Automotive MEMS interface for AutoDevKit applications with SPC5 MCU discovery boards

Introduction

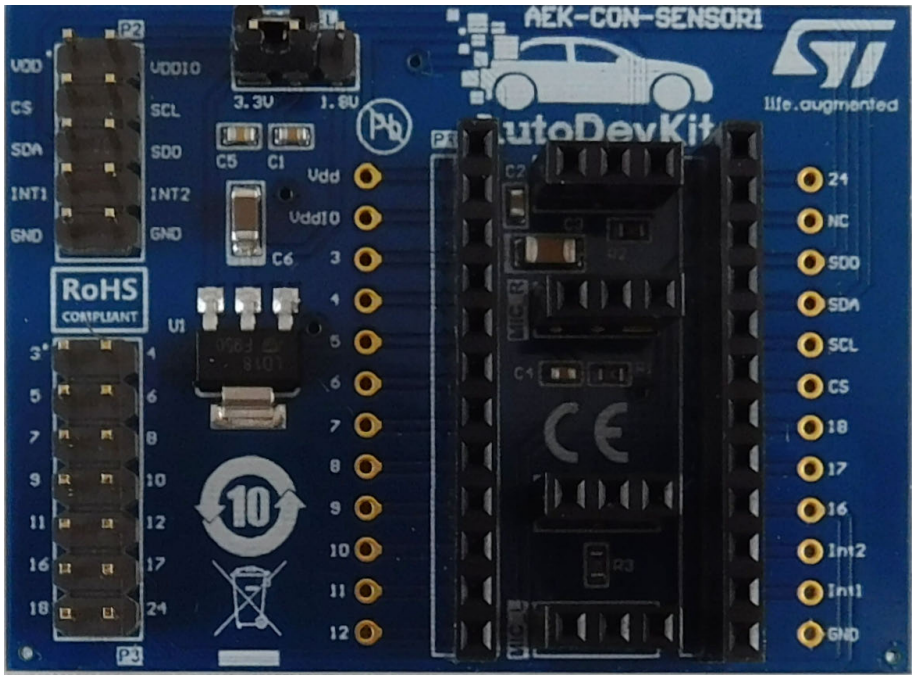
The AEK-CON-SENSOR1 connector board is designed to interface MEMS sensor boards in DIL 24 socket to an SPC5 MCU discovery board like the AEK-MCU-C4MLIT1 hosting a SPC58EC80E5 Chorus family automotive MCU with 4 MB flash.

The connector board includes a 1.8 V LDO voltage regulator to supply MEMS boards.

Two male strip connectors allow connecting the AEK-MCU-C4MLIT1: the main one (5x2) is used for power supply, SPI interface and signal interrupt, whereas the second one (7x2) is used for optional GPIO connections depending on the plugged MEMS sensor board.

For the MEMS sensor board connection, three female strip connectors are used: the main one (12x2) hosts the MEMS sensors in DIL 24 socket, whereas the other two (3x2) host digital microphones.

Figure 1. AEK-CON-SENSOR1



1 Overview

The AEK-CON-SENSOR1 interface board allows quick and easy connection of a variety of MEMS sensors in DIL-24 sockets with discovery boards embedding automotive-grade 32-bit SPC5 microcontrollers.

The board can host several industrial and automotive-grade MEMS sensors, allowing developers to quickly prototype safety and non-safety automotive applications using dedicated AutoDevKit sensor driver software for the following performance MEMS sensors:

- AIS2DW12: ultra-low-power 3-axis accelerometer for automotive applications
- ASM330LHH: automotive 6-axis inertial module: 3D accelerometer and 3D gyroscope
- ASM330LHHX: automotive 6-axis inertial module with embedded machine learning core and dual operating modes
- IIS2ICLX: high accuracy, high resolution, low-power, 2-axis digital inclinometer with embedded Machine Learning Core
- IIS3DWB: ultra-wide bandwidth, low-noise, 3-axis digital vibration sensor

Note: The drivers available in AutoDevKit can be easily extended to any ST MEMS sensors with similar architecture to those listed above.

2 Sensor connection procedure

The following table summarizes the connectors available on the interface board.

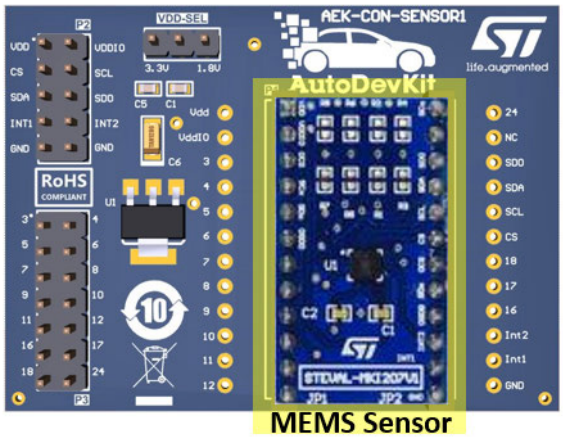
Table 1. AEK-CON-SENSOR1 components

Symbol on PCB	Description
P1	Connector dedicated to plug the MEMS sensor board in DIL-24 socket.
P2-P3	Connector dedicated to the communication with the MCU Board.
VDD-SEL	Jumpers to select the MEMS sensor board supply (VDD 3.3V or 1.8V).

Follow the procedure below to interface your SPC5 Chorus MCU discovery board with the sensor board you wish to evaluate.

Step 1. Place the sensor board on the DIL 24 connector, marked as P1 in the board serigraphy.

Figure 2. Connecting sensor board to interface board



Step 2. Connect the SPC5 discovery board via connectors P2 and P3 (if required).

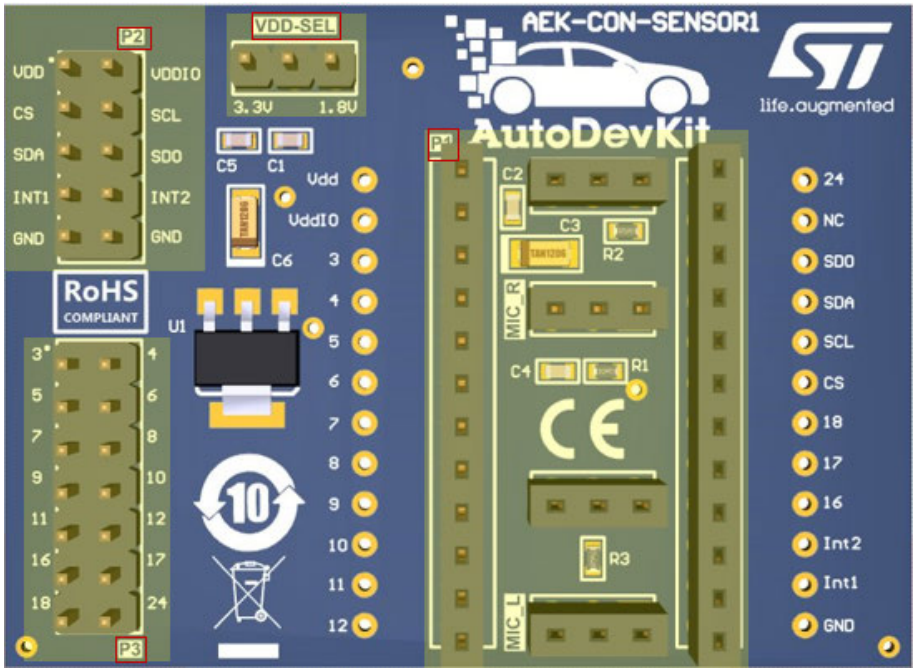
Only P2 connection is necessary in most cases as it provides the SPI communication pins (CS, SCL, SDA and SDO), the interrupt pins (INT1 and INT2), ground pin (GND) and supply pins (VDD and VDDIO). Some non-standard interface sensors will require the extra pin-out available on P3 for suitably interfacing with the MCU board.

