# Ultra Low Power Sensor Fusion Platform Data Sheet

## Description

The Sensor Fusion Platform (SFP) is small form factor integrated module containing a EM7180 *SENtral* Sensor Fusion Coprocessor, BMX055 9-degree-of-freedom (9-DOF) system in a package (3-axis gyroscope/accelerometer/ magnetometer) and a ST24256 32KB EEPROM containing the module firmware.

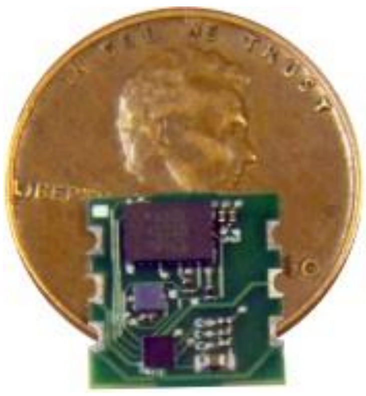
The *SENtral* Sensor Fusion Coprocessor fully controls and processes data from BMX055 9-DOF sensor. The primary data output from SFP are quaternions, which uniquely define device orientation, or Euler angles (heading, pitch, and roll). The quaternions easily can be also converted to the rotation vector, and the rotation matrix. Raw or calibrated sensor data are also provided to external Host which can control individual sensor rates and power states of the platform. External Host CPU can communicate with the SFP over high-speed I2C bus and obtain both fusion result and raw sensor data.

## Applications

- Robotics, Automation
- Sports activity (e.g. golf-swing)
- Motion-tracking-system
- Navigation systems
- Activity trackers
- Orientation-estimation (e.g. for binoculars)
- Air (remote) pointing devices
- Position identification (window handle)

## Features

- EM7180 *SENtral* state of the art sensor fusion coprocessor
- Ultra low power consumption: standby and still states reduce system power consumption dramatically
- VDD from 2.4V to 3.3V
- Industry leading heading and tracking accuracy
- 9 axis sensing with Bosch BMX055: 3-axis gyroscope, 3-axis accelerometer and 3-axis magnetometer
- Variety of outputs for accelerated application development
  - Quaternions
  - Heading, Pitch, Roll
  - Calibrated and raw sensor data
- Small Form Factor module - 10.16mm x 10.28mm
- I2C interface – 100 to 3400kbps - Standard, Fast, Fast Plus, and High Speed modes
- Host CPU driver source code available



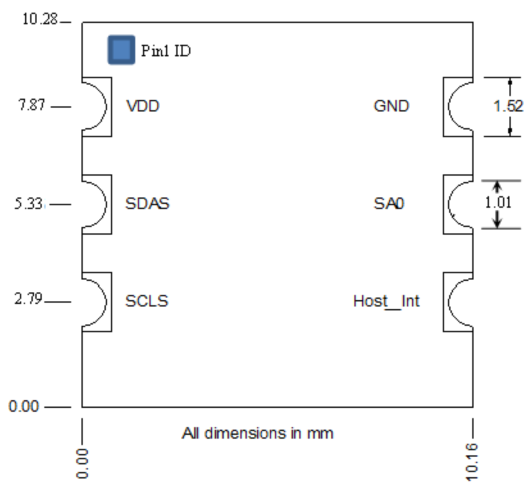


Fig. 1: *Actual Size*

## 1. Main characteristics

### 1.1. Pin Assignment

| Pin# | Pin Name | I/O Type (note 1) | Description |
|---|---|---|---|
| 1 | $V_{DD}$ | PWR | Supply voltage |
| 2 | $SDA_S$ | OD | $I^2C$ host bus SDA data line **(note 2)** |
| 3 | $SCL_S$ | OD | $I^2C$ host bus SCL clock line **(note 2)** |
| 4 | Host_Int | O | Interrupt to Host MCU **(note 3)** |
| 5 | $SA_0$ | I | I2C device address bit 0 **(note 3)** |
| 6 | GND | PWR | Ground connection |

*Table 1 SFP Pin Assignments*

**Note 1:** I/O Types are:

| | |
|---|---|
| PWR: | Power supply Connections |
| I: | Digital Input |
| O: | Digital Output |
| OD: | Open Drain – External Pull-up is required |

**Note 2:** Note that pull-ups on I2C lines (SDAs/SCLs) are not included on the module and need to be provided externally. See Section *"I2C Pull-Up Resistance"* for more information on the pull-up resistor sizes.

**Note 3:** Host_int signal should be connected to GPIO of Host MCU which can be used as an interrupt source. SA0 signal can be used to resolve unlikely I2C address alias problem on the Host I2C bus.

**Note 4:** In most applications SA0 can be connected to ground (see Section *"Host I2C Interface (Host Bus)"* for more details).

### 1.2. Block Diagram

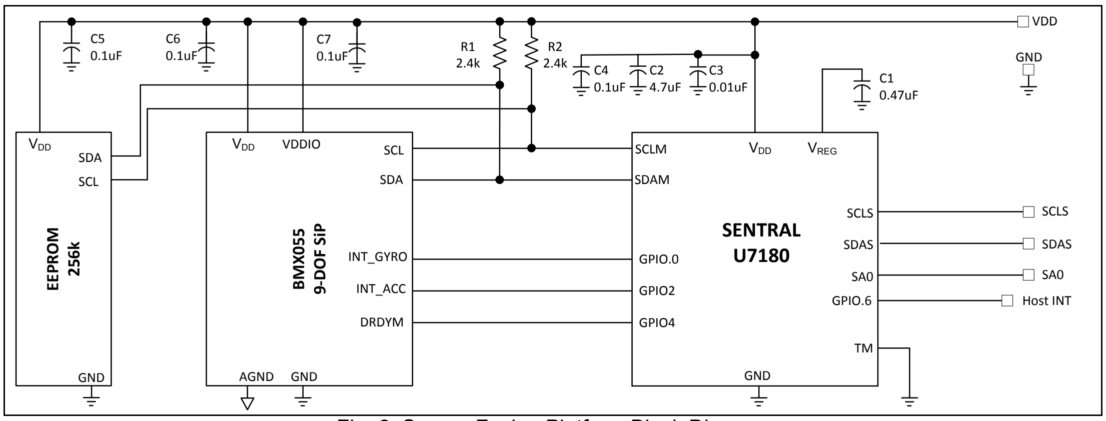


Fig. 2 Sensor Fusion Platform Block Diagram

### 1.3. Sensing axis orientations

The orientation of the module is defined by the BMX055 chip orientation on the board as shown in Fig. 3.

