InvenSense Device Driver library

3.8.4

Generated by Doxygen 1.8.11

Contents

Chapter 1

Main Page

1.1 Introduction

The InvenSense Device Driver library (aka libIDD) is dedicated to provide a unified API to control and retrieve data from InvenSense devices.

The library is composed of several layers which abstract specific features (protocol communication, registers access, ...) to the end user.

This library intends to be:

- Easy to use
- Highly modular
- C99 ANSI compliant

See the Supported devices page for the exhaustive list of supported device by libIDD.

1.2 Architecture overview

The main purpose of libIDD is to make any InvenSense devices easy to use, by mean of an abstract interface.

libIDD was designed to be used in embedded context, hence any system specific facility (SPI, I2C, delay, IRQ, ...) are abstracted. This also ease integration and portability.

The main API to be called from the application, in order to access to an InvenSense device is the Device API. The concrete implementation for the Device will call the associated low-level device driver.

Device and driver implementation may require to access to low-level system ressources and HW buses. This is achieved by mean of the Host Serial Interface. User is in charge to implement it (see ExampleHostSerif.c) depending on system capability and device interface.

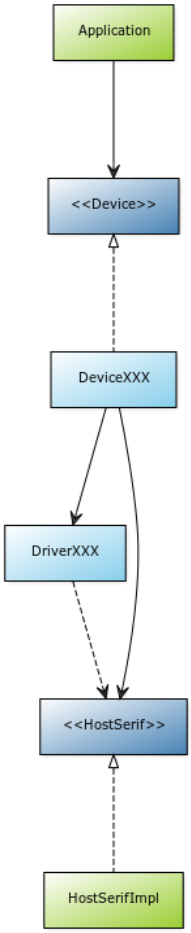


Figure 1.1 High level class diagram of libIDD

Legend:

- Green: user code
- Dark blue: abstract interface
- Light blue: specific device implementation

1.3 Importing and building libIDD

1.3.1 Requested library components

The library is delivered as sources. The directory structure should be preserved when imported into user project.

If only one device support is required, only files or directory with the device reference in it must be imported, in addition to the following files:

- Invn/IDDVersion.h
- Invn/InvBool.h

- Invn/InvError.h
- Invn/Devices/Device.h
- Invn/Devices/HostSerif.c
- Invn/Devices/HostSerif.h
- Invn/Devices/Sensor.c
- Invn/Devices/SensorTypes.h
- Invn/EmbUtils/DataConverter.c
- Invn/EmbUtils/DataConverter.h
- Invn/EmbUtils/ErrorHelper.c
- Invn/EmbUtils/ErrorHelper.h
- Invn/EmbUtils/Message.c
- Invn/EmbUtils/Message.h

1.3.2 Building the library

libIDD is coded in plain ANSI C99 without any uses of compiler extension. libIDD was built under gcc, vs2010, linaro for ARM and IAR for ARM.

Warning

libIDD uses 64bits integer and was not compiled for 8bits or 16bits architecture.

1.3.3 About debugging

Some library modules can output traces using the Message facility. Refer to dedicated page for help on how to use or disable this feature.

Chapter 2

Integration Guide

The library is using a serial interface instance to manage communication between targeted MCU and your InvenSense device.

The main goal will be to adress common communication functionalities.

2.1 Adapter

The first step is to create the adapter (also refered to as HostSerifImpl in the high level class diagram) . This file will describe how to connect to your specific hardware abstraction layer.

Please refer to ExampleHostSerif.c for a complete example of this file.

4 functions must be defined here :

```
int my_adapter_open(void) //Implementation of SPI/I2c open
{
        return Serial_open_low_level_driver()
}

int my_adapter_close(void) //Implementation of SPI/I2c close
{
        return Serial_close_low_level_driver()
}

int my_adapter_read_reg(uint8_t reg, uint8_t * rbuffer, uint32_t rlen) //Implementation of SPI/I2c read
{
        return Serial_read_low_level_driver(reg, rbuffer, rlen);
}

int my_adapter_write_reg(uint8_t reg, const uint8_t * wbuffer, uint32_t wlen) //Implementation of SPI/I2c
      write
{
        return Serial_write_low_level_driver(reg, wbuffer, wlen);
}
```

For advanced usage, more functions could be implemented such as an interrupt callback. This won't be covered in this guide.

2.2 Application

Once the Adapter is ready, we can start building the application.

Please refer to these files to have more complete example :

- ExampleDeviceIcm20602EMD.c
- ExampleDeviceIcm20603EMD.c
- ExampleDeviceIcm20690EMD.c

The application requires a instance of type inv_host_serif_t containing pointer to functions defined in the Adapter. In addition to the functions, this instance should contain :

- max_transaction_size : Hardware dependent value defining how many bytes are allowed per serial transaction
- serif_type : to be chosen amongst the inv_host_serif_type enumeration

```
// definition of the instance
static const inv_host_serif_t my_serif_instance = {
        my_adapter_open,
        my_adapter_close,
        my_adapter_read_reg,
        my_adapter_write_reg,
        MY_ADAPTER_SERIF_MAX_TRANSACTION_SIZE,
        MY_ADAPTER_SERIF_TYPE,
};
```

The structure being defined, it can now be used from the main to communicate with your InvenSense device. It then your responsability to open your serial interface at the very begenning and close it at the end depending on your needs (it might not be necessary to do it for your hardware) :

```
int main(void)
{
        rc = inv_host_serif_open(&my_serif_instance);

        // Your code here

        rc = inv_host_serif_close(&my_serif_instance);
}
```

You can then implement whatever you need using the LibIDD API as described in the example files.

Chapter 3

Supported devices

This page lists all devices supported by libIDD.

Two Base-Sensor devices are also supported:

- The ICM20602, a 6-axis MEMS able to report raw accelerometer and raw gyroscope data.
- The ICM20603, the next generation of 6-axis MEMS able to report raw accelerometer and raw gyroscope data
- The ICM20690, an advanced MEMS able to report raw accelerometer and raw gyroscope data and is also designed to support OIS and EIS feature.

The SmartMotion device is a virtual device that connects to other IDD devices and emulates missing motion features using software libraries.

3.1 ICM20602

The ICM20602 gives you access to raw accelerometer and raw gyroscope sensor. This device can be connected to the SmartMotion devices to emulate missing motion features using software libraries.

Primary interface:

- SPI: up to 6MHz, MSB first, CPOL=CPHA=1 (mode 3)
- I2C: 400 KHz, slave @ 0x68

GPIOs interrupts pinout;

- Data interrupt : The GPIO generates a active high pulse to indicate data ready.

For an example on how to use ICM20602 device with libIDD please see ExampleDeviceIcm20602EMD.c.

3.2 ICM20603

The ICM20603 gives you access to raw accelerometer and raw gyroscope sensor. This device can be connected to the SmartMotion devices to emulate missing motion features using software libraries.

Primary interface:

- SPI: up to 6MHz, MSB first, CPOL=CPHA=1 (mode 3)
- I2C: 400 KHz, slave @ 0x68

GPIOs interrupts pinout;

- Data interrupt : The GPIO generates a active high pulse to indicate data ready.

For an example on how to use ICM20603 device with libIDD please see ExampleDeviceIcm20603EMD.c.

3.3 ICM20690

The ICM20690 gives you access to raw accelerometer and raw gyroscope sensor. This parts is dedicated to O↩ IS support. The OIS feature allow to outputted gyroscope data to a camera module at high frequency for image stabilization. A FSYNC pin is also available to support the EIS feature and synchronize the gyroscope sampling. The ICM20690 supports auxiliary sensor such as magnetometer. This device can be connected to the SmartMotion devices to emulate missing motion features using software libraries.

Primary interface:

- SPI: up to 6MHz, MSB first, CPOL=CPHA=1 (mode 3)
- I2C: 400 KHz, slave @ 0x68

GPIOs interrupts pinout;

- Data interrupt : The GPIO generates a active high pulse to indicate data ready.

For an example on how to use ICM20690 device with libIDD please see ExampleDeviceIcm20690EMD.c.

3.4 SmartMotion

This Device can be connected to other IDD devices and emulates missing motion features using software libraries. It uses the InvenSense VSensor framework to connect a HW device to a InvenSense CModel processing graph.

Chapter 4

Deprecated List

Module HostSerif
Use SerifHal.h instead

Member inv_device_enable
use inv_device_enable_sensor

Member inv_device_icm20602_init (inv_device_icm20602_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)
Use innv_device_icm20602_init2() instead.

Member inv_device_icm20602_init_serif_ois (inv_device_icm20602_t *self, const inv_host_serif_t *serif_ois)
Use innv_device_icm20602_init_serif_ois2() instead.

Member inv_device_icm20603_init (inv_device_icm20603_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)
Use innv_device_icm20603_init2() instead.

Member inv_device_icm20603_init_serif_ois (inv_device_icm20603_t *self, const inv_host_serif_t *serif_ois)
Use innv_device_icm20603_init_serif_ois2() instead.

Member inv_device_icm20690_init (inv_device_icm20690_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)
Use innv_device_icm20690_init2() instead.

Member inv_device_icm20690_init_serif_ois (inv_device_icm20690_t *self, const inv_host_serif_t *serif_ois)
Use innv_device_icm20690_init_serif_ois2() instead.

Member INV_SENSOR_TYPE_ENERGY_EXPANDITURE

Member INV_SENSOR_TYPE_GYROMETER

Member INV_SENSOR_TYPE_META_DATA

Member INV_SENSOR_TYPE_UNCAL_GYROMETER

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

Module Documentation

7.1 Error code

Common error code.

Enumerations

7.1.1 Detailed Description

Common error code.

7.1.2 Enumeration Type Documentation

7.1.2.1 enum inv_error

Common error code definition.

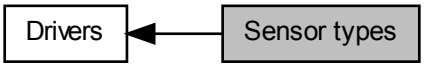
Enumerator

INV_ERROR_SUCCESS no error
INV_ERROR unspecified error
INV_ERROR_NIMPL function not implemented for given arguments
INV_ERROR_TRANSPORT error occured at transport level
INV_ERROR_TIMEOUT action did not complete in the expected time window
INV_ERROR_SIZE size/length of given arguments is not suitable to complete requested action
INV_ERROR_OS error related to OS
INV_ERROR_IO error related to IO operation
INV_ERROR_MEM not enough memory to complete requested action
INV_ERROR_HW error at HW level
INV_ERROR_BAD_ARG provided arguments are not good to perform requestion action
INV_ERROR_UNEXPECTED something unexpected happened
INV_ERROR_FILE cannot access file or unexpected format
INV_ERROR_PATH invalid file path
INV_ERROR_IMAGE_TYPE error when image type is not managed
INV_ERROR_WATCHDOG error when device doesn't respond to ping

7.2 Sensor types

Sensor related types definitions.

