ICM-426xx APEX Motion Functions: Description and Usage

TDK Corporation
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: AN-000173
Revision: 1.0
Rev. Date: 12/20/2019

TABLE OF CONTENTS

Document Number: AN-000173
Revision: 1.0

1 APEX MOTION OVERVIEW

APEX (**A**dvanced **P**edometer and **E**vent Detection – Ne**X**t Generation) Motion is a digital block inside the ICM-426xx family of Inertial Measurement Units. APEX processes accelerometer data, extracts measurements, and asserts motion-specific interrupts. APEX features use accelerometer data, not gyroscope data.

Each APEX feature has its own set of configuration bits and the features can run concurrently. Most APEX features assert an interrupt after a motion event (Significant Motion Detection asserts an interrupt but does not provide any measurements), while other features will extract data and provide to the user (Pedometer counts steps and can also assert interrupts).

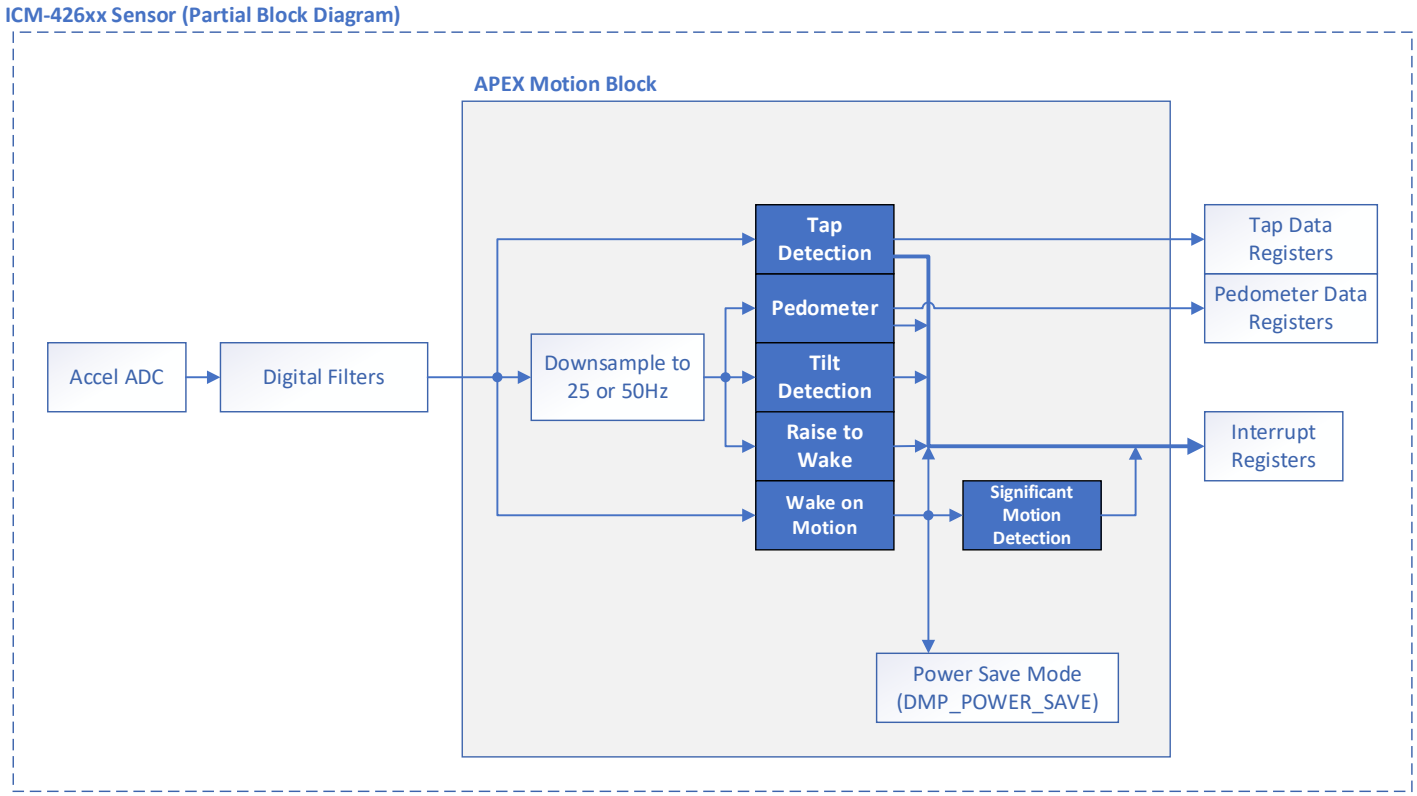

Figure 1. APEX simplified block diagram.

The **Pedometer** counts user steps, measures cadence, classifies if a user is walking or running, and issues an interrupt on a new step or step counter overflow. **Tilt Detection** issues an interrupt when it detects a >35° change in the direction of gravity. **Raise-To-Wake/Sleep (R2W)** issues an interrupt when the sensor is oriented to a face-up position (designed to identify the "raise" gesture of a user raising a mobile phone into view). **Tap detection** issues an interrupt when 1 or 2 pulses are observed on the accelerometer. **Wake-On-Motion (WoM)** issues an interrupt when the accelerometer change is greater than a programmable threshold, designed to alert the user upon transition from a stationary state to a moving state. **Significant Motion Detection (SMD)** asserts an interrupt when two sequential WoM events occur within a 1s or 3s window.

2 REGISTER CONFIGURATION REQUIREMENTS AND OPTIONS

APEX behavior is impacted (directly or indirectly) by several ICM-426xx register settings. Accelerometer configuration, digital filtering options, as well as software/hardware interrupt routing will change the conditions under which an interrupt is asserted.

ACCEL MODES: LOW NOISE (LN) VS LOW POWER (LP)

The ICM-426xx accelerometer functions in one of two modes: low noise mode ("LN mode") when ACCEL_MODE=0b11 or low power mode ("LP mode") when ACCEL_MODE=0b10. Low noise mode continuously samples at a fixed rate (either 8 kHz or 32 kHz depending on the model) and is decimated to the user's selected ODR (decimation controlled by ACCEL_UI_FILT_BW). In low power mode, the ADC sampling rate depends on the user's selected ODR.

LP Mode consumes less power because the ADC is duty cycled. LN Mode consumes constant power (because the ADC runs independently of ODR) and has lower noise because the output is decimated.

Accelerometer LP Mode is optional for ODR<=500 Hz, and accelerometer LN Mode is mandatory for ODR>=1 kHz.

ACCEL DIGITAL FILTERING OPTIONS

There are mutually exclusive filter paths for LN mode and LP mode. LN mode filtering options are shown in ICM-426xx datasheet Section 5: "Signal Path". LP mode has a single filter option which enables 16x averaging (see register GYRO_ACCEL_CONFIG0).

If a user deviates from the default filter settings, it may be possible to adversely impact APEX performance, particularly for Tap Detection where high frequency response is important[1].

OUTPUT DATA RATE (ODR) REQUIREMENTS

The 6 APEX features have different ODR requirements. Please see Table 1 below showing permissible ODRs (in green) for different features.

Sensor ODR	Tap Detection	Pedometer	Tilt Detection	Raise to Wake/Sleep (R2W)	Significant Motion Detection (SMD)	Wake on Motion (WoM)
< 25Hz	-	-	-	-		
25-100Hz	-					
200-1000Hz						
>1kHz	-	-	-	-	-	-

Table 1. APEX features and supported ODRs.

The user must keep in mind that **ODR=1kHz is supported in LN mode only** (ACCEL_MODE=0b11) and not in LP mode (ACCEL_MODE=0b10).

APEX DOWNSAMPLING OPTIONS (DMP_ODR)

There are two separate ODRs to configure when using Tilt/Pedometer/R2W. The first is the sensor's ODR (ACCEL_ODR), and the second is APEX's ODR (DMP_ODR). The accelerometer may sample at any ODR, but Tilt/Pedometer/R2W will only process data at 25 Hz or 50 Hz (controlled by DMP_ODR as indicated in Figure 2).

[1] For example: it is possible to program an improperly low bandwidth if the user sets the antialiasing filter ("AAF") to its lowest bandwidth (F_{BW}=10 Hz) while Tap Detection expects a higher ODR/bandwidth (ODR>=200 Hz).

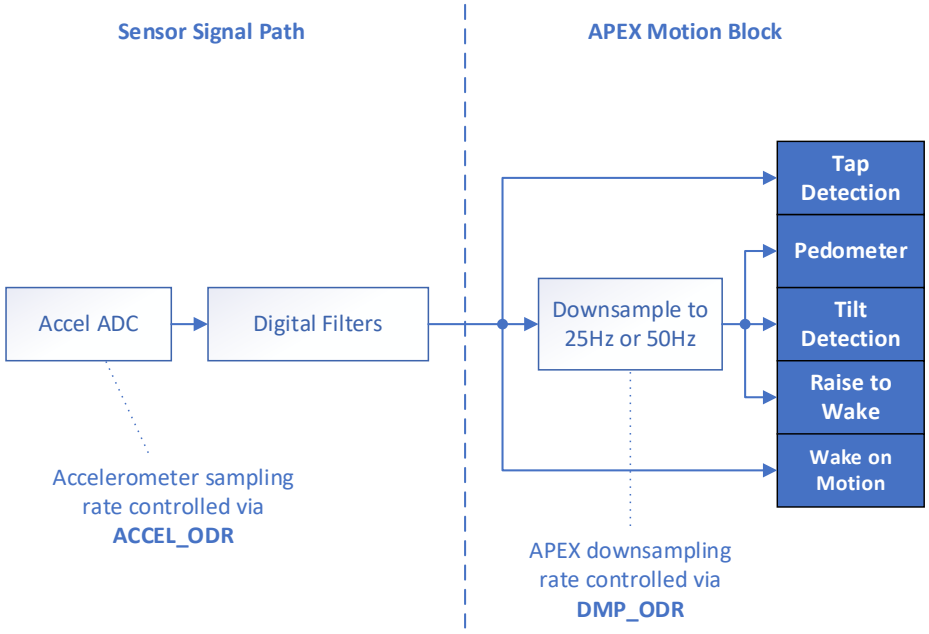

Figure 2. Differentiating sensor ODR and APEX ODR.