Step 3. Use the dedicated jumpers on VDD-SEL to adjust the supply to 3.3 V or to 1.8 V, according to the supply requirements of the sensor.

Step 4. Ensure the GND pin is connected.

Step 5. Supply 3.3 V through the VDD and VDDIO pins of P2.

Figure 3. AEK-CON-SENSOR1 pin details



To evaluate another sensor, simply swap the sensor board and adjust the supply voltage via the VDD-SEL jumpers, if necessary.

There are dedicated debug holes around the P1 connector that can be used to verify sensor signals.

The hardware is fully supported by a software ecosystem, which includes SPC5-STUDIO development environment, UDE Debug and firmware download tool and STSW-AUTODEVKIT Eclipse plug-in containing AEK-CON-SENSOR1 driver and sample application codes.

During the test of the sensor, you might have to physically move the AEK-CON-SENSOR1 board to trigger the interrupts, e.g. during the evaluation of accelerometers and gyroscopes. To ensure the connection is stable, you should use a cable instead of single female-to-female jumper wires.

3 AutoDevKit software library for automotive MEMS sensors

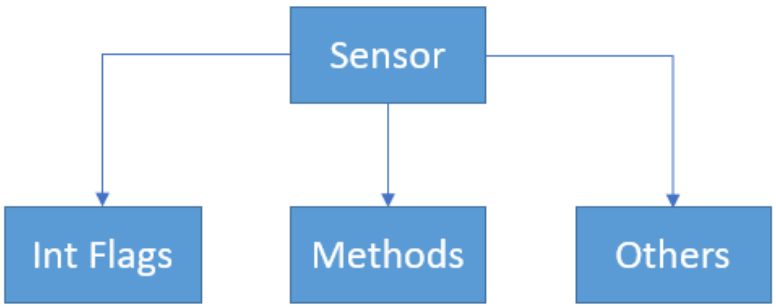
The drivers for the MEMS sensors supported by AEK-CON-SENSOR1 are included in a component of the AutoDevKit software (STSW-AUTODEVKIT) version 1.4.0. The library is written in MISRA C and the target software is generated automatically according to the code generation and pin allocation paradigm included in AutoDevKit design flow.

The drivers are structured so that they can be easily adapted to any ST MEMS sensor not directly supported in the software library.

3.1 Library features

The key component of each driver is a `Sensor STRUCT` containing a field called `methods` that can be accessed to select the API you want to use. The field `methods` is also a `struct` whose fields are function pointers to the sensor APIs. A different `Sensor STRUCT` is available for each sensor which references the relevant APIs for that sensor. The sensor also contains two flags that can be accessed to determine whether an interrupt was asserted by the sensor through INTPin1 or INTPin2. Other fields are used internally.

Figure 4. Sensor STRUCT elements



Before creating a `Sensor STRUCT`, a specific Methods structure for the individual sensor must be created, using the appropriate API provided in the AutoDevKit library. Once completed, the `Sensor STRUCT` can be built through a dedicated API.

3.2 Example Sensor STRUCT creation and configuration

```
/* Methods and Sensor declaration */
ais2dw12_methods_t methods;
ais2dw12_sensor_t sensor;
/* Methods initialization */
ais2dw12_methods_init(&methods);
/* Sensor STRUCT initialization.
   You need to pass your SPI handler and configuration
   generated with SPC5Studio Tool */
ais2dw12_sensor_init(&sensor, &SPID1, &spi_config_CS0, &methods);
/* Sensor configuration.
   This and all the other APIs can be accessed
   through the Sensor's methods field */
sensor.methods->configure_sensor(&sensor, power_mode_12bit,
                                 continuous,_100Hz, _2g, low_pass_1);
```

The example source code above shows how to create the struct and how to use the configuration APIs.

1. First, the methods and sensor structs are declared from the definitions in ais2dw12_api.h, according to the family the sensor belongs to:
   a. `ais2dw12_methods_t`
   b. `ais2dw12_sensor_t`
2. Then, from the same `.h` file, we invoke the APIs:
   a. `ais2dw12_methods_init`, which takes the methods struct address argument and initializes the struct itself.
   b. `ais2dw12_sensor_init`, which takes the sensor struct address argument (the struct to initialize in this case) and two other parameters according to the chosen SPI configuration set by the low-level driver component for the MCU. In this case, we pass the address of the SPIDriver structure `SPID1` because DSPI0 has been set and the address of the SPIConfig struct `spi_config_CS0`, taking the name assigned during the configuration of the specific chip select. These structures are automatically generated by AutoDevKit after configuring the GUI related to the SPI communication and can be found in the files spi_lld.c and spi_lld_cfg.c, respectively.

After the creation of the sensor struct, all the related APIs can be called via the name of the sensor and access the method field using the dot notation to show all the available APIs. In the previous example, the `configure_sensor` API is selected (you can see its description by hovering the mouse over the name of the API).

4 AutoDevKit ecosystem

Application development using the AEK-CON-SENSOR1 is based on the AutoDevKit ecosystem and the following components:

- SPC5-STUDIO Integrated Development Environment (IDE)
- SPC5-UDESTK-SW programmer and debugger
- AutoDevKit software library (STSW-AUTODEVKIT)
- AEK-CON-SENSOR1 driver

4.1 SPC5-STUDIO

SPC5-STUDIO is an integrated development environment (IDE) based on Eclipse designed to assist the development of embedded applications based on SPC5 Power Architecture 32-bit microcontrollers. The package includes an application wizard to initiate projects with all the relevant components and key elements required to generate the final application source code. It also contains straightforward software examples for each MCU peripheral.

Other advantages of SPC5-STUDIO are:

- ability to integrate other software products from the standard Eclipse marketplace
- free license GCC GNU C Compiler component
- support for industry-standard compilers
- support for multi-core microcontrollers
- PinMap editor to facilitate MCU pin configuration

Download the SPC5-UDESTK-SW software to run and debug applications created with SPC5-STUDIO.

4.2 STSW-AUTODEVKIT

The STSW-AUTODEVKIT plug-in for Eclipse extends SPC5-STUDIO for automotive applications. The main advantages are:

- Integrated hardware and software components, component compatibility checking and MCU and peripheral configuration tools
- Allows creation of new system solutions from existing solutions by adding or removing compatible function boards
- Hardware abstraction means new code can be generated immediately for any compatible MCU
- High-level application APIs to control each functional component, including the ones for the AEK-CON-SENSOR1 board.
- The GUI helps configure interfaces, including SPI, and can automatically manage all relevant pin allocation and deallocation operations.

RELATED LINKS

For more information, please refer to UM2623 (esp. ch 6 and 7)

You can also watch this video

4.2.1 Accessing the AEK_CON_SENSOR1 Component

The AEK_CON_SENSOR1 Component RLA with relevant drivers is available in STSW-AUTODEVKIT version 1.4.0 onwards.

