Addr,GPIO,C0,C1,C2,C3,C4,C5,C6,C7,C8,Off0,Off1,Off2,MaxRate
a48,spi0,25,2, 1, 0, 0, 0, -1, 0, 0, 0, -1, 0, 0, 0, -1.00000
```

Then bytes 14-18 of the Physical Sensor Information structure would be:

Byte 0x12: 01 (hexadecimal)

Byte 0x13: 00

Byte 0x14: 0F

Byte 0x15: 00

Byte 0x16: 0F

This Parameter can also be used to update the orientation matrix in use at runtime. When written, the following structure is used:

Table 64: Orientation Matrix Write Format

Field Name	Byte Offset	Description
Parameter ID	0x00 - 0x01	Physical Sensor Information: 0x0121 - 0x0160
Length	0x02 - 0x03	Number of bytes to follow = 8

Orientation Matrix	0x04 – 0x08	See bytes 0x12 to 0x16 in Table 69
Reserved	0x09 – 0x0B	-

For more details regarding axis remapping, please refer to Section 20.3 *Sensing axes and axes remapping*.

### 13.3.3 BSX Algorithm Parameters

The BSX Algorithm parameters are used to read and control various aspects of the Sensor Fusion algorithms in the BHA260AB. The interface directly with the BSX library in the BHA260AB firmware, which implements the fusion algorithms.

Parameters 0x0201 – 0x0240 contain calibration states for each physical sensor, matching the BSX input IDs in the range of 0x01 to 0x40. The BSX library currently supports accelerometer, gyroscope, and magnetometer as physical sensors, so the available parameters in this range are 0x0201, 0x0203, and 0x0205. Calibration state contains intermediate calibration state for each sensor data. It should be saved when the accuracy of the corresponding sensor reaches 3 and loaded on system boot to achieve the effect of warm start.

Besides the physical sensor calibration states it is also possible to write a soft iron calibration (SIC) matrix and read BSX version information.

Table 65: Algorithm Parameters

Parameter ID	Usage	Content Format	Access
0x0201	Calibration state for accelerometer	BSX State Exchange Structure	Read/Write
0x0203	Calibration state for gyroscope		Read/Write
0x0205	Calibration state for magnetometer		Read/Write
0x027D	SIC Matrix		Write only
0x027E	BSX Version	4 byte BSX Version number	Read only

#### 13.3.3.1 Reading and writing the BSX State Exchange Structure

The calibration state and SIC matrix are read / written through multiple operations due to the large size of the data. Each operation transfers one data block with the format of BSX state exchange structure.

During the read operation, the host is responsible for assembling the data together according to block number and block data length in BSX state exchange structure. Each block contains at most 64 bytes of data. The completion flag is set in the last block. When this bit is set, it notifies the host that the last block has been read, and the BSX state exchange structure is complete.

During the write operation, the host is responsible for breaking down the previously saved data to blocks in the BSX state exchange structure format with each block containing at most 64 bytes of data. All blocks but the last block should contain exactly 64 bytes of data. The last block contains the remaining data and the transfer length should still be an entire BSX state exchange structure. The completion flag must be set in the last block.

Table 66: BSX State Exchange Structure

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Calibration state: 0x0201 - 0x0240 SIC matrix: 0x027D	-
Length	0x02 - 0x03	Number of bytes to follow <= 68	-
Block information	0x04	Bit 0-6: Block number starting at 0 Bit 7: Completion flag	Read/Write
Block length	0x05	Valid data length in the current block	Read/Write
Structure length	0x06 - 0x07	Total data length for the entire data	Read/Write
Block data	0x08 - 0x47 max	Block data	Read/Write

**Note:**

The calibration state and SIC matrix use a BSX internal binary data format and are not intended to be interpreted by the host.

**13.3.3.2 BSX Version (0x027E)**

The 4-byte BSX Version information can be read with the following parameter:

Table 67: BSX Version

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	BSX Version: 0x027E	-
Length	0x02 - 0x03	Number of bytes to follow = 4	-
Major Version	0x04	BSX Major Version	Read only
Minor version	0x05	BSX Minor version	Read only
Major bug fix version	0x06	BSX Major bug fix version	Read only
Minor bug fix version	0x07	BSX Minor bug fix version	Read only

**13.3.4 Virtual Sensor Information Parameters (0x0301 – 0x0395)**

Each parameter in the range of 0x0301 to 0x0395 is read-only and refers to a specific Virtual Sensor ID as specified in Section 15. E.g., the parameter 0x0301 refers to virtual sensor ID 1, while 0x0395 refers to virtual sensor ID 149 (0x95).

The structure outlined in Table 74 is returned for every parameter and reports essential information about a virtual sensor. If the requested sensor ID is not supported by the current firmware image, then all fields of this structure are reported as zero. All values are of type unsigned integer, LSB first, with their respective length, unless specified differently.

Table 68: Virtual Sensor Information Structure

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Virtual Sensor Information: 0x0301 - 0x0395	-
Length	0x02 - 0x03	Number of bytes to follow = 28	-
Sensor ID	0x04	Same as (Parameter Number - 0x0300)	Read only

Driver ID	0x05	Unique per driver / vendor / part number	Read only
Driver Version	0x06	Denotes notable change in behavior	Read only
Power	0x07	Estimated current consumption of the virtual sensor. Scale factor is 0.1 mA	Read only
Max Range	0x08 - 0x09	Maximum range of sensor data in SI units	Read only
Resolution	0x0A - 0x0B	Number of bits of resolution of the underlying sensor	Read only
Max Rate	0x0C - 0x0F	Maximum supported output data rate in Hz, as 32-bit float value	Read only
FIFO Reserved	0x10 - 0x13	Number of samples of this virtual sensor for which FIFO space is reserved. This can be 0 in case of a single shared FIFO	Read only
FIFO Max	0x14 - 0x17	Maximum number of samples of this virtual sensor that can be stored when using the entire FIFO	Read only
Event Size	0x18	Number of bytes in FIFO for this virtual sensor's data packet (including Sensor Type)	Read only
Min Rate	0x19 - 0x1C	Minimum supported output data rate in Hz, as 32-bit float value	Read only
Reserved	0x1D - 0x1F	-	-

The Max Range field will be set to the maximum possible range, that the underlying physical sensor can attain if set to its highest range setting. This is a constant value that does not change based on the current Dynamic Range setting. It is stored in units commonly used for sensor ranges (Earth Gs, degrees/second,  $\mu$ T). The Resolution field is provided so that the host can determine the “smallest difference between two values reported by this sensor.” It contains the number of bits of resolution. With that, the host can determine the floating point resolution value by dividing the Max Range or current Dynamic Range by  $2^{\text{Resolution}}$  for unsigned values, or by  $2^{\text{Resolution}-1}$  for signed values.

### 13.3.5 Virtual Sensor Configuration Parameters (0x0501 – 0x0595)

Each parameter in the range of 0x0501 to 0x0595 is read-only and refers to a specific Virtual Sensor ID as specified in Section 15. E.g., the parameter 0x0501 refers to virtual sensor ID 1, while 0x0595 refers to virtual sensor ID 149.

The structure outlined in Table 75 is returned for every parameter and reports the current configuration of a virtual sensor. If the requested sensor ID is not supported by the current firmware image, then all fields of this structure are reported as zero. All values are of type unsigned integer, LSB first, with their respective length, unless specified differently

Table 69: Virtual Sensor Configuration Structure

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Virtual Sensor Configuration: 0x0501 - 0x0595	-
Length	0x02 - 0x03	Number of bytes to follow = 12	-



Sample Rate	0x04 - 0x07	Rate in Hz, $\frac{1}{\text{AndroidsampleRate}}$ , reads back the actual current sensor rate, as 32-bit float value	Read only
Max Report Latency	0x08 - 0x0B	Is 0 for non-batch mode (no latency), reads back actual current latency in ms	Read only
Reserved	0x0C - 0x0D	-	-
Dynamic Range	0x0E - 0x0F	Range setting for underlying physical sensor in their respective units. See Section 13.2.8	Read only

### 13.3.6 Physical Sensor Control Parameters (0x0E00 – 0x0EFF)

These parameters allow the host to set or get physical sensor control information. The Sensor Control Parameters are in the range of 0x0E00 to 0x0EFF, for which the lower byte is the physical sensor ID.

The meaning and nature of this information is defined by the implementation of the specific physical sensor driver. It can have registered callback functions, which are called when this parameter is written or read. Please refer to Reference 3 for details.

While the format of the data is defined by the design of the physical sensor driver, the only constraint placed on this format by the BHA260AB host interface and Event-driven Software Framework is that the first byte must contain a 8 bit code indicating the type of data to follow, and can be followed by 0 or more bytes of additional code-specific information.

The parameter is formatted as follows:

Table 70: Physical Sensor Control Parameters

Field Name	Byte Offset	Description	Access
Parameter ID	0x00 - 0x01	Calibration state: 0x0E00 - 0x0EFF	-
Length	0x02 - 0x03	Number of bytes to follow	-
Code	0x04	Bit 0-6: Sensor control code Bit 7: Direction: 0 = write (set_sensor_ctl callback function) 1 = read (get_sensor_ctl callback function)	Read/Write
Payload (optional)	0x05 - (0x05 + Length -1)	Data to be sent to the set_sensor_ctl function or to be retrieved from the get_sensor_ctl function	Read/Write

When setting the parameter, the Set Parameter command length must be a multiple of 4 bytes and must be large enough to include the code byte and any additional data bytes.

If the direction bit of the code, bit 7, is 0, then the host is requesting that the Fuser2 pass the code and data to the set\_sensor\_ctl() function for the physical driver whose ID is indicated as the LSB of the parameter number.

If the direction bit is 1 in the code byte of the Set Parameter command, then the host is requesting that the next Get Parameter from the host for the same sensor ID (as indicated in

the LSB of the parameter number) return the data obtained by calling the `get_sensor_ctl()` function after passing it the code set by the Set Parameter operation.

When getting the parameter, the Get Parameter command length should be 0 as usual. The Get Parameter Output Response's length field will indicate the number of sensor control parameter bytes that follow, including the original code requested earlier by the host in the Set Parameter command.

### 13.4 Command Error Response

This command response will be returned by the bootloader and main firmware when there is a problem with the received command.

If Error Value register (see Section 12.1.25) is 0xC0 (Command Error) or 0xC1 (Command Too Long), the following registers are also updated:

- Error Aux Register (see Section 12.1.26) = Command error value (see below)
- Debug Value Register (see Section 12.1.26) = Command ID in error (least significant byte)

Table 71: Command Error Response

Field Name	Byte Offset	Description	
Status Code	0x00-0x01	Command Error = 0x000F	
Length	0x02-0x03	4	
Command	0x04-0x05	The command ID with the error	
Error	0x06	Value	Command Error
		0x01	Incorrect Length
		0x02	Too Long (length specified is longer than input buffer <sup>1)</sup> ; recover by issuing an Abort Transfer on Channel 0)
		0x03	Parameter Write Error (incorrect page or unhandled parameter number or length provided is too short)
		0x04	Parameter Read Error (parameter size too big for output buffer, or invalid page or parameter specified)
		0x05	Invalid Command
		0x06	Invalid Parameter
		0xFF	Command Failed (did not complete successfully)
Reserved	0x07	Unused	

#### Note

1) The input buffer size is 128 bytes in bootloader and 1024 bytes when the Event-driven Software Framework is used.

## 14 FIFO Data Formats

When the host retrieves data from a FIFO by reading that FIFO's output stream, it will receive blocks of sensor data encapsulated by a FIFO Descriptor. This descriptor consists of the number of bytes to follow in this transfer and a Small Delta Timestamp with the delta set to 0, which is required to make the FIFO Descriptor length compatible with BHA260AB's DMA system.

Following the FIFO Descriptor, there will be one or more FIFO Blocks; the total size of data sent (including the initial small delta timestamp) will equal the FIFO Transfer Length field in the FIFO Descriptor. The maximum size of a single transfer is limited to the 16 bits of Transfer Length.

Each FIFO Block begins with a FIFO Block Header. This contains a Meta Event and a full 40 bit timestamp. The Meta Event will be a "spacer" Meta Event, with the two payload bytes set to a running 16 bit block count (the host can ignore this), unless one or more previous FIFO blocks were discarded due to a FIFO overflow, in which case the Meta Event will be a FIFO Overflow Meta Event. The full timestamp is provided in order to guarantee the host can always know the proper time for a block's sensor data regardless of any preceding FIFO overflows.

When more than one FIFO Blocks are sent in a transfer, all but the last one will be filled out to the full block size, which is 512 bytes including the Block Header, with one or more single byte Filler sensor IDs (0xFF). The host should ignore these filler bytes and continue parsing. The last (or only) FIFO block will often not be completely full of packed sensor data. In this case the packed sensor data will be padded out to the next 32 bit boundary with single byte Padding sensor IDs (0x00).