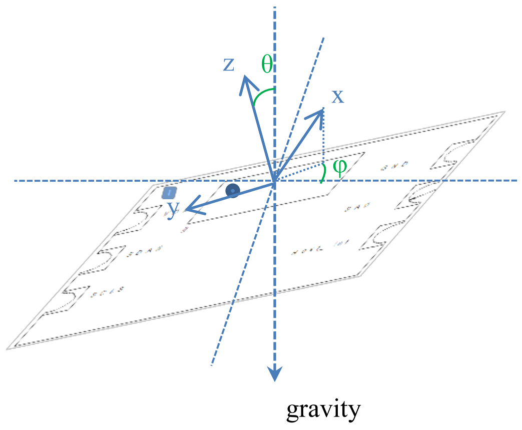


Fig. 3 Orientation of the module

### 1.4. Absolute Maximum Ratings

| Parameter | Symbol | Conditions |
|---|---|---|
| Voltage at $V_{DD}$ to GND | $V_{DD}$ | -0.3V to +3.6V |
| Minimum voltage at reset | $V_{MIN}$ | -0.6V |
| Maximum voltage at reset | $V_{MAX}$ | 3.6V |
| Storage Temperature Range | $T_{STG}$ | -50°C to +150°C |
| ESD HBM versus GND/VDD/ pin to pin | $V_{ESD}$ | +/-2000V |

*Table 2 Absolute Maximum Ratings*

Stresses above these listed maximum ratings may cause permanent damage to the device, operation cannot be guaranteed. Exposure beyond these absolute maximum ratings may affect device reliability or cause malfunction.

### 1.5. Handling Procedures

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

### 1.6. Operating Conditions

| Parameter | Symbol | Min | Max | Unit |
|---|---|---|---|---|
| Supply voltage | $V_{DD}$ | 2.4 | 3.3 | V |
| Operating Temperature | $T_A$ | -40 | +85 | °C |

*Table 3 Operating conditions*

### 1.7. Electrical Characteristics

Unless otherwise specified: $V_{DD}$= 2.4V to 3.3V, $T_A$=-40 to +85°C.

| Parameter | | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|---|
| High Level Input Voltage | | VIH | | 0.7*VDD | | VDD | VDC |
| Low Level Input Voltage | | VIL | | 0 | | 0.3*VDD | VDC |
| High Level Output Current | | IOH | VOH = VDD – 0.3V | | | -1 | mA |
| Low Level Output Current | | IOL | VOL = 0.3V | 1 | | | mA |
| Supply current | Moving (note 1) | IDDM | VDD=2.4V | | 7.9 | | mA |
| | Still (note 2) | IDDST | VDD=2.4V | | 300 | | µA |
| | Standby (note 3) | IDDSTD | VDD=2.4V | | 230 | | µA |
| I2C Interface Data Rate (note 4) | Host Bus | | | | | 3400 | kbits/sec |
| | Sensor Bus | | | | | 1000 | kbits/sec |
| | Pass-Through | | | | | 400 | kbits/sec |
| Decoupling Capacitor | | Creg | (ESR <2Ω) | 0.33 | 0.5 | 1.8 | µF |

*Table 4 Electrical characteristics*

**Note 1**: Sensor bus in I2C Fast mode @ 400 kbits/sec, accelerometer output data rate (ODR) = 100 Hz, gyroscope ODR=190 Hz, and magnetometer ODR=30 Hz. Operating current consumption is the average over 30 seconds while the device is in motion.

**Note 2:** SFP is in low power still state. In this state only the accelerometer is running and provides wakeup interrupt to *SENtral* when motion/shock is sensed. *SENtral* automatically transitions into full operational state after wakeup.

**Note 3:** Standby state is default state after power-up or when SFP operations are suspended.

**Note 4:** Sensor Bus is the communication bus between SENtral and sensors
Host bus is the communication bus between Host and SENtral
In Pass through state, the host communicates directly with sensors
See section I2C Pull-Up Resistance for a detailed description of the configuration taking into account pull-up resistance, capacitance and targeted Data rate

## 2. Modes of Operation

### 2.1. Initialization

Prior to entering Normal Operation *SENtral* must upload the Configuration File from on-board EEPROM into its Configuration RAM. This file contains information regarding how the user's sensor system is configured, such as sensor models, sensor slave addresses, GPIO pin assignments, etc. The Configuration File is pre-programmed into SFP EEPROM upon delivery, but updates might be available as discussed in "*Updating the on-board EEPROM*". Fig. *4* provides a flow chart of the initialization process, and a detailed discussion of the initialization process follows in "*Power-Up and Configuration File Upload*"

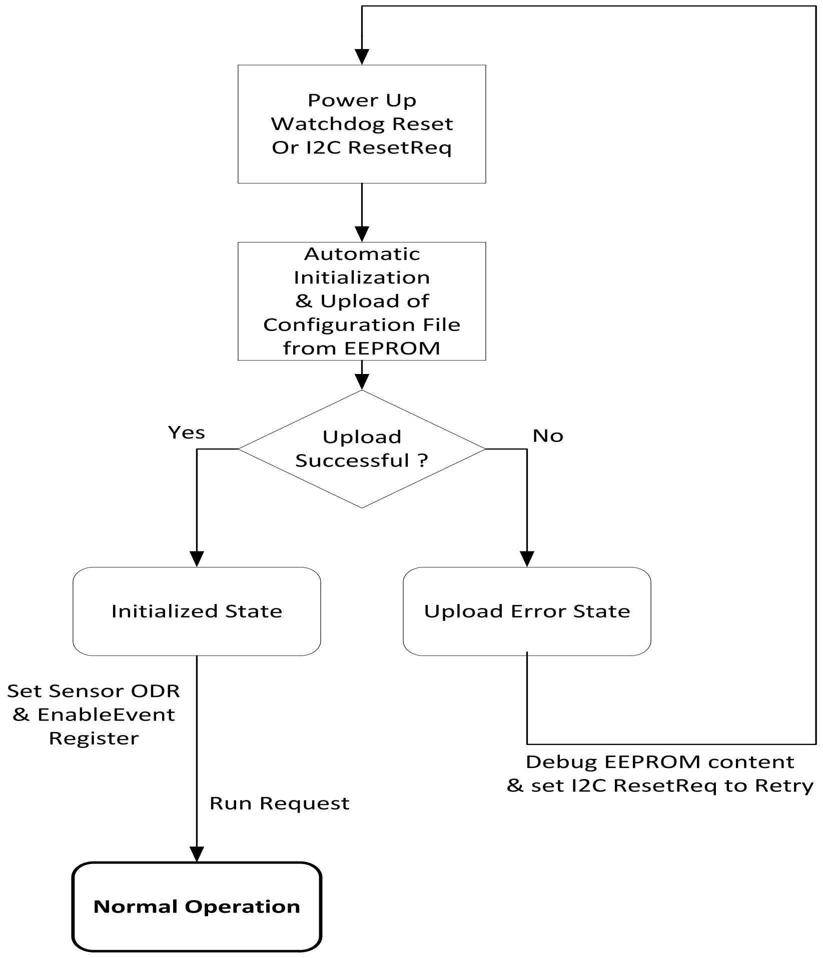

Power Up
Watchdog Reset
Or I2C ResetReq
Automatic Initialization & Upload of Configuration File from EEPROM
Upload Successful ?
Yes
No
Initialized State
Upload Error State
Set Sensor ODR & EnableEvent Register
Run Request
Debug EEPROM content & set I2C ResetReq to Retry
Normal Operation

