

# Air Motion Library (AML)

## Integration Guide

## TABLE OF CONTENTS

1	Overview .....	3
2	Description of the released software package .....	4
3	Algorithm integration .....	5
	Step 1: Include header files .....	5
	Step 2: Link library:.....	6
	Step 3: Modify example_aml.c file of printer and USB function.....	6
	Step 4: Transfer sensor axis to board/device axis.....	7
	Step 5: Implement read/write/sleep/delay function.....	9
	Step 6: Implement the flow for initialization and algorithm processing.....	10
4	Revision history.....	12

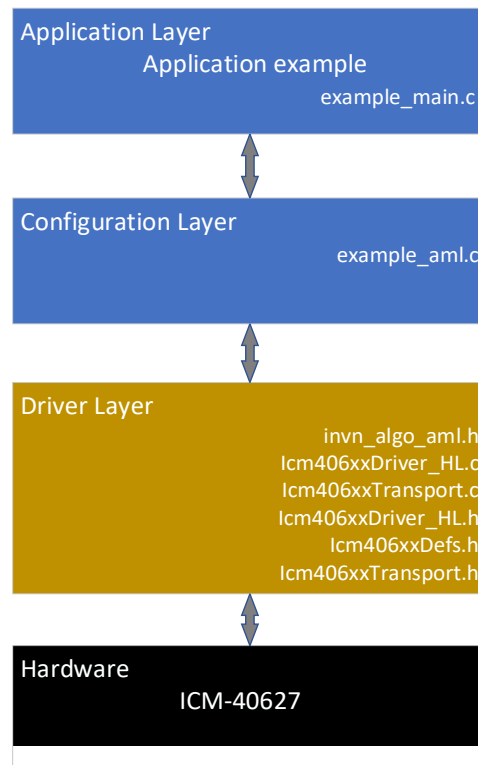
## 1 OVERVIEW

Air Motion Library (AML) is a library used to manage mouse cursor motion from a 3-axis gyroscope and a 3-axis accelerometer. It is responsible for converting motion sensor data into delta X and delta Y pointer movements.

The library is intended for use in free space pointing devices to operate in-air point and click navigation, the way a classic 2D mouser will do on a desk. In addition to its pointing feature, Air Motion Library is also capable of swipe motion recognition: 'Up', 'Down', 'Left', 'Right', 'Clockwise', and 'Counterclockwise'.








This document will provide a step-by-step integration guideline of the AML algorithm and the ICM-40627 driver. It includes the package introduction and the step-by-step integration guideline.

The block diagram below shows the architecture of the driver, algorithm, and application.



## 2 DESCRIPTION OF THE RELEASED SOFTWARE PACKAGE

The AML algorithm is bundled with ICM-40627. The software package includes the AML library and its header files, the driver for ICM-40627, documentation, and example code. The software package structure is shown below:

- ▼  Algo package for porting to different platforms
  - >  01\_doc      Documentation
  - >  02\_SensorA      Driver
  - ▼  03\_Algo      Algorithm
    -  inc
    -  lib
  - >  04\_Example      Example