Step 1. Go to the configuration portion of the component and add an entry on the Board List, selecting sensor family, DSPI and CS.

Step 2. Press the allocation button

AutoDevKit will allocate all needed pins on the MCU board included in the project.

Step 3. Use the PinMap editor to verify that all the relevant pins have been allocated (4 for the SPI and 2 for Interrupts).

Step 4. Use the Board Viewer to determine how to connect those pins to the selected MCU Board.

Step 5. Proceed to complete the `main()` function and build the application.

For each board added in the board list of component configuration GUI, SPC5Studio automatically creates a sensor struct of the selected family. The struct creation source code shown in Example 1 is automatically generated by AutoDevKit, therefore it sufficient to call the `init_mems()` API.

RELATED LINKS

4 AutoDevKit ecosystem on page 7

4.3 How to extend the sensor library to include a new family

Understanding the software structure helps to understand how to extend the library to new sensor part numbers.

As seen before, in the main file of the application, the `AEK_CON_SENSOR1.h` must be included to use the component features. It includes the dynamically generated component configuration, including in turn the related sensor libraries.

For example, we plan to use two AEK_CON_SENSOR1 interface boards with two AIS2DW12 sensors, so the configuration file will only include the drivers for this specific sensor, as shown in Figure 5. Software Architecture diagram based on AIS2DW12 sensor.

For simplicity, we will limit our focus to the AIS2DW12 family library, as drivers related to the other supported families have the same structure.

There are four files:

- two shared by all families (MEMS_API and COMMUNICATION)
- two related to the specific family (AIS2DW12_API and AIS2DW12_REG).

The content of each file is summarized in the following table.

Table 2. Sensor library file descriptions

File	Content
COMMUNICATION	Contains the primitive functions `write_register` and `read_register` used to write and read a sensor register through SPI communication, calling the SPI Low Level Drivers functions of the MCU. They are used by functions defined in AIS2DW12_REG file.
AIS2DW12_REG	Contains functions that read and write specific registers of the AIS2DW12 sensor family; e.g., `ais2dw12_get_device_id` reads the WHO_AM_I register.
MEMS API	Contains enumeration, return types and function pointer definitions that are shared by all sensor families.
AIS2DW12_API	Contains the high-level API definitions for the AIS2DW12 sensor family; e.g., `ais2dw12_methods_init`, `ais2dw12_sensor_init` and `configure_sensor`, previously seen in Example 1. It also includes the `ais2dw12_methods_t` and `ais2dw12_sensor_t` declarations seen previously.

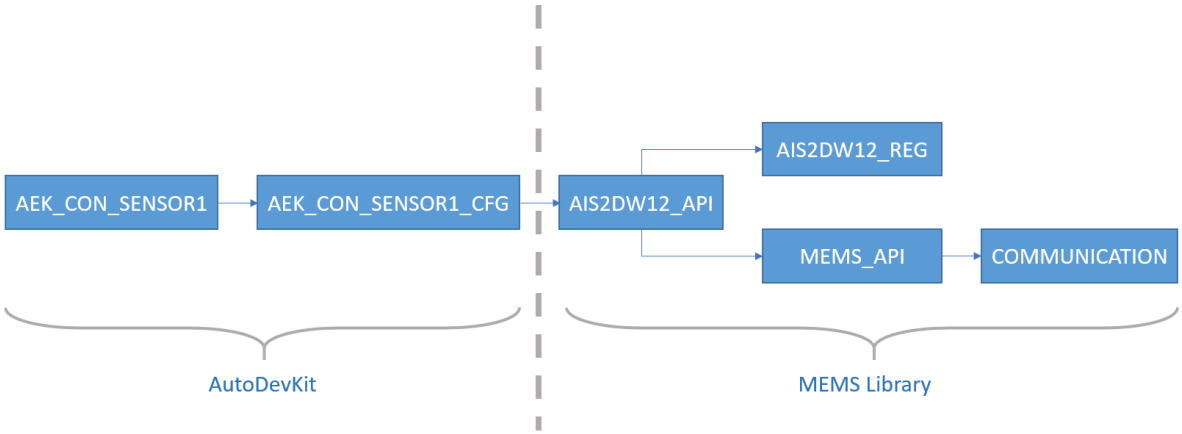
Note: The Methods structure only consists of function pointers declared in the MEMS_API file. The `ais2dw12_methods_init` API is used to assign to each function pointer the respective API defined in the AIS2DW12_API file.

The API file includes the Sensor and Methods struct definitions, as well as the API definitions based on register access functions defined in the REG file. The MEMS_API file stores general information common to all sensors, including the definition of APIs function pointers. Finally, the COMMUNICATION file contains primitives needed for SPI communication.

This architecture can be easily reused to extend the library to new sensors. You can modify the REG and API files to be compatible with different sensors by adding your APIs. The newly inserted method pointers must be added in the `MEMS_API` file.

Note: If you customize the library, you will not be able to use the features of AutoDevKit, so you must create your structures manually with `methods_init` and `sensor_init` APIs, appropriately modified.

Figure 5. Software Architecture diagram based on AIS2DW12 sensor



Note: File names are actually lower case, and every file has a header (.h) in addition to the .c (except the MEMS_API file that is just a header file).

RELATED LINKS

3.2 Example Sensor STRUCT creation and configuration on page 5

4.4 Supported sensors

The sensors currently supported are:

- AIS2DW12: ultra-low-power 3-axis accelerometer for automotive applications
- ASM330LHH: automotive 6-axis inertial module: 3D accelerometer and 3D gyroscope
- ASM330LHHX: automotive 6-axis inertial module with embedded machine learning core and dual operating modes
- IIS2ICLX: high-accuracy, high-resolution, low-power, 2-axis digital inclinometer with embedded machine learning core
- IIS3DWB: ultra-wide bandwidth, low-noise, 3-axis digital vibration sensor

Table 3. Sensor characteristics

Product	AIS2DW12	ASM330LHH	IIS2ICLX	IIS3DWB
Type	3D accelerometer	3D accelerometer and 3D gyroscope	2D inclinometer	3D vibration sensor
Full scale	±2 g, ±4 g	Up to ±16 g and up to ±4000 dps	Up to ±3 g	Up to ±16 g
Output data rate	up to 100 Hz	Up to 6.67 KHz	Up to 833 Hz	26.7 KHz
Event detection	Free-fall,wake-up, orientation change, activity-inactivity	Free-fall,wake-up, orientation change, activity-inactivity	Wake-up, single and double tap, activity-inactivity	Wake-up, activity-inactivity

RELATED LINKS

Appendix B Sensors supported by APIs on page 22

5 Sensor event detection

For each type of MEMS sensor, the user can configure two or more kinds of events, which can be reported to the MCU through the INT1 or INT2 interrupt pins (only INT1 for the AIS2DW12). To detect an event occurrence, you should therefore enable the corresponding Interrupt pin.