Table 72: FIFO Data Format

FIFO Descriptor		Header at Start of each FIFO Block		Contents of each FIFO block		0 or more additional FIFO Block Headers and FIFO Block Contents	
FIFO Transfer Length (16 bits)	Small Delta Timestamp (16 bits) – always 0	Meta Event (Spacer or Overflow)	Full Timestamp (40 bits)	Packed Sensor Data	Filler Bytes (0xFF)	...	0 – 3 Pad Bytes (0x00)

The format of the packed sensor data is described in Sections 15.1 to 15.3. All multi-byte fields are little-endian.

*Note:*

*If the host uses the Control FIFO Format Command (described in Section 13.2.9) to suppress the full timestamp in the header, then only a 4 byte Meta Event will be present at the start of each FIFO block. In this case, the proper full timestamp at which a FIFO overflow condition occurs will not be available. It is recommended that this feature is only be used when FIFO overflows cannot occur.*

### 14.1 Wake-up and Non-Wake-up FIFO

These FIFOs always comply with the format described in Section 14 above.

## 14.2 Status and Debug FIFO

The Status and Debug FIFO (Output Channel 3) has two operating modes: the synchronous mode and the asynchronous mode. The operating mode is controlled by the *Async Status Channel bit* of the *Host Interface Control (0x06)* register.

Each mode has a corresponding bit in the *Interrupt Status (0x2D)* register:

- Bit 5 indicates a synchronous status packet is available (as result of a command)
- Bit 6 indicates asynchronous data is available (as result of an error, debug output, or command)

Further, each bit can be masked using the *Host Interrupt Control (0x07)* register:

- Bit 2, if set, masks off (prevents) the synchronous status packet interrupt
- Bit 3, if set, masks off (prevents) the asynchronous debug FIFO data interrupt

When the host is about to transmit a command, it is recommended to clear the *Async Status Channel bit* of the *Host Interface Control (0x06)* register, in order to place the channel in synchronous mode. While in this mode, any asynchronous data will be stored in a memory buffer. Once the command is completed and any associated command response has been received, the host may switch back to asynchronous mode. The host may mask off the 'Status Available' Interrupt to not be alerted once the command response is available in synchronous mode by setting the bit 2 in the *Host Interrupt Control (0x07)* register.

### 14.2.1 Synchronous mode

In synchronous mode, the FIFO format described in *Table 78: FIFO Data Format* will NOT apply to data read from the Status and Debug FIFO. The only data that will appear in the Status and Debug FIFO (Output Channel 3) will be command responses as defined throughout *Section 13 Host Interface Commands*.

While in this mode, asynchronous status and debug messages will be queued in the FIFO in the background. These data will be transferred to the host once it switches the Status and Debug FIFO to Asynchronous mode. Presence of data will then be signaled via the Host Interrupt pin and Interrupt Status register.

### 14.2.2 Asynchronous mode

In Asynchronous mode, the FIFO format described in *Table 78: FIFO Data Format* will apply to data read from the Status and Debug FIFO. The content of the FIFO is the following:

- Sensor Error and System Error Meta events (see *Table 29: Error Value Register (0x2E)*)
- Debug events (Post Mortem Data)
- Command responses

The host has to parse the FIFO content to find the response corresponding to a recent command.

## 15 FIFO Data Types and Format

Table 73: Overview of FIFO Event IDs

FIFO Event Type	FIFO Event	ID (Non Wake-up)	ID (Wake-up)	Sensor Payload <sup>1)</sup> Format	Scale Factor	Bytes in FIFO	Reporting mode	Requires external sensor <sup>2)</sup>
Virtual Sensor Event	Rotation Vector	34	35	Quaternion+	Defined by format "Quaternion+"	11	Continuous	BMG250; BMM150 or AK09915
	Game Rotation Vector	37	38	Quaternion+	Defined by format "Quaternion+"	11	Continuous	BMG250
	Geomagnetic Rotation Vector	40	41	Quaternion+	Defined by format "Quaternion+"	11	Continuous	BMM150 or AK09915
	Orientation	43	44	Euler	Defined by format "Euler"	7	Continuous	BMG250; BMM150 or AK09915
	Accelerometer Passthrough	1	-	3D Vector	Unmodified raw data of sensor	7	Continuous	-
	Gyroscope Passthrough	10	-	3D Vector	Unmodified raw data of sensor	7	Continuous	BMG250
	Magnetometer Passthrough	19	-	3D Vector	Unmodified raw data of sensor	7	Continuous	BMM150 or AK09915
	Accelerometer Corrected	4	6	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	-
	Magnetometer Corrected	22	24	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMM150 or AK09915
	Gyroscope Corrected	13	15	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMG250
	Gravity	28	29	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMG250
	Linear Acceleration	31	32	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMG250
	Raw Accelerometer (AKA Uncalibrated)	3	7	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	-
	Raw Magnetometer	21	25	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMM150 or AK09915
	Raw Gyroscope	12	16	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMG250
	Accelerometer Offset	5	91	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	-
	Magnetometer Offset	23	93	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMM150 or AK09915
	Gyroscope Offset	14	92	3D Vector	Dynamic <sup>3)</sup>	7	Continuous	BMG250
	Light	146	148	16 bit unsigned integer	10000 Lux / 216	3	Continuous	TMG4903
	Proximity	147	149	8 bit	0: far, 1: near	2	Continuous	TMG4903
	Humidity	130	134	8 bit unsigned integer	1%RH	2	On-change	BME280 or BME680
	Step Counter	52	53	32 bit unsigned integer	1 step	5	On-change	-
Aux Step Counter	136	139	32 bit unsigned integer	1 step	5	On-change	-	
Temperature	128	132	16 bit signed integer	°C / 100 (range: -4000 to 8500)	3	On-change	BME280 or BME680	
Barometer	129	133	24 bit unsigned integer	1/128 Pa	4	Continuous	BME280 or BME680 or BMP280	

	Gas	131	135	32 bit unsigned integer	Ohms	5	Continuous	BME680
	Significant Motion	-	55	Event (none)	n.a.	1	One-shot	-
	Step Detector	50	94	Event (none)	n.a.	1	Special	-
	Tilt Detector	-	48	Event (none)	n.a.	1	Special	-
	Wake Gesture	-	57	Event (none)	n.a.	1	One-shot	-
	Glance Gesture	-	59	Event (none)	n.a.	1	One-shot	-
	Pick Up Gesture	-	61	Event (none)	n.a.	1	One-shot	-
	Aux Significant Motion	138	141	Event (none)	n.a.	1	One-shot	-
	Aux Step Detector	137	140	Event (none)	n.a.	1	Special	-
	Aux Any Motion	142	143	Event (none)	n.a.	1	One-shot	-
	Activity	-	63	Activity	n.a.	3	On-change	-
	Camera Shutter	144	-	8 bit count of interrupts	n.a.	2	On-change	shutter pulse
	GPS	145	-	Structure containing NMEA strings	n.a.	27	Continuous	SiRFStar V
	Wrist Tilt Gesture	-	67	Event (none)	n.a.	1	On-change	-
	Device Orientation	69	70	8 bit unsigned integer	0: Portrait upright 1: Landscape left 2: Portrait upside down 3: Landscape right	2	On-change	-
	Stationary Detect	-	75	Event (none)	n.a.	1	On-change	-
	Motion Detect	-	77	Event (none)	n.a.	1	On-change	-
Debug Data Event	Debug Data	250	-	Binary or string data (fwrite or printf)	n.a.	18	n.a.	-
Timestamp Event	Timestamp Small Delta	251	245	8 bit integer; incremental change from previous	1/64000 seconds	2	n.a.	-
	Timestamp Large Delta	252	246	16 bit integer; incremental change from previous	1/64000 seconds	3	n.a.	-
	Full Timestamp	253	247	40 bit unsigned integer; wraps every 198 days	1/64000 seconds	6	n.a.	-
Meta Event	Meta Events	254	248	Meta Event	n.a.	4	n.a.	-
Filler	Filler <sup>4)</sup>	255	255	n.a.	n.a.	1	n.a.	-
Padding	Padding <sup>5)</sup>	0	0	n.a.	n.a.	1	n.a.	-

**Notes:**

- 1) See Section 15.1 for definition of sensor value formats
- 2) Other physical sensor can be supported by creating a dedicated physical driver for the sensor using the SDK (Software Development Kit). See Reference 3 for more information
- 3) Dynamic: scaled to current dynamic range of sensor

- 4) Used to space FIFO blocks to even boundary; host should ignore but continue parsing.  
 5) Optionally used to mark end of a FIFO read; host should stop parsing here

## 15.1 Format of virtual sensor events

In general, every virtual sensor event in the FIFO data consists of a 1-Byte Sensor ID and a multi-byte payload. The size of the virtual sensor event is fixed per Sensor ID as described by the column “Bytes in FIFO” in the *Table 79: Overview of FIFO Event IDs* shown above. This value includes the Sensor ID byte.

For some of the Sensor IDs, the format and scaling of the payload (e.g. “1 bit unsigned integer” and “1%RW” for Humidity”) is described in the same *Table 79: Overview of FIFO Event IDs* shown above.

For Sensor IDs with complex payload the format is described in the following sections.

### 15.1.1 Format “Quaternion+”

For the three rotation vectors (Rotation Vector, Game Rotation Vector, Geomagnetic Rotation Vector), the “Quaternion+” format is used:

Table 74: Virtual Sensor Event Format “Quaternion+”

Byte Number	Contents	Format
0	Sensor ID (Rotation Vector, etc.)	8 bit unsigned, see Table 79: Overview of FIFO Event IDs.
1 .. 2	X component of quaternion	Signed 16 bit fixed point integer, least significant byte first, scaled by $2^{-14}$
3 .. 4	Y component of quaternion	Signed 16 bit fixed point integer, least significant byte first, scaled by $2^{-14}$
5 .. 6	Z component of quaternion	Signed 16 bit fixed point integer, least significant byte first, scaled by $2^{-14}$
7 .. 8	W component of quaternion	Signed 16 bit fixed point integer, least significant byte first, scaled by $2^{-14}$
9 .. 10	Estimated accuracy in radians	Unsigned 16 bit fixed point integer, least significant byte first, scaled by $2^{-14}$

For Game Rotation Vector the Estimated Accuracy in Radians is reported as 0.

### 15.1.2 Format “Euler”

The Orientation Sensor outputs the device orientation as Euler angles: heading, pitch, and roll.

Table 75: Virtual Sensor Event Format “Euler”

Byte Number	Contents	Format
0	Sensor ID (Orientation)	8 bit unsigned, see Table 79: Overview of FIFO Event IDs.
1 .. 2	Heading	Signed 16 bit fixed point integer, least significant byte first, scaled by $(360^\circ / 2^{-15})$

3 .. 4	Pitch	Signed 16 bit fixed point integer, least significant byte first, scaled by $(360^\circ / 2^{-15})$
5 .. 6	Roll	Signed 16 bit fixed point integer, least significant byte first, scaled by $(360^\circ / 2^{-15})$

### 15.1.3 Format “3D Vector”

For the many 3 axis sensors (e.g. Accelerometer, Magnetometer, Gyroscope, Gravity, the following layout is used. The scale factor of the values depends on the type of the sensor and optionally on the range setting. Please check Table 79 for details.

Table 76: Virtual Sensor Event Format “3D Vector”

Byte Number	Contents	Format
0	Sensor ID	8 bit unsigned, see Table 79: Overview of FIFO Event IDs.
1 .. 2	X	Signed 16 bit fixed point integer, least significant byte first
3 .. 4	Y	Signed 16 bit fixed point integer, least significant byte first
5 .. 6	Z	Signed 16 bit fixed point integer, least significant byte first

### 15.1.4 Format “Activity”

The activity sensor outputs a sensor whenever there is a change detected in activity. In the payload of the virtual sensor event bits are provided to indicate start the onset of an activity and the end of it. A bit value of 1 indicates the change of the activity (start or end), while a value of 0 indicates no change.

Table 77: Virtual Sensor Event Format “Activity”

Byte Number	Contents	Format
0	Sensor ID	8 bit unsigned, see Table 79: Overview of FIFO Event IDs.
1 .. 2	Activity change bitmap	16 bit field, least significant byte first

Table 78: Bitmap of activities

Bit Number	Contents
0	Still activity ended
1	Walking activity ended
2	Running activity ended
3	On Bicycle activity ended
4	In Vehicle activity ended



5	Tilting activity ended
6	In Vehicle still ended
7	Reserved
8	Still activity started
9	Walking activity started
10	Running activity started
11	On Bicycle activity started
12	In Vehicle activity started
13	Tilting activity started
14	In Vehicle still started
15	Reserved

### 15.1.5 Format of Scalar data

Many virtual sensor report a single signed or unsigned value. Currently, sensors with a payload of 1 to 5 bytes exist, corresponding to 8 to 40 bit signed or unsigned integer values.

The format of these events is:

Table 79: Virtual Sensor Event Format for scalar data

Byte Number	Contents	Format
0	Sensor ID	8 bit unsigned, see Table 79: Overview of FIFO Event IDs.
1 .. N	Scalar data	8 to 40 bits of signed or unsigned integer data, least significant byte first.