When the accel ODR is below 25 Hz, the DMP_ODR bits can be ignored because Tap/Pedometer/Raise-to-Wake are disabled. If the accelerometer's ODR is 25 Hz, set APEX's ODR to 25Hz (DMP_ODR=0). If the accel's ODR is 50 Hz or greater, set APEX's sampling rate to 50 Hz (DMP_ODR=2).

This does not apply to Tap, WoM, and SMD, as these features receive raw accelerometer data without any downsampling (indicated in Figure 2).

FULL SCALE RANGE (FSR)

APEX behavior is independent of accelerometer FSR. The FSR (controlled by ACCEL_FS_SEL) can be modified by the user without any impact to APEX performance.

APEX INITIALIZATION SEQUENCE

Tap/WoM/SMD can be arbitrarily enabled/disabled as they do not require any initialization commands.

Before using Tilt/Pedometer/R2W features, the user must sequentially set APEX's ODR (DMP_ODR), initialize APEX (DMP_INIT_EN=1), reset the memory (DMP_MEM_RESET_EN=1), and finally enable the feature of choice.

The user should issue these commands after hardware power-up, as well as after a software reset (SOFT_RESET_CONFIG=1). See the "Initialization Procedure" example code in this document, or our "eMD" reference drivers available from the InvenSense website.

APEX LOW POWER MODE

To reduce APEX power consumption when the part is motionless, the user can enable Power Save Mode (DMP_POWER_SAVE=0 and DMP_POWER_SAVE_TIME_SEL > 0b000) and WoM (SMD_MODE=0b01). Power Save Mode uses the WoM thresholds (ACCEL_WOM_X/Y/Z_THR) to enable/disable the other APEX routines during stationary periods, thus reducing the processing load on the chip.

HARDWARE/SOFTWARE INTERRUPT CONFIGURATION

By default, INT1 and INT2 interrupts are pulsed (auto-clearing), open-drain (pullup resistors required), and active low output. These settings are all reconfigurable in software. See the INT_SOURCE{x} and INT_CONFIG{x} registers for details.

Any APEX interrupt can be routed to either of the interrupt pins: INT1 or INT2. See the INT_SOURCE6 and INT_SOURCE7 registers in the datasheet's register map. This document's example code utilizes INT2.

One concept to be aware of is that INT2 is multiplexed with other features such frame synchronization input ("FSYNC"—available in all sensors) and external clock input ("CLKIN"—available in high performance variants). A user may want to route an interrupt to INT1 and leave INT2 available for other features depending on a system's design.

3 PEDOMETER

Pedometer measures three quantities: it counts steps, measures cadence (step-rate), and classifies motion activity as running/walking/"unknown" movement. Pedometer can assert an interrupt on any new step event, or if the step counter overflows its 16-bit register space. The default step-counting and activity classification algorithms are optimized for common locations (a user's wrist, pocket, and bag), and will function in any location as long as there's a sufficient step signal.

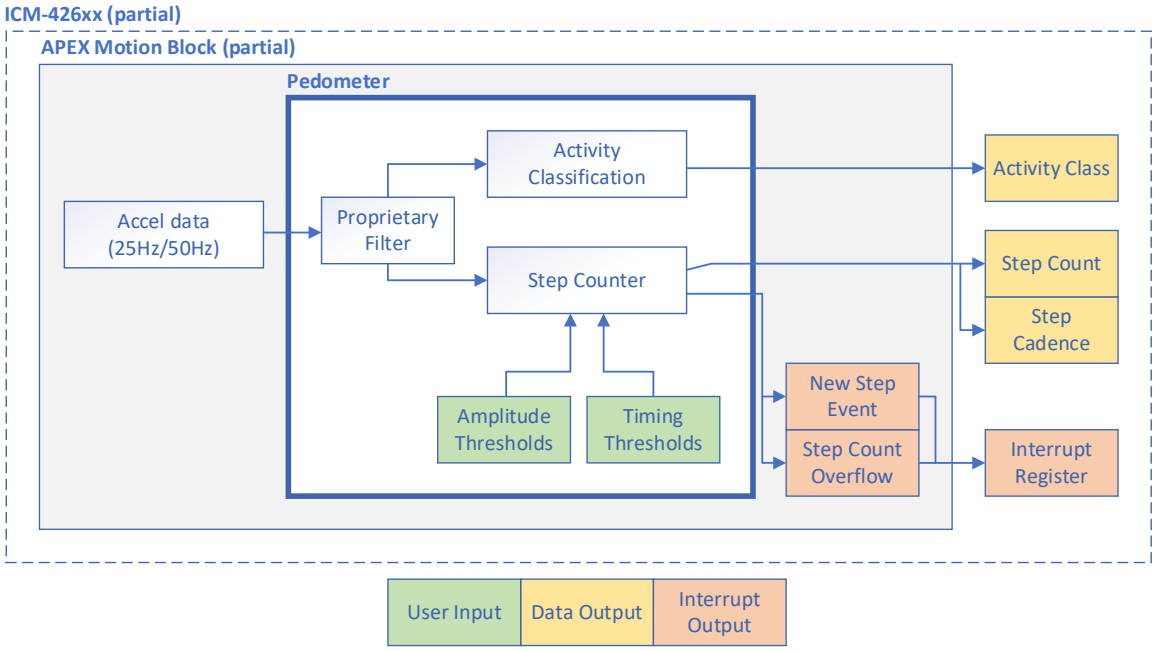

Figure 3. Pedometer block diagram.

Accelerometer data is downsampled, passed through a proprietary filter, and is then compared against user-programmable timing/signal thresholds that identify new step events. Pedometer has three user-readable output fields and two interrupt output bits.

The Pedometer has a default configuration to process accelerometer data and count steps. If more (or less) step sensitivity is desired, the amplitude/timing registers can be adjusted. Pedometer processes downsampled accelerometer data at 25 Hz or 50 Hz (per the DMP_ODR field).

PEDOMETER OUTPUT DATA EXAMPLE

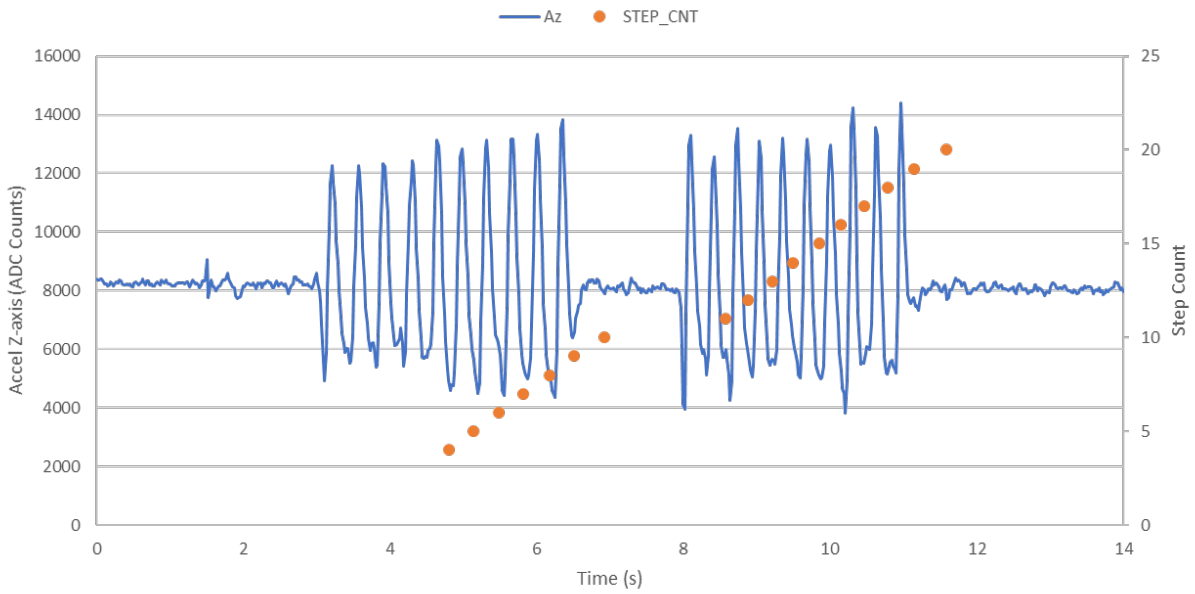

Figure 4. Plot of step count (STEP_CNT) changes during start/stop motion.

Note that the STEP_CNT register increases directly from 0 to 3, per the "Pedometer step detection threshold selection" field (PED_STEP_DET_TH_SEL). The Step Detection Threshold serves to reject spurious motion that might otherwise be qualified as a step.