The following events are supported by the software library:

- Data-ready
- Free-fall
- Wake-up
- Orientation change
- Activity-inactivity
- Single and double tap

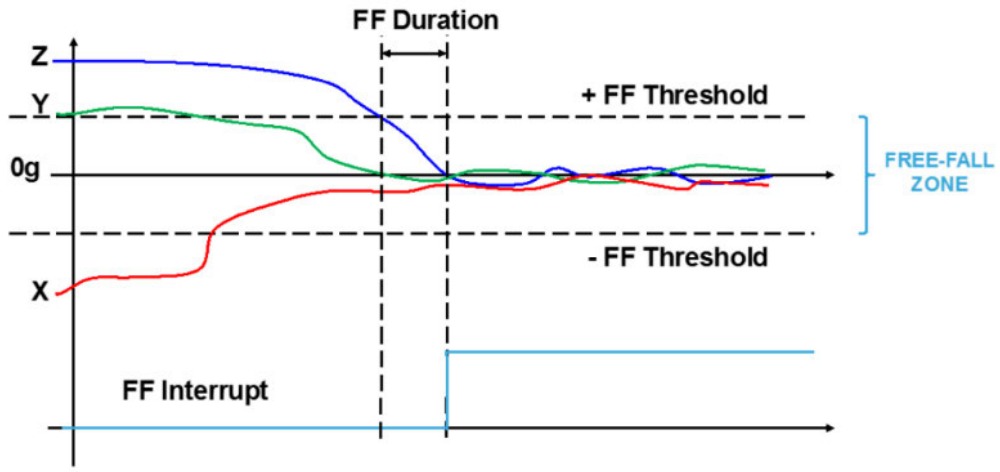
5.1 Data-ready

All supported MEMS sensors can detect the data-ready event, useful to understand if new data are available.

5.2 Free-fall

The AIS2DW12, ASM330LHH and ASM330LHHX MEMS sensors can detect when they are in free-fall according to selected thresholds and durations. The threshold determines the acceleration range of a free-fall condition. If the acceleration along the three axes fall into the free-fall zone for a certain time interval, a free-fall condition is signaled.

Figure 6. Free-fall signals

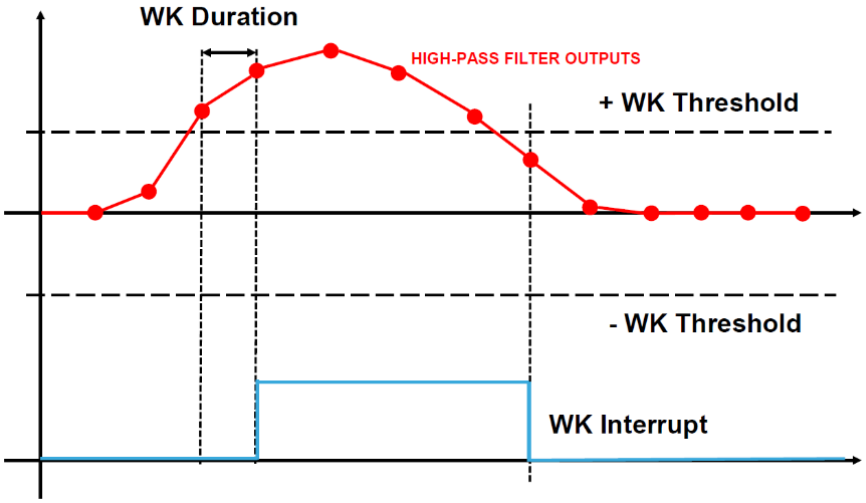


The FF Threshold and FF Duration are configurable parameters. The FF interrupt is generated when all the accelerations are small enough to be in the FREE-FALL zone for an FF Duration time.

5.3 Wake-up

All supported MEMS sensors can detect wake-up events that are triggered when the sensor data exceeds set threshold and duration values.

Figure 7. **Wake-up signals**



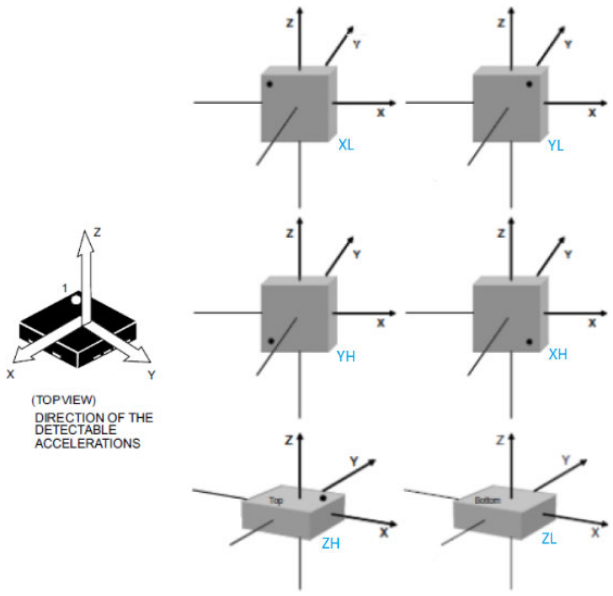
The WK Threshold and WK Duration are configurable parameters. A WK Interrupt signal is generated if a certain number of consecutive data exceed the configured threshold.

5.4 Orientation change

AIS2DW12, ASM330LHH and ASM330LHHX MEMS sensors can signal directional changes in their orientation when an angle threshold in degrees is exceeded for a mode indicating the number of dimensions considered, 6 or 4 (z axis disabled).

For example, there is a ZH (YH, XH) orientation change when the face perpendicular to the Z(Y,X) axis is almost flat and the magnitude of the acceleration measured on the Z(Y,X) axis is positive and greater than the threshold. Similarly, there is a ZL (YL, XL) orientation change when the face perpendicular to the Z(Y,X) axis is almost flat and the magnitude of the acceleration measured on the Z(Y,X) axis is negative and greater than the threshold.

Figure 8. **Sensor orientation**



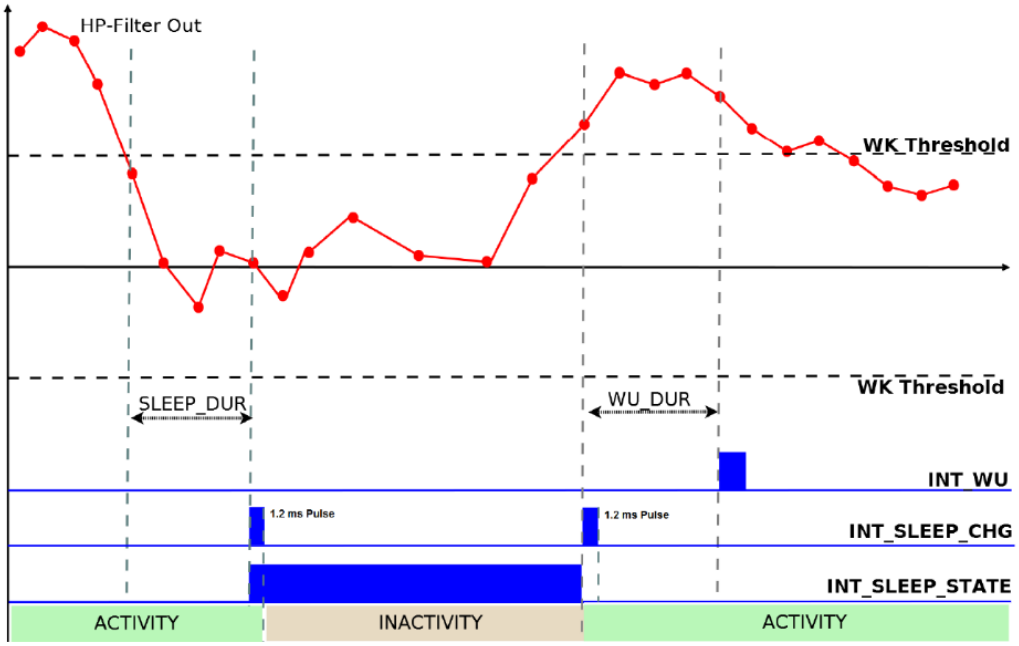
The OC Mode and OC Threshold are configurable parameters. The interrupt signal is asserted when only one axis exceeds the selected threshold and the acceleration values measured from the other two axes are lower than the threshold.