Collaboration diagram for Sensor types:



Classes

- struct inv_sensor_event

  Sensor event definition.
- struct inv_sensor_listener

  Sensor event listener definition.

Macros

- #define INV_SENSOR_TYPE_META_DATA INV_SENSOR_TYPE_RESERVED
- #define INV_SENSOR_TYPE_GYROMETER INV_SENSOR_TYPE_GYROSCOPE
- #define INV_SENSOR_TYPE_UNCAL_GYROMETER INV_SENSOR_TYPE_UNCAL_GYROSCOPE
- #define INV_SENSOR_TYPE_ENERGY_EXPANDITURE INV_SENSOR_TYPE_ENERGY_EXPENDITURE
- #define INV_SENSOR_TYPE_WU_FLAG (unsigned int)(0x80000000)

  Helper flag to indicate if sensor is a Wale-Up sensor.
- #define INV_SENSOR_EVENT_DATA_SIZE 64

  Maximum size of an event data.
- #define IVN_SENSOR_EVENT_DATA_SIZE INV_SENSOR_EVENT_DATA_SIZE

  For backward compatibility only - do not use.
- #define INV_SENSOR_ID_TO_TYPE(sensor) ((unsigned int)(sensor) & ~INV_SENSOR_TYPE_WU_FLAG)

  Helper macro to retrieve sensor type (without wake-up flag) from a sensor id.
- #define INV_SENSOR_IS_VALID(sensor) (INV_SENSOR_ID_TO_TYPE(sensor) < INV_SENSOR_TYPE_MAX)

  Helper macro that check if given sensor is of known type.
- #define INV_SENSOR_IS_WU(sensor) (((int)(sensor) & INV_SENSOR_TYPE_WU_FLAG) != 0)

  Helper macro that check if given sensor is a wake-up sensor.
- #define inv_sensor_2str inv_sensor_str

  Alias for inv_sensor_str.

Typedefs

- typedef struct inv_sensor_event inv_sensor_event_t

  Sensor event definition.
- typedef void(* inv_sensor_listener_event_cb_t) (const inv_sensor_event_t *event, void *context)

  Sensor listener event callback definition.
- typedef struct inv_sensor_listener inv_sensor_listener_t

  Sensor event listener definition.

Enumerations

Functions

- static void inv_sensor_listener_init (inv_sensor_listener_t *listener, inv_sensor_listener_event_cb_t event_↩ cb, void *context)

  Helper to initialize a listener object.
- static void inv_sensor_listener_notify (const inv_sensor_listener_t *listener, const inv_sensor_event_↩ t *event)

  Helper to notify a listener of a new sensor event.
- const char INV_EXPORT * inv_sensor_str (int sensor)

  Utility function that returns a string from a sensor id Empty string is returned if sensor is invalid.

7.2.1 Detailed Description

Sensor related types definitions.

7.2.2 Macro Definition Documentation

7.2.2.1 #define INV_SENSOR_TYPE_ENERGY_EXPANDITURE INV_SENSOR_TYPE_ENERGY_EXPENDITURE

Deprecated

7.2.2.2 #define INV_SENSOR_TYPE_GYROMETER INV_SENSOR_TYPE_GYROSCOPE

Deprecated

7.2.2.3 #define INV_SENSOR_TYPE_META_DATA INV_SENSOR_TYPE_RESERVED

Deprecated

7.2.2.4 #define INV_SENSOR_TYPE_UNCAL_GYROMETER INV_SENSOR_TYPE_UNCAL_GYROSCOPE

Deprecated

7.2.3 Typedef Documentation

7.2.3.1 typedef void(* inv_sensor_listener_event_cb_t) (const inv_sensor_event_t *event, void *context)

Sensor listener event callback definition.

Parameters

in	*event*	reference to sensor event
in	*context*	listener context

Returns

none

7.2.4 Enumeration Type Documentation

7.2.4.1 enum inv_sensor_status

Sensor status definition.

Enumerator

INV_SENSOR_STATUS_DATA_UPDATED new sensor data

INV_SENSOR_STATUS_STATE_CHANGED dummy sensor data indicating to a change in sensor state

INV_SENSOR_STATUS_FLUSH_COMPLETE dummy sensor data indicating a end of batch after a manual flush

INV_SENSOR_STATUS_POLLED_DATA sensor data value after manual request

7.2.4.2 enum inv_sensor_type

Sensor type identifier definition.

Enumerator

INV_SENSOR_TYPE_RESERVED Reserved ID: do not use.

INV_SENSOR_TYPE_ACCELEROMETER Accelerometer.

INV_SENSOR_TYPE_MAGNETOMETER Magnetic field.

INV_SENSOR_TYPE_ORIENTATION Deprecated orientation.

INV_SENSOR_TYPE_GYROSCOPE Gyroscope.

INV_SENSOR_TYPE_LIGHT Ambient light sensor.

INV_SENSOR_TYPE_PRESSURE Barometer.

INV_SENSOR_TYPE_TEMPERATURE Temperature.

INV_SENSOR_TYPE_PROXIMITY Proximity.

INV_SENSOR_TYPE_GRAVITY Gravity.

INV_SENSOR_TYPE_LINEAR_ACCELERATION Linear acceleration.

INV_SENSOR_TYPE_ROTATION_VECTOR Rotation vector.

INV_SENSOR_TYPE_HUMIDITY Relative humidity.

INV_SENSOR_TYPE_AMBIENT_TEMPERATURE Ambient temperature.

INV_SENSOR_TYPE_UNCAL_MAGNETOMETER Uncalibrated magnetic field.

INV_SENSOR_TYPE_GAME_ROTATION_VECTOR Game rotation vector.

INV_SENSOR_TYPE_UNCAL_GYROSCOPE Uncalibrated gyroscope.

INV_SENSOR_TYPE_SMD Significant motion detection.
INV_SENSOR_TYPE_STEP_DETECTOR Step detector.
INV_SENSOR_TYPE_STEP_COUNTER Step counter.
INV_SENSOR_TYPE_GEOMAG_ROTATION_VECTOR Geomagnetic rotation vector.
INV_SENSOR_TYPE_HEART_RATE Heart rate.
INV_SENSOR_TYPE_TILT_DETECTOR Tilt detector.
INV_SENSOR_TYPE_WAKE_GESTURE Wake-up gesture.
INV_SENSOR_TYPE_GLANCE_GESTURE Glance gesture.
INV_SENSOR_TYPE_PICK_UP_GESTURE Pick-up gesture.
INV_SENSOR_TYPE_BAC Basic Activity Classifier.
INV_SENSOR_TYPE_PDR Pedestrian Dead Reckoning.
INV_SENSOR_TYPE_B2S Bring to see.
INV_SENSOR_TYPE_3AXIS 3 Axis sensor
INV_SENSOR_TYPE_EIS Electronic Image Stabilization.
INV_SENSOR_TYPE_OIS Optical Image Stabilization.
INV_SENSOR_TYPE_RAW_ACCELEROMETER Raw accelerometer.
INV_SENSOR_TYPE_RAW_GYROSCOPE Raw gyroscope.
INV_SENSOR_TYPE_RAW_MAGNETOMETER Raw magnetometer.
INV_SENSOR_TYPE_RAW_TEMPERATURE Raw temperature.
INV_SENSOR_TYPE_CUSTOM_PRESSURE Custom Pressure Sensor.
INV_SENSOR_TYPE_MIC Stream audio from microphone.
INV_SENSOR_TYPE_TSIMU TS-IMU.
INV_SENSOR_TYPE_RAW_PPG Raw Photoplethysmogram.
INV_SENSOR_TYPE_HRV Heart rate variability.
INV_SENSOR_TYPE_SLEEP_ANALYSIS Sleep analysis.
INV_SENSOR_TYPE_BAC_EXTENDED Basic Activity Classifier Extended.
INV_SENSOR_TYPE_BAC_STATISTICS Basic Activity Classifier Statistics.
INV_SENSOR_TYPE_FLOOR_CLIMB_COUNTER Floor Climbed Counter.
INV_SENSOR_TYPE_ENERGY_EXPENDITURE Energy Expenditure.
INV_SENSOR_TYPE_DISTANCE Distance.
INV_SENSOR_TYPE_SHAKE Shake Gesture.
INV_SENSOR_TYPE_DOUBLE_TAP Double Tap.
INV_SENSOR_TYPE_CUSTOM0 Custom sensor ID 0.
INV_SENSOR_TYPE_CUSTOM1 Custom sensor ID 1.
INV_SENSOR_TYPE_CUSTOM2 Custom sensor ID 2.
INV_SENSOR_TYPE_CUSTOM3 Custom sensor ID 3.
INV_SENSOR_TYPE_CUSTOM4 Custom sensor ID 4.
INV_SENSOR_TYPE_CUSTOM5 Custom sensor ID 5.
INV_SENSOR_TYPE_CUSTOM6 Custom sensor ID 6.
INV_SENSOR_TYPE_CUSTOM7 Custom sensor ID 7.
INV_SENSOR_TYPE_WOM Wake-up on motion.
INV_SENSOR_TYPE_SEDENTARY_REMIND Sedentary Remind.
INV_SENSOR_TYPE_DATA_ENCRYPTION Data Encryption.
INV_SENSOR_TYPE_FSYNC_EVENT FSYNC event.
INV_SENSOR_TYPE_HIGH_RATE_GYRO High Rate Gyro.
INV_SENSOR_TYPE_CUSTOM_BSCD Custom BAC StepCounter Calorie counter and Distance counter.
INV_SENSOR_TYPE_HRM_LOGGER HRM ouput for logger.
INV_SENSOR_TYPE_PREDICTIVE_QUATERNION Predictive Quaternion.
INV_SENSOR_TYPE_MAX sentinel value for sensor type

7.3 Sensor Configuration

General sensor configuration types definitions.

Collaboration diagram for Sensor Configuration:

Classes

- struct inv_sensor_config_mounting_mtx

  Define mounting matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_MOUNTING_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.
- struct inv_sensor_config_gain

  Define gain matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.
- struct inv_sensor_config_offset

  Define offset vector value for 3-axis sensors (associated with INV_SENSOR_CONFIG_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.
- struct inv_sensor_config_context

  Define configuration context value (associated with INV_SENSOR_CONFIG_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implemetation.
- struct inv_sensor_config_fsr

  Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV_SENSOR_↩ CONFIG_FSR config ID) Value is expetcted to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.
- struct inv_sensor_config_powermode

  Define chip power mode (associated with INV_SENSOR_CONFIG_POWER_MODE config ID) Value is expetcted to be 0 for low power or 1 for low noise.
- struct inv_sensor_config_energy_expenditure

  Define the configuration for the energy expenditure's algorithm.
- struct inv_sensor_config_distance

  Define the configuration for the distance's algorithm.
- struct inv_sensor_config_bac

  Define the configuration for BAC.
- struct inv_sensor_config_stepc

  Define the configuration for steps counter.
- struct inv_sensor_config_shake_wrist

  Define the configuration for the shake wrist's algorithm.
- struct inv_sensor_config_double_tap

  Define the configuration for the double tap's algorithm.
- struct inv_sensor_config_BSCD

  Define the configuration for the BSCD virtual sensor.