PEDOMETER FIELDS

The amplitude/energy/timing thresholds are knobs that tune the pedometer sensitivity higher or lower. The default power-on configuration is appropriate for watch/phone/pocket/bag sensor locations, and **InvenSense recommends using the default Pedometer configuration**.

Input **amplitude** fields:

- *PED_AMP_TH_SEL*: the acceleration peak amplitude threshold above which the signal is considered to correspond to a valid step.
- *LOW_ENERGY_AMP_TH_SEL*: threshold to select a valid step depending on the energy of acceleration signal. This parameter helps improve the step detection during slow walk use case.
- *PED_HI_EN_TH_SEL*: the threshold on the signal's energy contained in the band of frequency associated to walk/run motion. This threshold is used to distinguish between acceleration generated by steps motion activities from other acceleration sources. This threshold helps reduce the false detection.
- *SENSITIVITY_MODE*: default is 0. If set to "1", provides greater sensitivity for a "slow walk" use-case, at the expense of higher false-positives during car/train travel (when the accelerometer shows motion but the user is not taking steps). Setting to a 1 is not recommended if motion-rejection is a key performance metric.

Input **timing** fields:

- *PED_STEP_CNT_TH_SEL*: the minimum number of steps needed to be initially detected before starting to increment the step count in real time. This minimum count of steps serves to reduce false positives (peaks that don't correspond to real steps.)
- *PED_STEP_DET_TH_SEL*: the minimum number of steps needed to be initially detected before starting to report step events.
- *PED_SB_TIMER_TH_SEL*: the maximum allowed time between 2 steps. If this amount of time elapses without detecting a new step, the step count buffer is reset to 0. This parameter serves to reduce false positives.

Output **data** fields:

- *STEP_CNT*: the accumulated number of steps (step count) during walk and/or run activities.

- *STEP_CADENCE*: averaged number of samples between consecutive steps, scaled by 4x. To obtain user's steps per minute (SPM), perform the following calculation: *SPM = ODR/(STEP_CADENCE/4)*60*. Example calculation: if STEP_CADENCE=100, and ODR=50Hz, 50/(100/4)*60 = 120 steps/minute by the user.
- *ACTIVITY_CLASS*: the detected user activity. 0: unknown, 1: walk, 2: run.

Output **interrupt** fields:

- *STEP_DET_INT*: interrupt asserted when a new step event is detected
- *STEP_CNT_OVF_INT*: interrupt asserted when the accumulated step count STEP_CNT overflows its 16-bit value, and is reset to 0.

PEDOMETER INITIALIZATION PROCEDURE

The code below sets accelerometer ODR = APEX ODR = 50 Hz, and the output is shown in Figure 4. Note that the ICM-426xx contains multiple register banks. By default the sensor is configured to bank 0. When the user desires to write a command to an alternate bank (see *INT_SOURCE7* line below), the user must first specify the bank by writing the bank number to register address 0x76.

Code Command	I2C Command(s)
`# Reset entire chip - restore default settings` `sensor.i2c_write(0, 0x11, 0x01)`	76w00; 11w01 (write 0x00 to address 0x76; write 0x01 to address 0x11)
`# Configure accel ODR=50Hz, FSR=4g` `sensor.i2c_write(0, 0x50, 0x49)`	50w49
`# ACCEL MODE - low power` `sensor.i2c_write(0, 0x4E, 0x02)`	4Ew02
`# APEX_CONFIG0 - DMP_POWER_SAVE=0, DMP_ODR=0b10 (50Hz)` `sensor.i2c_write(0, 0x56, 0x02)`	56w02
`# DMP_MEM_RESET_EN = 1` `sensor.i2c_write(0, 0x4B, 0x20)`	4Bw20
`# wait 1 millisecond` `sleep(0.001)`	
`# DMP_INIT_EN=1` `sensor.i2c_write(0, 0x4B, 0x40)`	4Bw40
`# Change to bank 4.  Set INT_SOURCE7 to assert interrupt on new step` `sensor.i2c_write(4, 0x4E, 0x20)`	76w04; 4Ew20
`# APEX_CONFIG0 - Enable pedometer` `sensor.i2c_write(0, 0x56, 0x22)`	76w00; 56w22
`# wait 50ms for 1 sample to elapse` `sleep(0.050)`	

Table 2. Pedometer initialization example.

The output data fields (STEP_CNT, STEP_CADENCE, ACTIVITY_CLASS) are located at bank 0 with register addresses 0x32, 0x33, 0x34 (APEX_DATA1, APEX_DATA2, APEX_DATA3).

4 TILT DETECTION

Tilt Detection measures the angle between two accelerometer vectors: the current sample and the sample at the time of the previous tilt interrupt. Tilt Detection asserts an interrupt if it calculates a change of >35°. There is a single input parameter—Tilt Wait Time—which controls how long a valid "tilt" event must be held before an interrupt is initiated.

Large acceleration without a change in sensor orientation (relative to gravity) should not trigger a tilt event. For example, a rotation of the sensor about the observer's z-axis (direction of gravity) should not result in a tilt interrupt. A sharp turn while driving a car, or a user turning around rapidly while walking should not trigger a tilt event if the rotation is about the axis of gravity.

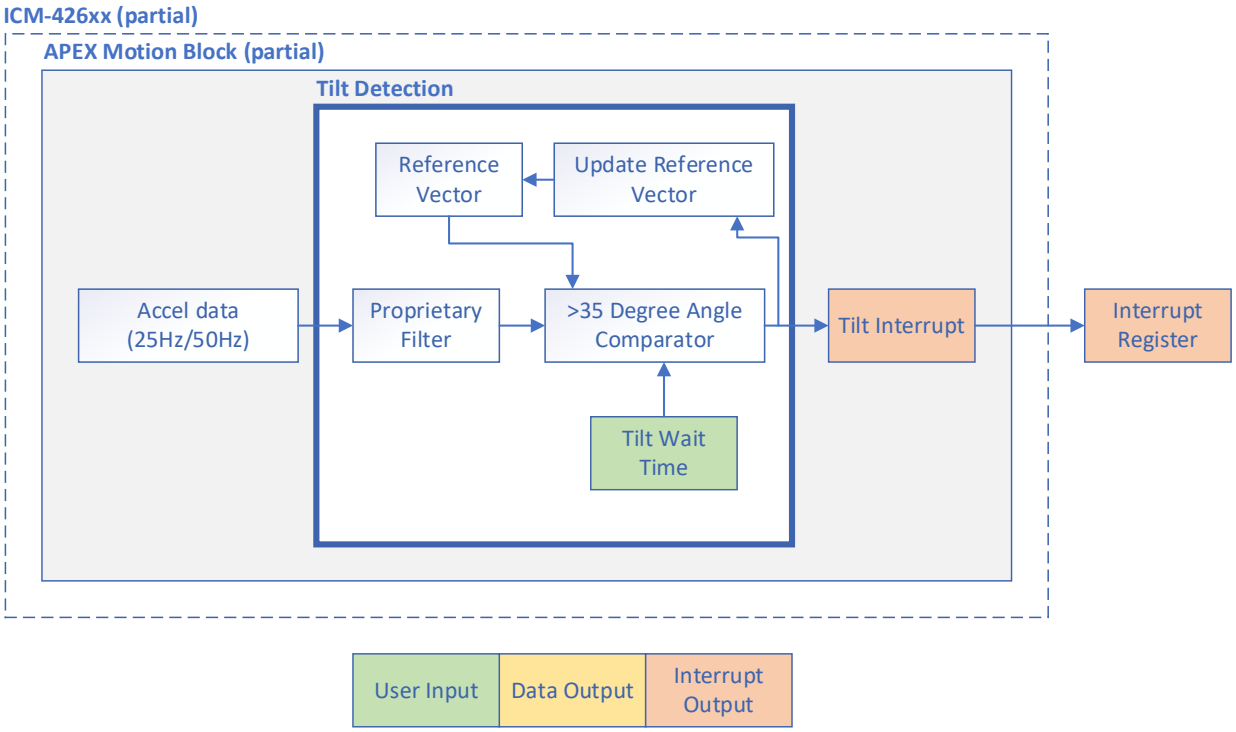

Figure 5. Tilt interrupt block diagram.