### 15.1.6 Format of sensors without payload

Some virtual sensors generate no payload data, i.e. the virtual sensor event notifies the occurrence of the event. The format of these events consists of the Sensor ID only:

Table 80: Virtual Sensor Event Format with no payload

Byte Number	Contents	Format
0	Sensor ID	8 bit unsigned, see Table 79: Overview of FIFO Event IDs.

## 15.2 Retrieving timestamps of virtual sensor events

Every virtual sensor event has a timestamp associated to it, indicating at which time the event has occurred. The timestamps are not part of the event payload, but transferred as separate events. This allows for multiple virtual sensor events having the same timestamp, transferring the timestamp only once in order to save FIFO space.

As a general rule, if a timestamp is transferred, it is valid for all following virtual sensor events, until the next timestamp is transferred.

The format of a timestamp is a 40 bit unsigned value with a resolution of typical 1/64000 second, starting from 0 when the system has booted. The timestamp wraps around after approximately 198 days of continuous operation.

In order to save further FIFO space, 3 different types of timestamp events have been defined, one (“Full Timestamp”) providing the full 40 bit absolute timestamp, while the two other (“Timestamp Small delta”, “Timestamp Large Delta”) provide an 8 bit or 16 bit increment to the previous timestamp.

See *Table 79: Overview of FIFO Event IDs* for the definition of the timestamps and their format.

### 15.3 Format of Meta Events

Meta Events indicate asynchronous, low periodicity events. They can be individually enabled or disabled with the *13.3.2.1 Meta Event Control (0x0101, 0x0102)* Parameter.

Meta Events can also be configured to trigger or cancel a host interrupt whenever the Meta Event occur. In this case the host interrupt would be issued even if there were no pending virtual sensor events in the FIFOs.

The “Initialized” Meta Event will be the first event in the FIFO (following the current timestamp) after initialization. After the host receives this, it is safe to send Host Commands, e.g. to configure the device or to enable virtual sensors.

There are two types of events: System Meta Events, and Meta Events related to a specific physical sensor.

System Meta Events will be placed in the output channel 3 (Status and Debug FIFO). These Meta Events are Error and Sensor Error. These Meta Events are enabled by default, and wake up the AP by default.

Meta events related to a specific physical sensor (generated as a result of a sensor configuration request from the host) such as the Sample Rate Changed, Power Mode Changed, and Dynamic Range Changed, are always sent to the appropriate FIFO. For example, if the host requests to turn on the wake-up accelerometer, and it was not enabled before this, all three events (if enabled and the state changes) may be generated in the wake-up FIFO.

Table 81: Overview of Meta Events

Meta Event Type		Event-Specific Values		Wake-up FIFO		Non-Wake-up FIFO		Status FIFO	
Name	Byte 1	Byte 2	Byte 3	Default Enable State	Default Int Enable State	Default Enable State	Default Int Enable State	Default Enable State	Default Int Enable State
Not used	0								
Flush Complete	1	Sensor Type from FIFO_FLUSH register	Not used	Enabled		Enabled			
Sample Rate Changed	2	Sensor Type	Sample Rate	Enabled		Enabled			
Power Mode Changed	3	Sensor Type	Power Mode	Enabled		Enabled			

System Error	4	Error Register	Interrupt State Register					Enabled	Enabled
Algorithm Events	5	Sub Event	Event Payload	Enabled		Enabled			
Sensor Status	6	Sensor Type	Status	Enabled		Enabled			
Reserved	7								
Reserved	8								
Reserved	9								
Reserved	10								
Sensor Error	11	Sensor Type	Error Register					Enabled	Enabled
FIFO Overflow	12	Loss Count LSB	Loss Count MSB	Enabled <sup>1)</sup>		Enabled <sup>1)</sup>			
Dynamic Range Changed	13	Sensor Type	Not used	Enabled		Enabled			
FIFO Watermark	14	Bytes Remaining LSB	Bytes Remaining MSB	Enabled		Enabled			
Reserved	15								
Initialized	16	RAM Ver LSB	RAM Ver MSB	Enabled	Enabled	Enabled	Enabled		
Transfer Cause	17	Sensor Type	Not used						
Event-driven Software Framework	18	Sensor Type	Condition			Enabled			
Reset	19	0	Reset Cause	Enabled	Enabled	Enabled	Enabled		
Spacer	20	Block Count LSB	Block Count MSB	Enabled <sup>1)</sup>		Enabled <sup>1)</sup>			

*Notes:*

1) These Meta Events cannot be disabled

### 15.3.1 Meta Event: Flush Complete

This Meta Event will be inserted in the appropriate FIFO(s) after a Flush FIFO request, whether there is any data in the FIFO or not. In line with the Android terminology, flushing in this context means “transfer to host”, and not “discard.”

### 15.3.2 Meta Event: Sample Rate Changed

This Meta Event occurs when a given virtual sensor’s sample rate has been set for the first time, and/or when a requested change to the rate actually occurs.

Byte 1 indicates the sensor type whose rate changed.

Byte 2 indicates the new sample rate (rounded down) for the sensor, saturated at 255. To determine the exact sample rate, the virtual sensor information should be read if 255 is reported or if a fractional value is needed.

This Meta Event will be placed in the same FIFO as the related virtual sensor reports to, e.g. if a virtual sensor reports its events into the wake-up FIFO, this event will also be placed in the wake-up FIFO.

### 15.3.3 Meta Event: Power Mode Changed

This Meta Event indicates when a given sensor powers up or down; the Sensor ID of the related virtual sensor is passed in byte 2.

This Meta Event will be placed in the same FIFO as the related virtual sensor reports to, e.g. if a virtual sensor reports its events into the wake-up FIFO, this event will also be placed in the wake-up FIFO.

### 15.3.4 Meta Event: System Error

This Meta Event reports when a system error has occurred. It is reported in the status FIFO.

See Table 29 for error codes (Byte 2) and Table 28 for the description of the interrupt status (Byte 3)

### 15.3.5 Meta Event: Algorithm Events

This Meta Event is reserved for algorithm specific reports. The current BSX Fusion Library does not use this feature. It is reserved for future extensions.

### 15.3.6 Meta Event: Sensor Status

This Meta Event indicates the data quality of the specified virtual sensor. It is sent whenever the status changes. It will be reported to the same FIFO as the virtual sensor reports to.

The Sensor ID of the related virtual sensor is in Byte 2, and the sensor status is in Byte 3. The following sensor status values are used:

Table 82: Sensor Status Values

Status Value	Meaning
0	Unreliable
1	Accuracy Low
2	Accuracy Medium
3	Accuracy High

### 15.3.7 Meta Event: Sensor Error

This Meta Event reports when a sensor error has occurred. It is reported in the status FIFO.

Byte 2 is the Physical Sensor ID, as specified in Section 13.3.2.6. Byte 3 is the error code according to Table 29.

### 15.3.8 Meta Event: FIFO Overflow

This Meta Event indicates when data loss has occurred due to the host being unable to read out FIFO data quickly enough. This may be intentional, for example when the host is suspended, or it may be due to having too many sensors on at high sample rates with a slow host I2C rate or slow driver implementation.

Bytes 2 and 3 report a saturating count of lost bytes. Following this Meta Event will be a full 40 bit timestamp, to ensure the host will be able to report accurate timestamps after the area of data loss.

The BHA260AB, when it detects a FIFO overflow, automatically discards a block of FIFO data (maximum size 512 bytes) in order to make room for more data, make room for the FIFO Overflow and Timestamp events, and ensure that at least some new data will appear in the FIFO between FIFO Overflow events, rather than become saturated with such events in worst case conditions.

The Meta Event will be placed in the appropriate wake vs. non-wake FIFO.

### 15.3.9 Meta Event: Dynamic Range Changed

This Meta Event will be placed in the FIFO as soon as a requested change in dynamic range has occurred. The host may wish to wait for this event before changing the scale factor, in the event that a sensor whose dynamic range was changed was already on. Otherwise, the host could apply the wrong scale factor on some samples, and report invalid data as a result.

This event is inserted in the FIFOs for all enabled virtual sensors whose physical source is the sensor whose dynamic range was changed. For example, if both the wake and non-wake accelerometer corrected sensors are enabled, and the accelerometer dynamic range is changed by the host, then a dynamic range changed Meta Event will be issued to the wake FIFO for the wake accelerometer corrected sensor as well as to the non-wake FIFO for the non-wake accelerometer corrected sensor.

#### 15.3.10 Meta Event: FIFO Watermark

This Meta Event occurs when the specified watermark level of a FIFO has been reached. Due to synchronization issues, this Meta Event may be displaced by a few dozen bytes, i.e. some other events may be reported after the watermark has been triggered and before the Meta Event occurs.

The Meta Event will be placed in the appropriate wake vs. non-wake FIFO.

#### 15.3.11 Meta Event: Initialized

This is the first Meta Event reported after the firmware has completed initialization.

It will be placed in both the Wake-up and Non-Wake-up FIFOs. It indicates the end of the initialization phase and shall be read from both FIFOs before any other operation, like e.g. enabling a virtual sensor, is performed. See Section 5 for details of the initialization procedure.

#### 15.3.12 Meta Event: Transfer Cause

The Meta Event occurs when,

- a host transfer has begun
- the reason is because of an on-change sensor, and
- the *Transfer Cause* Meta Event is enabled

In this case, the first event in the FIFO will be a Transfer Cause Meta Event. The sensor ID of the sensor causing the transfer will be included in byte 2.

It will be placed in the appropriate wake vs. non-wake FIFO.

### 15.3.13 Meta Event: Software Framework

This Meta Event will be issued for various Event-driven Software Framework-related errors.

The sensor ID for the sensor which is associated with the error is reported in byte 2, and the specific framework error is reported in byte 3.

Software Framework errors are:

- 1 = virtual sensor trigger was delayed (CPU is heavily loaded)
- 2 = virtual sensor trigger was dropped (an entire sample period passed without time to trigger the sensor; CPU overloaded)
- 3 = because the requested rate is so low, the hang detection logic has been disabled for this sensor (otherwise not an error)
- 4 = the parent of the specified virtual sensor is unexpectedly not enabled

### 15.3.14 Meta Event: Reset

This Meta Event is placed by the boot loader in the wake and non-wake streams, to increase the chances that the host will notice a watchdog reset during a period where the host is expecting to receive sensor data.

Byte 2 is always 0. Byte 3 indicates the Reset cause:

Table 83: List of Reset Causes

Byte 3 value	Reset Cause
0	Power-On Reset
1	External Reset (RESETN pin)
2	Reset by host command
4	Watchdog Reset

### 15.3.15 Meta Event: Spacer

This Meta Event is used to maintain a fixed size for the FIFO block header. If a previous block was discarded due to FIFO overflow, the FIFO block header will instead contain a FIFO Overflow Meta Event. The host should ignore the Spacer Meta Event.

## 15.4 Debug Data

This event contains binary or string debug data that has been created by using the `fwrite()` or `printf()` functions for debugging during the software development phase.

Table 84: Debug Data Format

Byte Number	Contents	Format
0	Sensor ID	8 bit unsigned, see Table 79: Overview of FIFO Event IDs.
1	Flags	8 bit unsigned integer

2 .. 17	Data	8 bit unsigned integer
---------	------	------------------------

The flags consist of 6 bits of valid length, which indicates the number of bytes in the Data area that are valid, and 1 bit indicating the type (binary or string):

Table 85: Flags in Debug Data Format

Bit Number	Contents
0-5	Valid length
6	Format (1 = binary, 0 = string)
7	Reserved

## 16 Reading FIFO Data

The FIFO content is read from Output Channel 1-3, which contain the wake-up, non-wake-up, and Status and Debug FIFO streams. The information about the amount of data in each channel is given by the first two bytes of each FIFO stream at the start of a host transfer (see Section 14 *FIFO Data Formats*).

When the host receives the host interrupt from the BHA260AB, it can optionally first read the Interrupt Status register to determine which channel has available data (or some other reason for the interrupt). It can then read at least the first 2 bytes of the channels that have data pending to determine how many bytes follow. It should then read all those bytes in one or more consecutive SPI or I2C read operations.

FIFO data is stored in one or more FIFO blocks by BHA260AB, as needed. The host only needs to start reading the appropriate channel register, and continue reading until the entire transfer is complete. The host may break this read into smaller blocks at its convenience, as long as it eventually reads the entire pending amount.

### Notes:

*Reading more data than the Transfer Length indicates is allowed; all bytes read beyond the Transfer Length will be 0. However, this is only true until the end of the current read transaction (until the host de-asserts the chip select pin in SPI mode, or until it issues the I2C STOP condition in I2C mode). A subsequent read transaction could return the beginning of the FIFO Descriptor of the next transfer.*

*Stopping to read before the total amount of bytes is read, will block any new host interrupt until the remainder of the FIFO is read, or until a transfer abort is requested (which might cause data loss)*

### 16.1 Host Interrupt Behavior

During normal operation, the host interrupt signal will be asserted when data is available in any of the FIFOs and it is time to notify the host.