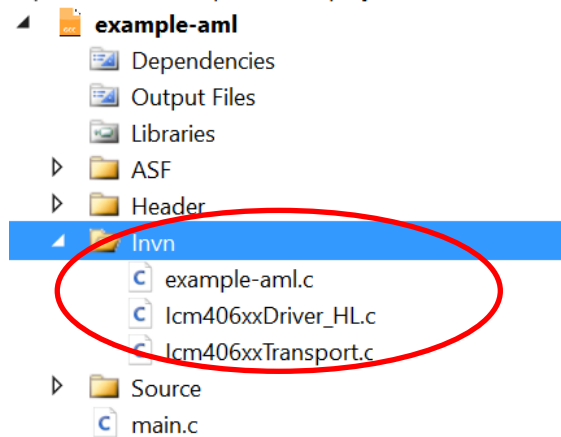
SOURCE FILES	FOLDER	DESCRIPTION	NOTE
libInvnAlgoAML_627_gcc-arm-none-eabi-cm0-1.4.0.a	03_Algo_lib	Algorithm library	No need to change (select the library that applies to your platform for porting.)
libInvnAlgoAML_627_gcc-arm-none-eabi-cm3-1.4.0.a			
libInvnAlgoAML_627_gcc-arm-none-eabi-cm4-fpu-1.4.0.a			
libInvnAlgoAML_627_iar-cm0-1.4.0.a			
libInvnAlgoAML_627_iar-cm3-1.4.0.a			
libInvnAlgoAML_627_iar-cm4-fpu-1.4.0.a			
libInvnAlgoAML_627_keil-cm0-1.4.0.a			
libInvnAlgoAML_627_keil-cm3-1.4.0.a			
libInvnAlgoAML_627_keil-cm4-fpu-1.4.0.a			
Invn_algo_aml.h	03_Algo_inc	Header file for AML algo	No need to change
Icm406xxDriver_HL.c	02_SensorAPI	Driver API	No need to change
Icm406xxDriver_HL.h	02_sensorAPI	Header of Driver API	No need to change
Icm406xxTransport.c	02_sensorAPI	Communication API	No need to change
Icm406xxTransport.h	02_sensorAPI	Header of Communication API	No need to change
Icm406xxVersion.h	02_sensorAPI	Header of driver version	No need to change
example_aml.c	04_example	Sensor and algorithm configuration API	Printer function and USB function adjust according to customer platform
example_main.c	04_example	Algorithm call main example	Example to be referenced

### 3 ALGORITHM INTEGRATION

The steps below use the Microchip studio platform as an example. For other platforms, apply the platform-specific method for the step below to implement the integration.

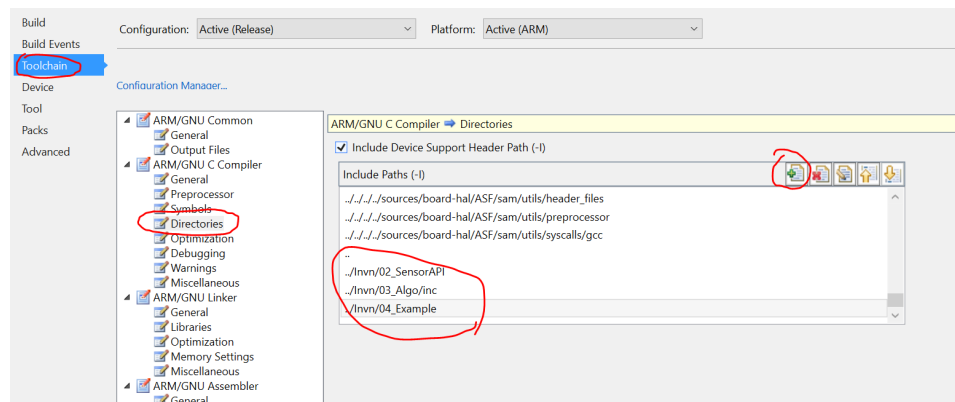
#### STEP 1: INCLUDE HEADER FILES

- Copy the software package to the target IDE projects, adding the source files below to target IDE platform:



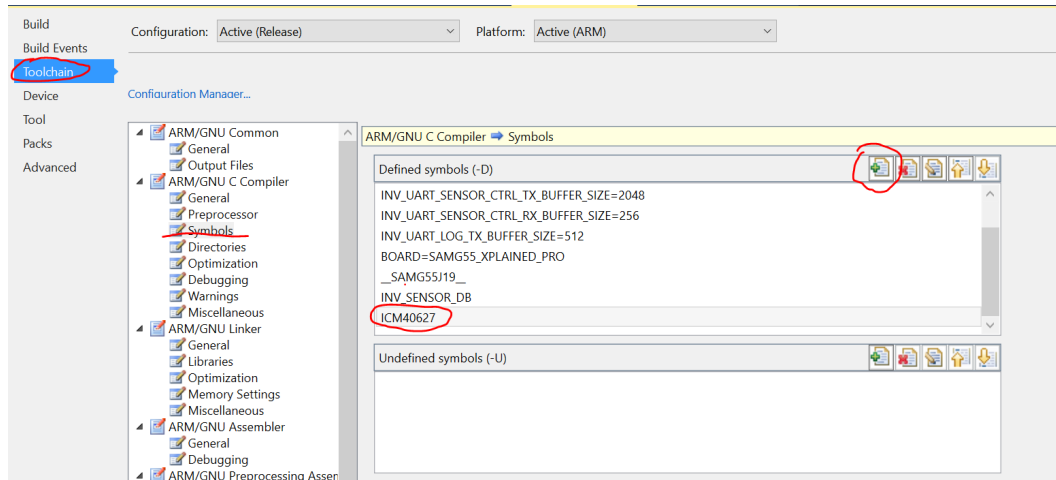
- Add the directory of the header files to the compiler directories.