Document Number: AN-000173
Revision: 1.0

TILT DETECTION OUTPUT DATA EXAMPLE

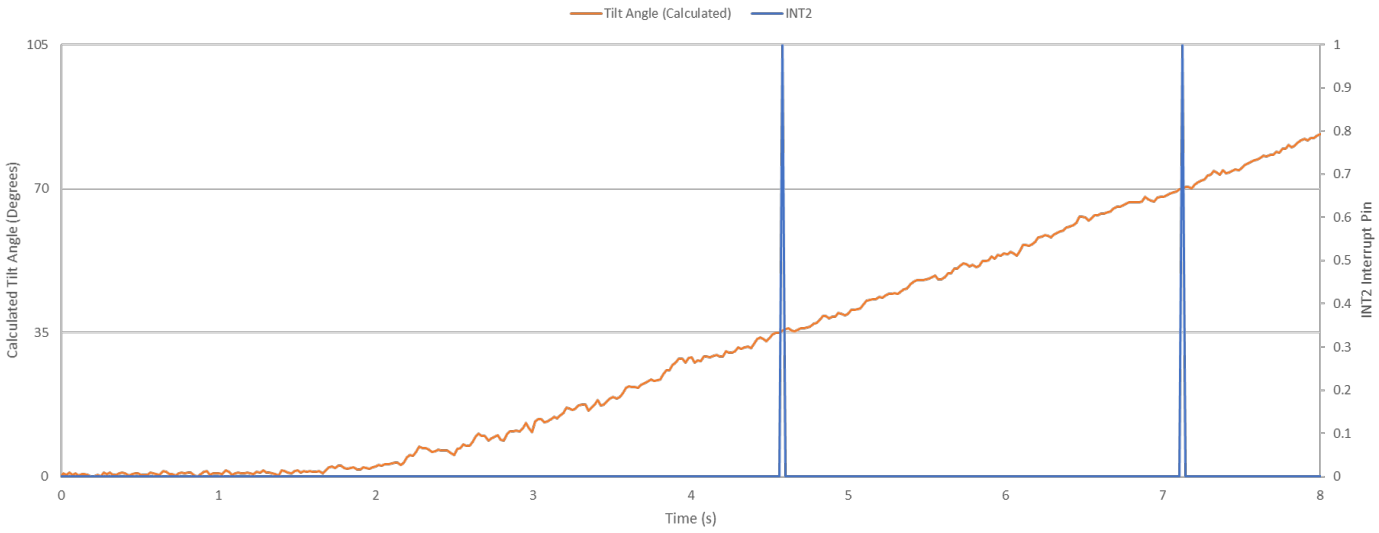

Figure 6. Tilt interrupt output while sensor rotates.

Tilt angle (orange) is calculated from accelerometer vector, and INT2 interrupt pin (blue) is measured.

TILT_WAIT_TIME_SEL=0 for an immediate interrupt at 35-degree tilt angle limits.

Figure 6 shows the tilt angle (calculated in software—not directly output by the ICM-426xx) while rotating the sensor. Tilt Angle represents the angle between two accelerometer vectors: (a) the previous sample at time of last interrupt, compared to (b) the current sample. Tilt Wait Time was programmed to 0 seconds for an immediate interrupt after a tilt event.

Tilt Detection passes the accelerometer signal through a proprietary lowpass filter (shown in Figure 5), which will add a small delay (no delay observed in Figure 6 due to the slow rate of sensor rotation).

The user should wait ~2 seconds in between enabling tilt (TILT_ENABLE=1) and enabling hardware interrupt output (INT_SOURCE6/7 registers) to allow the Tilt feature to stabilize.

TILT DETECTION FIELDS

Input field:

- *TILT_WAIT_TIME_SEL*: defines the polling period when the sensor checks for a tilt event. Longer polling periods result in fewer, and more delayed, interrupts. (Note: the default time is 2 seconds, but for illustration purposes Figure 6 shows behavior with 0 second tilt wait time).

Output interrupt field:

- *TILT_DET_INT*: interrupt status dedicated to the Tilt event.

TILT DETECTION INITIALIZATION PROCEDURE

Code Command	I2C Command(s)
`# Reset entire chip - restore default settings` `sensor.i2c_write(0, 0x11, 0x01)`	76w00; 11w01 (write 0x00 to address 0x76; write 0x01 to address 0x11)
`# APEX_CONFIG4.. TILT_WAIT_TIME_SEL: 0s for immediate interrupt` `sensor.i2c_write(4, 0x43, 0x00)`	76w04; 43w00
`# ODR=50Hz, FSR=4g` `sensor.i2c_write(0, 0x50, 0x49)`	76w00; 50w49
`# LOW POWER MODE` `sensor.i2c_write(0, 0x4E, 0x02)`	4Ew02
`# APEX_CONFIG0 - DMP_POWER_SAVE=0, DMP_ODR=0b10 (50Hz)` `sensor.i2c_write(0, 0x56, 0x02)`	56w02
`# DMP_MEM_RESET_EN=1` `sensor.i2c_write(0, 0x4B, 0x20)`	4Bw20
`# wait 1 millisecond` `sleep(0.001)`	
`# DMP_INIT_EN=1` `sensor.i2c_write(0, 0x4B, 0x40)`	4Bw40
`# APEX_CONFIG0 - Enable TILT` `sensor.i2c_write(0, 0x56, 0x12)`	76w00; 56w12
`# wait 2000ms for TILT feature to stabilize` `sleep(2.000)`	
`# INT_SOURCE7 -- route software TILT interrupt to hardware INT2` `sensor.i2c_write(4, 0x4E, 0x08)`	76w04; 4Ew08

Table 3. Tilt Detection initialization example.

5 RAISE TO WAKE/SLEEP (R2W)

Raise-To-Wake/Sleep (R2W) is designed to detect when a user raises a product (cell phone, smartwatch, tablet) to their face in an upright orientation, and issue an interrupt.

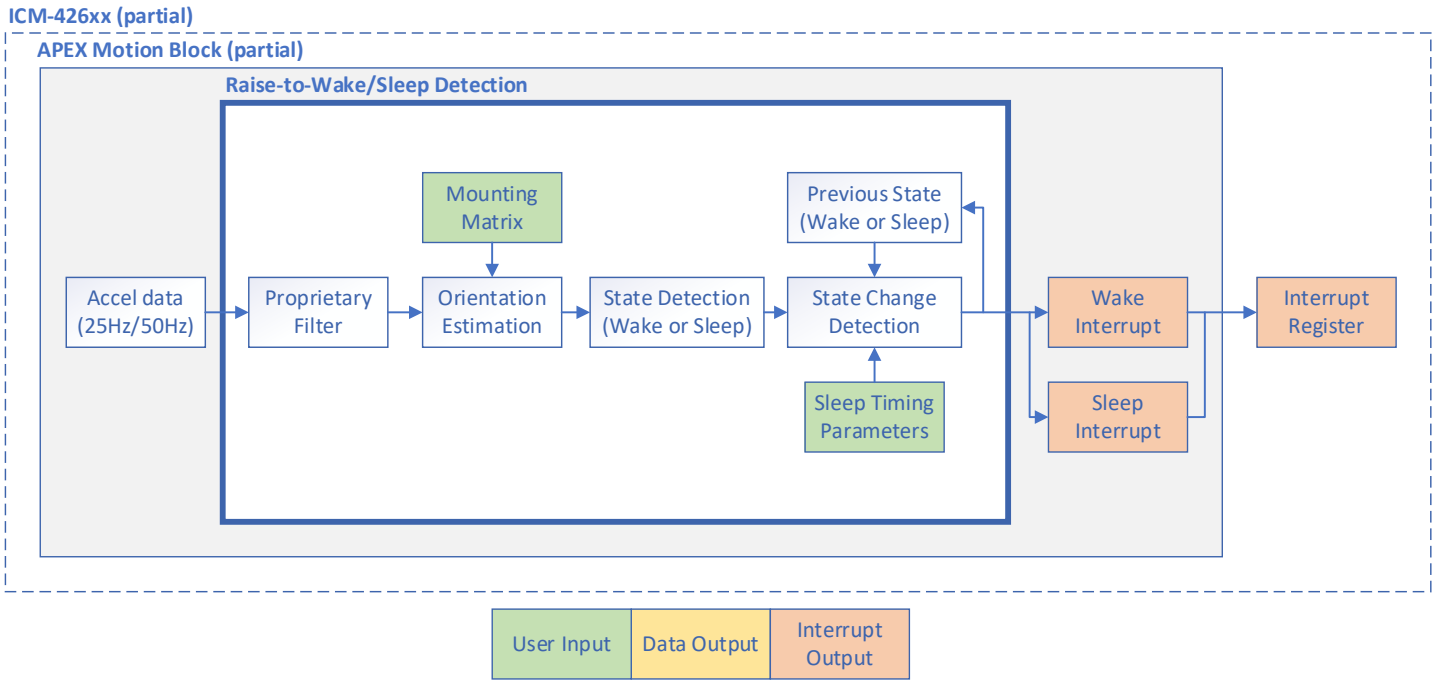

Figure 7. Raise to wake block diagram.

A wake gesture is defined by the direction of gravity changing by > 80° and coming to rest within a ±30° cone of a user-selected axis (x-axis by default). A sleep gesture requires that the sensor is outside of the ±30° cone and *either* the SLEEP_GESTURE_DELAY threshold or the SLEEP_TIME_OUT threshold have been crossed.