E.g. this is the case when:

1. An enabled virtual sensor has a zero max report latency (timeout) and has generated a sample
2. A sensor has a non-zero max report latency, it has a sample in the FIFO, and it has timed out before any other sensor with a shorter latency or zero latency generates a sample
3. One (or more) virtual sensor has a non-zero max report latency, it has samples in the FIFO, the FIFO Watermark is non-zero, and it has been exceeded before the latencies timed out
4. A Meta Event has occurred which has its interrupt enable set (by default, only internal firmware errors or sensor hardware errors can generate interrupts)
5. The AP is in suspend mode, one or more wake-up sensors are enabled, and one or more wake-up events have occurred (no other event except unexpected reset will generate an interrupt in this state), and the wake-up FIFO interrupt disable bit is clear in the Host Interface Control register
6. The AP is in suspend mode, the wake-up FIFO watermark is non-zero, and all enabled sensors have non-zero latency; when the watermark is reached, the AP will be woken up

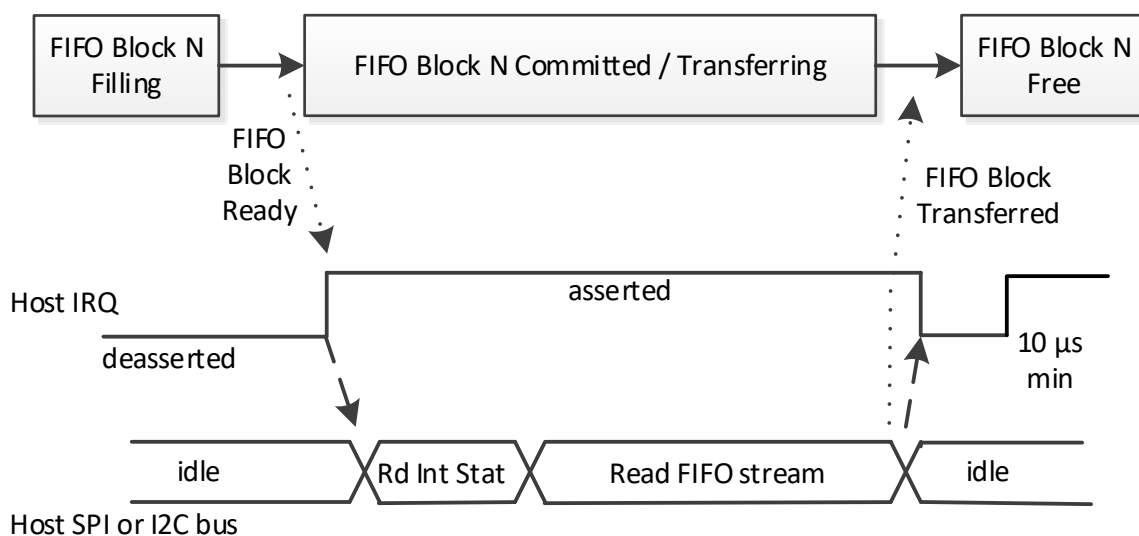


Upon power-up the host interrupt signal is only asserted if a watchdog timeout or other fatal error caused an unplanned reset, then using the previous interrupt configuration. However, a host-initiated reset due to power-on reset, reset pad, or write to the reset register will not result in an interrupt being raised.

In level interrupt mode, the host interrupt signal will remain asserted until the host has emptied the FIFO or has aborted the transfer with the Abort Transfer bit in the Host Interface Control register.

It may then reassert immediately depending on the notification criteria above.

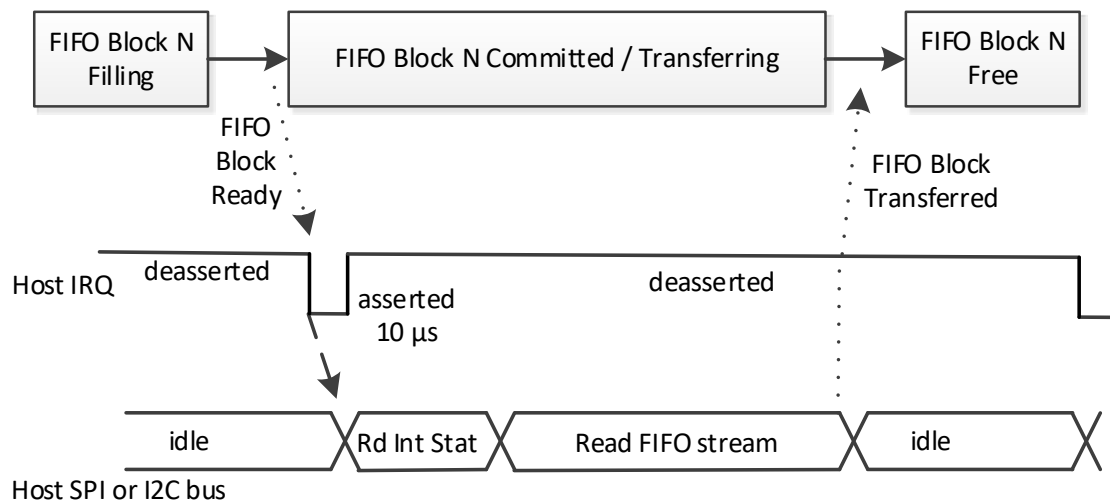
Figure 18: Active High Level Host Interrupt Example



**Note:**

The host interrupt may take up to 150us to de-assert after the host interface has been handled.

Figure 19: Active Low Edge Triggered Host Interrupt Example



In edge triggered mode, the host interrupt will pulse for 10 µs to start the transfer. It will pulse again after the host has emptied the FIFO or aborted the transfer, if there is more data to be read.

The host interrupt will always go low for a minimum of 10 µs between the time the host receives and empties the FIFO and any subsequent transfer. This is to ensure that either level or edge-triggered interrupts can be used by the host.

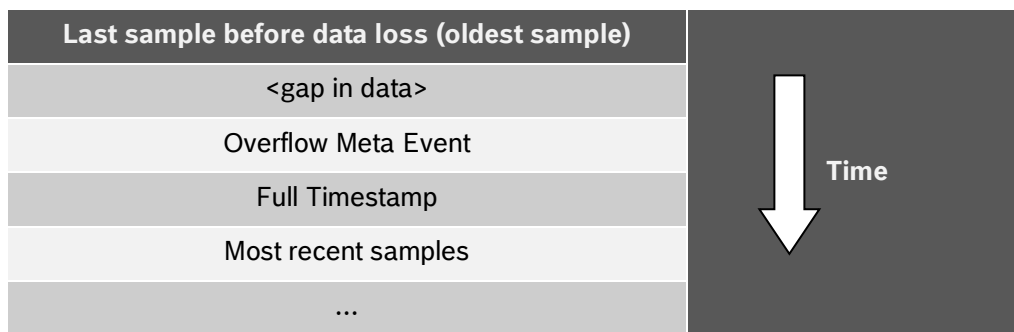
The time at which the rising edge of the interrupt occurred will be accessible from the Host Interrupt Timestamp Parameter, see Section 12.1.22.

## 16.2 FIFO Overflow Handling

In the event that the FIFO overflows, due to too high sample rates and/or slow reading by the host, or simply the host being asleep, BHA260AB will discard the oldest data in the FIFO first. The data will be discarded in full FIFO Blocks, following the FIFO format described in Section 14. In the event a host interrupt is pending or a host transfer is actively underway, some amount of the oldest data, namely the currently transferred and the following FIFO block, will still be transferred to the host and are not affected by the discard. Only in the rare case that besides the currently transferred block, only one other FIFO block is present, then this block will be discarded.

The Event-driven Software Framework will insert an Overflow Meta Event (if enabled) into the header of the FIFO Block at the point of data loss, followed immediately by the Full Timestamp for the continuation point, followed finally by the new data sampled after the overflow. This way, the host can know that data was lost and has an accurate timestamp for the data that continues after the data loss.

Table 86: Data Chronology after FIFO Overflow



### 16.3 Application Processor Suspend Mode

When the application processor goes into suspend mode, it should inform BHA260AB by first writing a 1 to the AP Suspend bit in the Host Interface Control register, so that only wake-up sensors issue a host interrupt. It is recommended that the host precedes this with a FIFO Flush and complete emptying of the FIFO, so that the host is not immediately woken by mistake.

When the AP wakes up, it should notify BHA260AB by clearing the AP Suspend bit again, so that all enabled sensors can (if configured) issue a host interrupt as needed. The BHA260AB will detect this action and automatically prepare the output channels and assert the host interrupt, if data is pending. If the AP did not notify BHA260AB of entering and leaving suspend mode, but instead masked the host interrupt line, the AP may find the host-interrupt line already asserted when it comes out of suspend. The first pending FIFO transfer will contain a smaller Transfer Length than the total available. Once the host has transferred this smaller amount of bytes, the next host transfer will cover the remaining amount to be transferred.

### 16.4 Loss of sync recovery

It might occur that the host can't interpret the FIFO data anymore that it is reading. In this case it may have lost sync to the FIFO structure. To recover from this state, the host should abort the current transfer by writing the appropriate *Abort Transfer* bit in the *Host Interface Control* (0x06) register to 1. This may cause data loss of 32 bytes of data.

Alternatively, the host can also finish the current transfer, discard what it is receiving, and then wait for the next transfer as signaled by a host interrupt.

After that, the next transfer will start again with a whole block of data as specified in Section 14 *FIFO Data Formats*.

## 17 Error detection and recovery

The BHA260AB can, under extraordinary circumstances, reach a fatal error condition during the boot or execution of firmware. Therefore, the host software or driver need to be able to detect and recover from such a condition.

It is recommended for a production system to make the BHA260AB driver self-monitor the recovery process. It should back off recovery attempts over longer and longer time intervals, and eventually stop recovery attempts after some reasonable amount of time, and log a fatal error. This is to prevent an infinite loop of recovery attempts caused by a continuous fatal hardware error or corrupted firmware file on the host system. The following three scenarios should be addressed in the host:

1) Watchdog / Power On Reset occurred during operation. With one of the following symptoms:

- Host IRQ is received with 0 bytes to transfer
- Firmware Idle is set in the Boot Status register
- Kernel version number is now 0
- Error register contains a non-zero error code
- Wake-up and Non-Wake-up FIFOs contain Reset Meta Events

As a reaction, the following steps should be taken:

1. Log registers 0x04 to and including 0x31
2. Issue Download Post Mortem Data command, receive Crash Dump status packet, store for later analysis
3. Reload firmware image
4. Configure system parameters (FIFO watermark, etc.)
5. Restart virtual sensors

2) Errors detected by BHA260AB firmware:

- Error Event in a FIFO (precondition: the error Meta Events are not masked)
- Non-zero state of Error register

Reaction for different error categories should be:

- Fatal errors: issue reset over SPI or I2C (write 0x01 to register 0x9B or assert HW reset pin), then the same as item 1
- Hardware errors: same as previous point
- Programming errors: should never occur in the field; recovery same as previous point
- Temporary errors: It is acceptable to ignore and continue. It is recommended to log the error state for later analysis.

3) Unexpected / Unknown State / Live-lock

- Detection only by software watchdog in the host driver

Reaction should be:

- Follow the same recovery as item 2) for Fatal Errors

## 18 Physical and Electrical Specifications

### 18.1 Absolute Maximum Ratings

Table 87: Absolute maximum ratings

Parameter	Condition	Min	Max	Unit
Voltage at Supply Pin	VDD Pin	-0.3	4.25	V
	VDDIO Pin	-0.3	2.75	V
Voltage at any Logic Pin	Non-Supply Pin	-0.3	VDDIO+0.3	V
Passive Storage Temp. Range	<=65% rel. H.	-50	150	°C
None-volatile memory (NVM) Data Retention	T = 85°C, after 15 cycles	10		y
Mechanical Shock	Duration 200 µs, half sine		10,000	g
	Duration 1.0 ms, half sine		2,000	g
	Free fall onto hard surfaces		1.8	m
ESD	HBM		2	kV
	CDM		500	V
	MM		200	V

*Note:*

Stress above these limits may cause damage to the device. Exceeding the specified electrical limits may affect the device reliability or cause malfunction.