5.5 Activity-inactivity

All supported MEMS sensors can detect activity-inactivity events, which are especially for enabling AIS2DW12, ASM330LHH and ASM330LHHX power down modes. If inactivity is detected on these sensors, the output data rate frequency (and consequent power consumption) can be reduced until activity is detected.

Once the Wake-Up Threshold, Duration, and Sleep Duration are configured, if sensor data does not exceed the threshold for a Sleep Duration Time, the sensor enters inactivity mode, while a wake-up event pulls the sensor into activity mode. If configured, the mode change will trigger an interrupt.

Figure 9. Activity-inactivity signals

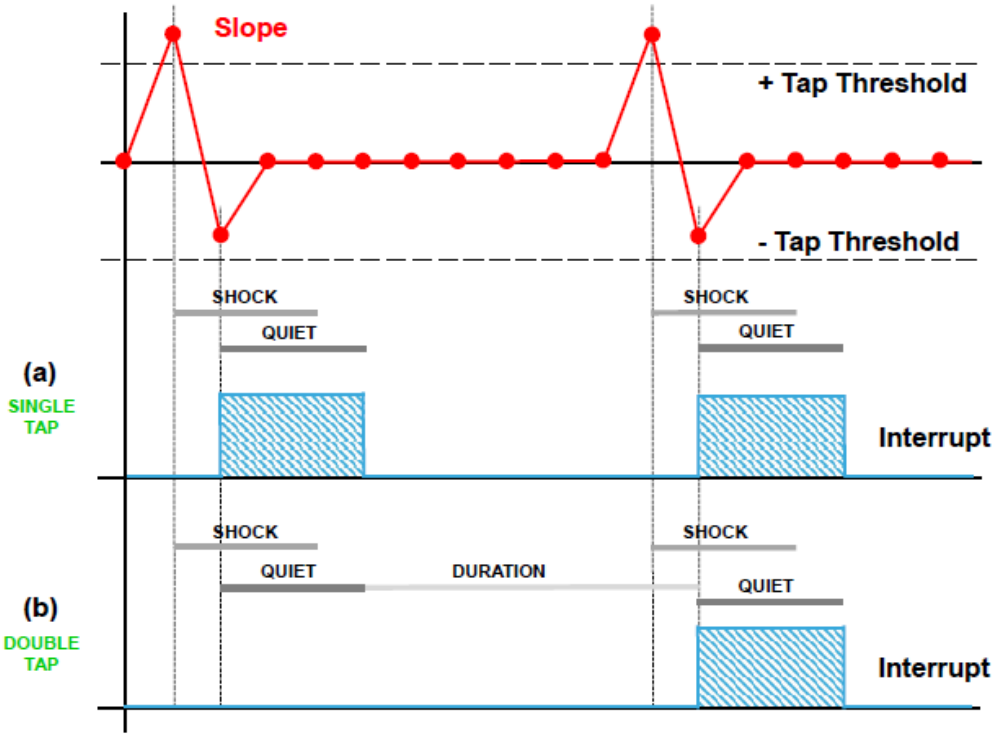


The AI Mode, WK Threshold, WK Duration, and Sleep Duration are configurable parameters. The interrupt signal is asserted when the device switches from one sleep state to another, so where a sleep change occurs.

5.6 Single and double tap

The IIS2ICLX sensor can detect single and double tap events. A tap event is detected when data exceeds the programmed threshold and then returns to the area delimited by the threshold itself within a set Shock time. A double tap event is detected if there is a single tap and then another after a Quiet time interval, but before the end of the Duration time window.

Figure 10. **Tap description**



The Tap Threshold, Shock Time, Quiet Time, and Duration Time are configurable parameters. A single tap interrupt is generated when the slope data of the selected channel exceeds the programmed threshold and returns below it within the Shock time window. After the first tap has been recognized, the second tap detection procedure is delayed for an interval defined by the Quiet time. This means that after the first tap has been recognized, the second tap detection procedure is only triggered when the slope data persists for longer than the Quiet window time, but less than the Duration window time.

6 AEK-CON-SENSOR1 sample application

6.1 How to create a simple AEK-CON-SENSOR1 application

In this example, we will create an application to detect when an AIS2DW12 MEMS sensor is in the free-fall condition. Our sensor board is plugged onto an AEK-CON-SENSOR1 board, which is also connected to an AEK-MCU-C4MLIT1 MCU discovery board.

Step 1. Create a new SPC5-STUDIO application for the SPC58EC series microcontroller and add the following components:

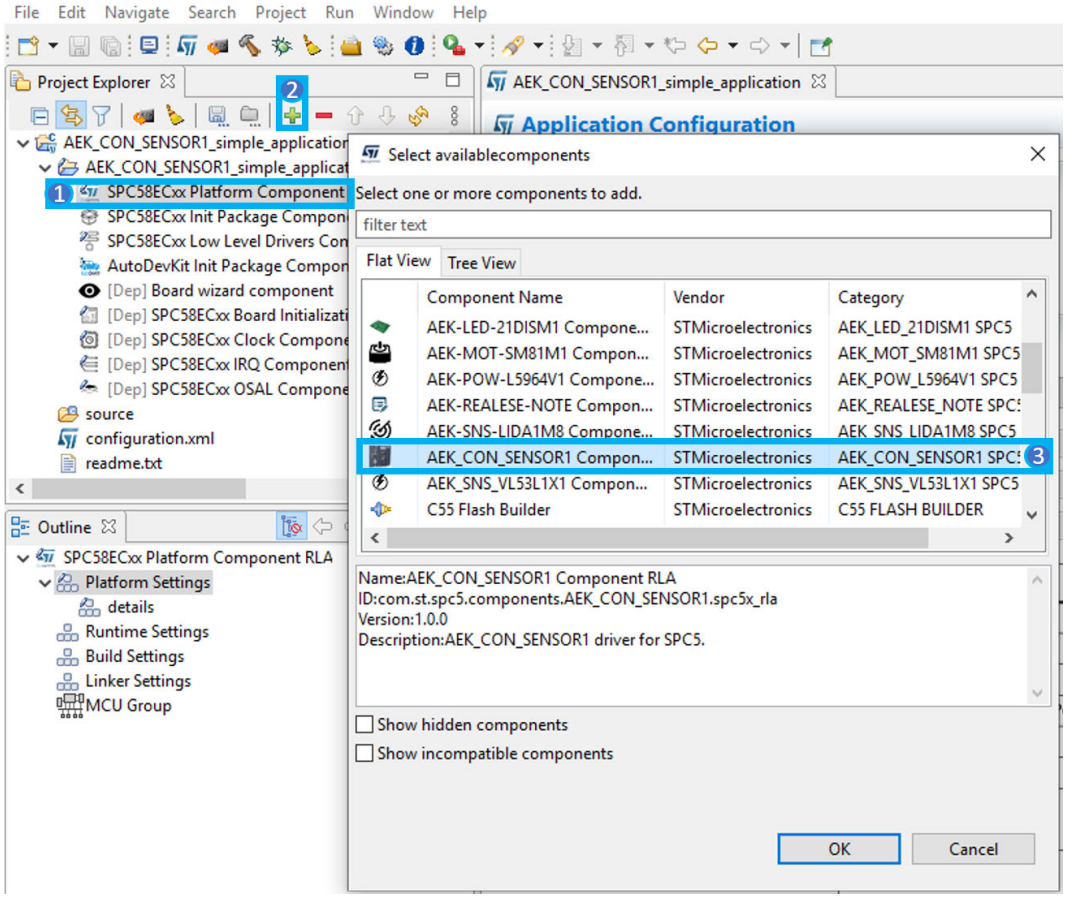
- SPC58ECxx Init Package Component RLA
- SPC58ECxx Low Level Drivers Component RLA