Typedefs

- typedef struct inv_sensor_config_mounting_mtx inv_sensor_config_mounting_mtx_t

  Define mounting matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_MOUNTING_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.
- typedef struct inv_sensor_config_gain inv_sensor_config_gain_t

  Define gain matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.
- typedef struct inv_sensor_config_offset inv_sensor_config_offset_t

  Define offset vector value for 3-axis sensors (associated with INV_SENSOR_CONFIG_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.
- typedef struct inv_sensor_config_context inv_sensor_config_context_t

  Define configuration context value (associated with INV_SENSOR_CONFIG_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implemetation.
- typedef struct inv_sensor_config_fsr inv_sensor_config_fsr_t

  Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV_SENSOR_↩ CONFIG_FSR config ID) Value is expetcted to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.
- typedef struct inv_sensor_config_powermode inv_sensor_config_powermode_t

  Define chip power mode (associated with INV_SENSOR_CONFIG_POWER_MODE config ID) Value is expetcted to be 0 for low power or 1 for low noise.
- typedef struct inv_sensor_config_energy_expenditure inv_sensor_config_energy_expenditure_t

  Define the configuration for the energy expenditure's algorithm.
- typedef struct inv_sensor_config_distance inv_sensor_config_distance_t

  Define the configuration for the distance's algorithm.
- typedef struct inv_sensor_config_bac inv_sensor_config_bac_t

  Define the configuration for BAC.
- typedef struct inv_sensor_config_stepc inv_sensor_config_stepc_t

  Define the configuration for steps counter.
- typedef struct inv_sensor_config_shake_wrist inv_sensor_config_shake_wrist_t

  Define the configuration for the shake wrist's algorithm.
- typedef struct inv_sensor_config_double_tap inv_sensor_config_double_tap_t

  Define the configuration for the double tap's algorithm.
- typedef struct inv_sensor_config_BSCD inv_sensor_config_BSCD_t

  Define the configuration for the BSCD virtual sensor.

Enumerations

7.3.1 Detailed Description

General sensor configuration types definitions.

7.3.2 Typedef Documentation

7.3.2.1 typedef struct inv_sensor_config_bac inv_sensor_config_bac_t

Define the configuration for BAC.

Parameters

enableNotify	enable disable notify

7.3.2.2 typedef struct inv_sensor_config_BSCD inv_sensor_config_BSCD_t

Define the configuration for the BSCD virtual sensor.

Parameters

Age	age in year; Range is (0;100). Default is 35.
Gender	gender is 0 for men, 1 for female. Default is 0
Height	height in centimeter; Range is (50;250). Default is 175.
Weight	weight in kg; Range is (3;300). Default is 75
enableNotify	bitmask to enable/disable notify on a a specific sensor event bit 0 (1): enable/disable notify on BAC event bit 1 (2): enable/disable notify on step counter event bit 2 (4): enable/disable notify on energy expenditure event bit 3 (8): enable/disable notify on distance event

7.3.2.3 typedef struct inv_sensor_config_distance inv_sensor_config_distance_t

Define the configuration for the distance's algorithm.

Parameters

user_height	height of the user in cm
enableNotify	enable disable notify

7.3.2.4 typedef struct inv_sensor_config_double_tap inv_sensor_config_double_tap_t

Define the configuration for the double tap's algorithm.

Parameters

minimum_threshold	This parameter sets the minimum threshold to reach in order to start a Tap detection. Default value is 2000, recommended range [500 ; 2500]
t_max	This parameter sets the maximum time after a Tap event in [sample]. Default value is 100, recommended range [30 ; 200].

7.3.2.5 typedef struct inv_sensor_config_energy_expenditure inv_sensor_config_energy_expenditure_t

Define the configuration for the energy expenditure's algorithm.

Parameters

age	age in year; Range is (0;100).
gender	gender is 0 for men, 1 for female.
height	height in centimeter; Range is (50;250)
weight	weight in kg; Range is (3;300)
enableNotify	enable disable notify

7.3.2.6 typedef struct inv_sensor_config_mounting_mtx inv_sensor_config_mounting_mtx_t

Define mounting matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_MOUNTING_MATR↩ IX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.

Value is expetcted to be a rotation matrix.

7.3.2.7 typedef struct inv_sensor_config_offset inv_sensor_config_offset_t

Define offset vector value for 3-axis sensors (associated with INV_SENSOR_CONFIG_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.

If applied to RAW sensor, value is expected to be in lsb. If applied to other sensor, value is expected to be in sensor unit (g, uT or dps).

7.3.2.8 typedef struct inv_sensor_config_shake_wrist inv_sensor_config_shake_wrist_t

Define the configuration for the shake wrist's algorithm.

Parameters

max_period	This parameter sets the maximal duration for half oscillation to detect a Shake wrist. The default value is 20, recommend range [15 ; 40], 15 for the lower sensitivity and 40 for the higher sensitivity. Notice that increasing the sensitivity will increase the number of false detection, and also slightly increase response time.
dummy_padding	Dummy byte for padding. Set it to 0.

7.3.2.9 typedef struct inv_sensor_config_stepc inv_sensor_config_stepc_t

Define the configuration for steps counter.

Parameters

enableNotify	enable disable notify

7.3.3 Enumeration Type Documentation

7.3.3.1 enum inv_sensor_config

Sensor type identifier definition.

Enumerator

INV_SENSOR_CONFIG_RESERVED Reserved config ID: do not use.

INV_SENSOR_CONFIG_MOUNTING_MATRIX 3x3 mounting matrix

INV_SENSOR_CONFIG_GAIN 3x3 gain matrix (to correct for cross-axis defect)

INV_SENSOR_CONFIG_OFFSET 3d offset vector

INV_SENSOR_CONFIG_CONTEXT arbitrary context buffer

INV_SENSOR_CONFIG_FSR Full scale range.

INV_SENSOR_CONFIG_RESET Reset the specified service.

INV_SENSOR_CONFIG_POWER_MODE Low Power or Low Noise mode.

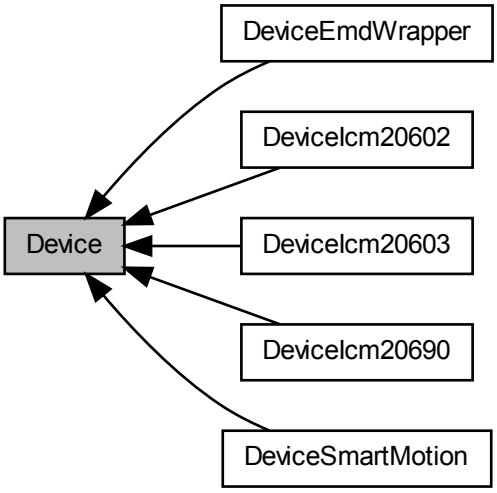
INV_SENSOR_CONFIG_CUSTOM Configuration ID above this value are device specific.

INV_SENSOR_CONFIG_MAX Absolute maximum value for sensor config.

7.4 Device

Abstract device interface definition.

Collaboration diagram for Device:



Modules

- DeviceSmartMotion

  Device implementation that connects to other IDD devices and emulates missing motion features using software libraries.
- DeviceEmdWrapper

  emd wrapper implementation for device interface
- DeviceIcm20602

  Concrete implementation of the 'Device' interface for Icm20602 devices.
- DeviceIcm20603

  Concrete implementation of the 'Device' interface for Icm20603 devices.
- DeviceIcm20690

  Concrete implementation of the 'Device' interface for Icm20690 devices.

Classes

- struct inv_fw_version

  FW version structure definition.
- struct inv_device_vt

  Device virtual table definition.
- struct inv_device

  Abtract device object definition.

Macros

- #define inv_device_enable inv_device_enable_sensor

  Alias of inv_device_enable_sensor() for backward compatibility.

Typedefs

- typedef struct inv_fw_version inv_fw_version_t

  FW version structure definition.
- typedef struct inv_device_vt inv_device_vt_t

  Device virtual table definition.
- typedef struct inv_device inv_device_t

  Abtract device object definition.

Functions

- static int inv_device_whoami (const inv_device_t *dev, uint8_t *whoami)

  Gets WHO AM I value.
- static int inv_device_reset (const inv_device_t *dev)

  Resets the device to a known state.
- static int inv_device_setup (const inv_device_t *dev)

  Performs basic device initialization.
- static int inv_device_cleanup (const inv_device_t *dev)

  Shutdown the device and clean-up internal states.
- static int inv_device_poll (const inv_device_t *dev)

  Polls the device for data.
- static int inv_device_load (const inv_device_t *dev, int type, const uint8_t *image, uint32_t size, inv_bool_t verify, inv_bool_t force)

  Begins loading procedure for device's image(s)
- static int inv_device_get_fw_info (const inv_device_t *dev, struct inv_fw_version *version)

  Gets device FW version.
- static int inv_device_set_running_state (const inv_device_t *dev, inv_bool_t state)

  Indicates to device current RUN/SUSPEND state of the host.
- static int inv_device_ping_sensor (const inv_device_t *dev, int sensor)

  Checks if a sensor is supported by the device.
- static int inv_device_enable_sensor (const inv_device_t *dev, int sensor, inv_bool_t start)

  Enable/Disable a sensor.
- static int inv_device_start_sensor (const inv_device_t *dev, int sensor)

  Starts a sensor.
- static int inv_device_stop_sensor (const inv_device_t *dev, int sensor)

  Stops a sensor.
- static int inv_device_set_sensor_period_us (const inv_device_t *dev, int sensor, uint32_t period)

  Configure sensor output data period.
- static int inv_device_set_sensor_period (const inv_device_t *dev, int sensor, uint32_t period)

  Configure sensor output data period.
- static int inv_device_set_sensor_timeout (const inv_device_t *dev, int sensor, uint32_t timeout)

  Configure sensor output timeout.
- static int inv_device_set_sensor_timeout_us (const inv_device_t *dev, int sensor, uint32_t timeout)

  Configure sensor output timeout.
- static int inv_device_flush_sensor (const inv_device_t *dev, int sensor)

Forces flush of devices's internal buffers.