*Fig. 4 SFP Initialization Sequence*

Once the initialization sequence is complete, there are three states in which *SENtral* may reside: Normal Operation, Standby, and Pass-Through. Fig. *5* indicates the recommended way to get from one state to another, and these states are discussed in detail in Sections *"Normal Operation" "Standby State", and "Pass-Through State*"

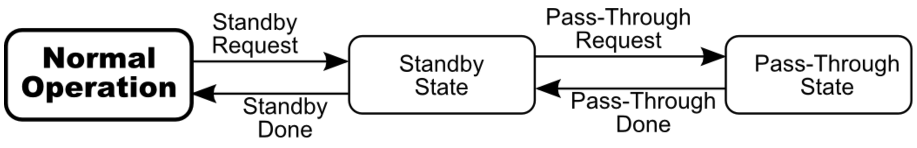


*Fig. 5 SENtral Operational States*

## 2.2. Power-Up and Configuration File Upload

After powering up or issuing a ResetReq command, *SENtral* automatically initializes the registers, and then uploads Configuration file from an EEPROM on the sensor bus, as indicated in Fig. *4*. *SENtral* sets Host_int interrupt pin high when the upload is completed (signal is cleared after enabling Normal Operation or setting ResetReq).

Host MCU should check result of EEPROM upload by reading *Sentral*Status register (see Table 5). Correct configuration file upload completion is indicated by value 0x0B in *Sentral*Status register (EEPROM detected, EEUploadDone and Idle bits are set). If the upload was successful the Host can enter Normal Operation (as described in section *"Normal Operation"*).

| Register Name | Address | Register Value |
|---|---|---|
| *Sentral*Status | 0x37 | [0] EEPROM. 1 = EEPROM detected<br>[1] EEUploadDone. 1 = EEPROM upload completed<br>[2] EEUploadError. 1 = Calculated CRC of EEPROM is correct. Only valid when EEUploadDone = 1.<br>[3] Idle. 1 = Device in Upload Error or Initialized state.<br>[4] NoEEPROM. 1 = No EEPROM detected. |
| ResetReq | 0x9B | [0] ResetRequest. 1 = Emulate a hard power down/power up. |

*Table 5 Configuration File Upload from EEPROM Registers*

Error during upload is indicated either by value 0x18 (No EEPROM and Idle bits are set) or value 0x0D (EEPROM detected, EEUploadError and Idle bits are set).

If the Configuration File upload failed, try the following:

- Send a Reset command by writing 0x01 to the ResetReq register. This would retry the initialization process.
- Download the Configuration File from the EEPROM and verify its integrity, as discussed in Section *"Updating the on-board EEPROM"*
- Reload the Configuration File from the host into the EEPROM.

## 2.3. Normal Operation

During Normal Operation the sensors and *SENtral* algorithm will run and it will be possible to obtain real-time orientation and motion-tracking data from *SENtral*. The registers used in Normal Operation are given in Table 6.

| Register Name | Address | Register Value |
|---|---|---|
| HostControl | 0x34 | [0] 1 = Run Request<br>0 = Enable Initialized State |
| MagRate | 0x55 | Requested magnetometer output data rate |
| AccelRate | 0x56 | Requested accelerometer output data rate divided by 10 |
| GyroRate | 0x57 | Requested gyroscope output data rate divided by 10 |
| ActualMagRate | 0x45 | Actual magnetometer output data rate |
| ActualAccelRate | 0x46 | Actual accelerometer output data rate divided by 10 |
| ActualGyroRate | 0x47 | Actual gyroscope output data rate divided by 10 |
| QRateDivisor | 0x32 | Along with GyroRate, establishes output data rate for quaternion data. |
| EnableEvents | 0x33 | ‘1’ indicates an interrupt to the host will be generated for the event.<br>[0] CPUReset. Non-maskable<br>[1] Error<br>[2] QuaternionResult<br>[3] MagResult<br>[4] AccelResult<br>[5] GyroResult<br>[6] Reserved<br>[7] Reserved |
| EventStatus | 0x35 | ‘1’ indicates a new event has been generated.<br>[0] CPUReset<br>[1] Error<br>[2] QuaternionResult<br>[3] MagResult<br>[4] AccelResult<br>[5] GyroResult |

*Table 6 Normal Operation Registers*

Prior to entering the Normal Operation state it is necessary to perform the following:

- Set the sensor output data rates (ODRs): MagRate, AccelRate, and GyroRate. If a sensor rate is set to 0x00, *SENtral* will shutdown the sensor and disable *SENtral* background calibration. There are two major points regarding setting these registers:
  - The AccelRate and GyroRate register values should be 1/10th the desired rate, while the MagRate value should match the desired ODR. For example, if the desired ODR is 30 Hz for the magnetometer, 100 Hz for the accelerometer, and 200 Hz for the gyroscope, then the respective register values should be 0x1E (30d), 0x0A (10d), and 0x14 (20d).
  - The actual accelerometer and gyro ODRs are limited to the ODRs supported by the specific sensors (in this case BMX055). If the AccelRate or GyroRate register values do not correspond to a supported ODR, then the next highest ODR will be used. For instance, if the GyroRate register is set to 0x14, which corresponds to 200 Hz, but the gyro supports 95 Hz, 190 Hz, and 380 Hz, then the actual gyro ODR will be 380 Hz since this is the closest supported rate above that requested by the register. During Normal Operation, the actual sensor rates can be read from ActualMagRate, ActualAccelRate and ActualGyroRate registers.
- Establish the quaternion output data rate, where the quaternion output data rate equals GyroRate divided by QRateDivisor. The default for QRateDivisor is 0x00, which is interpreted as ‘1’ and results in the quaternion output data rate equalling GyroRate.
- Establish which events will trigger an interrupt to the host by configuring the EnableEvent register. It is specifically recommended to enable bit [1], the Error interrupt bit, in addition to whichever other interrupts the user wants.