Step 2. Add the following additional components:

- AutoDevKit Init Package Component
- SPC58ECxx Platform Component RLA
- AEK-CON-SENSOR1 Component RLA

Figure 11. Adding AEK_CON_SENSOR1 Component in a SPC5Studio project

1. SPC58ECxx Platform Component RLA
2. Open available components
3. AEK-CON-SENSOR1 Component RLA

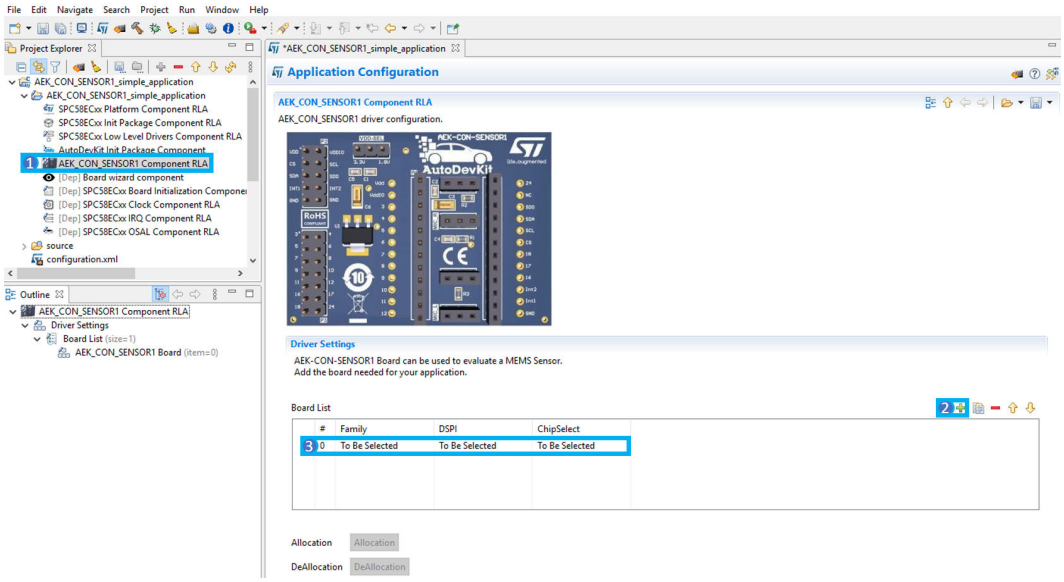


Step 3. Select [**AEK_CON_SENSOR1 Component RLA**] to open the [**Application Configuration**] window.

Step 4. Click on [+] to add a new element to the board list.

Figure 12. AEK_CON_SENSOR1 component configuration

1. AEK-CON-SENSOR1 component
2. add new element icon
3. new entry

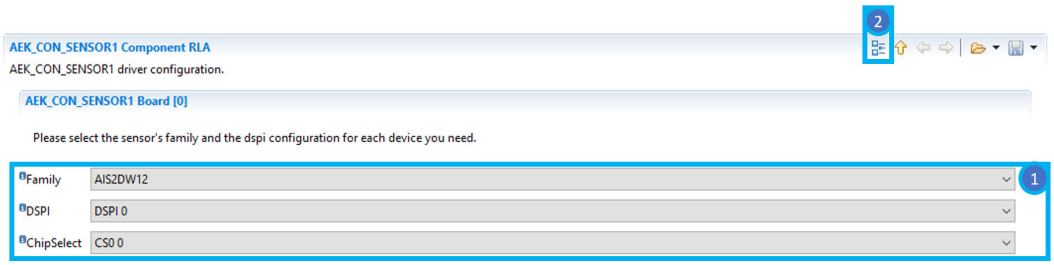


Step 5. Double click on the newly added element to configure the board.

Step 6. Select the sensor part number and configure the SPI selecting DSPI and CS.

Figure 13. AEK-CON-SENSOR1 sensor configuration

1. Select AIS2DW12 sensor family.
2. Select DSPI0 and CS0.
3. Confirm configuration



Step 7. Click the [**Allocation**] button below the AEK-CON-SENSOR1 list and click [**OK**] in the confirmation window.

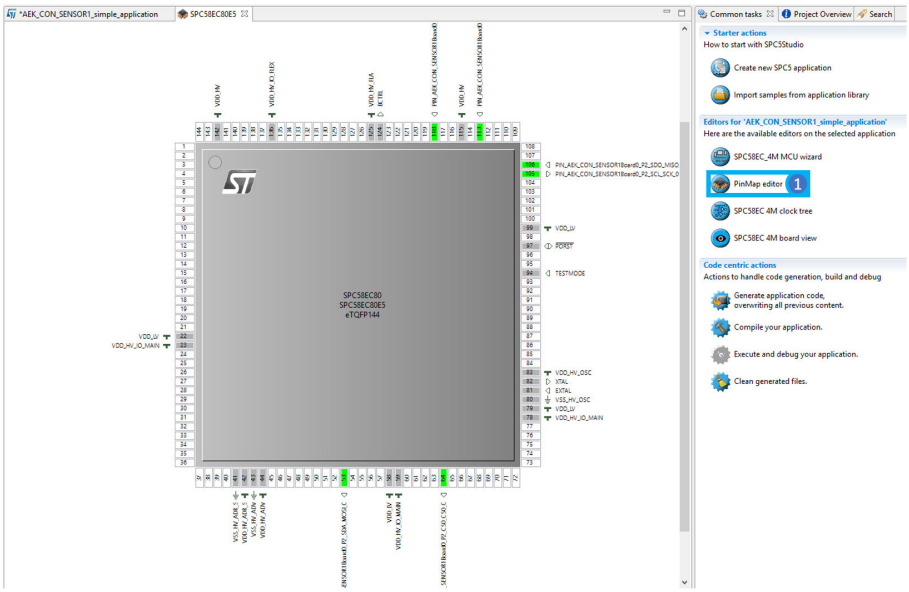
This operation delegates automatic pin allocation to AutoDevKit.

Note: The sensor struct allocated will have the name `ais2dw12_0`*, according to the rule:* `sensorStructName = sensorFamily_indexBoardsTable`.

Step 8. Click on the PinMap editor icon to check that required pins have been properly allocated:

- DSPI SCL
- DSPI MOSI
- DSPI MISO
- DSPI CS
- 2 EIRQ pins for interrupts.

Figure 14. AEK-CON-SENSOR1 PinMap display



Step 9. Close the PinMap Editor and save the application.

Step 10. Generate and build the application using the appropriate icons in SPC5-STUDIO.

Step 11. The project folder will be populated with new files, including main.c and the components folder with AEK-CON-SENSOR1 drivers.