Project→Properties→Toolchain→ARM/GNC C Compiler→Directorie→Include Paths



- Define ICM40627 macro globally.

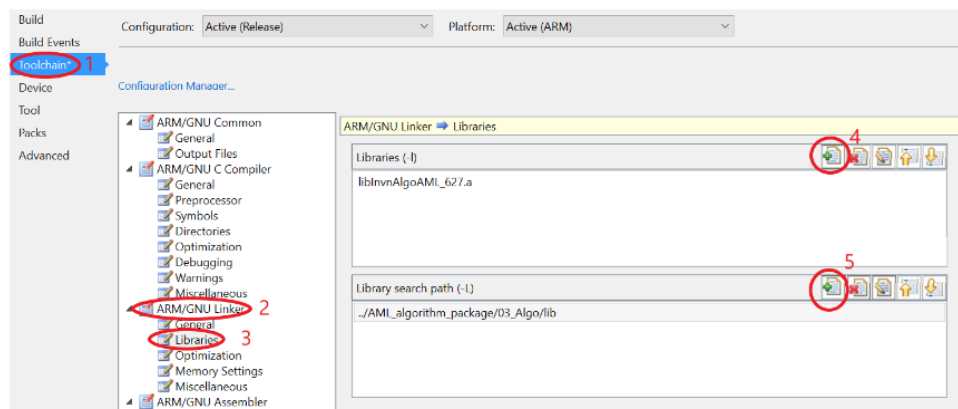
Project→Properties→Toolchain→ARM/GNC C Compiler→Symbols→



## STEP 2: LINK LIBRARY:

Link AML library and the path of library in the Linker. There are multiple libraries inside 03/Algo/lib, so please link the right library for your target platform.

Project→Properties→Toolchain→ARM/GNC Linker→Libraries→



## STEP 3: MODIFY EXAMPLE\_AML.C FILE OF PRINTER AND USB FUNCTION

Please implement the function below in your platform to enable to printer or modify the example\_aml.c to remove all the printer.

```
extern void msg_printer(const char * str, ...);
```