- static int inv_device_set_sensor_bias (const inv_device_t *dev, int sensor, const float bias[3])

  Configure bias value for a sensor.
- static int inv_device_get_sensor_bias (const inv_device_t *dev, int sensor, float bias[3])

  Gets bias value for a sensor.
- static int inv_device_set_sensor_mounting_matrix (const inv_device_t *dev, int sensor, const float matrix[9])

  Sets the mounting matrix information for a multi-axis sensor.
- static int inv_device_get_sensor_data (const inv_device_t *dev, int sensor, inv_sensor_event_t *event)

  Retrieve last known sensor event for a sensor.
- static int inv_device_self_test (const inv_device_t *dev, int sensor)

  Perform self-test procedure for MEMS component of the device.
- static int inv_device_set_sensor_config (const inv_device_t *dev, int sensor, int settings, const void *arg, uint16_t size)

  Generic method to configure a sensor.
- static int inv_device_get_sensor_config (const inv_device_t *dev, int sensor, int settings, void *value, uint16_t size)

  Generic method to retrieve configuration value for a sensor.
- static int inv_device_write_mems_register (const inv_device_t *dev, int sensor, uint16_t reg_addr, const void *data, uint16_t len)

  Set the MEMS register.
- static int inv_device_read_mems_register (const inv_device_t *dev, int sensor, uint16_t reg_addr, void *data, uint16_t len)

  Read register of underlying MEMS or HW sensor.

7.4.1 Detailed Description

Abstract device interface definition.

All functions declared in this file are virtual. They aim to provide a unified way of accessing InvenSense devices. All functions shall return a int for which 0 indicates success and a negative value indicates an error as described by enum inv_error

If a particular device implementation does not support any of the method declared here, it shall return INV_ERROR_NIMPL.

Implementation is not expected to be thread-safe.

Refer to concrete device implementation for additionnal and specific information about API usage related to a particular device.

7.4.2 Macro Definition Documentation

7.4.2.1 #define inv_device_enable inv_device_enable_sensor

Alias of inv_device_enable_sensor() for backward compatibility.

Deprecated use inv_device_enable_sensor

7.4.3 Function Documentation

7.4.3.1 static int inv_device_cleanup (const inv_device_t * *dev*) [inline], [static]

Shutdown the device and clean-up internal states.

Parameters

in	*dev*	pointer to device object instance

Returns

0 on success INV_ERROR_TIMEOUT if clean-up does not complete in time INV_ERROR_TRANSPORT in case of low level serial error

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.2 static int inv_device_enable_sensor (const inv_device_t * *dev*, int *sensor*, inv_bool_t *start*) [inline], [static]

Enable/Disable a sensor.

Send a command to start or stop a sensor. See inv_device_start_sensor() and inv_device_stop_sensor()

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*start*	true to start the sensor, false to stop the sensor

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

7.4.3.3 static int inv_device_flush_sensor (const inv_device_t * *dev*, int *sensor*) [inline], [static]

Forces flush of devices's internal buffers.

Send a command a flush command to device. Device will imediatly send all sensor events that may be store in its internal buffers.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

7.4.3.4 static int inv_device_get_fw_info (const inv_device_t * *dev*, struct inv_fw_version * *version*) [inline], [static]

Gets device FW version.

Parameters

in	*dev*	pointer to device object instance
out	*version*	version structure placeholder

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time

7.4.3.5 static int inv_device_get_sensor_bias (const inv_device_t * *dev*, int *sensor*, float *bias[3]*) [inline], [static]

Gets bias value for a sensor.

Bias configuration makes sense only for few sensor types:

- INV_SENSOR_TYPE_ACCELEROMETER
- INV_SENSOR_TYPE_MAGNETOMETER
- INV_SENSOR_TYPE_GYROSCOPE Bias unit is the same as the corresponding sensor unit.

See also

inv_sensor_event_t for details.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

out	*bias*	returned bias

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_BAD_ARG if sensor is not supported by the implementation

7.4.3.6 static int inv_device_get_sensor_config (const inv_device_t * *dev*, int *sensor*, int *settings*, void * *value*, uint16_t *size*) [inline], [static]

Generic method to retrieve configuration value for a sensor.

For common settings, setting value is expected to be a value from

See also

enum inv_sensor_config. Settings data is expected to point the proper type as describes in SensorConfig.h

For specific settings, refer to concrete device implementation for supported sensor and available configuration settings parameters.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*settings*	settings to configure
out	*value*	the value for the corresponding setting
in	*size*	maximum buffer size pointed by value

Returns

0 on success >0 indicating success and the number of byte written to value INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_BAD↩ _ARG if sensor is not supported by the implementation INV_ERROR_SIZE provided buffer is not big enough INV_ERROR if configuration has failed

7.4.3.7 static int inv_device_get_sensor_data (const inv_device_t * *dev*, int *sensor*, inv_sensor_event_t * *event*) [inline], [static]

Retrieve last known sensor event for a sensor.

Depending on device capability, a call to this function may have no effect or return an error.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

out	*event*	last known event data

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_BAD_ARG if sensor is not supported by the implementation INV_E↩ RROR if and event was received but unmanaged by the implementation

7.4.3.8 static int inv_device_load (const inv_device_t * *dev,* int *type,* const uint8_t * *image,* uint32_t *size,* inv_bool_t *verify,* inv_bool_t *force*) [inline], [static]

Begins loading procedure for device's image(s)

Will start the process of loading an image to device's memory. Type of images to load will depend on the device type and/or FW version.

Parameters

in	*dev*	pointer to device object instance
in	*type*	type of image to load. Can vary from one implementation to another. Refer to specific implementation for details.
in	*image*	pointer to image (or image chunk) data
in	*size*	size of image (or size of image chunk)
in	*verify*	true to perform image integrity verification, false to skip it
in	*force*	true to load image even if identical to current image, false to compare image first

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_SIZE if image size does not fit in device memory

7.4.3.9 static int inv_device_ping_sensor (const inv_device_t * *dev,* int *sensor*) [inline], [static]

Checks if a sensor is supported by the device.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR if sensor is not supported by the device INV_ERROR_BAD_ARG if sensor is not supported by the implementation

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.10 static int inv_device_poll (const inv_device_t * *dev*) [inline], [static]

Polls the device for data.

Will read device interrupt registers and data registers or FIFOs. Will parse data and called sensor events handler provided at init time. Handler will be called in the same context of this function.

Warning

Care should be taken regarding concurency. If this function is called in a dedicated thread, suitable protection must be used to avoid concurent calls to poll() or any other device methods.

Parameters

in	*dev*	pointer to device object instance

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_UNEXPECTED in case of bad formated or un-handled data frame

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.11 static int inv_device_read_mems_register (const inv_device_t * *dev*, int *sensor*, uint16_t *reg_addr*, void * *data*, uint16_t *len*) [inline], [static]

Read register of underlying MEMS or HW sensor.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*reg_addr*	the register that should be read
in	*data*	pointer to buffer to hold read data
in	*length*	length of data to read

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_BAD_ARG if sensor is not supported by the implementation INV_E↩ RROR_SIZE request length is above device capability INV_ERROR if configuration has failed

7.4.3.12 static int inv_device_reset (const inv_device_t * *dev*) [inline], [static]

Resets the device to a known state.

Will perform an HW and SW reset of device, and reset internal driver states To know value. Should be called before setup or when device state is unknown.

Parameters

in	*dev*	pointer to device object instance

Returns

0 on success INV_ERROR_TIMEOUT if reset does not complete in time INV_ERROR_TRANSPORT in case of low level serial error

7.4.3.13 static int inv_device_self_test (const inv_device_t * *dev*, int *sensor*) [inline], [static]

Perform self-test procedure for MEMS component of the device.

Available MEMS vary depend on the device. Use following sensor type for the various MEMS:

- INV_SENSOR_TYPE_ACCELEROMETER: for HW accelerometer sensor
- INV_SENSOR_TYPE_MAGNETOMETER : for HW magnetometer sensor
- INV_SENSOR_TYPE_GYROSCOPE : for HW gyroscope sensor
- INV_SENSOR_TYPE_PRESSURE : for HW pressure sensor

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR if self test has failed

7.4.3.14 static int inv_device_set_running_state (const inv_device_t * *dev*, inv_bool_t *state*) [inline], [static]

Indicates to device current RUN/SUSPEND state of the host.

If SUSPEND state (false) is set, device should not notify any sensor events (besides event comming from a wake-up source). If RUNNING state (true) is set, all sensor events will be notify to host. Device will consider host to be in RUNNING state after a reset/setup.

Parameters

in	*dev*	pointer to device object instance
in	*state*	RUNNING (true) or SUSPEND (false) state

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error

7.4.3.15 static int inv_device_set_sensor_bias (const inv_device_t * *dev*, int *sensor*, const float *bias[3]*) [inline], [static]

Configure bias value for a sensor.

Bias configuration makes sense only for few sensor types:

- INV_SENSOR_TYPE_ACCELEROMETER
- INV_SENSOR_TYPE_MAGNETOMETER
- INV_SENSOR_TYPE_GYROSCOPE Bias unit is the same as the corresponding sensor unit.

See also

inv_sensor_event_t for details.

If this feature is supported by the implementation but not by the device, behavior is undefined (but will most probably have no effect).

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*bias*	bias to set

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

7.4.3.16 static int inv_device_set_sensor_config (const inv_device_t * *dev*, int *sensor*, int *settings*, const void * *arg*, uint16_t *size*) [inline], [static]

Generic method to configure a sensor.

Allow to configure a sensor (HW or virtual), such as FSR, BW, ...

For common settings, setting value is expected to be a value from

See also

enum inv_sensor_config. Settings data is expected to point the proper type as describes in SensorConfig.h

For specific settings, refer to concrete device implementation for supported sensor and available configuration settings parameters.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*settings*	settings to configure
in	*arg*	pointer to settings value
in	*size*	settings value size

Returns

0 on sucess INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_BAD_ARG if sensor is not supported by the implementation INV_E↩ RROR_SIZE size is above intenral buffer size / device capability INV_ERROR if configuration has failed

7.4.3.17 static int inv_device_set_sensor_mounting_matrix (const inv_device_t * *dev*, int *sensor*, const float *matrix[9]*) [inline], [static]

Sets the mounting matrix information for a multi-axis sensor.

Allow to specify the mounting matrix for multi-axis sensor in order to align axis of several sensors in the same reference frame. Sensor types allowed:

- INV_SENSOR_TYPE_ACCELEROMETER
- INV_SENSOR_TYPE_MAGNETOMETER
- INV_SENSOR_TYPE_GYROSCOPE Depending on device capability, called to this function may have no effect.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*matrix*	mounting matrix to apply

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_BAD_ARG if sensor is not supported by the implementation or if one of the mounting matrix value is not in the [-1;1] range

7.4.3.18 static int inv_device_set_sensor_period (const inv_device_t * *dev*, int *sensor*, uint32_t *period*) [inline], [static]

Configure sensor output data period.