Step 12. Open the main.c file and include AEK-CON-SENSOR1.h file.

```
#include "components.h"
#include "AEK_CON_SENSOR1.h"
/*
 * Application entry point.
 */
int main(void) {
/* Initialization of all the imported components in the order specified in
   the application wizard. The function is generated automatically.*/
  componentsInit();
  /* Enable Interrupts */
  irqIsrEnable();
  /* Sensor boot time waiting */
  osalThreadDelayMilliseconds(AIS2DW12_BOOT_TIME);
  /* MEMS initialization */
  init_mems();
    /* Sensor configuration */
  ais2dw12_0.methods->configure_sensor(&ais2dw12_0, power_mode_4,
                                       continuous, _25Hz, _2g, low_pass_1);
  ais2dw12_0.methods->enable_interrupt_for_event(&ais2dw12_0,
                                                 free_fall_to_pin_int_1);
  ais2dw12_0.methods->configure_interrupts_mode(&ais2dw12_0,latched_mode);
  ais2dw12_0.methods->configure_freefall(&ais2dw12_0, _250mg, 6);
  /* Application main loop.*/
  for ( ; ; ) {
  if(ais2dw12_0.flag_int1 == 1){
            /* Interrupt generated. Do something.*/
            /* Reset Interrupt 1 Flag.*/
      ais2dw12_0.flag_int1 = 0;
  }
  ff_result = ais2dw12_0.methods->detect_freefall(&ais2dw12_0);
  if(result == detected){
            /* Free Fall Detected. Do something. */
  }
  }
}
```

Step 13. Place the previous source code inside the main() function.

Step 14. Save, generate, and compile the application.

Step 15. Open the Board View Editor provided by AutoDevKit.

Step 16. This provides a graphical point-to-point guide on how to wire the boards.

Step 17. Connect the AEK-MCU-C4MLIT1 to a USB port on your PC using a mini-USB to USB cable.

Step 18. Launch SPC5-UDESTK-SW and open the debug.wsx file in the AEK-MCU-C4MLIT1– Application/UDE folder.

Step 19. Run and debug your code.

6.2 Available demos for AEK-CON-SENSOR1

Please update your AutoDevKit versions as more demos may be available with new AutoDevKit releases.

To read the sensor outputs, each demo includes a serial communication configuration. Connect a USB cable from the AEK-MCU-C4MLIT1 to your PC and use a console like Termite to read incoming data.

Note: Set the baud rate to 38400.

6.2.1 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect activity and get accelerations – Test application

This demo involves two AEK-CON-SENSOR1 boards:

with AIS2DW12 sensor to read the accelerations data and receive an interrupt when data are ready

with ASM330LHH sensor to monitor activity and inactivity events and receive an interrupt when the sleep state of the sensor changes.

The main used APIs in this demo are:

- `init_mems`: initializes the sensors structures and SPI interface.
- `configure_sensor`: to configure the sensor.
- `configure_offsets`: to set appropriate offsets for each axes of acceleration sample.
- `enable_interrupt_for_event`: to route an event to interrupt pin 1 or 2.
- `configure_interrupts_mode`: to choose between latched and pulsed interrupts.
- `get_accelerations`: to read acceleration samples.
- `detect_activity_inactivity`: to detect when activity or inactivity events occur.

6.2.2 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect free-fall – Test application

This demo involves one AEK-CON-SENSOR1 board with AIS2DW12 sensor to detect and receive an interrupt in a free-fall event.

The main used APIs in this demo are:

- `init_mems`: initializes the sensors structures and SPI interface.
- `configure_sensor`: to configure the sensor.
- `enable_interrupt_for_event`: to route an event to interrupt pin 1 or 2.
- `configure_interrupts_mode`: to choose between latched and pulsed interrupts.
- `configure_freefall`: to configure freefall detection event.
- `detect_freefall`: to detect if freefall event occurred.

Note: SPC582Bxx_RLA AEK-CON-SENSOR1 – Detect free-fall - Test application and SPC584Bxx_RLA AEK-CON-SENSOR1 – Detect free-fall - Test application demos implement the same functionality as this demo. The only difference is in the microcontroller used, as indicated in demo name.

6.2.3 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect tap – Test application

This demo involves oneAEK-CON-SENSOR1 board with an IIS2ICLX sensor to detect single tap events and receive the corresponding interrupts.

The main used APIs in this demo are:

- `init_mems`: initializes the sensors structures and SPI interface.
- `configure_sensor`: to configure the sensor.
- `enable_interrupt_for_event`: to route an event to interrupt pin 1 or 2.
- `configure_interrupts_mode`: to choose between latched and pulsed interrupts.
- `configure_tap`: to configure tap detection event.
- `detect_tap`: to detect if tap event occurred.

6.2.4 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect wake-up and orientation change – Test application

This demo involves two AEK-CON-SENSOR1 boards:

with an AIS2DW12 sensor to detect wake-up events and receive corresponding interrupts.

with an IIS3DWB sensor to detect orientation change events and receive corresponding interrupts.

The main used APIs in this demo are:

- `init_mems`: initializes the sensors structures and SPI interface.
- `configure_sensor`: to configure the sensor.
- `enable_interrupt_for_event`: to route an event to interrupt pin 1 or 2.
- `configure_interrupts_mode`: to choose between latched and pulsed interrupts.
- `configure_orientation_change`: to configure orientation change event.
- `configure_wakeup`: to configure wake-up event.
- `detect_orientation_change`: to detect if orientation change event occurred.

- `detect_wakeup`: to detect if wake-up event occurred.

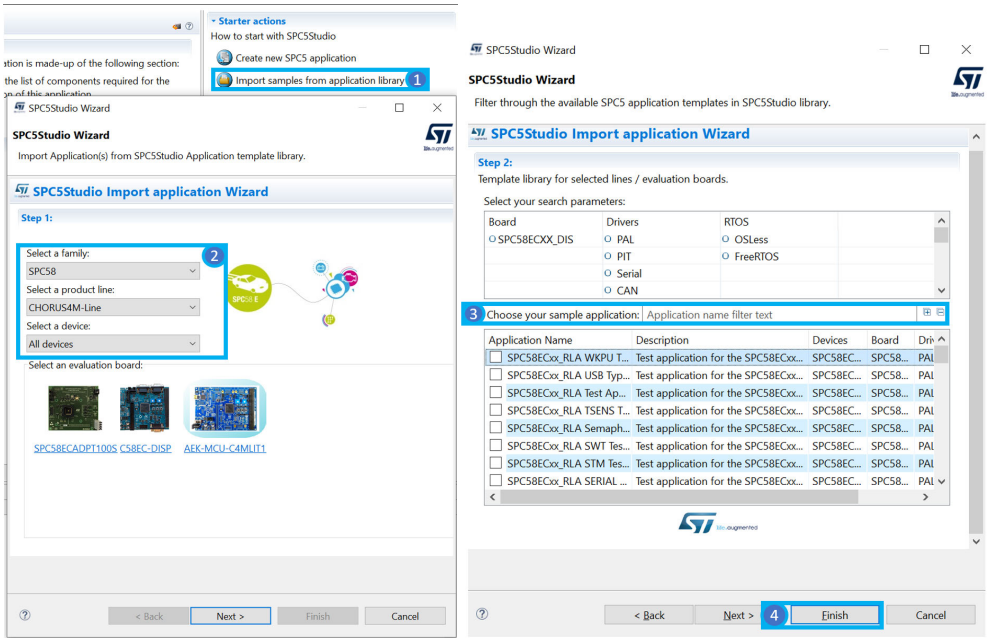
6.3 How to upload the demos for AEK-CON-SENSOR1

Follow the procedure below to import the demos into SPC5-STUDIO.