### 18.2 Operating Conditions

Table 88: Operating conditions

Parameter	Symbol	Min	Typ	Max	Unit
Supply Voltage IMU Analog Domain	V <sub>DD</sub>	1.71	1.8	3.6	V
Supply Voltage I/O Domain and Fuser2 Core	V <sub>DDIO</sub>	1.71	1.8	1.89	V
External regulator decoupling capacitor	C <sub>reg</sub>	0.1	0.22	0.5	µF
	ESR			0.5	Ω
Operating Temperature	T <sub>A</sub>	-40		85	°C
RESETN pulse duration (active low)	t <sub>RSTN</sub>	100			ns

### 18.3 Electrical characteristics

Table 89: Electrical characteristics

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Brown-out detection level rising	V <sub>BROUT-R</sub>	T <sub>A</sub> = -40°C to 85°C	1.3		1.6	V

Voltage Input Low Level	V <sub>IL</sub>				0.3VDDIO	-
Voltage Input High Level	V <sub>IH</sub>			0.7VDDIO		-
Voltage Output Low Level, auxiliary IMU interface (pads ASDX, ASCX, OCSB, OSDO)	V <sub>OAL</sub>	VDDIO=1.71V, IOL=3mA			0.2VDDIO	-
Voltage Output High Level, auxiliary IMU interface (pads ASDX, ASCX, OCSB, OSDO)	V <sub>OAH</sub>	VDDIO=1.71V, IOH=3mA	0.8VDDIO			-
Voltage Output Low Level, all other outputs	V <sub>OML</sub>	VDDIO=1.71V, IOL=4mA for LDS <sup>3)</sup> , IOL=6mA for HDS <sup>4)</sup>			0.3VDDIO	-
Voltage Output High Level, all other outputs	V <sub>OMH</sub>	VDDIO=1.71V, IOL=4mA for LDS <sup>3)</sup> , IOL=6mA for HDS <sup>4)</sup>	0.7VDDIO			-
Internal Pull-Up resistors	R <sub>UP</sub>		71	121	214	kΩ
Internal Pull-down resistor (HSDO pin)	R <sub>DN</sub>		62	109	208	kΩ
Pad Capacitance, all inputs except pads M1SCX, M1SDX, M1SDI, ASDX, ASCX, OCSB, OSDO, RESV	C <sub>PAD</sub>				1.8	pF
Pad Capacitance M1SCX, M1SDX, M1SDI, RESV1/2/3	C <sub>PAD,M1</sub>				6.8	pF
Pad Capacitance ASDX, ASCX, OCSB, OSDO	C <sub>PAD,AUX</sub>				5	pF
System Oscillator Frequency	f <sub>SYSLR</sub>	Long run mode, T <sub>A</sub> =25°C	18.4	20	21.6	MHz
	f <sub>SYSTEM</sub>	Turbo Mode, T <sub>A</sub> =25°C	46	50	54	MHz
System Oscillator Temperature Drift	DF <sub>SYSLR,TE</sub> MP	Drift from 25°C for T <sub>A</sub> = -40°C to 85°C	-6		6	%
System Oscillator Start-up time <sup>(1)</sup>	OSC <sub>SYSTEM,S</sub> TART	T <sub>A</sub> =25°C			4	μs
Timer Oscillator Frequency	f <sub>TMR</sub>	T <sub>A</sub> =25°C	125	128	131	kHz
Timer Oscillator Temperature Drift	DF <sub>TMR,TE</sub> MP	Drift from 25°C for T <sub>A</sub> = -40°C to 85°C	-5		5	%
Timer Oscillator Start-up time <sup>(1)</sup>	OSC <sub>TMR,START</sub>	T <sub>A</sub> =25°C			250	μs
Timer Oscillator trim step size <sup>(5)</sup>	d <sub>TRIM,TMR</sub>	T <sub>A</sub> =25°C		0.9		%
Current consumption, total on pins VDD and VDDIO	(I <sub>DD</sub> +I <sub>DDIO</sub> )	VDD=VDDIO=1.8V, Accel in suspend, Fuser2 in Deep Sleep <sup>2)</sup> , 32KBytes RAM retention, T <sub>A</sub> = 25°C			7.8	μA
		VDD=VDDIO=1.8V, Accel in suspend, Fuser2 in Regular Sleep, 32KBytes RAM retention, T <sub>A</sub> = 25°C			8.1	μA
		VDD=VDDIO=1.8V, Accel in APS <sup>6)</sup> mode, ODR 25 Hz, Fuser2 in deep sleep <sup>2)</sup> , T <sub>A</sub> =25°C			15	μA
		VDD=VDDIO=1.8V, Accel in APS <sup>6)</sup> mode, ODR 50 Hz, Fuser2 in deep sleep <sup>2)</sup> , T <sub>A</sub> =25°C			19	μA

		VDD=VDDIO=1.8V, Accel in full operation mode, Fuser2 in deep sleep <sup>2)</sup> , T <sub>A</sub> =25°C		151		μA
		VDD=VDDIO=1.8V, Fuser2 executing matrix multiplication in Long Run mode, Accel in suspend, T <sub>A</sub> = 25°C,		840		μA
		VDD=VDDIO=1.8V, Fuser2 executing Coremark in Long Run mode, Accel in suspend, T <sub>A</sub> = 25°C		950		μA
		VDD=VDDIO=1.8V, Fuser2 executing Coremark in Turbo mode, Accel in suspend, T <sub>A</sub> = 25°C		2.8		mA
Fuser2 CPU benchmark		Metaware compiler ccac, compiler options set for maximum performance		3.67		CoreMark / MHz

**Notes:**

- 1) Start-up time is the time from oscillator enable until the first rising edge of the oscillator. The first cycle will be within 10% of the final frequency.
- 2) Fuser2 in Deep Sleep: all oscillators are disabled.
- 3) LDS: low drive strength configuration of output driver
- 4) HDS: high drive strength configuration of output driver
- 5) Expressed as percentage of the oscillator frequency
- 6) APS: Advanced Power Saving mode of accel sensor

**18.4 Physical Characteristics and Measurement Performance**

If not stated otherwise, the given values are over lifetime and full performance temperature and voltage ranges, minimum/maximum values are ±3σ.

Table 90: Operating conditions accelerometer

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Acceleration Range	g <sub>FS2g</sub>	Selectable via serial digital interface		±2		g
	g <sub>FS4g</sub>			±4		g
	g <sub>FS8g</sub>			±8		g
	g <sub>FS16g</sub>			±16		g
Start-up time	t <sub>A,su</sub>	Suspend/low power mode to normal mode		3.2		ms

Table 91: Output signal accelerometer

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Resolution				16		bit



Sensitivity	S <sub>2g</sub>	g <sub>FS2g</sub> , T <sub>A</sub> =25°C		16384		LSB/g
	S <sub>4g</sub>	g <sub>FS4g</sub> , T <sub>A</sub> =25°C		8192		LSB/g
	S <sub>8g</sub>	g <sub>FS8g</sub> , T <sub>A</sub> =25°C		4096		LSB/g
	S <sub>16g</sub>	g <sub>FS16g</sub> , T <sub>A</sub> =25°C		2048		LSB/g
Sensitivity Error	S <sub>A_err</sub>	T <sub>A</sub> =25°C Nominal VDD supplies, all ranges		±0.8	±2	%
Sensitivity Temperature Drift	TCS <sub>A</sub>	g <sub>FS2g</sub> , Nominal VDD supplies best fit straight line		±0.01		%/K
Sensitivity change over supply voltage	S <sub>A,VDD</sub>	T <sub>A</sub> =25°C, VDD,min ≤ VDD ≤ VDD,max best fit straight line		0.02		%/V
Zero-g Offset	Off <sub>A,init</sub>	g <sub>FS8g</sub> , T <sub>A</sub> =25°C, nominal VDD supplies, component level		±20		mg
	Off <sub>A,life</sub>	g <sub>FS8g</sub> , T <sub>A</sub> =25°C, nominal VDD supplies, soldered, over life time <sup>1)</sup>		±50		mg
Zero-g Offset Temperature Drift	TCO <sub>A</sub>	g <sub>FS8g</sub> , Nominal VDD supplies best fit straight line		±0.5		mg/K
Nonlinearity	NL <sub>A</sub>	Best fit straight line, g <sub>FS2g</sub>		±0.5		%FS
Output Noise	n <sub>A,rms</sub>	g <sub>FS8g</sub> , T <sub>A</sub> =25°C, nominal VDD, Normal mode, Filter setting 80 Hz, ODR 200 Hz		1.2		mg-rms
Cross Axis Sensitivity	S <sub>A</sub>	Relative contribution between any two of the three axes		1		%
Alignment Error	E <sub>A</sub>	Relative to package outline		±0.5		°
Nominal Output Data rate (set of x,y,z rate)	ODR <sub>A</sub>		12.5		1600	Hz

Note:

1) Values taken from qualification, according to Reference 5 of Section 26

## 18.5 Timing Characteristics

### 18.5.1 Fuser2 Power-Up and Power-Down Timing Characteristics

Table 92: Fuser2 Power-Up and Power-Down Timing Characteristics

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Start-up time <sup>1)</sup>	T <sub>start</sub>				500	us
Bootloader boot time	T <sub>boot_bl_host</sub>	Host boot <sup>2)</sup>			1300	us
Firmware load time						

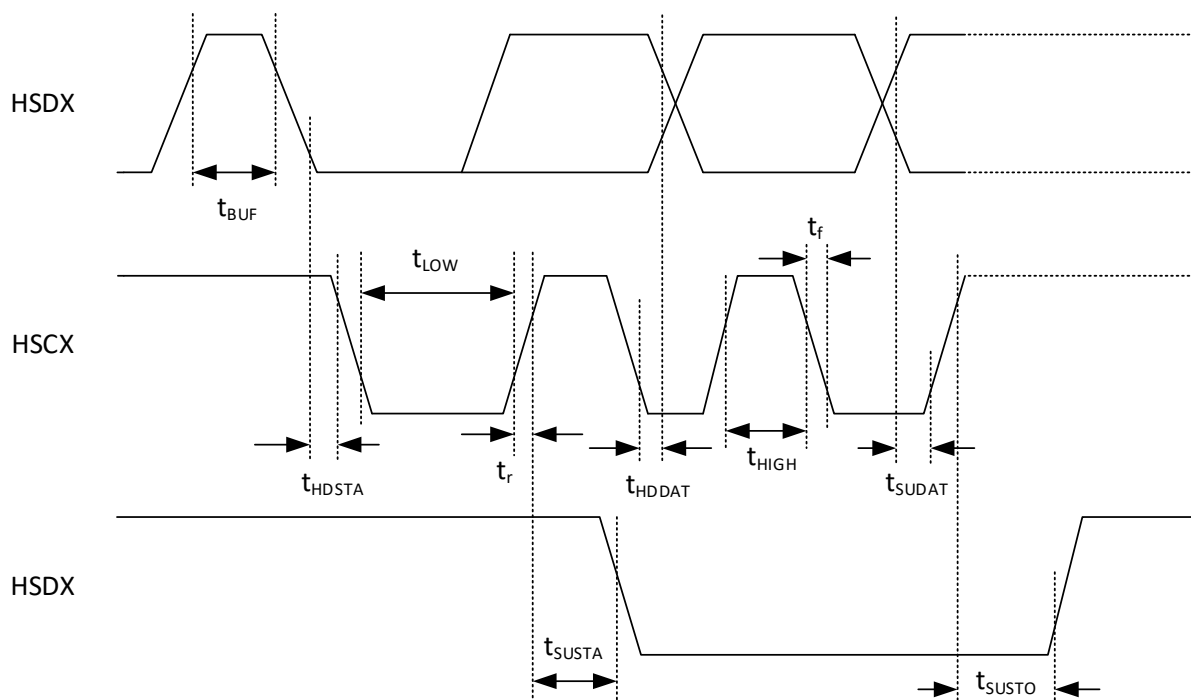
	T_load_fw_host	Host firmware load and verify <sup>3)</sup>		121		ms
Firmware boot time	T_boot_fw_host	Host firmware boot <sup>4)</sup>		81		ms
Run-level switch time to turbo mode	T_switch_turbo	--			3	us
Run-level switch time to long-run mode	T_switch_longrun	--			3	us
Wake-up time from sleep / deep-sleep	T_wake	--			5	us
Time between RAM bank power-on in long-run mode	T_PwrRam_long_run	--			8	CLK <sup>5)</sup>
Time between RAM bank power-on in turbo mode	T_PwrRam_turbo	--			20	CLK <sup>5)</sup>

Notes:

- 1) The start-up time is measured from the point of time when the supply voltage reaches the power-on-reset trip level until the system oscillator is enabled
- 2) The host boot time is measured from the release of the reset until the device is ready to receive a firmware upload (bit "Host Interface Ready" in register "Boot Status" gets high)
- 3) Long run mode, Firmware load and verify time for a 76 kByte firmware file, using host interface in SPI mode at 5 Mbit clock rate.
- 4) Long run mode, firmware boot time, measured from sending the command "Boot firmware" until the "Host Interface Ready" bit gets high. Please note that the firmware verification is started already during upload of the firmware, i.e. this time depends on T\_load\_fw\_host
- 5) CLK is the clock period of the system clock, depending on the mode setting (Long Run vs. Turbo)

### 18.5.2 Host I2C Interface Timing

Figure 20: Host I2C interface timing



Unless otherwise specified, see I2C Specification, Reference 1 in Section 26 for details.