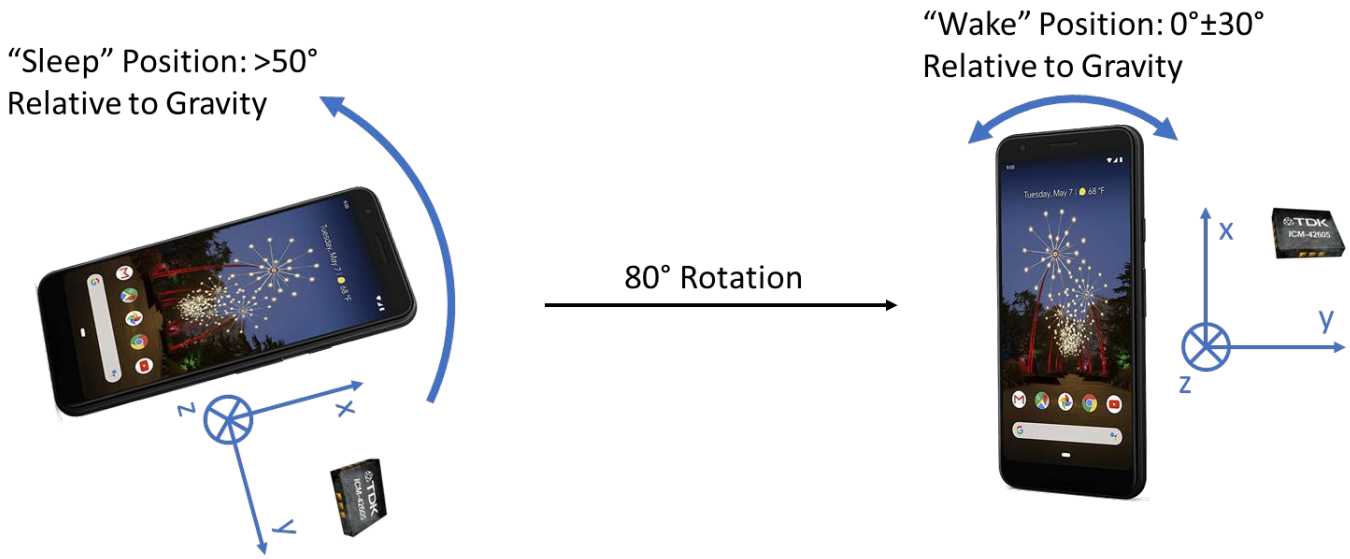

Figure 8. Diagram showing "wake" gesture for default configuration (interrupt when [Ax, Ay, Az] = [1, 0, 0]g).

The user may define which raised axis produces an interrupt: ±x or ±y (but not ±z). Set *MOUNTING_MATRIX* to align the sensor's axes with the product's axes per Figure 9, and R2W will assert an interrupt when the device is raised accordingly.

Document Number: AN-000173
Revision: 1.0

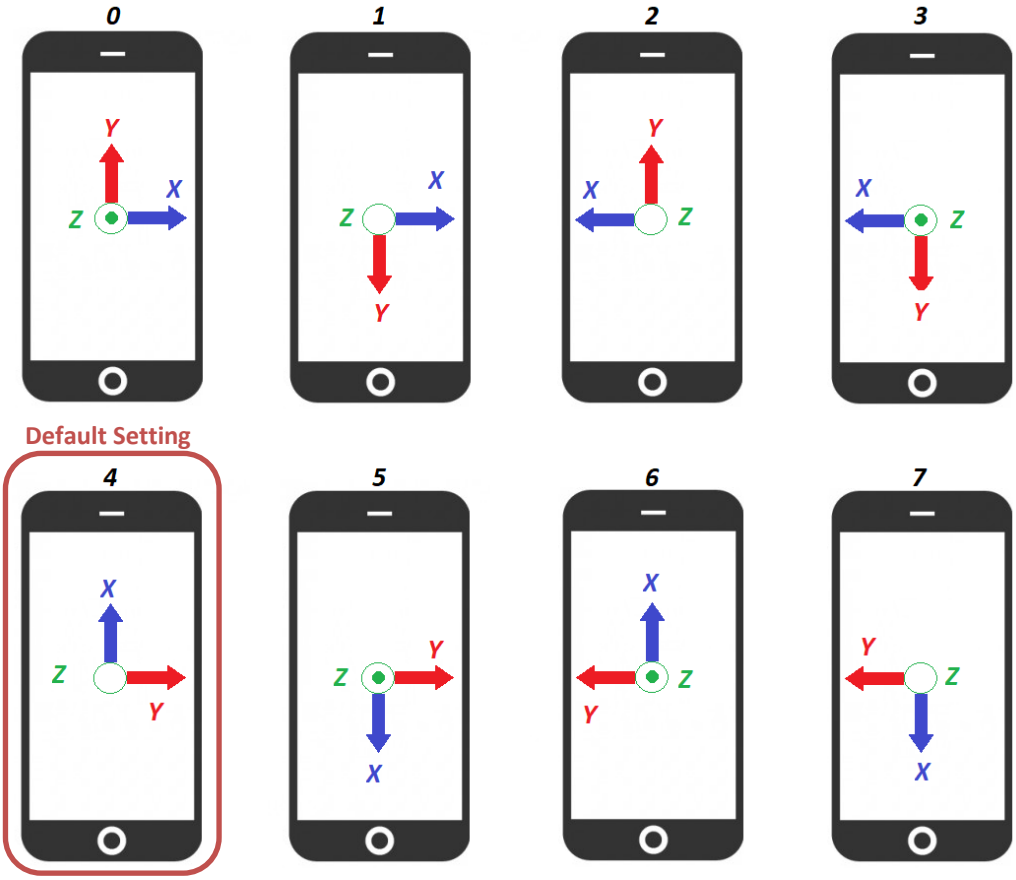

Figure 9. Visualization of MOUNTING_MATRIX options for accel x/y/z orientations.

The displayed image number corresponds to the MOUNTING_MATRIX register setting.

Power-on default (MOUNTING_MATRIX=4) asserts interrupt when accelerometer x-axis is raised.

A sleep gesture interrupt requires that a minimum duration has elapsed between “wake” and “sleep” gestures (*SLEEP_TIME_OUT*), or that the sensor remained in the “sleep” position for a specified duration (*SLEEP_GESTURE_DELAY*). A wake gesture interrupt will fire promptly as shown in Figure 10 below—there are no timing options for “wake” interrupts.

A sleep event will only happen if the last event was wake, and a wake will only happen if the last event was a sleep gesture—it’s not possible to have 2 consecutive sleep events or 2 consecutive wake events.

R2W OUTPUT DATA

The sensor output shown in Figure 10 was captured using the default MOUNTING_MATRIX setting, where [Ax, Ay, Az] = [1, 0, 0]g will trigger a “wake” interrupt.

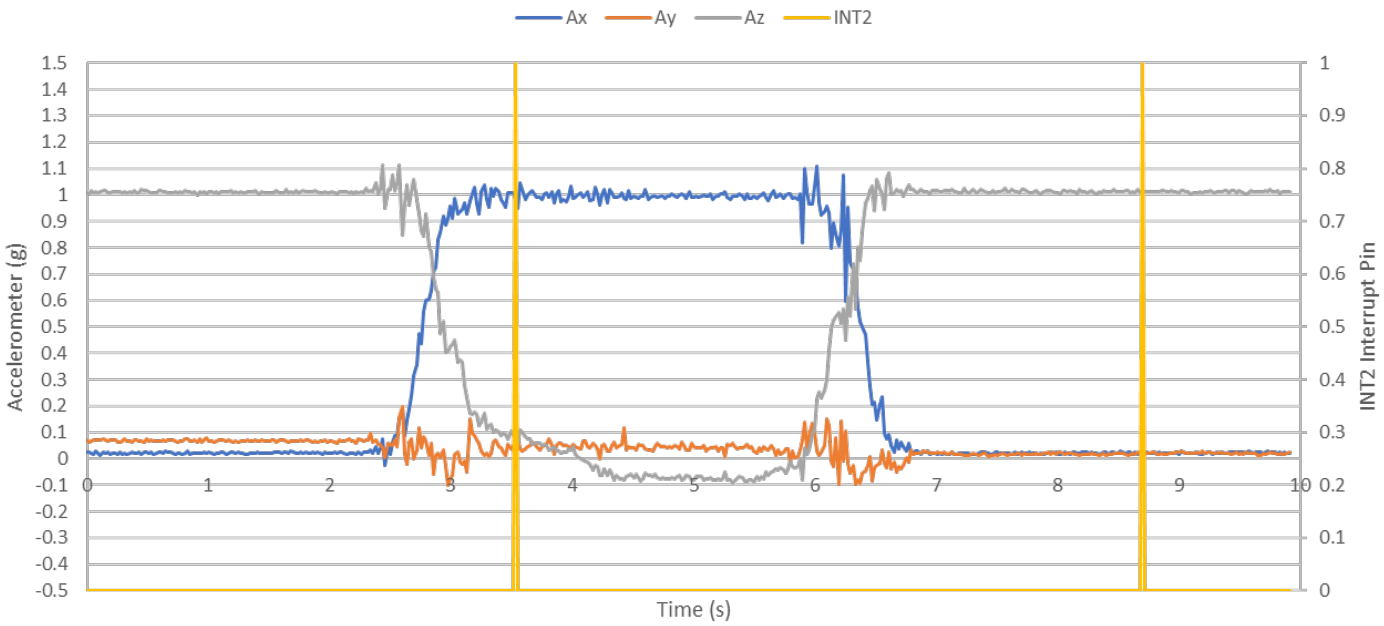

Figure 10. Plot showing sensor output during two R2W events with default register configuration.

The first interrupt is a "wake" event, and the second interrupt is a "sleep" event.

R2W FIELDS

Input timing fields:

- *SLEEP_GESTURE_DELAY*: after "wake" event, wait an amount of time controlled by SLEEP_GESTURE_DELAY (0b000: 0.32 seconds up to 0b111: 2.56 seconds in linear 0.32 second increments), then R2W conditionally asserts a sleep interrupt if the sensor is in sleep orientation.
- *SLEEP_TIME_OUT*: if the sensor is in the "wake" state and has remained in this state for a time controlled by SLEEP_TIME_OUT (0b000: 1.28 seconds up to 0b111 10.24 seconds in linear 1.28 second increments), then assert a "sleep" interrupt.

Please note that if SLEEP_GESTURE_DELAY triggers a sleep interrupt, then SLEEP_TIME_OUT becomes irrelevant because R2W will only assert one sleep interrupt.