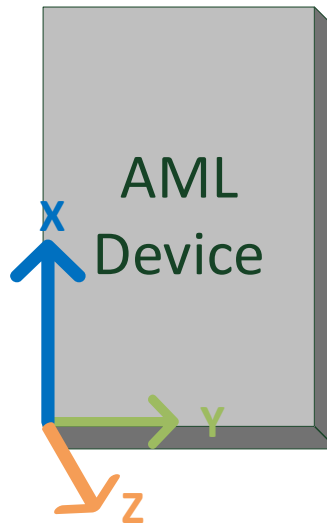
Replace or implement the two functions below to fulfill the mouse move and click. If you're using AML for other applications, you can also replace these two functions.

```
/* !!! Should be modified based on your own platform for usb hid function*/
inv_usb_hid_mouse_move(output.delta[0], output.delta[1]);

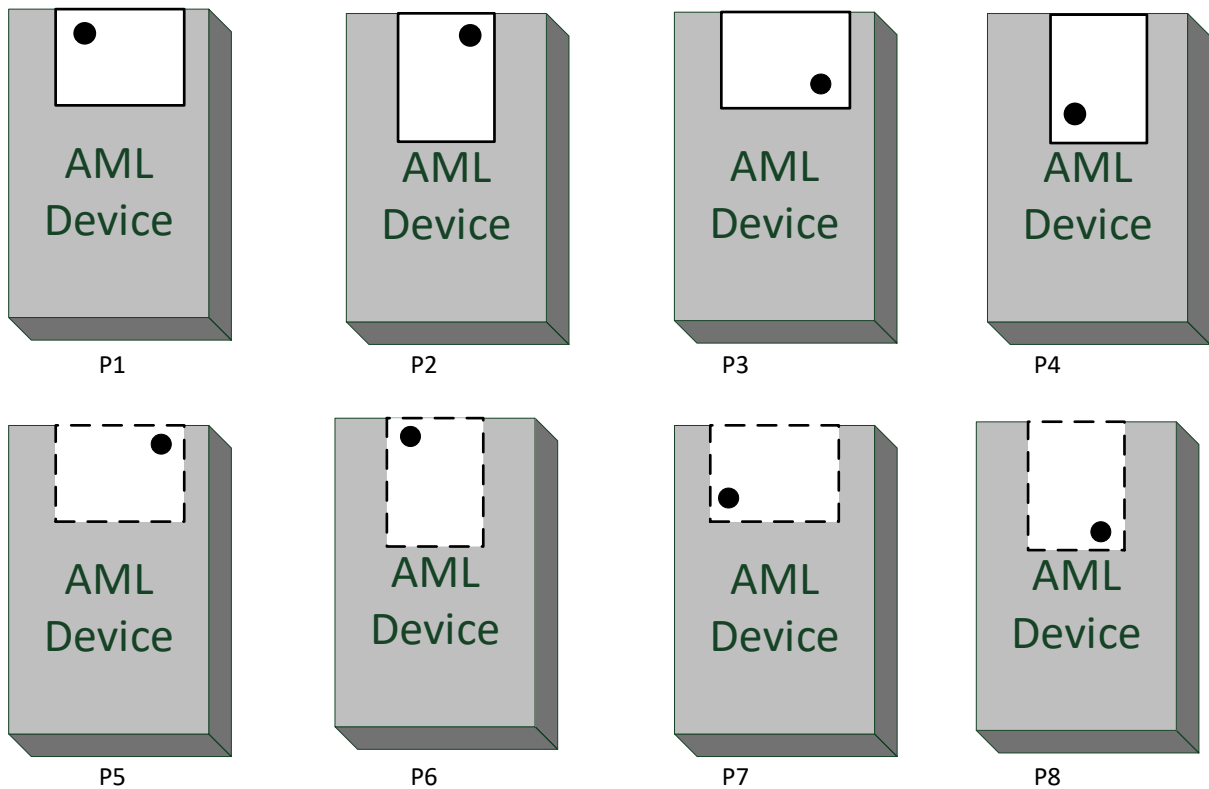
/* Update USB HID interface with the button click detected */
inv_usb_hid_mouse_button_click(input.click_button, false);
```

#### STEP 4: TRANSFER SENSOR AXIS TO BOARD/DEVICE AXIS

Sensor frame needs to transfer to device frame. AML device/board axis is defined below. In the figure below, +X is the axis that points to the screen.



Below are 8 possible positions for the sensor with the white rectangle representing the sensor. P1 to P4 have the sensor on the top layer of PCB, which aligns with the AML device. P5 to P8 have the sensor on the bottom layer of PCB which is on the back of the AML device. The dotted lines represent the transparent view from the top.