- Once setting of ODR and EventEnable register is completed, Host MCU can enable Normal Operation by setting bit Run Request in HostControl register (writing 0x01 into the register). This would also clear initial Host_int (set after completion of configuration file upload).

Example steps to enter Normal Operation are given below:

- Write 0x1E0A0F to the MagRate register. Since *SENtral* automatically increments to the next register, this also populates the AccelRate and GyroRate registers. This sets MagRate to 30 Hz, AccelRate to 100 Hz, and GyroRate to 150 Hz.
- Write 0x01 to the QRateDivisor Register. This sets the quaternion output data rate to equal the GyroRate. For writing 0x01 this step is optional, since the default also sets the quaternion output data rate equal to GyroRate.
- Write 0x07 to the EnableEvents register. This sets up the host to receive interrupts from *SENtral* whenever the quaternion results registers are updated, an error has been detected, or when *SENtral* needs to be reset.

After performing the steps listed above, *SENtral* is ready to start generating orientation data. Below are the steps to follow when operating in Normal Operation state. Write 0x01 to the HostControl register. This sets the RunRequest bit to '1' and enables the sensors and the *SENtral* algorithm. Read the ActualMagRate, ActualAccelRate, and ActualGyroRate registers to ensure the ODRs are as expected. If operating in an interrupt-driven mode, then the Host MCU waits until it receives an interrupt signal from *SENtral*. Alternatively the host may operate on a polling basis, rather than an interrupt-driven basis, in which case the interrupt line may not be used. Once an interrupt is received by the Host or the Host otherwise decides to read new data, read the EventStatus register.

a) Interpret and act on the EventStatus register in the priority shown in Fig. *6*. If bit [1], the Error bit, is '1' or if bit [0], the CPUReset bit, is '1', see *"Section Error and CPUReset"*. If bits [2], [3], [4], or [5], the Results bits, are '1', read the desired data (see Section "*Read Results*").

b) Repeat steps c – e until new orientation data is not needed and/or the host decides to enter a different state.

**Reading the EventStatus register clears it**. It is possible for more than one bit position to be '1' in the EventStatus register, especially if the host does not always read the EventStatus register after receiving an interrupt. Similarly, if multiple bits are set to '1' in the EventStatus register, once the register is read all the bits will be set to '0'. For this reason the EventStatus register should be processed in the priority shown in Fig. *6*, as information will be cleared for events that are not handled.

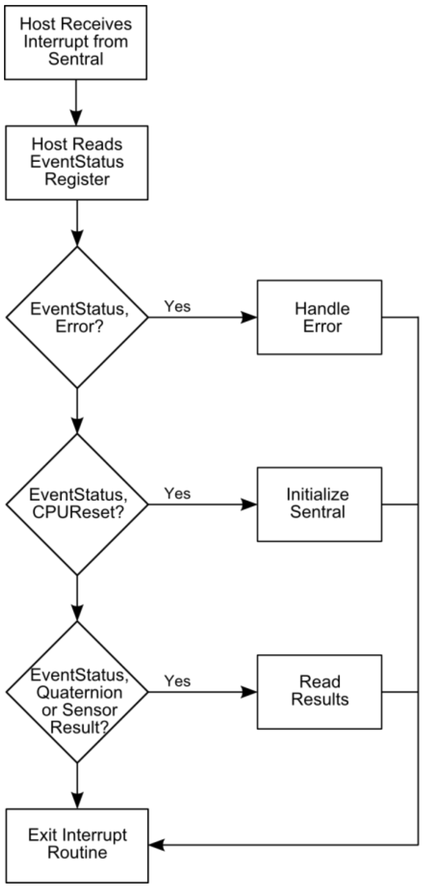


*Fig.* 6 *SENtral Normal Operation Flow*

A discussion of how to handle the various events follows.

- ***Error and CPUReset***

In the event of an error, *SENtral* will trigger an error interrupt and *SENtral* will enter Standby State. When CPUReset bit is seen it also indicates potential problem (watchdog reset occurred) See the Section "Updating the on-board EEPROM" for recommendations on Troubleshooting and/or reset *SENtral* by sending 0x01 to the ResetReq register, at address 0x9B.

- ***Read Results***

The Results Registers' addresses, formats, and full-scale ranges are given below in Table 7. *SENtral* allows switching between Quaternions and Heading/Pitch/Roll (Euler Angles) output and between calibrated and raw sensor data using AlgorithmControl register (see Table 8). For an explanation of how to convert the quaternions to the rotation vector, or the rotation matrix, see corresponding application note. The resolution is 32 kHz for all timestamps.

***Note:*** *All multi-byte elements are stored and transmitted using the Little Endian convention: the least significant byte is stored at the lowest address and transmitted first over the I$^2$C bus.*

| Name | Address (Hex) | Description | Format | Full-Scale Range |
|---|---|---|---|---|
| QX | 00 – 03 | Normalized Quaternion – X, or Heading | Float32 | 0.0 – 1.0, or ±π |
| QY | 04 – 07 | Normalized Quaternion – Y, or Pitch | Float32 | 0.0 – 1.0, or ±π/2 |
| QZ | 08 – 0B | Normalized Quaternion – Z, or Roll | Float32 | 0.0 – 1.0 or ±π |
| QW | 0C – 0F | Normalized Quaternion – W, or 0.0 | Float32 | 0.0 – 1.0 |
| QTime | 10 – 11 | Quaternion Data Timestamp | UInt16 | 0 – 2048 msec |
| MX | 12 – 13 | Magnetic Field – X Axis | Int16 | ±1000 µT |
| MY | 14 – 15 | Magnetic Field – Y Axis | Int16 | ±1000 µT |
| MZ | 16 – 17 | Magnetic Field – Z Axis | Int16 | ±1000 µT |
| MTime | 18 – 19 | Magnetometer Interrupt Timestamp | UInt16 | 0 – 2048 msec |
| AX | 1A – 1B | Linear Acceleration – X Axis | Int16 | ±16 g |
| AY | 1C – 1D | Linear Acceleration – Y Axis | Int16 | ±16 g |
| AZ | 1E – 1F | Linear Acceleration – Z Axis | Int16 | ±16 g |
| ATime | 20 – 21 | Accelerometer Interrupt Timestamp | UInt16 | 0 – 2048 msec |
| GX | 22 – 23 | Rotational Velocity – X Axis | Int16 | ±5000°/sec |
| GY | 24 – 25 | Rotational Velocity – Y Axis | Int16 | ±5000°/sec |
| GZ | 26 – 27 | Rotational Velocity – Z Axis | Int16 | ±5000°/sec |
| GTime | 28 – 29 | Gyroscope Interrupt Timestamp | UInt16 | 0.0 – 2.048 sec |