Similar to inv_device_set_sensor_period_us() except period is specified in ms. Will simply call inv_device_set_↩ sensor_period_us() after converting input period.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*period*	requested data period in ms

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.19 static int inv_device_set_sensor_period_us (const inv_device_t * *dev*, int *sensor*, uint32_t *period*) [inline], [static]

Configure sensor output data period.

Send a command to set sensor output data period. Period is a hint only. Depending on sensor type or device capability, the effective output data might be different. User shall refer to sensor events timestamp to determine effective output data period.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*period*	requested data period in us

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

7.4.3.20 static int inv_device_set_sensor_timeout (const inv_device_t * *dev*, int *sensor*, uint32_t *timeout*) [inline], [static]

Configure sensor output timeout.

Send a command to set sensor maximum report latency (or batch timeout). This allows to enable batch mode. Provided timeout is a hint only and sensor events may be notified at a faster rate depending on sensor type or device capability or other active sensors.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*timeout*	allowed timeout in ms

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

7.4.3.21 static int inv_device_set_sensor_timeout_us (const inv_device_t * *dev*, int *sensor*, uint32_t *timeout*) [inline], [static]

Configure sensor output timeout.

Similar to inv_device_set_sensor_timeout() except period is specified in ms. Will simply call inv_device_set_↩ sensor_timeout() after converting input period.

Warning

If input timeout is < 1000, value will be rounded to 0.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*timeout*	allowed timeout in us

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

7.4.3.22 static int inv_device_setup (const inv_device_t * *dev*) [inline], [static]

Performs basic device initialization.

Except if device's flash memory is outdated, devuie should be able to handle request after setup() is complete. If devices's flash memory need to be updated, load_begin()/load_continue()/ load_end() methods must be called first with suitable argument.

Parameters

in	*dev*	pointer to device object instance

Returns

0 on success INV_ERROR_TIMEOUT if setup does not complete in time INV_ERROR_TRANSPORT in case of low level serial error

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.23 static int inv_device_start_sensor (const inv_device_t * *dev*, int *sensor*) [inline], [static]

Starts a sensor.

Send a command to start a sensor. Device will start sending events if sensor is supported (ie: ping() returns 0 for this sensor type).

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.24 static int inv_device_stop_sensor (const inv_device_t * *dev*, int *sensor*) [inline], [static]

Stops a sensor.

Send a command to stop a sensor. Device will stop sending events if sensor was previously started.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_BAD_ARG if sensor is not supported by the implementation

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.25 static int inv_device_whoami (const inv_device_t * *dev*, uint8_t * *whoami*) [inline], [static]

Gets WHO AM I value.

Can be called before performing device setup

Parameters

in	*dev*	pointer to device object instance
out	*whoami*	WHO AM I value

Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

7.4.3.26 static int inv_device_write_mems_register (const inv_device_t * *dev*, int *sensor*, uint16_t *reg_addr*, const void * *data*, uint16_t *len*) [inline], [static]

Set the MEMS register.

Parameters

in	*dev*	pointer to device object instance
in	*sensor*	sensor type (as defined by

See also

inv_sensor_type_t)

Parameters

in	*reg_addr*	the register that should be written
in	*data*	data to write at reg_addr
in	*length*	length of data to write

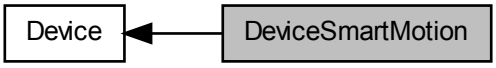
Returns

0 on success INV_ERROR_TRANSPORT in case of low level serial error INV_ERROR_TIMEOUT if device does not respond in time INV_ERROR_BAD_ARG if sensor is not supported by the implementation INV_E↩ RROR_SIZE request length is above device capability INV_ERROR if configuration has failed

7.5 DeviceSmartMotion

Device implementation that connects to other IDD devices and emulates missing motion features using software libraries.

Collaboration diagram for DeviceSmartMotion:



Classes

- struct inv_device_smart_motion_vlistener

  Internal definition for graph leaves (based on VSensorListener)
- struct inv_device_smart_motion_vsensor

  Internal definition for graph roots (based on VSensor)
- struct inv_device_smart_motion

  States for SmartMotion device implementation.

Typedefs

- typedef struct inv_device_smart_motion inv_device_smart_motion_t

  States for SmartMotion device implementation.
- typedef uint32_t inv_smartmotion_config_value_sensitivity_t

  Expected type for SENSITIVITY config settings Value is coded in Q16 and expressed SI unit / LSB SI unit being g for ACC, uT for MAG and dps for GYR.

Enumerations

Functions

- void INV_EXPORT inv_device_smart_motion_init (inv_device_smart_motion_t *self, inv_device_t *hw_↩ device)

  Initialize states for SmartMotion device implementation.
- void INV_EXPORT inv_device_smart_motion_init2 (inv_device_smart_motion_t *self, inv_device_t *hw_↩ device, void *opt)

  Initialize states for SmartMotion device implementation.
- static inv_device_t * inv_device_smart_motion_get_base (inv_device_smart_motion_t *self)

  Return generic handle to base inv_device_t.
- int INV_EXPORT inv_device_smart_motion_setup (void *context)

  Initialize SmartMotion graph.
- void INV_EXPORT inv_device_smart_motion_init_aar (void *context)

  Initialize SmartMotion leaves related to Advanced Activity Recognition.
- void INV_EXPORT inv_device_smart_motion_init_pickup (void *context)

  Initialize SmartMotion graph leaves related to Pickup.
- void INV_EXPORT inv_device_smart_motion_init_eis (void *context)

  Initialize SmartMotion graph leaves related to Electronic Image Stabilisation.

Variables

- VSensorListener inv_device_smart_motion_vlistener::vlistener

  base VSensorListener states
- void(* inv_device_smart_motion_vlistener::build_event)(const void *, inv_sensor_event_t *)

  callback to convert VSensorData to IDD sensor event
- uint8_t inv_device_smart_motion_vlistener::idd_type

  corresponding IDD sensor type
- InvList inv_device_smart_motion_vlistener::node

  pointer to next element when stored in a list
- VSensor inv_device_smart_motion_vsensor::vsensor

  base VSensor states
- void(* inv_device_smart_motion_vsensor::build_vsensor_data)(const inv_sensor_event_t *, void *)

  callback to convert IDD sensor event to VSensorData
- uint8_t inv_device_smart_motion_vsensor::idd_type

  corresponding IDD sensor type
- inv_device_t inv_device_smart_motion::base

  base inv_device_t states
- inv_device_t * inv_device_smart_motion::hw_device

  reference to inv_device_t that will provide sensor data for base sensor
- uint8_t inv_device_smart_motion::base_chip_info [3]

  parameter given at init time to identify underlying ICM
- inv_sensor_listener_t inv_device_smart_motion::private_idd_listener

  internal IDD listener to catch events from HW device
- InvList inv_device_smart_motion::leaves_list

  list of graph leaves (corresponds to sensors not present at hardware level and that are emulated)
- struct inv_device_smart_motion_vsensor inv_device_smart_motion::s

  placeholder for roots VSensor
- struct inv_device_smart_motion_vlistener inv_device_smart_motion::l

  placeholder for leaves corresponding to root VSensor
- struct {
  struct inv_device_smart_motion_vsensor inv_device_smart_motion::s
  placeholder for roots VSensor
  struct inv_device_smart_motion_vlistener inv_device_smart_motion::l
  placeholder for leaves corresponding to root VSensor
  } inv_device_smart_motion::graph_roots [5]

  placehold holder for graph roots objects
- struct inv_device_smart_motion_vlistener inv_device_smart_motion::l

  placeholder for leaves VSensorListener
- struct {
  struct inv_device_smart_motion_vlistener inv_device_smart_motion::l
  placeholder for leaves VSensorListener
  } inv_device_smart_motion::graph_leaves [24]

  placeholder for leaves corresponding to emulated sensors
- uint64_t inv_device_smart_motion::hw_sensor_available_mask

  mask to keep track of HW available sensor from underlying device

7.5.1 Detailed Description

Device implementation that connects to other IDD devices and emulates missing motion features using software libraries.

It uses the InvenSense VSensor framework to connect a HW device to a InvenSense CModel processing graph.

The graph consists in:

- Few roots that get data from underlying IDD device
- Leaves for all emulated sensors
- Intermediary nodes that call SmartMotion library

Upon SmartMotion device initialization, you must provide a reference to an already intialiazed and setup inv_↩ device_t (eg: ICM20602).

This device implementation takes ownership of ther underlying device. Except for very specific methods (eg: setup(), load(), self_test(), ...), handle to SmartMotion device should be used to control sensors.

When calling setup(), SmartMotion device will ping all sensors for the underlying HW device and keep track their availibitly.

When starting a sensor, if is is present at HW level, SmartMotion device will forward the request to the underlying device. If not, it will enable the corresponding algorithm and start required base sensor to provide requested data. Eg: if user requests 'GAME_ROTATION_VECTOR' and it is not directly available from the HW device, both 'RA↩ W_GYROSCOPE' and 'RAW_ACCELEROMETER' will be started at HW level, and 'GAME_ROTATION_VECTOR' algorithm enabled.

SmartMotion device only emulates the following sensors:

- INV_SENSOR_TYPE_ACCELEROMETER
- INV_SENSOR_TYPE_GYROSCOPE
- INV_SENSOR_TYPE_UNCAL_GYROSCOPE
- INV_SENSOR_TYPE_MAGNETOMETER
- INV_SENSOR_TYPE_UNCAL_MAGNETOMETER
- INV_SENSOR_TYPE_GAME_ROTATION_VECTOR
- INV_SENSOR_TYPE_ROTATION_VECTOR
- INV_SENSOR_TYPE_GEOMAG_ROTATION_VECTOR
- INV_SENSOR_TYPE_BAC
- INV_SENSOR_TYPE_STEP_COUNTER
- INV_SENSOR_TYPE_STEP_DETECTOR
- INV_SENSOR_TYPE_SMD
- INV_SENSOR_TYPE_TILT_DETECTOR
- INV_SENSOR_TYPE_PICK_UP_GESTURE
- INV_SENSOR_TYPE_GRAVITY

- INV_SENSOR_TYPE_LINEAR_ACCELERATION
- INV_SENSOR_TYPE_ORIENTATION

Required base sensor are:

- INV_SENSOR_TYPE_RAW_ACCELEROMETER
- INV_SENSOR_TYPE_RAW_GYROSCOPE
- INV_SENSOR_TYPE_RAW_MAGNETOMETER

Warning

Batching is currently not emulated
Only one instance of this device can be used in the same application SmartMotion algorithms are currently integrated using static variables. Only one instance of this device must be used at once, to avoid re-entrancy issue.

7.5.2 Enumeration Type Documentation

7.5.2.1 enum inv_smartmotion_config_setting

Allowed config settings for DeviceSmartMotion.