Below are the mounting matrixes that you can use in your project by selecting the corresponding matrix in your device:

P1:	Accel_matrix:	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$
P2	Accel_matrix:	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$
P3	Accel_matrix:	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$
P4	Accel_matrix:	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$
P5	Accel_matrix:	$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
P6	Accel_matrix:	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
P7	Accel_matrix:	$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
P8	Accel_matrix:	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	Gyro_matrix:	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Once you've decided on your matrix, replace the one in example\_aml.c:

```

/* ICM mounting matrix for AML referential.
 * For AML algorithm, accelerometer data are inverted compared to ICM convention,
 * It expects gravity minus acceleration and ICM measures acceleration minus
gravity */
static int32_t accel_mounting_matrix[9]= {0,    1,    0,
                                           1,    0,    0,
                                           0,    0,    1 };

static int32_t gyro_mounting_matrix[9]= { 0,   -1,    0,
                                           -1,    0,    0,
                                           0,    0,   -1 };

```



## STEP 5: IMPLEMENT READ/WRITE/SLEEP/DELAY FUNCTION

The four functions, which are needed by the driver, need to be implemented in your project. When you're implementing them, remember to keep the function definition identical.

Below is an example of how we implemented them in the microchip studio project.

```
int inv_icm406xx_io_hal_read_reg(struct inv_icm406xx_serif * serif, uint8_t reg,
uint8_t * rbuffer, uint32_t rlen)
{
    switch (serif->serif_type) {
        case ICM406XX_UI_SPI4:
            return inv_spi_master_read_register(INV_SPI_AP, reg, rlen, rbuffer);
        case ICM406XX_UI_I2C:
            while(inv_i2c_master_read_register(ICM_I2C_ADDR, reg, rlen, rbuffer)) {
                inv_delay_us(32000); // Loop in case of I2C timeout
            }
            return 0;
        default:
            return -1;
    }
}

int inv_icm406xx_io_hal_write_reg(struct inv_icm406xx_serif * serif, uint8_t reg,
const uint8_t * wbuffer, uint32_t wlen)
{
    int rc;

    switch (serif->serif_type) {
        case ICM406XX_UI_SPI4:
            for(uint32_t i=0; i<wlen; i++) {
                rc = inv_spi_master_write_register(INV_SPI_AP, reg+i, 1, &wbuffer[i]);
                if(rc)
                    return rc;
            }
            return 0;
        case ICM406XX_UI_I2C:
            while(inv_i2c_master_write_register(ICM_I2C_ADDR, reg, wlen, wbuffer)) {
                inv_delay_us(32000); // Loop in case of I2C timeout
            }
            return 0;
        default:
            return -1;
    }
}

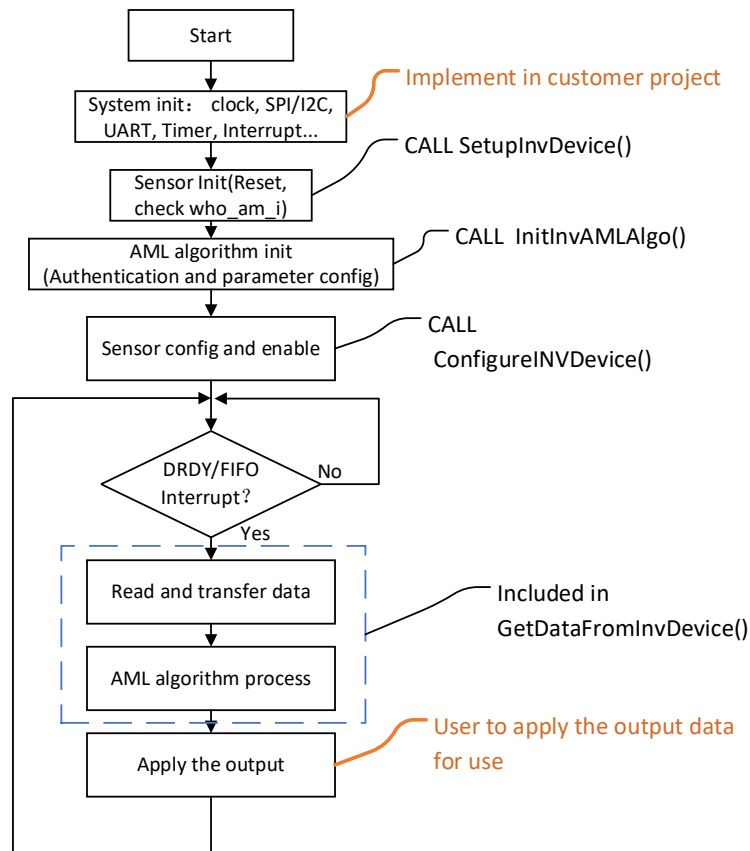
/*
 * Icm406xx driver needs to get time in us. Let's give its implementation here.
 */
uint64_t inv_icm406xx_get_time_us(void)
{
    return inv_timer_get_counter(TIMEBASE_TIMER);
}

/*
 * Icm406xx driver needs a sleep feature from external device. Thus
 */
inv_icm406xx_sleep_us
```

```
* is defined as extern symbol in driver. Let's give its implementation here.
*/
void inv_icm406xx_sleep_us(uint32_t us)
{
    inv_delay_us(us);
}
```