*Table 7 Results Registers*

### 2.4. Standby State

In Standby State overall system power consumption is dramatically reduced because both the *SENtral* algorithm and the sensors are shut down. Table 8 provides the registers associated with Standby State.

| Register Name | Address | Register Value |
|---|---|---|
| AlgorithmControl | 0x54 | [2] 1 = HPR output<br>0 = Quaternion output (default)<br>[1] 1 = Raw Data Output<br>0 = Calibrated Data Output (default)<br>[0] 1 = StandbyEnable<br>0 = Disable Standby State (default) |
| AlgorithmStatus | 0x38 | [0] 1 = *SENtral* in Standby State<br>0 = *SENtral* not in Standby State |

*Table 8 Standby and Result Control Registers*

The steps to enter and exit Standby State are given below:

- Set bit [0] of the AlgorithmControl register. This places *SENtral* in Standby State.
- Read the AlgorithmStatus register. If bit [0] is ‘1’, then *SENtral* is in Standby State. This step is optional.
- When you are ready to exit Standby State, clear the bit [0] of the AlgorithmControl register. This takes *SENtral* out of Standby State and normally will place it back into Normal Operation.
- Read the AlgorithmStatus register. If bit [0] is ‘0’, then *SENtral* is not in Standby State. This step is optional.

### 2.5. Pass-Through State

*SENtral* can be configured so the host communicates directly with devices on the sensor bus by placing *SENtral* into Pass-Through State. In Pass-Through State, *SENtral*’s sensor and host interfaces are connected by internal switches so the host system communicates directly with the sensors and/or dedicated EEPROM. To enter Pass-Through State, *SENtral* first should be in either standby, initialized, or unprogrammed State. Consequently, in Pass-Through State the *SENtral* algorithm, host interrupt line, and sensors are disabled, unless a sensor is directly turned on by the host. When exiting Pass-Through State, *SENtral* will return to its prior state.

***Note:*** *When entering Pass-Through State the sensor’s registers retain the values established by SENtral, and when exiting Pass-Through State any register changes will be retained.*

Uses for the Pass-Through State include:

- Direct control of BMX055 sensors, if desired.
- Debugging.
- Communication with the on-board EEPROM. Specifically, if a new Configuration File is generated, the host can write this into the EEPROM when in Pass-Through State.

Since operating in Pass-Through State requires stopping the *SENtral* algorithm, Pass-Through State is not recommended for accessing sensor data unless reliable heading data is not required. If sensor data and reliable heading data are both desired, scaled sensor data can be accessed during Normal Operation from the Results Registers, as given in Table 7.

Table 9 provides the registers associated with Pass-Through State.

| Register Name | Address | Register Value |
|---|---|---|
| AlgorithmStatus | 0x38 | [0] 1 = *SENtral* in Standby State<br>0 = *SENtral* not in Standby State |
| PassThroughControl | 0xA0 | [0] 1 = Enable Pass-Through State<br>0 = Disable Pass-Through State |
| PassThroughStatus | 0x9E | [0] 1 = *SENtral* in Pass-Through State.<br>0 = *SENtral* not in Pass-Through State. |

*Table 9 Pass-Through Registers*

The steps to go in and out of Pass-Through State are given below.

- Enter Standby State by setting of AlgorithmControl register (see section *"Standby State"*) Pass-Through can be also entered from Initialized or Upload Error State (see section *"Power-Up and Configuration File Upload"*)
- Write 0x01 to the PassThroughControl register. This places *SENtral* in Pass-Through State.
- Read the PassThroughStatus register. If bit [0] is '1', then *SENtral* is in Pass-Through State. This step is optional.
- When you are done in Pass-Through State, write 0x00 to the PassThroughControl register. This terminates Pass-Through mode and returns *SENtral* to Standby State.
- Restore setting of the AlgorithmControl register. This takes *SENtral* out of Standby State and normally will place it back into Normal Operation.

## 2.6. Updating the on-board EEPROM

This section describes method to check or update Configuration File stored in on-board EEPROM device.

*Checking Configuration File Revision*

SFP module is always delivered with a stable and tested version of the Configuration File revision. However, updated Configuration File might be available on *SENtral* product website (please ask your contact person for more detail).

| Register Name | Register Address | Register Value |
|---|---|---|
| Product ID | 0x90 | 0x80 |
| Revision ID | 0x91 | Hardware Revision |
| ROM Version | 0x70 - 0x71 | ROM Revision |
| RAM Version | 0x72 - 0x73 | RAM Version – Configuration File Version |

*Table 10 Revision Registers*

RAM Version register can be read during Normal Operation of the device. It contains a 16bit Configuration File Version which can be compared with SFP Configuration File versions which are available for download from the *SENtral* product website.

*Uploading Configuration File into EEPROM*

SFP board contains a ST24256 256Kbit (32KB) serial EEPROM. Its I2C device address is 0xA0. The user is encouraged to refer to the ST24256 datasheet for details about correct programing sequence and timing.

The steps to program EEPROM are given bellow.

- Enter Pass-Through State (as described in the related Section "*Pass-Through State*"). Make sure that *SENtral* is not in Normal Operation (it can be either in Initialized or Standby State)
- Copy the content of Configuration File into EEPROM, starting with EEPROM byte address 0x0000. Best efficiency is achieved by utilizing page write mode (up to 64B at a time). Please refer to ST24256 datasheet for details about correct timing between writes.
- Verify the programing success by read back of the EEPROM content.

- Leave Pass-Through State and issue I2C ResetReq (Write 0x01 into ResetReq register). This will force *SENtral* to reinitialize from EEPROM (see Section *"Power-Up and Configuration File Upload"*)

## 3. Communication

### 3.1. Timing Characteristics

Communication between the Host MCU and SFP utilizes I2C bus and Host_int interrupt line. The Host_int interrupt line lets the Host MCU know when *SENtral* has updated the quaternions or when new raw sensor data are available. Conditions for Host_int interrupt are programmable.