Enumerator

INV_SMARTMOTION_CONFIG_SENSITIVITY sensitivity (LSB to SI unit factor)

7.5.3 Function Documentation

7.5.3.1 void INV_EXPORT inv_device_smart_motion_init (inv_device_smart_motion_t * *self,* inv_device_t * *hw_device*)

Initialize states for SmartMotion device implementation.

Assumes underlying device is already initialized. Will take ownership of the underlying device and re-use its listener. No HW access are done in this function.

Parameters

in	*self*	handle to current device
in	*hw_device*	reference to inv_device_t used to retrieve base data

7.5.3.2 void INV_EXPORT inv_device_smart_motion_init2 (inv_device_smart_motion_t * *self,* inv_device_t * *hw_device,* void * *opt*)

Initialize states for SmartMotion device implementation.

Assumes underlying device is already initialized. Will take ownership of the underlying device and re-use its listener. No HW access are done in this function.

Parameters

in	*self*	handle to current device
in	*hw_device*	reference to inv_device_t used to retrieve base data
in	*opt*	optional parameter given at init time (can be NULL)

7.5.3.3 int INV_EXPORT inv_device_smart_motion_setup (void * *context*)

Initialize SmartMotion graph.

Will perform HW access through underlying device (eg: calling ping()). Underlying device should be setup and all FW/images loaded before calling this method

7.5.4 Variable Documentation

7.5.4.1 struct inv_device_smart_motion_vlistener inv_device_smart_motion::l

placeholder for leaves corresponding to root VSensor

placeholder for leaves VSensorListener

7.6 DeviceEmdWrapper

emd wrapper implementation for device interface

Collaboration diagram for DeviceEmdWrapper:

Classes

- struct inv_device_emd_wrap_icm20xxx
- struct inv_device_emd_wrap_icm20xxx_serial

Functions

- void INV_EXPORT inv_device_emd_wrap_icm20xxx_init (inv_device_emd_wrap_icm20xxx_t *self, const inv_sensor_listener_t *listener, const struct inv_device_emd_wrap_icm20xxx_serial *serial, void *serial↩ _cookie)

  constructor-like function for EMD Wrapper device

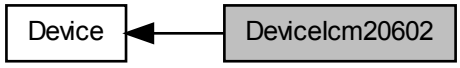
7.6.1 Detailed Description

emd wrapper implementation for device interface

7.7 DeviceIcm20602

Concrete implementation of the 'Device' interface for Icm20602 devices.

Collaboration diagram for DeviceIcm20602:



Classes

- struct inv_device_icm20602

  States for Icm20602 device.
- struct inv_device_icm20602_config_bias_st

  Bias collected during self-test (config INV_DEVICE_ICM20602_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

Typedefs

- typedef struct inv_device_icm20602 inv_device_icm20602_t

  States for Icm20602 device.
- typedef struct inv_device_icm20602_config_bias_st inv_device_icm20602_config_bias_st_t

  Bias collected during self-test (config INV_DEVICE_ICM20602_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.
- typedef int32_t inv_device_icm20602_config_wom_threshold_t

  WOM threshold value (expressed in LSB)

Enumerations

Functions

- void INV_EXPORT inv_device_icm20602_init2 (inv_device_icm20602_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener)

  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20602_init (inv_device_icm20602_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)

  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20602_init_serif_ois2 (inv_device_icm20602_t *self, const inv_serif_hal↩_t *serif_ois)

  Set reference to Serial Interface object for OIS interface.
- void INV_EXPORT inv_device_icm20602_init_serif_ois (inv_device_icm20602_t *self, const inv_host_serif↩_t *serif_ois)

  Set reference to Serial Interface object for OIS interface.
- void INV_EXPORT inv_device_icm20602_init_aux_compass (inv_device_icm20602_t *self, int aux_↩compass_id, uint8_t aux_compass_addr)

  Register a compass as AUX senosr.
- static inv_device_t * inv_device_icm20602_get_base (inv_device_icm20602_t *self)

  Helper function to get handle to base object.

7.7.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20602 devices.

See ExampleDeviceIcm20602.c example.

7.7.2 Function Documentation

7.7.2.1 void INV_EXPORT inv_device_icm20602_init (inv_device_icm20602_t * *self*, const inv_host_serif_t * *serif*, const inv_sensor_listener_t * *listener*)

constructor-like function for basesensor device

Deprecated Use innv_device_icm20602_init2() instead.

Will initialize inv_device_icm20602_t object states to default value for basesensor.

Parameters

in	*self*	handle to device
in	*serif*	reference to legacy Host Serial Interface object
in	*listener*	reference to Sensor Event Listener object

7.7.2.2 void INV_EXPORT inv_device_icm20602_init2 (inv_device_icm20602_t * *self*, const inv_serif_hal_t * *serif*, const inv_sensor_listener_t * *listener*)

constructor-like function for basesensor device

Will initialize inv_device_icm20602_t object states to default value for basesensor.

Parameters

in	*self*	handle to device
in	*serif*	reference to Serial Interface object
in	*listener*	reference to Sensor Event Listener object

Examples:

ExampleDeviceIcm20602EMD.c.

7.7.2.3 void INV_EXPORT inv_device_icm20602_init_aux_compass (inv_device_icm20602_t * *self*, int *aux_compass_id*, uint8_t *aux_compass_addr*)

Register a compass as AUX senosr.

Should be called after inv_device_icm20602_init() but before device setup.

Parameters

in	*aux_compass_id*	auxiliary compass id (as of enum inv_icm20602_compass_id)
in	*aux_compass_add*	I2C slave address for compass

7.7.2.4 void INV_EXPORT inv_device_icm20602_init_serif_ois (inv_device_icm20602_t * *self*, const inv_host_serif_t * *serif_ois*)

Set reference to Serial Interface object for OIS interface.

Deprecated Use innv_device_icm20602_init_serif_ois2() instead.

Parameters

in	*self*	handle to device
in	*serif*	reference to legacy Host Serial Interface object for OIS interface

7.7.2.5 void INV_EXPORT inv_device_icm20602_init_serif_ois2 (inv_device_icm20602_t * *self*, const inv_serif_hal_t * *serif_ois*)

Set reference to Serial Interface object for OIS interface.

When set, a call to the poll() method will also retrieve and report OIS data (if enabled)

Should be called after inv_device_icm20602_init2() but before device setup.

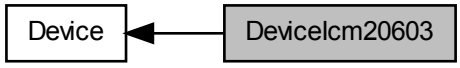
Parameters

in	*self*	handle to device
in	*serif_ois*	reference to Serial Interface object for OIS interface

7.8 DeviceIcm20603

Concrete implementation of the 'Device' interface for Icm20603 devices.

Collaboration diagram for DeviceIcm20603:



Classes

- struct inv_device_icm20603

  States for Icm20603 device.
- struct inv_device_icm20603_config_bias_st

  Bias collected during self-test (config INV_DEVICE_ICM20603_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

Typedefs

- typedef struct inv_device_icm20603 inv_device_icm20603_t

  States for Icm20603 device.
- typedef struct inv_device_icm20603_config_bias_st inv_device_icm20603_config_bias_st_t

  Bias collected during self-test (config INV_DEVICE_ICM20603_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.
- typedef int32_t inv_device_icm20603_config_wom_threshold_t

  WOM threshold value (expressed in LSB)

Enumerations

Functions

- void INV_EXPORT inv_device_icm20603_init2 (inv_device_icm20603_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener)

  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20603_init (inv_device_icm20603_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)

  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20603_init_serif_ois2 (inv_device_icm20603_t *self, const inv_serif_hal↩_t *serif_ois)

  Set reference to Serial Interface object for OIS interface.
- void INV_EXPORT inv_device_icm20603_init_serif_ois (inv_device_icm20603_t *self, const inv_host_serif↩_t *serif_ois)

  Set reference to Serial Interface object for OIS interface.
- void INV_EXPORT inv_device_icm20603_init_aux_compass (inv_device_icm20603_t *self, int aux_↩compass_id, uint8_t aux_compass_addr)

  Register a compass as AUX senosr.
- static inv_device_t * inv_device_icm20603_get_base (inv_device_icm20603_t *self)

  Helper function to get handle to base object.

7.8.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20603 devices.

See ExampleDeviceIcm20603.c example.

7.8.2 Function Documentation

7.8.2.1 void INV_EXPORT inv_device_icm20603_init (inv_device_icm20603_t * *self*, const inv_host_serif_t * *serif*, const inv_sensor_listener_t * *listener*)

constructor-like function for basesensor device

Deprecated Use innv_device_icm20603_init2() instead.

Will initialize inv_device_icm20603_t object states to default value for basesensor.

Parameters

in	*self*	handle to device
in	*serif*	reference to legacy Host Serial Interface object
in	*listener*	reference to Sensor Event Listener object

7.8.2.2 void INV_EXPORT inv_device_icm20603_init2 (inv_device_icm20603_t * *self*, const inv_serif_hal_t * *serif*, const inv_sensor_listener_t * *listener*)

constructor-like function for basesensor device

Will initialize inv_device_icm20603_t object states to default value for basesensor.

Parameters

in	*self*	handle to device
in	*serif*	reference to Serial Interface object
in	*listener*	reference to Sensor Event Listener object

Examples:

ExampleDeviceIcm20603EMD.c.

7.8.2.3 void INV_EXPORT inv_device_icm20603_init_aux_compass (inv_device_icm20603_t * *self*, int *aux_compass_id*, uint8_t *aux_compass_addr*)

Register a compass as AUX senosr.

Should be called after inv_device_icm20603_init() but before device setup.

Parameters

in	*aux_compass_id*	auxiliary compass id (as of enum inv_icm20603_compass_id)
in	*aux_compass_add*	I2C slave address for compass

7.8.2.4 void INV_EXPORT inv_device_icm20603_init_serif_ois (inv_device_icm20603_t * *self*, const inv_host_serif_t * *serif_ois*)

Set reference to Serial Interface object for OIS interface.

Deprecated Use innv_device_icm20603_init_serif_ois2() instead.

Parameters

in	*self*	handle to device
in	*serif*	reference to legacy Host Serial Interface object for OIS interface

7.8.2.5 void INV_EXPORT inv_device_icm20603_init_serif_ois2 (inv_device_icm20603_t * *self*, const inv_serif_hal_t * *serif_ois*)

Set reference to Serial Interface object for OIS interface.

When set, a call to the poll() method will also retrieve and report OIS data (if enabled)

Should be called after inv_device_icm20603_init2() but before device setup.

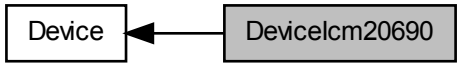
Parameters

in	*self*	handle to device
in	*serif_ois*	reference to Serial Interface object for OIS interface

7.9 DeviceIcm20690

Concrete implementation of the 'Device' interface for Icm20690 devices.