Input mounting field:

- *MOUNTING_MATRIX*: defines the axis that triggers wake/sleep interrupts. By default R2W asserts an interrupt when the x-axis is aligned with the gravity vector. **To assert an interrupt when gravity is aligned to a different axis, refer to Figure 10 to see different sensor/product orientation options.**

Output interrupt fields:

- *WAKE_INT*: interrupt for "wake" event.
- *SLEEP_INT*: interrupt for "sleep" event.

R2W INITIALIZATION PROCEDURE

Code Command	I2C Command(s)
`# Reset entire chip - restore default settings` `sensor.i2c_write(0, 0x11, 0x01)`	76w00; 11w01 (write 0x00 to address 0x76; write 0x01 to address 0x11)
`# ODR=50Hz, FSR=4g` `sensor.i2c_write(0, 0x50, 0x40)`	50w49
`# LOW POWER MODE` `sensor.i2c_write(0, 0x4E, 0x02)`	4Ew02
`# APEX_CONFIG0 - DMP_POWER_SAVE=0, DMP_ODR=0b10 (50Hz)` `sensor.i2c_write(0, 0x56, 0x02)`	56w02
`# DMP_MEM_RESET_EN=1` `sensor.i2c_write(0, 0x4B, 0x20)`	4Bw20
`# Wait 1 millisecond` `sleep(0.001)`	
`# DMP_INIT_EN=1` `sensor.i2c_write(0, 0x4B, 0x40)`	4Bw40
`# INT_SOURCE7 -- tie SLEEP+WAKE interrupts to INT2` `sensor.i2c_write(4, 0x4E, 0x06)`	76w04; 4Ew06
`# APEX_CONFIG0 - Enable R2W` `sensor.i2c_write(0, 0x56, 0x0A)`	76w00; 56w0A
`# wait 50ms for 1 sample to elapse` `sleep(0.050)`	

Table 4 - R2W initialization example.

6 TAP DETECTION

Tap Detection monitors the accelerometer for sharp pulses (of 150 ms~375 ms in length with a > 0.125g jump in acceleration) and extracts the number of taps (one or two), the time duration between the taps, the primary axis and direction of the tap (±x, ±y, or ±z), and asserts an interrupt.

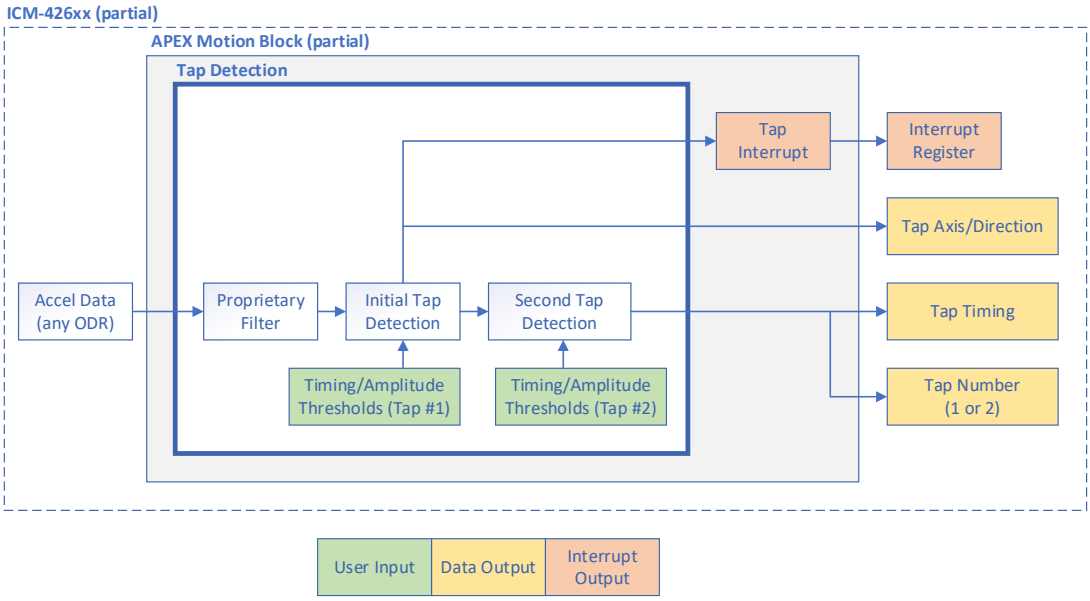

Figure 11. Tap Detection block diagram.

There are a variety of timing windows and amplitude thresholds to determine what is (or isn't) a tap gesture. The default power-on values are the recommended settings for most consumer applications.

TAP DETECTION OUTPUT DATA

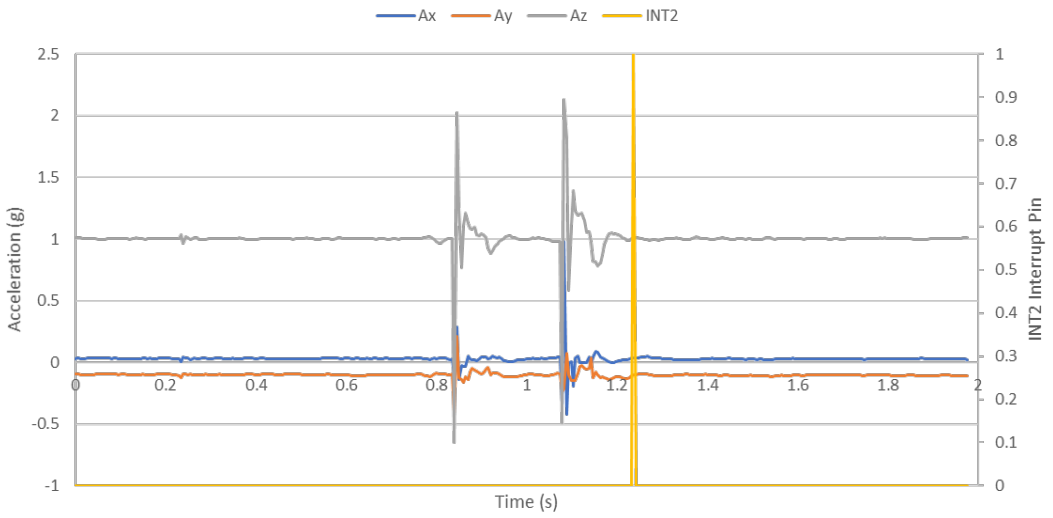

Figure 12. Example of a double-tap gesture and corresponding interrupt.

For the signal shown in Figure 13, the sensor's ODR is 200 Hz. The sensor reported 2 taps (*TAP_NUM*=2) with the initial tap in the negative-z axis direction (*TAP_AXIS*=2, *TAP_DIR*=1), and the second tap occurring 240 ms after the first (*TAP_TIMING*=3).

For highest accuracy and precision, configure the accel ODR to 1 kHz.

Document Number: AN-000173
Revision: 1.0

TAP DETECTION TIMING DIAGRAM

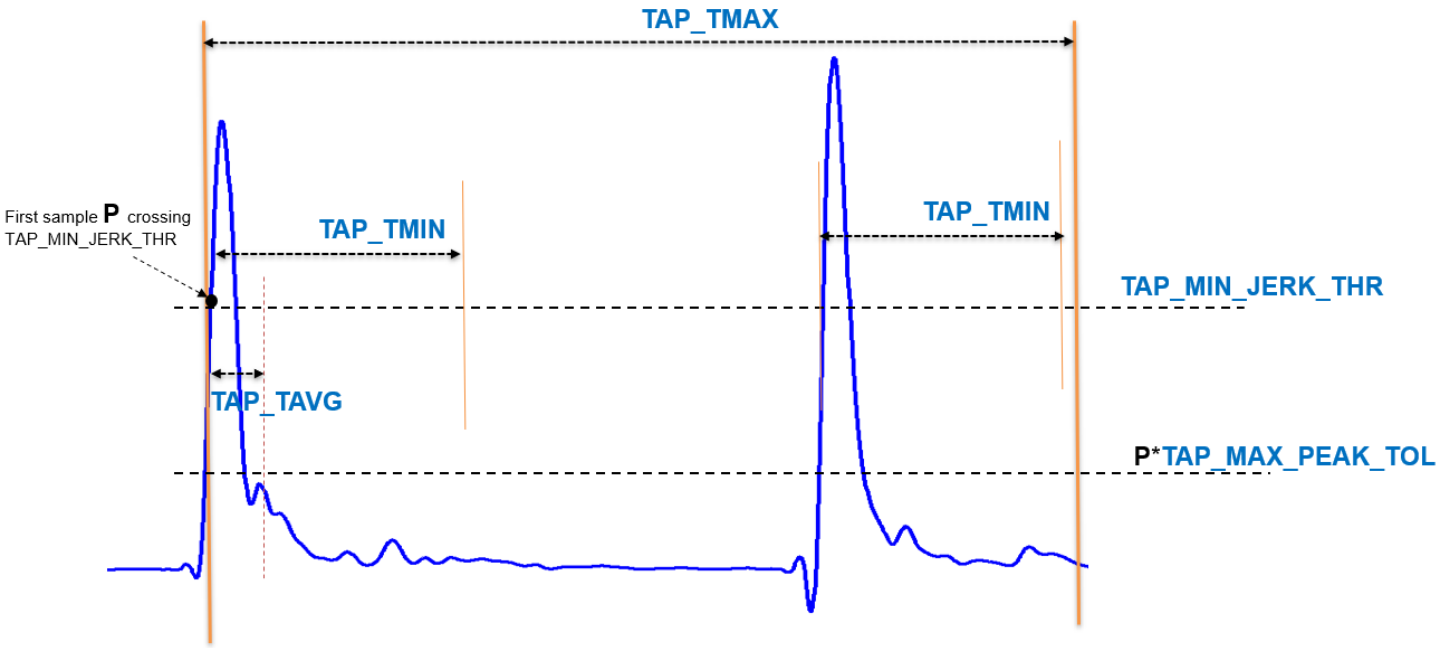

Figure 13 - Timing diagram showing configurable input parameters.