SFP I2C interface complies with NXP's UM10204 specification and user manual, rev 04. Standard, Fast, Fast Plus, and High Speed modes of the I2C protocol are supported by SFP I2C host interface. Host MCU does not need to support clock-stretching when connected to SFP. Below is a link to this document.

http://www.nxp.com/documents/user_manual/UM10204.pdf

| Parameter | Symbol | Standard | | Fast | | FastPlus | | Unit |
|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | Min | Max | |
| SCL Clock | $f_{SCL}$ | 0 | 100 | 0 | 400 | 0 | 1000 | kHz |
| SDA & SCL Rise Time | $t_r$ | - | 1000 | 20 | 300 | | 120 | ns |
| SDA & SCL Fall Time | $t_f$ | - | 300 | 20*(VDD/5.5V) | 300 | 20*(VDD/5.5V) | 120 | ns |
| LOW period of SCL Clock | $t_{LOW}$ | 4.7 | - | 1.3 | - | 0.5 | - | µs |
| HIGH period of SCL Clock | $t_{HIGH}$ | 4.0 | - | 0.6 | - | 0.26 | - | µs |
| Hold time (repeated) START | $t_{HD;STA}$ | 4.0 | - | 0.6 | - | 0.26 | - | µs |
| Data hold time | $t_{HD;DAT}$ | 0 | - | 0 | - | 0 | - | µs |
| Data set-up time | $t_{SU:DAT}$ | 250 | - | 100 | - | 50 | - | ns |
| Set-Up time for repeated Start | $t_{SU;STA}$ | 4.7 | - | 0.6 | - | 0.26 | - | µs |
| Stop set-up time | $t_{SU;STO}$ | 4.0 | - | 0.6 | - | 0.26 | - | µs |
| Bus free time between STOP & START | $t_{BUF}$ | 4.7 | - | 1.3 | - | 0.5 | - | µs |

*Table 11 Timing characteristics*

### 3.2. Timing Waveforms

SFP I2C timing requirements are set forth below, in Fig. 7. For the timing requirements shown in Fig. 7, transitions are 30% and 70% of VDD.

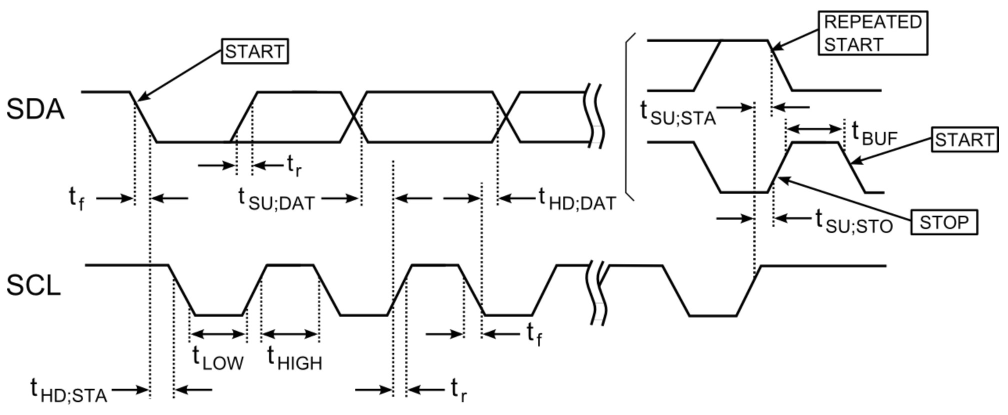


*Fig. 7 I2C Timing Diagram*

### 3.3. Host I²C Interface (Host Bus)

The host will control SFP on the host bus via *SENtral*'s I2C host interface. The host interface consists of 2 wires: the serial clock, SCLS, and the serial data line, SDAS. Both lines are bi-directional. *SENtral* is connected to the host bus via the SDAS and SCLS pins, which incorporate open drain drivers within the device. The host bus lines must be externally connected to a positive supply voltage (Vdd) via a pull-up resistor. See Section *"I2C Pull-Up Resistance"* for more on the pull-up resistor.

*SENtral*'s 7-bit I2C slave address is 0b010100x, where the most significant 6 bits of the slave address are pre-defined in hardware and are the same for all *SENtral* devices. The least significant bit is user-configurable, using the SA0 pin to set the bit to '0' or '1'. For example, grounding the SA0 pin ('0' value) results in the 7-bit address of 0b0101000. This should be set so the *SENtral* slave address is unique to any other devices on the host bus.

Data transfer is always initiated by the host. Data is transferred between the host and *SENtral* serially through the data line (SDAS) in an 8-bit transfer format. The transfer is synchronized by the serial clock line, SCLS. Supported transfer formats are single-byte read, multiple-byte read, single-byte write, and multiple-byte write. *SENtral* as part of SFP is not utilizing clock stretching feature of I2C bus and SCLS is driven by the Host MCU only.

**Host I²C Transfer formats**

Fig. *8* illustrates writing data to registers in single-byte or multiple-byte mode.

| START | SLAVE ADDRESS | | | | | | | RW | ACK | REGISTER ADDRESS (N) | | | | | | | | ACK | DATA TO REGISTER (N) | | | | | | | | ACK | DATA TO REGISTER (N+1) | | | | | | | | ACK | STOP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 0 | 0 | R7 | R6 | R5 | R4 | R3 | R2 | R1 | R0 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 0 | P |

▨ From Host to *SENtral*

□ From *SENtral* to Host

------------ Data Transferred (n bytes + acknowledge) ------------

*Fig. 8 I2C Slave Write Example*

The I2C host interface supports both a read sequence using repeated START conditions, shown in Fig. 9, and a sequence in which the register address is sent in a separate sequence than the data, shown in Fig. 10 and Fig. *11*.

| START | SLAVE ADDRESS | | | | | | | RW | ACK | REGISTER ADDRESS (N) | | | | | | | | ACK | START | SLAVE ADDRESS | | | | | | | RW | ACK | DATA FROM REGISTER (N) | | | | | | | | NACK | STOP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 0 | 0 | R7 | R6 | R5 | R4 | R3 | R2 | R1 | R0 | 0 | SR | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 1 | P |

Data Transferred
(n bytes + acknowledge)

*Fig. 9 I2C Slave Read Example, with Repeated START*

| START | SLAVE ADDRESS | | | | | | | RW | ACK | REGISTER ADDRESS (N) | | | | | | | | ACK | STOP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 0 | 0 | R7 | R6 | R5 | R4 | R3 | R2 | R1 | R0 | 0 | P |

*Fig. 10 I2C Slave Write Register Address Only*

| START | SLAVE ADDRESS | | | | | | | RW | ACK | DATA FROM REG. (N) | | | | | | | | ACK | DATA FROM REG. (N+1) | | | | | | | | NACK | STOP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 1 | P |

▨ From Host to *SENtral*

□ From *SENtral* to Host

-------------- Data Transferred (n bytes + acknowledge) --------------

*Fig. 11 I2C Slave read register from current address*

## I2C Sensor Interface (Sensor Bus)

*SENtral* coprocessor communicates with BMX055 sensor and serial EEPROM over I2C sensor bus (internal to SFP). Host MCU can communicate directly with EEPROM and BMX055 sensor when the *SENtral* coprocessor is put into Pass-Through mode. Note that Pass-Through mode is supported when *SENtral* is in Standby state (normal operation is suspended). Communication speed in Pass-Through mode is limited to 400 kbit/s (Fast Mode).