Collaboration diagram for DeviceIcm20690:



Classes

- struct inv_device_icm20690

  States for Icm20690 device.
- struct inv_device_icm20690_config_bias_st

  Bias collected during self-test (config INV_DEVICE_ICM20690_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

Typedefs

- typedef struct inv_device_icm20690 inv_device_icm20690_t

  States for Icm20690 device.
- typedef struct inv_device_icm20690_config_bias_st inv_device_icm20690_config_bias_st_t

  Bias collected during self-test (config INV_DEVICE_ICM20690_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.
- typedef int32_t inv_device_icm20690_config_wom_threshold_t

  WOM threshold value (expressed in LSB)

Enumerations

Functions

- void INV_EXPORT inv_device_icm20690_init2 (inv_device_icm20690_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener)

  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20690_init (inv_device_icm20690_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)

  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20690_init_serif_ois2 (inv_device_icm20690_t *self, const inv_serif_hal_t *serif_ois)

  Set reference to Serial Interface object for OIS interface.
- void INV_EXPORT inv_device_icm20690_init_serif_ois (inv_device_icm20690_t *self, const inv_host_serif_t *serif_ois)

  Set reference to Serial Interface object for OIS interface.
- void INV_EXPORT inv_device_icm20690_init_aux_compass (inv_device_icm20690_t *self, int aux_compass_id, uint8_t aux_compass_addr)

  Register a compass as AUX senosr.
- static inv_device_t * inv_device_icm20690_get_base (inv_device_icm20690_t *self)

  Helper function to get handle to base object.

7.9.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20690 devices.

See ExampleDeviceIcm20690.c example.

7.9.2 Function Documentation

7.9.2.1 void INV_EXPORT inv_device_icm20690_init (inv_device_icm20690_t * *self*, const inv_host_serif_t * *serif*, const inv_sensor_listener_t * *listener*)

constructor-like function for basesensor device

Deprecated Use innv_device_icm20690_init2() instead.

Will initialize inv_device_icm20690_t object states to default value for basesensor.

Parameters

in	*self*	handle to device
in	*serif*	reference to legacy Host Serial Interface object
in	*listener*	reference to Sensor Event Listener object

7.9.2.2 void INV_EXPORT inv_device_icm20690_init2 (inv_device_icm20690_t * *self*, const inv_serif_hal_t * *serif*, const inv_sensor_listener_t * *listener*)

constructor-like function for basesensor device

Will initialize inv_device_icm20690_t object states to default value for basesensor.

Parameters

in	*self*	handle to device
in	*serif*	reference to Serial Interface object
in	*listener*	reference to Sensor Event Listener object

Examples:

ExampleDeviceIcm20690EMD.c.

7.9.2.3 void INV_EXPORT inv_device_icm20690_init_aux_compass (inv_device_icm20690_t * *self*, int *aux_compass_id*, uint8_t *aux_compass_addr*)

Register a compass as AUX senosr.

Should be called after inv_device_icm20690_init() but before device setup.

Parameters

in	*aux_compass_id*	auxiliary compass id (as of enum inv_icm20690_compass_id)
in	*aux_compass_add*	I2C slave address for compass

7.9.2.4 void INV_EXPORT inv_device_icm20690_init_serif_ois (inv_device_icm20690_t * *self*, const inv_host_serif_t * *serif_ois*)

Set reference to Serial Interface object for OIS interface.

Deprecated Use innv_device_icm20690_init_serif_ois2() instead.

Parameters

in	*self*	handle to device
in	*serif*	reference to legacy Host Serial Interface object for OIS interface

7.9.2.5 void INV_EXPORT inv_device_icm20690_init_serif_ois2 (inv_device_icm20690_t * *self*, const inv_serif_hal_t * *serif_ois*)

Set reference to Serial Interface object for OIS interface.

When set, a call to the poll() method will also retrieve and report OIS data (if enabled)

Should be called after inv_device_icm20690_init2() but before device setup.

Parameters

in	*self*	handle to device
in	*serif_ois*	reference to Serial Interface object for OIS interface

7.10 Host Serial Interface

Virtual abstraction of host adapter for serial interface.

Collaboration diagram for Host Serial Interface:

Classes

- struct inv_host_serif

  Serial Interface interface definition.

Typedefs

- typedef struct inv_host_serif inv_host_serif_t

  Serial Interface interface definition.

Enumerations

Functions

- void INV_EXPORT inv_host_serif_set_instance (const inv_host_serif_t *instance)

  Set global instance for Serial Interface.
- const inv_host_serif_t * inv_host_serif_get_instance (void)

  Return global instance for Serial Interface.
- static int inv_host_serif_open (const inv_host_serif_t *instance)

  Helper method to call open() method of a Serial Interface object.
- static int inv_host_serif_close (const inv_host_serif_t *instance)

  Helper method to call close() method of a Serial Interface object.
- static int inv_host_serif_read_reg (const inv_host_serif_t *instance, uint8_t reg, uint8_t *data, uint32_t len)

  Helper method to call read_reg() method of a Serial Interface object.
- static int inv_host_serif_write_reg (const inv_host_serif_t *instance, uint8_t reg, const uint8_t *data, uint32_t len)

  Helper method to call write_reg() method of a Serial Interface object.
- static int inv_host_serif_register_interrupt_callback (const inv_host_serif_t *instance, void(*interrupt_cb)(void *context, int int_num), void *context)

  Helper method to call register_interrupt_callback() method of a Serial Interface object.
- static int inv_host_serif_get_type (const inv_host_serif_t *instance)

  Helper method to get serial interface type of a Serial Interface object.
- static uint32_t inv_host_serif_get_max_read_transaction_size (const inv_host_serif_t *instance)

Helper method to get max read size value of a Serial Interface object.

- static uint32_t inv_host_serif_get_max_write_transaction_size (const inv_host_serif_t *instance)

  Helper method to get max write size value of a Serial Interface object.
- static int inv_host_serif_is_i2c (const inv_host_serif_t *instance)

  Helper method to check if serial interface type is I2C for a Serial Interface object.
- static int inv_host_serif_is_spi (const inv_host_serif_t *instance)

  Helper method to check if serial interface type is SPI for a Serial Interface object.

7.10.1 Detailed Description

Virtual abstraction of host adapter for serial interface.

Deprecated Use SerifHal.h instead

7.11 Data Converter

Helper functions to convert integer.

Functions

- uint8_t INV_EXPORT * inv_dc_int32_to_little8 (int32_t x, uint8_t *little8)
  Converts a 32-bit long to a little endian byte stream.
- uint8_t INV_EXPORT * inv_dc_int16_to_little8 (int16_t x, uint8_t *little8)
  Converts a 16-bit integer to a little endian byte stream.
- uint8_t INV_EXPORT * inv_dc_int32_to_big8 (int32_t x, uint8_t *big8)
  Converts a 32-bit long to a big endian byte stream.
- int32_t INV_EXPORT inv_dc_little8_to_int32 (const uint8_t *little8)
  Converts a little endian byte stream into a 32-bit integer.
- int16_t INV_EXPORT inv_dc_le_to_int16 (const uint8_t *little8)
  Converts a little endian byte stream into a 16-bit integer.
- int16_t INV_EXPORT inv_dc_big16_to_int16 (uint8_t *data)
  Converts big endian on 16 bits into an unsigned short.
- void INV_EXPORT inv_dc_sfix32_to_float (const int32_t *in, uint32_t len, uint8_t qx, float *out)
  Converts an array of 32-bit signed fixed-point integers to an array of floats.
- void INV_EXPORT inv_dc_float_to_sfix32 (const float *in, uint32_t len, uint8_t qx, int32_t *out)
  Converts an array of floats to an array of 32-bit signed fixed-point integers.

7.11.1 Detailed Description

Helper functions to convert integer.

7.11.2 Function Documentation

7.11.2.1 void INV_EXPORT inv_dc_float_to_sfix32 (const float * *in*, uint32_t *len*, uint8_t *qx*, int32_t * *out*)

Converts an array of floats to an array of 32-bit signed fixed-point integers.

Parameters

in	*in*	Pointer to the first element of the array of floats
in	*len*	Length of the array
in	*qx*	Number of bits used to represent the decimal part of the fixed-point integers
out	*out*	Pointer to the memory area where the output will be stored

7.11.2.2 void INV_EXPORT inv_dc_sfix32_to_float (const int32_t * *in*, uint32_t *len*, uint8_t *qx*, float * *out*)

Converts an array of 32-bit signed fixed-point integers to an array of floats.

Parameters

in	*in*	Pointer to the first element of the array of 32-bit signed fixed-point integers
in	*len*	Length of the array
in	*qx*	Number of bits used to represent the decimal part of the fixed-point integers
out	*out*	Pointer to the memory area where the output will be stored

7.12 Error Helper

Helper functions realted to error code.

Functions

- const char INV_EXPORT * inv_error_str (int error)
  Returns string describing error number.

7.12.1 Detailed Description

Helper functions realted to error code.

7.12.2 Function Documentation

7.12.2.1 const char INV_EXPORT* inv_error_str (int *error*)

Returns string describing error number.

See also

enum inv_error

7.13 Message

Utility functions to display and redirect diagnostic messages.

Macros

- #define INV_MSG_DISABLE 1

  For eMD target, disable log by default If compile switch is set for a compilation unit messages will be totally disabled by default.
- #define INV_MSG(level, ...) _INV_MSG(level, __VA_ARGS__)

  Allow to force enabling messaging using INV_MSG_ENABLE define.
- #define INV_MSG_SETUP(level, printer) _INV_MSG_SETUP(level, printer)

  Helper macro for calling inv_msg_setup() If INV_MSG_DISABLE compile switch is set for a compilation unit messages will be totally disabled.
- #define INV_MSG_SETUP_LEVEL(level) _INV_MSG_SETUP_LEVEL(level)

  Helper macro for calling inv_msg_setup_level() If INV_MSG_DISABLE compile switch is set for a compilation unit messages will be totally disabled.
- #define INV_MSG_SETUP_DEFAULT() _INV_MSG_SETUP_DEFAULT()

  Helper macro for calling inv_msg_setup_default() If INV_MSG_DISABLE compile switch is set for a compilation unit messages will be totally disabled.
- #define INV_MSG_LEVEL _INV_MSG_LEVEL

  Return current level.

Typedefs

- typedef void(* inv_msg_printer_t) (int level, const char *str, va_list ap)

  Prototype for print routine function.

Enumerations

Functions

- void INV_EXPORT inv_msg_setup (int level, inv_msg_printer_t printer)

  Set message level and printer function.
- void INV_EXPORT inv_msg_printer_default (int level, const char *str, va_list ap)

  Default printer function that display messages to stderr Function uses stdio.
- static void inv_msg_setup_level (int level)

  Set message level Default printer function will be used.
- static void inv_msg_setup_default (void)

  Set default message level and printer.
- int INV_EXPORT inv_msg_get_level (void)

  Return current message level.
- void INV_EXPORT inv_msg (int level, const char *str,...)