Table 93: Host I2C interface timing

Parameter	Symbol	Condition	Min	Typ	Max	Unit
HSCX frequency	$f_{i2c\_sm}$	STM, FM, FM+ <sup>1)</sup>		400	1000	kHz
HSCX frequency	$f_{i2c\_hs}$	HSM <sup>1)</sup>			3400	kHz
Bus load capacitor	Cb	on HSDX and HSCX			400	pF
HSCX low time	$t_{LOW}$	HSM / Cb <= 100pF	160			ns
HSDX input setup time <sup>2)</sup>	$t_{SU;DAT}$	STM, FM, FM+	85			ns
		HSM	15			ns
HSDX output hold time	$t_{HD;DAT}$	STM, FM, FM+ / Cb <= 100pF	70			ns
		STM, FM, FM+ / Cb <= 400pF	90			ns
		HSM / Cb <= 100pF	24			ns
		HSM / Cb <= 400pF	27			ns
HSDX output fall time	$t_{OF}$	Cb <= 100pF	1.5			ns
		Cb <= 400pF	6.5			ns
HSDX-HSCX input delta	$t_{HD;STA}$	STM, FM, FM+	50	70	90	ns
		HSM	6	11	20	ns

**Notes:**

- 1) STM = standard mode, FM = fast mode, FM+ = fast+ mode and HSM = high speed mode
- 2) Measured with transition times of 16ns from 0% VDD on SDA to 0% VDD on SCL for SDA rising and 100% VDD on SDA to 0% VDD on SCL for SDA falling.

### 18.5.3 Host SPI Interface Timing

Figure 21: Write Transaction on 4-Wire Host SPI

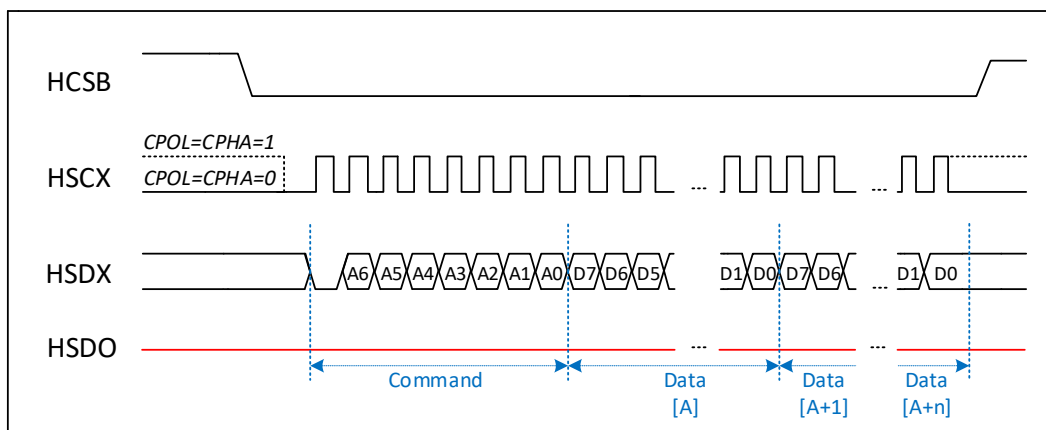


Figure 22: Read Transaction on 4-Wire Host SPI

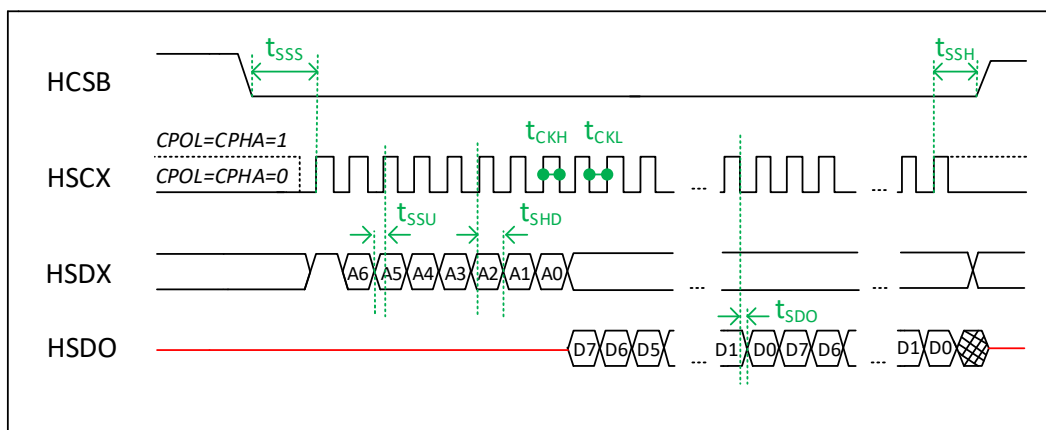


Figure 23: Read Transaction on 3-Wire Host SPI

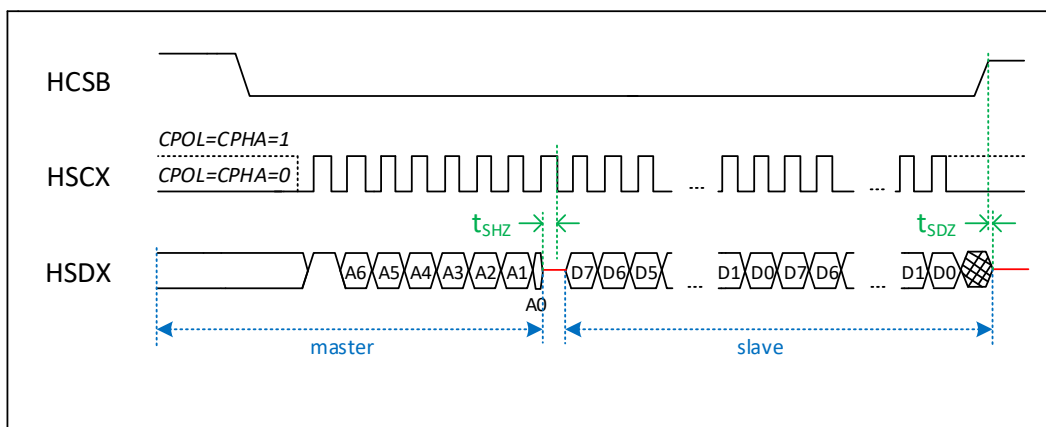


Table 94: Host SPI Interface Timing Parameters in Long-Run (LR) and Turbo mode

Parameter	Symbol	Condition	Min		Max		Unit
			LR	Turbo	LR	Turbo	
SPI clock frequency	$f_{SCK}$				20	50	MHz

SPI clock high time	$t_{CKH}$		25	10			ns
SPI clock low time	$t_{CKL}$		25	10			ns
SPI select setup time	$t_{SSS}$		50	20			ns
SPI select hold time	$t_{SSH}$		50	20			ns
HSDX input setup time	$t_{SSU}$	max trans: 1.5ns	17	6			ns
HSDX input hold time	$t_{SHD}$	max load: 12pF (for 3-wire SPI)	7	4			ns
HSDX output to Hi-Z time	$t_{SHZ}$		15	5			ns
HSDO output valid time	$t_{SDO}$	max load: 12pF			17	9	ns
SPI select to MISO Hi-Z time	$t_{SDZ}$				20	10	ns

### 18.5.4 Master Interface I2C Timing

The I2C Master Timing is compliant with Reference 1 in Section 26, Standard, Fast and Fast+ Mode.

### 18.5.5 Master Interface SPI Timing

Figure 24: Master Interface SPI Clock Timing

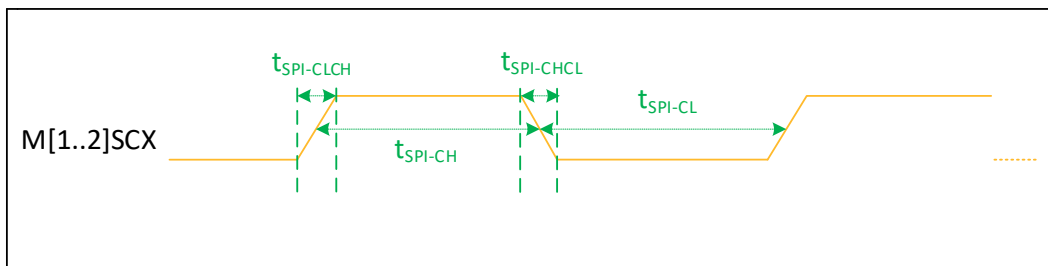


Figure 25: Master Interface SPI Timing

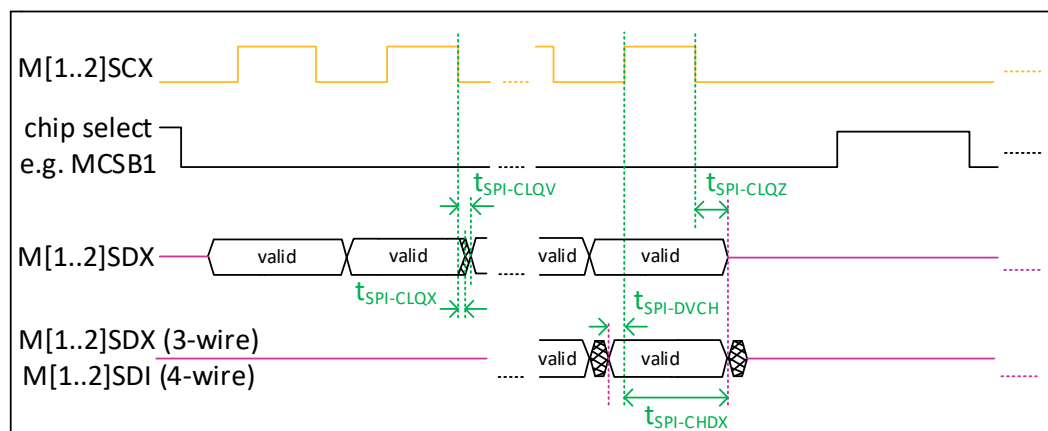


Table 95: Master Interface SPI Timing Parameters for Long-Run (LR) and Turbo Mode

Parameter	Symbol	Condition	Min <sup>1)</sup>	Max <sup>1)</sup>	Unit
-----------	--------	-----------	-------------------	-------------------	------

			LR	Turbo	LR	Turbo	
SPI clock frequency	f <sub>SPI</sub>	no division	18	45	22	55	MHz
Clock rise time	t <sub>SPI-CLCH</sub>	load: 12pF			2.1	2.1	ns
Clock fall time	t <sub>SPI-CHCL</sub>				2.1	2.1	ns
Clock high time	t <sub>SPI-CH</sub>	no division	20	8	29	12	ns
Clock low time	t <sub>SPI-CL</sub>		20	8	29	12	ns
Input data valid to clock rising edge	t <sub>SPI-DVCH</sub>	max trans <sup>2)</sup>	4	4			ns
Clock rising edge to input data valid <sup>3)</sup>	t <sub>SPI-CHDX</sub>	load: 12pF	50	50			%T
Clock falling edge to output data invalid	t <sub>SPI-CLQX</sub>		-4	-4			ns
Clock falling edge to output data valid	t <sub>SPI-CLQV</sub>	load: 12pF			5	5	ns
Clock falling edge to output data Hi-Z	t <sub>SPI-CLQZ</sub>		-3	-3			ns

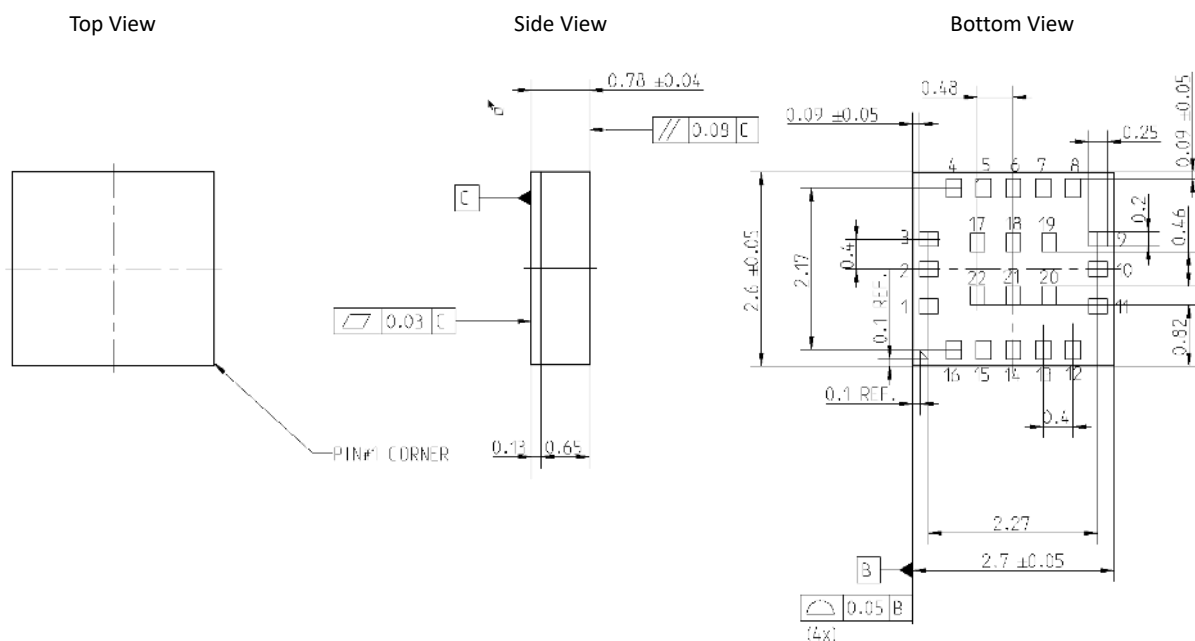
**Notes:**

- 1) Timing pertains to both Master Interface 1 and 2. Timing parameters are specified for CPOL/CPHA of 00 and 11; they also hold for CPOL/CPHA of 01 and 10 with according edge adjustment.