TAP DETECTION FIELDS

Like the Pedometer feature, Tap Detection has carefully selected default values. InvenSense recommends using the default tap configuration.

InvenSense recommends setting the accelerometer ODR=1kHz for highest accuracy. Using an ODR of 200 Hz or 500 Hz will save power, but the timing precision (see *DOUBLE_TAP_TIMING*) will be lower.

Input timing fields:

- *TAP_TMAX*: controls the maximum time for a double-tap sequence. Lowering this time will result in the sensor recognizing two separate single taps (with two interrupts). Default setting corresponds to 0.375 seconds.
- *TAP_TAVG*: an averaging window used to determine the dominant axis (±x, ±y, or ±z). Applies to the first tap, not the second. Default setting corresponds to a smoothing window of 2 samples.
- *TAP_TMIN*: controls the minimum amount of time a single tap can occur. If multiple taps occur in this period, the pulses are *not* recognized as a valid tap gesture. Default setting corresponds to 0.150 seconds.

Input amplitude fields:

- *TAP_MIN_JERK_THR*: a change in acceleration above this threshold qualifies a tap gesture as valid. Default value corresponds to change in acceleration of a[n] – a[n-1] = 0.125g delta between samples.
- *TAP_MAX_PEAK_TOL*: defines a percentage of the first accel value that crosses TAP_MIN_JERK_THR, used as a threshold to measure the duration of the pulse. Default value is 25%.

Output data fields:

- *TAP_NUM*: the number of taps measured by the sensor (0-2).
- *TAP_AXIS*: the measured dominant axis of the first tap (0: x-axis, 1: y-axis, 2: z-axis).
- *TAP_DIR*: the direction of the tap (1: tap pushes sensor in negative direction, 0: tap pushes sensor in positive direction), for the axis specified in TAP_AXIS.
- *DOUBLE_TAP_TIMING*: the duration between two double-tap pulses (in seconds: DOUBLE_TAP_TIMING * 16 / ODR)

Please note that these **output data fields are clear-on-read**. After you read from these registers, they will revert to their default value.

Document Number: AN-000173
Revision: 1.0

Output interrupt field:

- *TAP_DET_INT*: interrupt after a valid tap gesture.

TAP DETECTION INITIALIZATION PROCEDURE

Code Command	I2C Command(s)
`# Reset entire chip - restore default settings` `sensor.i2c_write(0, 0x11, 0x01)`	76w00; 11w01 (write 0x00 to address 0x76; write 0x01 to address 0x11)
`# ODR=1kHz, FSR=16g - note: this line re-writes default value. Not required since chip was just reset, but it's included for consistency with other examples.` `sensor.i2c_write(0, 0x50, 0x06)`	50w06
`# Turn accel on, in low noise mode. Low noise mode is mandatory since ODR=1kHz` `sensor.i2c_write(0, 0x4E, 0x03)`	4Ew03
`# APEX_CONFIG0 - Enable TAP` `sensor.i2c_write(0, 0x56, 0x40)`	56w40
`# INT_SOURCE7 -- tie TAP interrupt to INT2` `sensor.i2c_write(4, 0x4E, 0x04)`	76w04; 4Ew06

Table 5. Tap detection initialization example.

After the user has performed a "tap" gesture and an interrupt is generated, the user can read from *APEX_DATA4* (bank 0, register address 0x35) and *APEX_DATA5* (bank 0, register address 0x36) to obtain the tap number/axis/direction information.

7 WAKE ON MOTION (WOM)

Wake on Motion detects when the change in accelerometer output exceeds a user-programmable threshold.

WoM can assert an interrupt if the accelerometer *1st order difference* is greater than a threshold[2] or if the accelerometer change (relative to the first sample after WoM is enabled) is greater than a threshold[3].

The user can program independent acceleration thresholds for all 3 axes and configure the part to assert an interrupt when ANY axis trips its threshold, or when ALL axes trip their thresholds simultaneously.

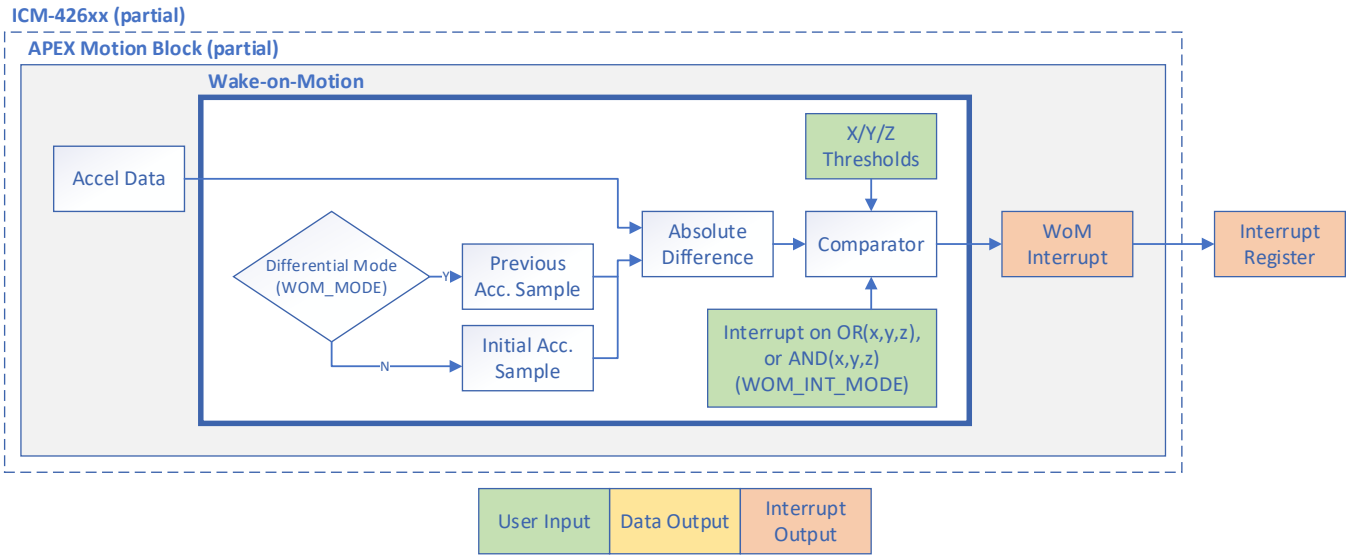

Figure 14. Wake-on-Motion block diagram.

WOM OUTPUT DATA EXAMPLE

The plot below shows two different Wake-On-Motion modes: **absolute (WOM_MODE=0)** and **differential (WOM_MODE=1)**. Differential mode asserts an interrupt when the *1st order difference* in accel signal is greater than a programmed threshold. Absolute mode asserts an interrupt when the current sample is greater than the initial sample after WoM is enabled.

The data below was simulated (not measured) for the sake of illustration. The simulated data was created with sensor ODR = APEX ODR = 50 Hz and with a WOM threshold = 0.4g.

[2] In differential mode when *WOM_MODE*=1, if |a(t) - a(t-1)| > threshold, assert interrupt
[3] In absolute mode when *WOM_MODE*=0, if |a(t) - a(t_0)| > threshold, assert interrupt

Document Number: AN-000173
Revision: 1.0

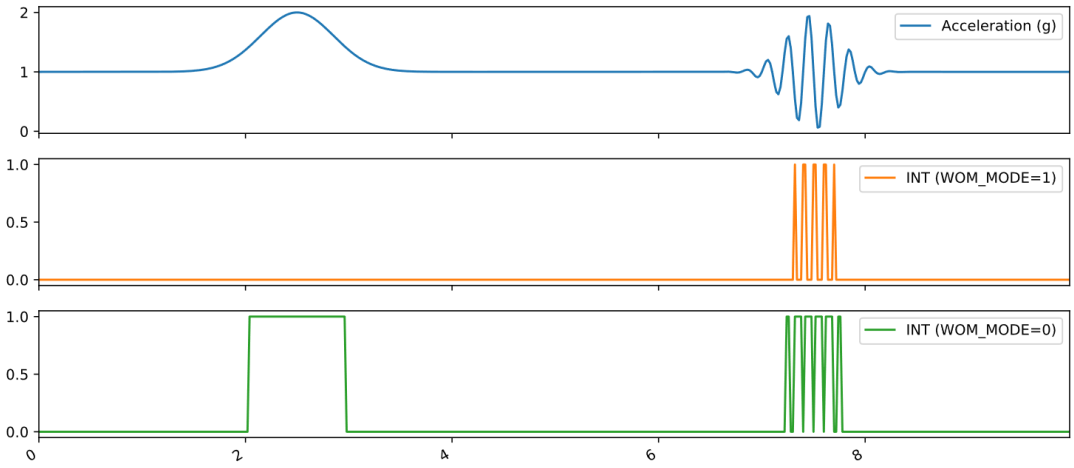

Figure 15. Simulated data comparing absolute WOM_MODE=0 vs differential WOM_MODE=1 with 0.4g threshold.