Step 1. Select [**Import samples from application library**] from the Common tasks pane. An Import application Wizard appears.

Step 2. Insert the appropriate product MCU family details.

Figure 15. AEK-CON-SENSOR1 demo upload



Step 3. Select the desired application from the library.

Step 4. Click the [**Finish**] button.

Appendix A Available APIs for AEK-CON-SENSOR1 board

init_mems:

Initializes the sensor structs and SPI protocol. The rule used to name the structs is: sensorStructName = sensorFamily_indexBoardsTable. For example, if you use two AEK-CON-SENSOR1 boards where the first holds an ASM330LHH MEMS sensor plugged and the second holds an IIS3DWB, then the struct sensor names will be `asm330lhh_0` and `iis3dwb_1` respectively.

configure_sensor:

Configures base sensor parameters: power mode, output data rate, full scale and others if required by the specific product.

configure_interrupts_mode:

Chooses between latched interrupts (default), where the signal is reset after reading the relevant register, and pulsed interrupts, where the signal is automatically reset when the interrupt condition no longer persists or after a certain amount of time.

enable_interrupt_for_event:

Enables interrupt generation through INTPin1 or INTPin2 from the sensor when a chosen event happens.

configure_offsets:

Configures the data offsets for the data sampled by the sensor along all available axes.

get_accelerations:

Gets the accelerations along all available axes.

get_accelerations_and_angular_rates:

Gets the accelerations and angular rates along all available axes (for gyroscope type only).

configure_freefall:

Configures the free-fall Threshold and Duration parameters.

detect_freefall:

Detects free-fall events according to the chosen configuration in the appropriate register.

configure_wakeup:

Configures the wake-up Threshold and Duration parameters.

detect_wakeup:

Detects wake-up events according to the chosen configuration in the appropriate register.

configure_orientation_change:

Configures orientation change parameters: you can choose between 6-4 dimensions mode and a Threshold.

detect_orientation_change:

Detects orientation change events and direction according to the chosen configuration in the appropriate register.

configure_activity_inactivity:

Configures the activity-inactivity mode, wake-up threshold, wake-up and sleep duration parameters.

detect_activity_inactivity:

Detects activity/inactivity events according to the chosen configuration in the appropriate register.

configure_tap:

Configures single and double tap threshold and shock, quiet and duration time parameters. If single mode is selected, unnecessary parameters are ignored.

detect_tap:

Detects single and double tap events according to the chosen configuration in the appropriate register.

read_register:

Performs an SPI read of a specific register.

write_register:

Performs an SPI write in a specified register.

Appendix B Sensors supported by APIs

Note: Configure each sensor with the `configure_sensor` API before using it. Configure an event and enable its interrupt before trying to detect it. Advanced features of each sensor can be directly accessed with `read_register` and `write_register` APIs.

Data bit Default value

Table 4. APIs and supported sensors

API	Supporting sensor order codes
configure_sensor	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB
configure_interrupts_mode	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB
enable_interrupt_for_event	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB
configure_offsets	AIS2DW12, IIS2ICLX, IIS3DWB
get_accelerations	AIS2DW12, IIS2ICLX, IIS3DWB
get_accelerations_and_angular_rates	ASM330LHH
configure_freefall	AIS2DW12, ASM330LHH, ASM330LHHX
detect_freefall	AIS2DW12, ASM330LHH, ASM330LHHX
configure_wakeup	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB
detect_wakeup	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB
configure_orientation_change	AIS2DW12, ASM330LHH, ASM330LHHX
detect_orientation_change	AIS2DW12, ASM330LHH, ASM330LHHX
configure_activity_inactivity	AIS2DW12, ASM330LHH, IIS2ICLX, IIS3DWB
detect_activity_inactivity	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB
configure_tap	IIS2ICLX
detect_tap	IIS2ICLX
read_register	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB
write_register	AIS2DW12, ASM330LHH, ASM330LHHX, IIS2ICLX, IIS3DWB

Revision history

Table 5. **Document revision history**

Date	Version	Changes
10-Feb-2021	1	Initial release.
26-Sep-2022	2	Updated Section 1 Overview, Section 4.4 Supported sensors, Section 5.2 Free-fall, Section 5.4 Orientation change, Section 5.5 Activity-inactivity, and Section Appendix B Sensors supported by APIs. Added references to the ASM330LHHX sensor.

Contents

1 **Overview** 2
2 **Sensor connection procedure** 3
3 **AutoDevKit software library for automotive MEMS sensors** 5
3.1 Library features 5
3.2 Example Sensor STRUCT creation and configuration 5
4 **AutoDevKit ecosystem** 7
4.1 SPC5-STUDIO 7
4.2 STSW-AUTODEVKIT 7
4.2.1 Accessing the AEK_CON_SENSOR1 Component 7
4.3 How to extend the sensor library to include a new family 8
4.4 Supported sensors 9
5 **Sensor event detection** 10
5.1 Data-ready 10
5.2 Free-fall 10
5.3 Wake-up 10
5.4 Orientation change 11
5.5 Activity-inactivity 12
5.6 Single and double tap 12
6 **AEK-CON-SENSOR1 sample application** 14
6.1 How to create a simple AEK-CON-SENSOR1 application 14
6.2 Available demos for AEK-CON-SENSOR1 17
6.2.1 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect activity and get accelerations – Test application 17
6.2.2 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect free-fall – Test application 18
6.2.3 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect tap – Test application 18
6.2.4 SPC58ECxx_RLA AEK-CON-SENSOR1 – Detect wake-up and orientation change – Test application 18
6.3 How to upload the demos for AEK-CON-SENSOR1 19
Appendix A **Available APIs for AEK-CON-SENSOR1 board** 20
Appendix B **Sensors supported by APIs** 22

Revision history 23

List of figures

Figure 1. AEK-CON-SENSOR1 . . . 1
Figure 2. Connecting sensor board to interface board . . . 3
Figure 3. AEK-CON-SENSOR1 pin details . . . 4
Figure 4. Sensor STRUCT elements . . . 5
Figure 5. Software Architecture diagram based on AIS2DW12 sensor . . . 9
Figure 6. Free-fall signals . . . 10
Figure 7. Wake-up signals . . . 11
Figure 8. Sensor orientation . . . 11
Figure 9. Activity-inactivity signals . . . 12
Figure 10. Tap description . . . 13
Figure 11. Adding AEK_CON_SENSOR1 Component in a SPC5Studio project . . . 14
Figure 12. AEK_CON_SENSOR1 component configuration . . . 15
Figure 13. AEK-CON-SENSOR1 sensor configuration . . . 15
Figure 14. AEK-CON-SENSOR1 PinMap display . . . 16
Figure 15. AEK-CON-SENSOR1 demo upload . . . 19

List of tables

Table 1. AEK-CON-SENSOR1 components . . . 3
Table 2. Sensor library file descriptions . . . 8
Table 3. Sensor characteristics . . . 9
Table 4. APIs and supported sensors . . . 22
Table 5. Document revision history . . . 23