## I2C Pull-Up Resistance

The pull-up resistor value for both the host and sensor bus will depend on the I2C data rate and the number of devices on the bus.

| **I2C mode** | | **Rate (kbit/s)** | **Rise Time (ns)** | **Max Cb(pF)** | |
|---|---|---|---|---|---|
| | | | | **4.7kΩ pull-up** | **2.47kΩ pull-up** |
| Standard | | 100 | 1000 | 251.1 | 491.8 |
| Fast | | 400 | 300 | 75.3 | 147.5 |
| Fast Plus | | 1000 | 120 | 30.1 | 59.0 |
| High Speed- 1.7 MHz | Clock | 1700 | 80 | 20.1 | 39.3 |
| | Data | 1700 | 160 | 40.2 | 78.7 |
| High Speed- 3.4 MHz | Clock | 3400 | 40 | 10.0 | 19.7 |
| | Data | 3400 | 80 | 20.1 | 39.3 |

Table 12 provides the maximum acceptable bus capacitance, as a function of bus rate, which can be accommodated with a 4.7 kΩ or 2.4 kΩ pull-up resistor. As a general rule, each device connected to the bus represents 10 pF of capacitance on the bus, so a bus with 4 devices would require a “Max Cb” value of >40 pF.

| **I2C mode** | | **Rate (kbit/s)** | **Rise Time (ns)** | **Max Cb(pF)** | |
|---|---|---|---|---|---|
| | | | | **4.7kΩ pull-up** | **2.47kΩ pull-up** |
| Standard | | 100 | 1000 | 251.1 | 491.8 |
| Fast | | 400 | 300 | 75.3 | 147.5 |
| Fast Plus | | 1000 | 120 | 30.1 | 59.0 |
| High Speed- 1.7 MHz | Clock | 1700 | 80 | 20.1 | 39.3 |
| | Data | 1700 | 160 | 40.2 | 78.7 |
| High Speed- 3.4 MHz | Clock | 3400 | 40 | 10.0 | 19.7 |
| | Data | 3400 | 80 | 20.1 | 39.3 |

*Table 12 I2C Pull-Up Resistance Table*

As the table implies, for most Standard and Fast Mode implementations a 4.7 kΩ pull-up should work well, while a 2.4 kΩ pull-up normally should be used for Fast Plus. See Section 7.1 of NXP’s UM10204 specification for additional information.

## 4. Troubleshooting

This section provides guidance in troubleshooting *SENtral*, and is divided into hardware-related and software-related errors.

### 4.1. Hardware-Related Error Conditions

Possible indications of a hardware-related problem are given below in Table 13.

| Register Name | Address | Error Indication |
|---|---|---|
| EventStatus | 0x35 | [0] 1 = CPUReset. Watchdog or Power on Reset occurred and EEPROM upload failed. See Section *“Power-Up and Configuration File Upload”*. |
| *Sentral*Status | 0x37 | [2] 1 = EEUploadError. Issue with uploading from the dedicated EEPROM.<br>[4] 1 = No EEPROM detected<br>See Section *“Power-Up and Configuration File Upload”*. |
| MagRate | 0x55 | 0x00 – Value lost |
| AccelRate | 0x56 | 0x00 – Value lost |
| GyroRate | 0x57 | 0x00 – Value lost |

*Table 13 Hardware-Related Error Indications*

In the event of such errors, *SENtral* will enter Standby State, shut down the sensors, and generate an interrupt to the host. Possible reasons for hardware-related errors include problems with an external EEPROM upload, power transients detected by power management, and errors in software detected by Watchdog. Often the error can be cleared by sending the ResetReq command and reloading the Configuration File.

### 4.2. Software-Related Error Conditions

Possible indications of software-related errors are given below in Table 14:

| Register Name | Address | Error Indication |
|---|---|---|
| EventStatus | 0x35 | [1] 1 = Error. |
| SensorStatus | 0x36 | Non-zero value indicates sensor-related error. Check sensors by communicating in Pass-Through State. See Table 15 |
| ErrorRegister | 0x50 | Non-zero value indicated an error. See Table 16. |

*Table 14 Software-Related Error Indications*

If the SensorStatus register indicates a non-zero value, then the value provides additional information on the sensor that is causing a problem, as given in Table 15.

| Register Name | Address | Error Indication |
|---|---|---|
| SensorStatus | 0x36 | [0] MagNACK. 1 = NACK from magnetometer<br>[1] AccelNACK. 1 = NACK from accelerometer<br>[2] GyroNACK. 1 = NACK from gyroscope<br>[4] MagDeviceIDErr. 1 = Unexpected DeviceID from magnetometer<br>[5] AccelDeviceIDErr. 1 = Unexpected DeviceID from accelerometer<br>[6] GyroDeviceIDErr. 1 = Unexpected DeviceID from gyroscope. |

*Table 15 SensorStatus Register Indications*

If the ErrorRegister indicates a non-zero value, then the value provides additional information on the sensor that is causing a problem, as given in Table 16.

| Value | Error Condition | Response |
|---|---|---|
| 0x00 | No error | |
| 0x80 | Invalid sample rate selected | Check sensor rate settings. |
| 0x30 | Mathematical Error | Check for software updates |
| 0x21 | Magnetometer initialization failed | This error can be caused by a wrong driver, physically bad sensor connection, or incorrect I$^2$C device address in the driver |
| 0x22 | Accelerometer initialization failed | |
| 0x24 | Gyroscope initialization failed | |
| 0x11 | Magnetometer rate failure | This error indicates the given sensor is unreliable and has stopped producing data. |
| 0x12 | Accelerometer rate failure | |
| 0x14 | Gyroscope rate failure | |