In absolute mode (green line; *WOM_MODE*=0), WoM compares the accelerometer sample at time t=0s to all subsequent samples. The sensor value at time t=2 seconds is 0.4g greater than the sensor value at time t=0 seconds, which triggers the absolute interrupt but does not trigger the differential interrupt.

In differential mode (orange line; WOM_MODE=1) at time t=2, the sensor's output changes slowly. The changes are not >0.4g relative to the immediately previous sample, keeping the (WOM_MODE=1) interrupt from triggering. The faster output changes at time t=7s trigger both the differential and absolute interrupts.

Note that ODR can directly impact the interrupt behavior in differential mode (but not absolute mode), because the difference between samples will get smaller as the sampling rate increases. For example: in differential mode, a 0.4g threshold at ODR=50 Hz will trigger under the same acceleration conditions as a 0.2g threshold will trigger at ODR=100 Hz.

WOM FIELDS

Input fields

- *WOM_X_TH*: threshold value for the X-axis Wake on Motion Interrupt. WoM thresholds range from 0g~1g in increments of 1/256 g, so WOM_X_TH=10 represents 39.1 mg (10/256 *1000 mg).
- *WOM_Y_TH*: threshold value for the Y-axis Wake on Motion Interrupt. Same scale as x-axis.
- *WOM_Z_TH*: threshold value for the Z-axis Wake on Motion Interrupt. Same scale as x-axis.
- *WOM_INT_MODE*: Set WoM interrupt on the logical OR of all enabled accelerometer thresholds (when set to 0), or logical AND (when set to 1).
- *WOM_MODE*: Controls if WoM operates as differential (when set to 0) or absolute (when set to 1)
- *SMD_MODE*: Set value to 1 when only using WoM (and not SMD) or set value to 0 to deactivate.

Interrupt output:

- *WOM_X_INT*: Wake on Motion Interrupt on x-axis
- *WOM_Y_INT*: Wake on Motion Interrupt on y-axis
- *WOM_Z_INT*: Wake on Motion Interrupt on z-axis

WOM INITIALIZATION PROCEDURE

Code Command	I2C Command(s)
`# Reset entire chip - restore default settings` `sensor.i2c_write(0, 0x11, 0x01)`	76w00; 11w01 (write 0x00 to address 0x76; write 0x01 to address 0x11)
`# ODR=50Hz, FSR=4g` `sensor.i2c_write(0, 0x50, 0x49)`	50w49
`# LOW POWER MODE` `sensor.i2c_write(0, 0x4E, 0x02)`	4Ew02
`# ACCEL_WOM_X_THR: threshold=~0.4g` `sensor.i2c_write(4, 0x4A, 0x66)`	76w04; 4Aw66
`# ACCEL_WOM_Y_THR: threshold=~0.4g` `sensor.i2c_write(4, 0x4B, 0x66)`	4Bw66
`# ACCEL_WOM_Z_THR: threshold=~0.4g` `sensor.i2c_write(4, 0x4C, 0x66)`	4Cw66
`# INT_SOURCE4 -- tie WoM x/y/z interrupt to INT2` `sensor.i2c_write(0, 0x69, 0x07)`	76w00; 69w07
`# SMD_CONFIG: differential WoM, interrupt on any axis` `sensor.i2c_write(0, 0x57, 0x05)`	57w05

Table 6. WoM initialization example.

8 SIGNIFICANT MOTION DETECTION (SMD)

Significant Motion Detection (SMD) asserts an interrupt when two sequential WoM events occur within a user-programmable time window. The two selectable options are t=1 second or t=3 seconds.

SMD OUTPUT DATA EXAMPLE

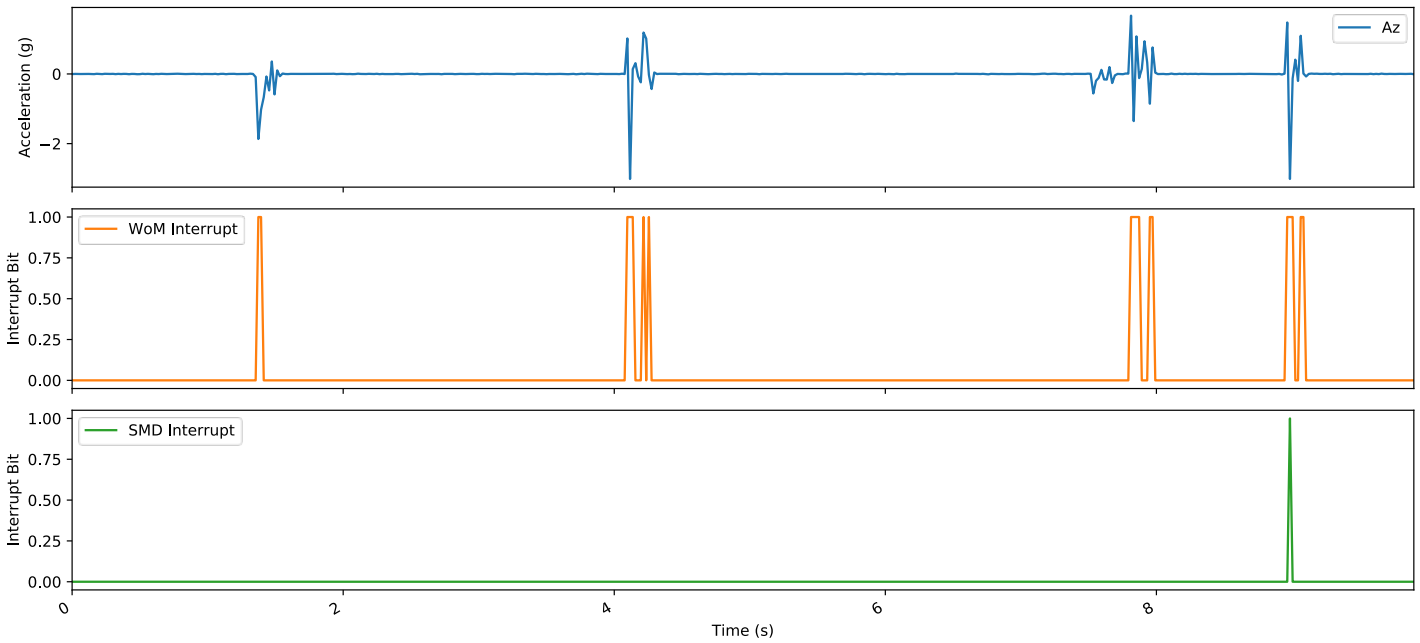

Figure 16. Plot showing 2 WoM interrupts in a 1-second window triggering an SMD interrupt.

SMD FIELDS

All WoM fields apply to SMD, since SMD will assert an interrupt based on WoM events.

Input fields:

- *SMD_MODE*: Asserts an SMD interrupt when two WOM gestures are detected 1 second apart (*SMD_MODE*=2) or 3 seconds apart (*SMD_MODE*=3). SMD is deactivated by default (*SMD_MODE*=0).

Interrupt output field:

- *SMD_INT*: Significant Motion Detection Interrupt

SMD INITIALIZATION PROCEDURE

Code Command	I2C Command(s)
`# Reset entire chip - restore default settings` `sensor.i2c_write(0, 0x11, 0x01)`	76w00; 11w01 (write 0x00 to address 0x76; write 0x01 to address 0x11)
`# ODR=50Hz, FSR=4g` `sensor.i2c_write(0, 0x50, 0x49)`	50w49
`# LOW POWER MODE` `sensor.i2c_write(0, 0x4E, 0x02)`	4Ew02
`# ACCEL_WOM_X_THR: threshold=~0.4g` `sensor.i2c_write(4, 0x4A, 0x66)`	76w04; 4Aw66
`# ACCEL_WOM_Y_THR: threshold=~0.4g` `sensor.i2c_write(4, 0x4B, 0x66)`	4Bw66
`# ACCEL_WOM_Z_THR: threshold=~0.4g` `sensor.i2c_write(4, 0x4C, 0x66)`	4Cw66
`# INT_SOURCE4: tie SMD interrupt to INT2` `sensor.i2c_write(0, 0x69, 0x08)`	76w00; 69w08
`# SMD_CONFIG: SMD short (1s), differential WoM, interrupt on any axis` `sensor.i2c_write(0, 0x57, 0x06)`	57w06

Table 7. SMD initialization example.

9 EXAMPLE CODE IN C

Detailed C-code examples are available for our SmartMotion Development Platform (https://www.invensense.com/smartmotion-platform/). See the “eMD” reference drivers on the InvenSense website Developer’s Corner (https://www.invensense.com/developers/). Once the files are downloaded and uncompressed, find the “Examples” folder for example projects showcasing each APEX feature.

Android drivers are available for select sensors and require a software licensing agreement (SLA).

10 REVISION HISTORY

REVISION DATE	REVISION	DESCRIPTION
12/20/2019	1.0	Initial release

DECLARATION DISCLAIMER

InvenSense believes the environmental and other compliance information given in this document to be correct but cannot guarantee accuracy or completeness. Conformity documents substantiating the specifications and component characteristics are on file. InvenSense subcontracts manufacturing, and the information contained herein is based on data received from vendors and suppliers, which has not been validated by InvenSense.

©2019 InvenSense. All rights reserved.