- 2) Maximum transition time = 1.5ns; load number are for 3-wire SPI
- 3) T = clock period

## 19 Mechanical Specifications

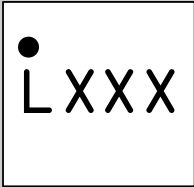
### 19.1 Outline Dimensions

Figure 26: Outline dimensions



### 19.2 Device Marking

Table 96: BHA260AB Device Marking

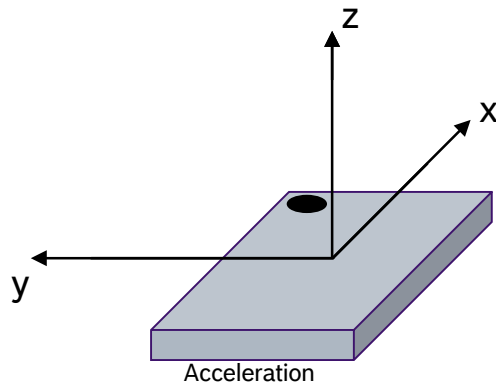
Labeling	Name	Symbol	Remark
	Pin 1 dot	•	Pin 1 indicator; fixed; left alignment
	L	L	Product identifier; fixed, center alignment; font OCRA
	XXX	XXX	Lot identifiers (3 digits, no reset, Alphanumerical 0-Z by device): dynamic, center alignment; font OCRA

### 19.3 Sensing axes and axes remapping

The default axis orientation is shown in Figure 31. This orientation is valid for all sensor outputs (both physical and virtual). For any other sensor connected to the BHA260AB, e.g. gyroscope or magnetometer, the physical alignment of the corresponding sensor and its axis should be according to Figure 31 to guarantee correct 9DoF data fusion.

In case that the default axis orientation of BHA260AB and/or its sensor extensions does not match with the target device coordinate system, axis remapping can be performed to reassign the sensing axes of BHA260AB so that they match with the axes defined by the target device coordinate system

Figure 27: Sensing Axes



Possible placement and remapping options are given in Figure 32.

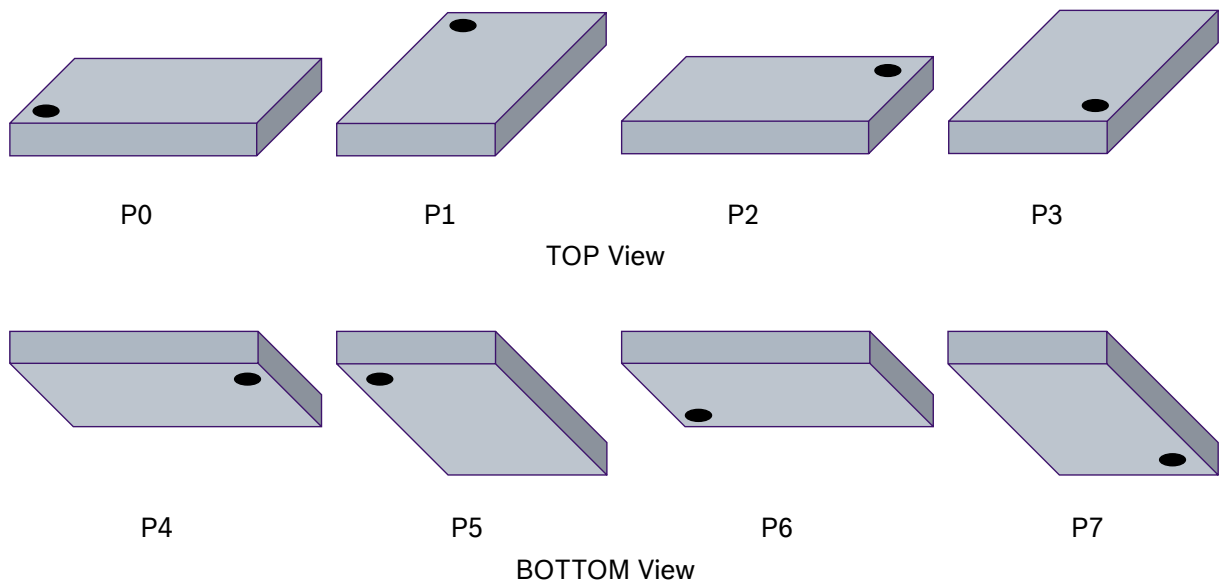
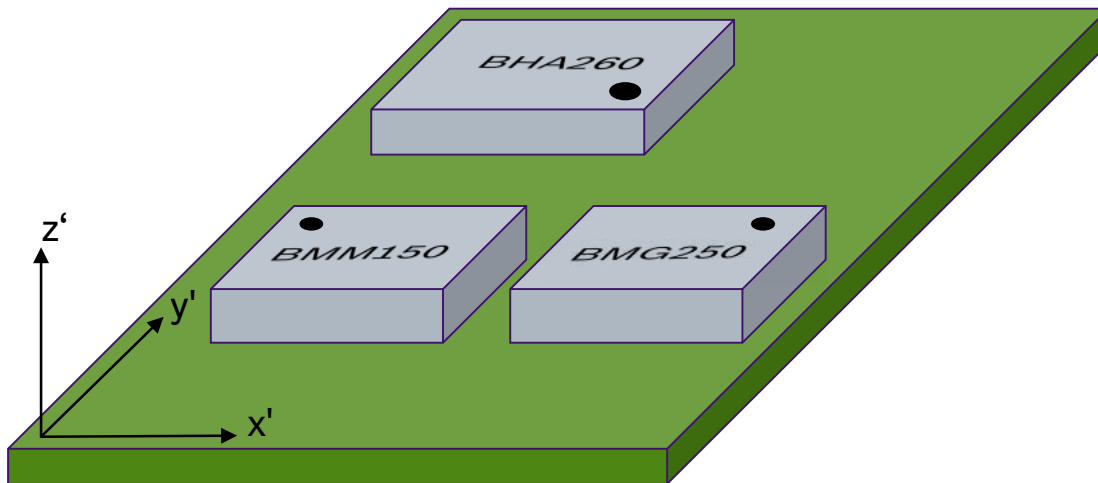


Figure 28: Placement Options with Respect to Device Orientation



Figure 29: Example component placement



If the sensing axes of the sensor do not match with the coordinate system of the target device or the sensor extensions which are potentially connected to the BHA260AB, several tools can be used to remap the sensing assignment in the FW file. This enables a subsequent axis alignment of all virtual and physically integrated and/or attached sensors to match with the target coordinate system.

The following equation transforms the device axes ( $x \ y \ z$ ) into the target coordinate system ( $x' \ y' \ z'$ ). It can also be used to transform the axes of sensors that are attached to the BHA260AB:

$$(x' \ y' \ z') = (x \ y \ z) \cdot \begin{pmatrix} x_x & x_y & x_z \\ y_x & y_y & y_z \\ z_x & z_y & z_z \end{pmatrix}$$

For the example shown in Figure 33 the coordinate system of BHA260AB needs to be remapped to the target device as in placement option P3 shown in Figure 32. The corresponding remapping matrix is shown below. The same applies for BMM150 (see BMM150 datasheet for axis orientation). The axes orientation of BMG250 matches with the target coordinate system, therefore the remapping matrix looks as below.

$$M_{BHA260} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad M_{BMM150} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad M_{BMG250} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

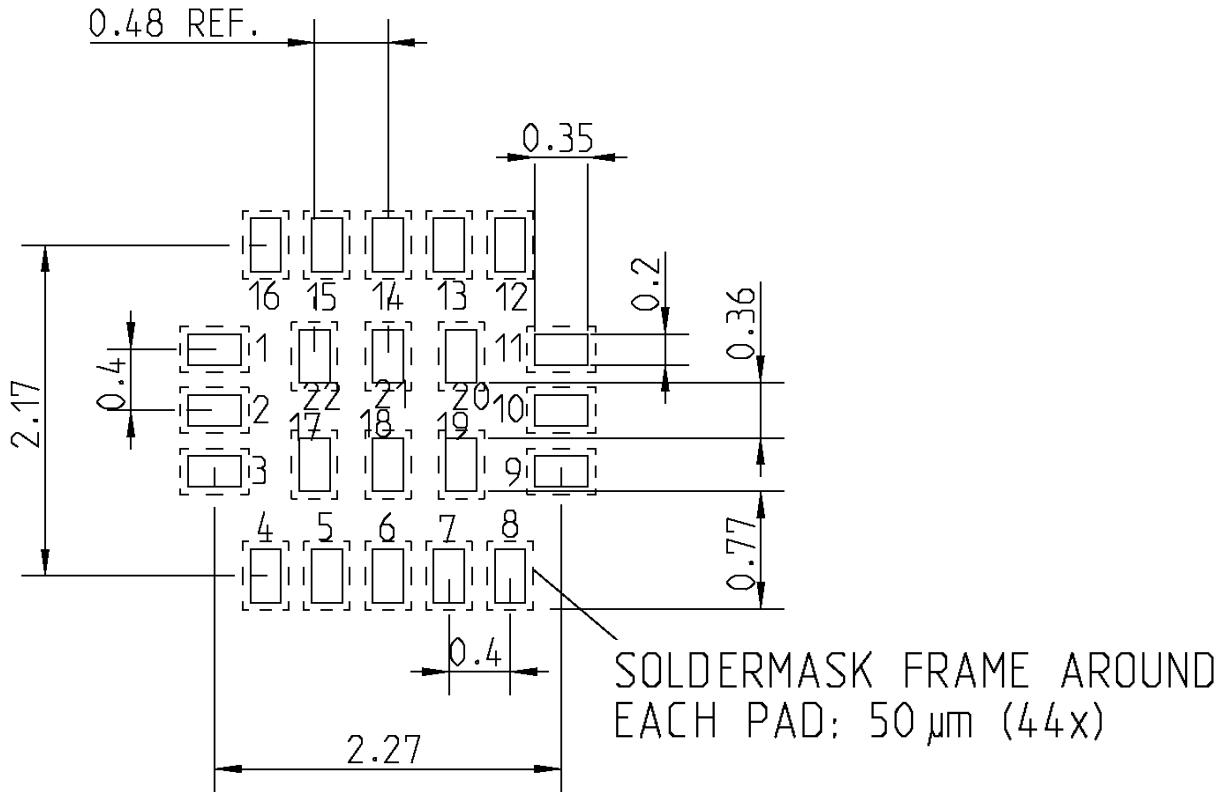
To apply the remapping defined by the matrix  $\begin{pmatrix} x_x & x_y & x_z \\ y_x & y_y & y_z \\ z_x & z_y & z_z \end{pmatrix}$  for any triaxial sensor, the matrix elements have to be inserted in the specific board configuration file. The values Cal0 ( $c_0$ ) to Cal8 ( $c_8$ ) for a sensor correspond to the matrix elements as follows:

$$\begin{pmatrix} x_x & x_y & x_z \\ y_x & y_y & y_z \\ z_x & z_y & z_z \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 \\ c_3 & c_4 & c_5 \\ c_6 & c_7 & c_8 \end{pmatrix}$$

For further information on how to apply this matrix, please refer to Section 13.3.2.7. In case of questions and for technical support, please contact our regional offices, distributors, and sales representatives.

### 19.4 PCB Footprint recommendation

Figure 30: Footprint recommendation<sup>1)</sup>



**Note:**

1) Dimensions are in mm, unless otherwise noted.

## 20 Packaging Specifications (Tape & Reel)

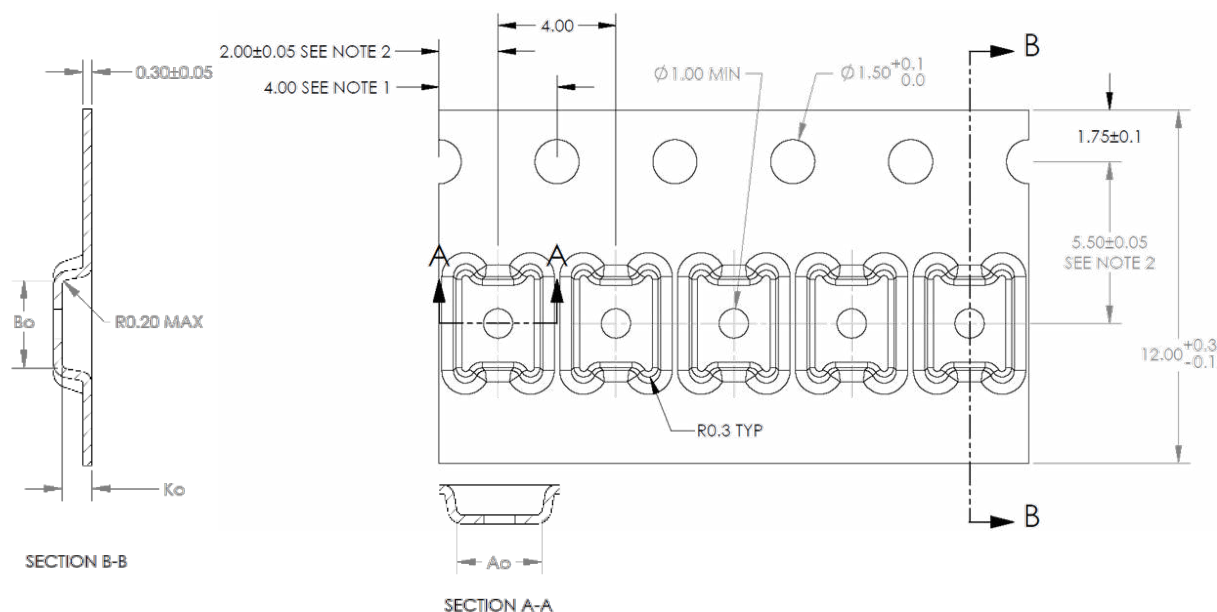
BHA260AB is shipped in standard cardboard box.