*Table 16 ErrorRegister Indications*

## 5. Typical Application Circuit

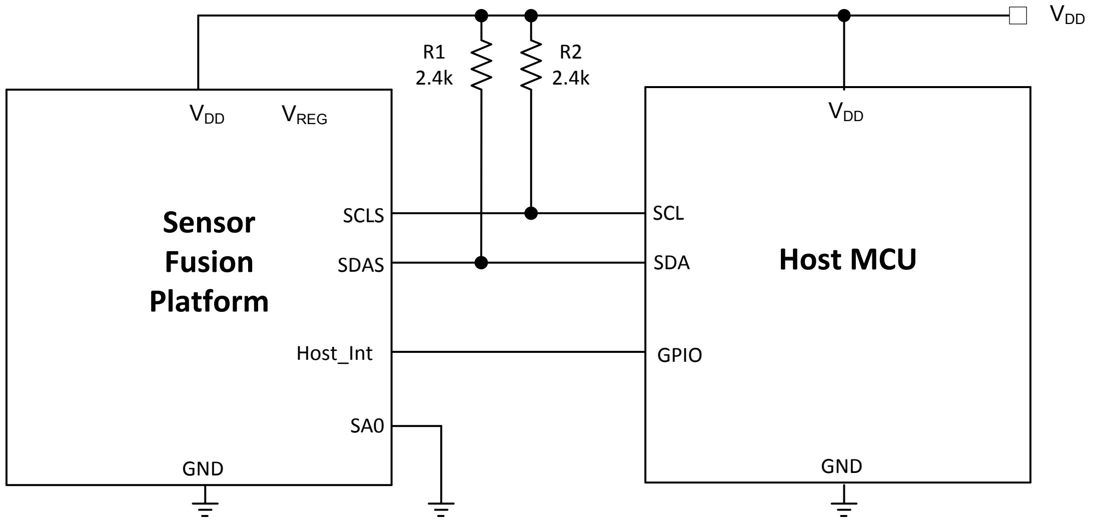


*Fig. 12 Sensor Fusion Platform Connection to Host MCU*

## 6. Package Information

### 6.1. Module dimensions

The 2 following figures describe the sensor fusion platform dimensions and recommended assembly footprint.
The Module thickness is max 2.1mm (at BMX055 sensor location).

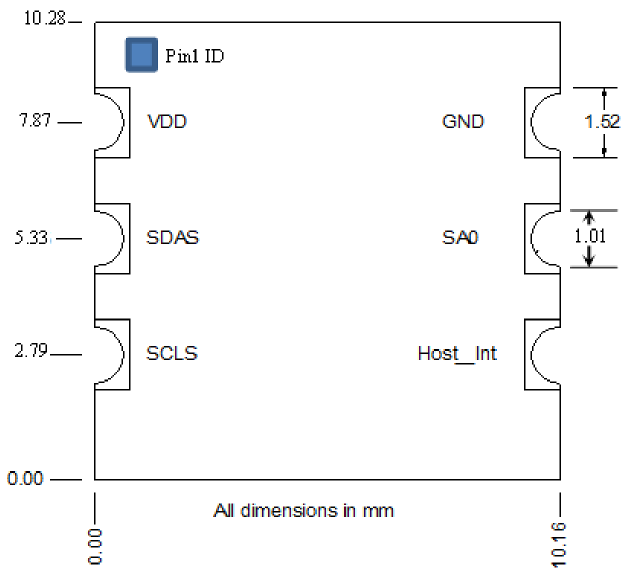


*Fig. 13 Reference outline dimensions (top view, component side)*

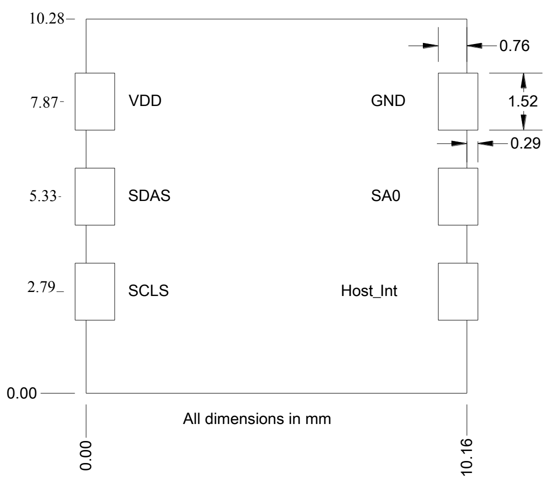


*Fig. 14 Landing pattern recommendation*

The SFP pins are organized on a standard 0.1" pitch which is compatible with many prototyping boards.

### 6.2. Soldering and usage information

The SFP module is rated MSL1 and complies with lead free reflow profile according to Jedec J-STD-020.

### 6.3. Module marking

As shown in Fig. 15, each module is marked with a lot number made of 5 alphanumeric characters as follows:

- YY= year of assembly
- M= Month of assembly
- AB= Lot number code

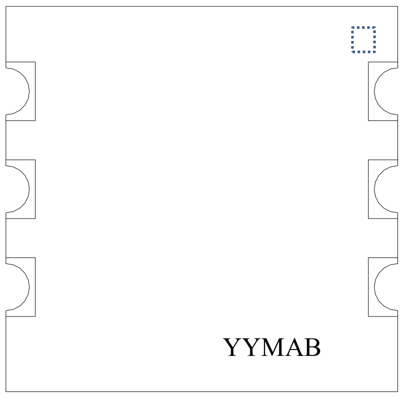
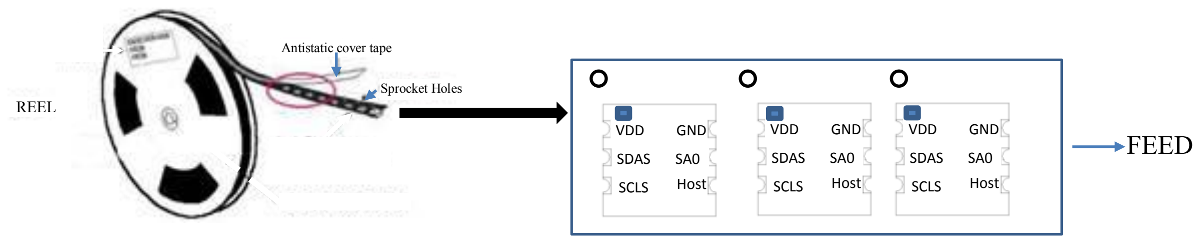


*Fig. 15 Bottom side view of the module*

## 7. Packing and labelling

The Sensor Fusion Platform modules are packed in Tape and Reel.

### 7.1. Module orientation in the carrier Tape



### 7.2. Reel information

| | |
|---|---|
| size of the reel | D330mm, W16mm, HUB 4” |
| quantity of unit per reel | 1000 pcs |

## 8. Ordering Information

The ordering code for the module is EM7180SFP01B