## STEP 6: IMPLEMENT THE FLOW FOR INITIALIZATION AND ALGORITHM PROCESSING

example\_main.c shows an example of the function calling flow. The flow chart is below:



The code example for the integration process follows:

```
int rc = 0;
struct inv_icm406xx_serif icm406xx_serif;

/* Initialize MCU */
rc = SetupMCUHardware(&icm406xx_serif);
/* Initialize ICM device */
rc |= SetupInvDevice(&icm406xx_serif);
/* Initialize algorithm */
rc |= InitInvAMLAlgo();
/* Configure ICM device */
rc |= ConfigureInvDevice();

check_rc(rc, "Error while configuring ICM device");
msg_printer("OK");
```

```
ready_to_go = true;
/* Print reminder on how to use example */
print_help();
msg_printer( "Start processing");
do {
    /* Check Icm406xx IRQ */
    if (irq_from_device & TO_MASK(INV_GPIO_INT1)) {
        inv_disable_irq();
        irq_from_device &= ~TO_MASK(INV_GPIO_INT1);
        inv_enable_irq();

        rc = GetDataFromInvDevice();
        check_rc(rc, "error while getting data from ICM device");

        /* Clear SW0 button IRQ */
        if (irq_event_sw0_button) {
            /* Button is active low */
            if (!inv_gpio_get_status(INV_GPIO_SW0_BUTTON)) {
                ClickButton(true);
            } else {
                ClickButton(false);

                inv_disable_irq();
                irq_event_sw0_button = false;
                inv_enable_irq();
            }
        }
    }
}

process_user_command();
} while(1);
```

#### **4 REVISION HISTORY**

REVISION DATE	REVISION	DESCRIPTION
03/25/2021	1.0	Initial release

This information furnished by InvenSense or its affiliates ("TDK InvenSense") is believed to be accurate and reliable. However, no responsibility is assumed by TDK InvenSense for its use, or for any infringements of patents or other rights of third parties that may result from its use. Specifications are subject to change without notice. TDK InvenSense reserves the right to make changes to this product, including its circuits and software, in order to improve its design and/or performance, without prior notice. TDK InvenSense makes no warranties, neither expressed nor implied, regarding the information and specifications contained in this document. TDK InvenSense assumes no responsibility for any claims or damages arising from information contained in this document, or from the use of products and services detailed therein. This includes, but is not limited to, claims or damages based on the infringement of patents, copyrights, mask work and/or other intellectual property rights.

Certain intellectual property owned by InvenSense and described in this document is patent protected. No license is granted by implication or otherwise under any patent or patent rights of InvenSense. This publication supersedes and replaces all information previously supplied. Trademarks that are registered trademarks are the property of their respective companies. TDK InvenSense sensors should not be used or sold in the development, storage, production or utilization of any conventional or mass-destructive weapons or for any other weapons or life threatening applications, as well as in any other life critical applications such as medical equipment, transportation, aerospace and nuclear instruments, undersea equipment, power plant equipment, disaster prevention and crime prevention equipment.

©2021 InvenSense. All rights reserved. InvenSense, MotionTracking, MotionProcessing, MotionProcessor, MotionFusion, MotionApps, DMP, AAR, and the InvenSense logo are trademarks of InvenSense, Inc. The TDK logo is a trademark of TDK Corporation. Other company and product names may be trademarks of the respective companies with which they are associated.



©2021 InvenSense. All rights reserved.