Box dimensions for one reel: L x W x H = 35 cm x 35 cm x 6 cm

Quantity: 10000 pieces per reel.

Figure 35 shows the dimensions of the tape used for shipping BHA260AB sensor. The tape is made of conductive polystyrene (IV).

Figure 35: Tape specification



**Notes:**

Dimensions are in mm, unless otherwise noted

Ao and Bo are measured on a plane at a distance “R” above the bottom of the pocket.

Tolerances unless specified: 1  $Pl \pm 0.2$  2  $Pl \pm 0.10$

1) 10 sprocket hole pitch cumulative tolerance +/-0.2

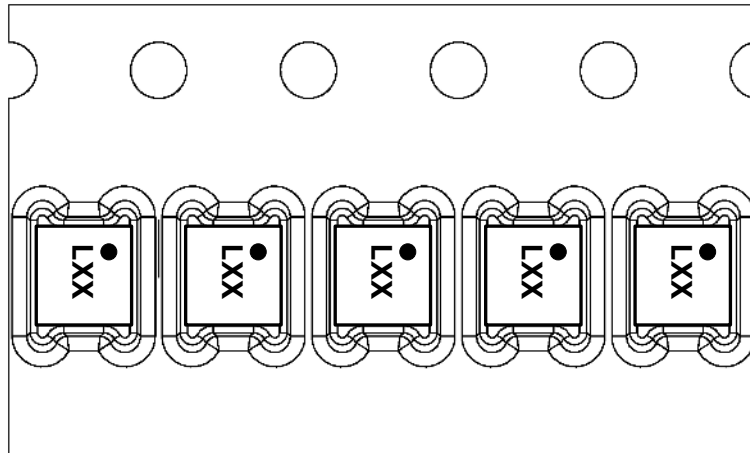
2) Pocket position relative to sprocket hole measured as true position of pocket, not pocket hole.

Table114: Tape Pocket Dimensions

Symbol	DIM	+/-
Ao	2.80	0.05
Bo	2.95	0.05
Ko	1.00	0.10

## 20.1 Orientation within the tape

Figure 31: Part orientation within tape



## 20.2 Multiple sourcing

In order to improve its products and secure the mass product supply, Bosch Sensortec employs multiple sources in the supply chain.

While Bosch Sensortec takes care that all of the parameters described above are 100% identical for all sources, there can be differences in device marking and bar code labeling.

However, as secured by the extensive product qualification process of Bosch Sensortec, this has no impact to the usage or to the quality of the product.

## 21 Handling, Soldering and Environmental Guidelines

### 21.1 Handling instructions

Micromechanical sensors are designed to sense acceleration with high accuracy even at low amplitudes and contain highly sensitive structures inside the sensor element. The MEMS sensor can tolerate mechanical shocks up to several thousand g's. However, these limits might be exceeded in conditions with extreme shock loads such as e.g. hammer blow on or next to the sensor, dropping of the sensor onto hard surfaces etc.

We recommend avoiding g-forces beyond the specified limits during transport, handling and mounting of the sensors in a defined and qualified installation process.

This device has built-in protections against high electrostatic discharges or electric fields (e.g. 2kV HBM); however, anti-static precautions should be taken as for any other CMOS component. Unless otherwise specified, proper operation can only occur when all terminal voltages are kept within the supply voltage range. Unused inputs must always be tied to a defined logic voltage level.

For more details on recommended handling, soldering and mounting please refer to the corresponding "Handling, soldering and mounting instructions" document or contact our regional offices, distributors and sales representatives.

### 21.2 Soldering guidelines

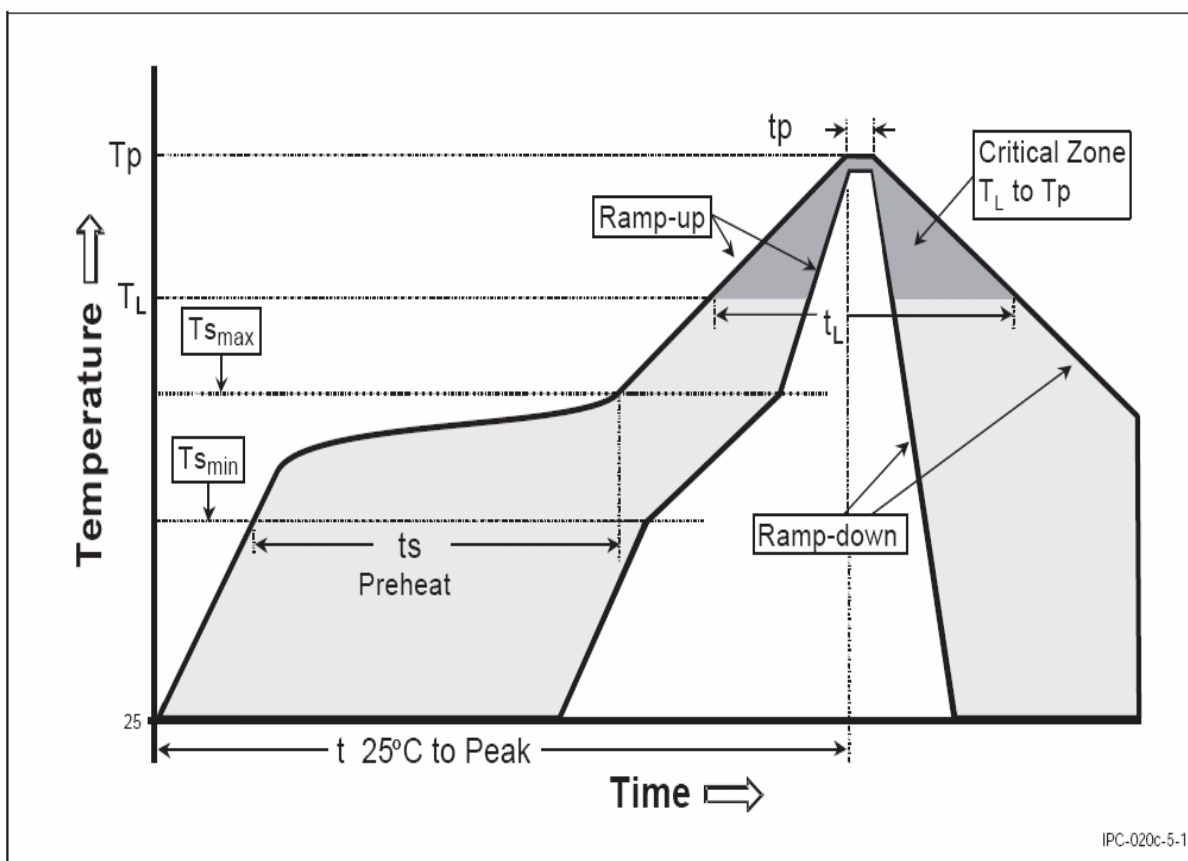
The moisture sensitivity level of the BHA260AB sensors corresponds to JEDEC Level 1, see also Reference 5 and Reference 6 in Section 26.

The sensor fulfils the lead-free soldering requirements of the above-mentioned IPC/JEDEC standard, i.e. reflow soldering with a peak temperature up to 260°C.

Figure 32: Soldering profile for BHA260AB

Profile Feature		Pb-Free Assembly
Average Ramp-Up Rate (T <sub>s_max</sub> to T <sub>p</sub> )		3° C/second max.
<b>Preheat</b> - Temperature Min (T <sub>s_min</sub> ) - Temperature Max (T <sub>s_max</sub> ) - Time (t <sub>s_min</sub> to t <sub>s_max</sub> )		150 °C 200 °C 60-180 seconds
Time maintained above: - Temperature (T <sub>L</sub> ) - Time (t <sub>L</sub> )		217 °C 60-150 seconds
Peak/Classification Temperature (T <sub>p</sub> )		See Table 4.2
Time within 5 °C of actual Peak Temperature (t <sub>p</sub> )		20-40 seconds
Ramp-Down Rate		6 °C/second max.
Time 25 °C to Peak Temperature		8 minutes max.

Note 1: All temperatures refer to topside of the package, measured on the package body surface.



### 21.3 Environmental Safety

The BHA260AB sensors meet the requirements of the EC restriction of hazardous substances (RoHS) directive, see also:

RoHS - Directive 2011/65/EU and its amendments, including the amendment 2015/863/EU on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

The BHA260AB is halogen-free. For more details on the analysis results please contact our regional offices, distributors and sales representatives.

# LEGAL DISCLAIMER

## 22 Engineering Samples

Engineering Samples are marked with an asterisk (\*), (E) or (e). Samples may vary from the valid technical specifications of the product series contained in this data sheet. They are therefore not intended or fit for resale to third parties or for use in end products. Their sole purpose is internal client testing. The testing of an engineering sample may in no way replace the testing of a product series. Bosch Sensortec assumes no liability for the use of engineering samples. The Purchaser shall indemnify Bosch Sensortec from all claims arising from the use of engineering samples.



## 23 Product use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or safety-critical systems. Safety-critical systems are those for which a malfunction is expected to lead to bodily harm, death or severe property damage. In addition, they shall not be used directly or indirectly for military purposes (including but not limited to nuclear, chemical or biological proliferation of weapons or development of missile technology), nuclear power, deep sea or space applications (including but not limited to satellite technology).

Bosch Sensortec products are released on the basis of the legal and normative requirements relevant to the Bosch Sensortec product for use in the following geographical target market: BE, BG, DK, DE, EE, FI, FR, GR, IE, IT, HR, LV, LT, LU, MT, NL, AT, PL, PT, RO, SE, SK, SI, ES, CZ, HU, CY, US, CN, JP, KR, TW. If you need further information or have further requirements, please contact your local sales contact.

The resale and/or use of Bosch Sensortec products are at the purchaser's own risk and his own responsibility. The examination of fitness for the intended use is the sole responsibility of the purchaser.

The purchaser shall indemnify Bosch Sensortec from all third party claims arising from any product use not covered by the parameters of this product data sheet or not approved by Bosch Sensortec and reimburse Bosch Sensortec for all costs in connection with such claims.

The purchaser accepts the responsibility to monitor the market for the purchased products, particularly with regard to product safety, and to inform Bosch Sensortec without delay of all safety-critical incidents.

## 24 Application examples and hints

With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Bosch Sensortec hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights or copyrights of any third party. The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. They are provided for illustrative purposes only and no evaluation regarding infringement of intellectual property rights or copyrights or regarding functionality, performance or error has been made.

## 25 References

- Reference 1: UM10204 I2C-Bus Specifications and User Manual , Rev 6 (April 2014)  
<https://www.nxp.com/docs/en/user-guide/UM10204.pdf>
- Reference 2: <https://www.highintegratingsystems.com/openrtos/>
- Reference 3: BHI260AB-BHA260AB Programmer's Manual (BST-BHI260AB-AN002)
- Reference 4: ARC v2 Programmer's Reference Manual from Synopsys
- Reference 5: IPC/JEDEC J-STD-020E "Joint Industry Standard: Moisture/Reflow Sensitivity Classification for non-hermetic Solid-State Surface Mount Devices"
- Reference 6: IPC/JEDEC J-STD-033A "Joint Industry Standard: Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices"
- Reference 7: BHI260AB-BHA260AB SDK Quick Start Guide (BST-BHI260AB-AN000)
- Reference 8: BHI260AB-BHA260AB Evaluation Setup Guide (BST-BHI260AB-AN001)
- Reference 9: MSS – Handling, soldering & mounting instructions (BST-MSS-HS-000)
- Reference 10: BHI260: Shipment packaging details (BST-BHI260AB-SP000)
- )

# DOCUMENT HISTORY AND MODIFICATION

## 26 Document History

Table 97: Document History

Rev. No	Section	Description of modification/changes	Date
0.6	All	Preliminary Release	7.08.19
1.0	All	Main release	4.02.20
1.4	All	Minor typos in chapters: 5, 6, 12, 21, 22	20.02.20
1.5	All	Fixed typos and updated Table 79	21.04.20
1.7	22,23,24	New disclaimer	24.11.20



Bosch Sensortec GmbH  
Gerhard-Kindler-Straße 9  
72770 Reutlingen / Germany

[www.bosch-sensortec.com](http://www.bosch-sensortec.com)

Modifications reserved  
Preliminary - specifications subject to change without  
notice  
Document number: BST-BHA260AB-DS000-07  
Revision: 1.7