  Display a message (through means of printer function)

7.13.1 Detailed Description

Utility functions to display and redirect diagnostic messages.

Use INV_MSG_DISABLE or INV_MSG_ENABLE define before including this header to enable/disable messages for a compilation unit.

Under Linux, Windows or Arduino, messages are enabled by default. Use INV_MSG_DISABLE to disable them.

Under orther environmment, message are disabled by default. Use INV_MSG_ENABLE to disable them.

7.13.2 Macro Definition Documentation

7.13.2.1 #define INV_MSG(*level,* ...) _INV_MSG(level, __VA_ARGS__)

Allow to force enabling messaging using INV_MSG_ENABLE define.

Helper macro for calling inv_msg() If INV_MSG_DISABLE compile switch is set for a compilation unit messages will be totally disabled

7.13.2.2 #define INV_MSG_LEVEL _INV_MSG_LEVEL

Return current level.

Warning

This macro may expand as a function call

7.13.3 Function Documentation

7.13.3.1 void INV_EXPORT inv_msg (int *level,* const char * *str,* ...)

Display a message (through means of printer function)

Parameters

in	*level*	for the message
in	*str*	message string
in	...	optional arguments

Returns

none

7.13.3.2 int INV_EXPORT inv_msg_get_level (void)

Return current message level.

Returns

current message level

7.13.3.3 void INV_EXPORT inv_msg_printer_default (int *level,* const char * *str,* va_list *ap*)

Default printer function that display messages to stderr Function uses stdio.

Care must be taken on embeded platfrom. Function does nothing with IAR compiler.

Returns

none

7.13.3.4 void INV_EXPORT inv_msg_setup (int *level,* inv_msg_printer_t *printer*)

Set message level and printer function.

Parameters

in	*level*	only message above level will be passed to printer function
in	*printer*	user provided function in charge printing message

Returns

none

7.13.3.5 static void inv_msg_setup_default (void) [inline], [static]

Set default message level and printer.

Returns

none

7.13.3.6 static void inv_msg_setup_level (int *level*) [inline], [static]

Set message level Default printer function will be used.

Parameters

in	*level*	only message above level will be passed to printer function

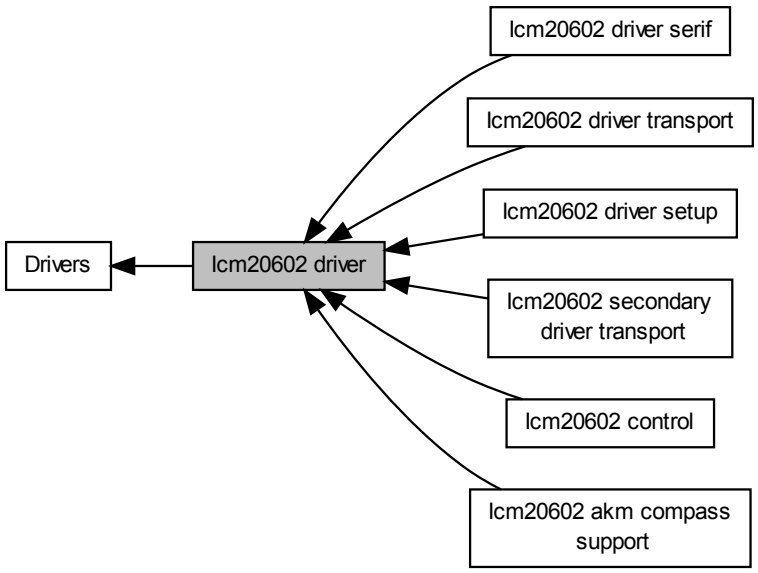
Returns

none

7.14 Icm20602 driver

Low-level driver for Icm20602 devices.

Collaboration diagram for Icm20602 driver:



Modules

- Icm20602 akm compass support

  Low-level Icm20602 aux sensor access.
- Icm20602 secondary driver transport

  Low-level Icm20602 secondary interface access.
- Icm20602 control

  Low-level function to control a Icm20602 device.
- Icm20602 driver serif

  Interface for low-level serial (I2C/SPI) access.
- Icm20602 driver setup

  Low-level function to setup an Icm20602 device.
- Icm20602 driver transport

  Low-level Icm20602 register access.

Classes

- struct inv_icm20602

  Icm20602 driver states definition.

Typedefs

- typedef struct inv_icm20602 inv_icm20602_t

  Icm20602 driver states definition.

Functions

- void inv_icm20602_sleep (int ms)

  Hook for low-level system sleep() function to be implemented by upper layer.
- void inv_icm20602_sleep_us (int us)

  Hook for low-level high res system sleep() function to be implemented by upper layer ~100us resolution is sufficient.
- uint64_t inv_icm20602_get_time_us (void)

  Hook for low-level system time() function to be implemented by upper layer.
- uint64_t inv_icm20602_get_dataready_interrupt_time_us (void)

  Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get_time_us()
- static void inv_icm20602_reset_states (struct inv_icm20602 *s, const struct inv_icm20602_serif *serif)

  Reset and initialize driver states.
- static void inv_icm20602_reset_states_serif_ois (struct inv_icm20602 *s, const struct inv_icm20602_serif *serif_ois)

  Register secondary SERIF object for OIS access Must be called after inv_icm20602_reset_states()

7.14.1 Detailed Description

Low-level driver for Icm20602 devices.

7.14.2 Function Documentation

7.14.2.1 uint64_t inv_icm20602_get_dataready_interrupt_time_us (void)

Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get_time_us()

Returns

data ready interrupt timestamp in us

7.14.2.2 uint64_t inv_icm20602_get_time_us (void)

Hook for low-level system time() function to be implemented by upper layer.

Returns

monotonic timestamp in us

7.14.2.3 static void inv_icm20602_reset_states (struct inv_icm20602 * *s*, const struct inv_icm20602_serif * *serif*) [inline], [static]

Reset and initialize driver states.

Parameters

in	*s*	handle to driver states structure
in	*serif*	handle to SERIF object for underlying register access

7.14.2.4 static void inv_icm20602_reset_states_serif_ois (struct inv_icm20602 * *s*, const struct inv_icm20602_serif * *serif_ois*) [inline], [static]

Register secondary SERIF object for OIS access Must be called after inv_icm20602_reset_states()

Parameters

in	*s*	handle to driver states structure
in	*serif*	handle to SERIF object for underlying register access to OIS

7.14.2.5 void inv_icm20602_sleep (int *ms*)

Hook for low-level system sleep() function to be implemented by upper layer.

Parameters

in	*ms*	number of millisecond the calling thread should sleep

7.14.2.6 void inv_icm20602_sleep_us (int *us*)

Hook for low-level high res system sleep() function to be implemented by upper layer ~100us resolution is sufficient.

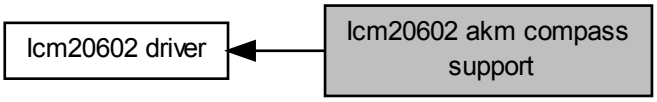
Parameters

in	*us*	number of us the calling thread should sleep

7.15 Icm20602 akm compass support

Low-level Icm20602 aux sensor access.

Collaboration diagram for Icm20602 akm compass support:



Enumerations

Functions

- void INV_EXPORT inv_icm20602_register_aux_compass (struct inv_icm20602 *s, enum inv_icm20602_↩ compass_id compass_id, uint8_t compass_i2c_addr)

  Register AUX compass.
- int INV_EXPORT inv_icm20602_is_compass_registered (struct inv_icm20602 *s)

  Return non-zero value is AUX compass was regitered, 0 if not If compass was registered but setup failed, will also return 0.
- int INV_EXPORT inv_icm20602_setup_compass_akm (struct inv_icm20602 *s)

  Initializes the compass.
- int INV_EXPORT inv_icm20602_check_akm_self_test (struct inv_icm20602 *s)

  Self test for the compass.
- int INV_EXPORT inv_icm20602_read_akm_scale (struct inv_icm20602 *s, int *scale)

  Reads the scale of the compass.
- int INV_EXPORT inv_icm20602_suspend_akm (struct inv_icm20602 *s)

  Stops the compass.
- int INV_EXPORT inv_icm20602_resume_akm (struct inv_icm20602 *s)

  Starts the compass.
- char INV_EXPORT inv_icm20602_compass_getstate (struct inv_icm20602 *s)

  Get compass power status.
- int INV_EXPORT inv_icm20602_get_compass_data (struct inv_icm20602 *s, const unsigned char *packet, unsigned char *raw_compass)

  Parse compass data packet.
- int INV_EXPORT inv_icm20602_get_compass_bytes (struct inv_icm20602 *s)

  Get data packet size according to comass type.

7.15.1 Detailed Description

Low-level Icm20602 aux sensor access.

7.15.2 Enumeration Type Documentation

7.15.2.1 enum inv_icm20602_compass_id

Supported auxiliary compass identifer.

Enumerator

INV_ICM20602_COMPASS_ID_NONE no compass

INV_ICM20602_COMPASS_ID_AK09911 AKM AK09911.

INV_ICM20602_COMPASS_ID_AK09912 AKM AK09912.

7.15.3 Function Documentation

7.15.3.1 int INV_EXPORT inv_icm20602_check_akm_self_test (struct inv_icm20602 * *s*)

Self test for the compass.

Returns

0 in case of success, -1 for any error

7.15.3.2 char INV_EXPORT inv_icm20602_compass_getstate (struct inv_icm20602 * *s*)

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

7.15.3.3 int INV_EXPORT inv_icm20602_get_compass_bytes (struct inv_icm20602 * *s*)

Get data packet size according to comass type.

Returns

size in bytes

7.15.3.4 int INV_EXPORT inv_icm20602_get_compass_data (struct inv_icm20602 * *s*, const unsigned char * *packet*, unsigned char * *raw_compass*)

Parse compass data packet.

Parameters

in	*data*	packet pointer to compass data packet
in	*raw_compass*	pointer to raw compass. big endian 2 bytes for each axis.

Returns

0 in case of success, -1 for any error

7.15.3.5 int INV_EXPORT inv_icm20602_read_akm_scale (struct inv_icm20602 * *s*, int * *scale*)

Reads the scale of the compass.

Parameters

out	*scale*	pointer to recuperate the scale

Returns

0 in case of success, -1 for any error

7.15.3.6 void INV_EXPORT inv_icm20602_register_aux_compass (struct inv_icm20602 * *s*, enum inv_icm20602_compass_id *compass_id*, uint8_t *compass_i2c_addr*)

Register AUX compass.

Will only set internal states and won't perform any transaction on the bus. Must be called before inv_icm20602_↩ initialize().

Parameters

in	*compass_id*	Compass ID
in	*compass_i2c_addr*	Compass I2C address

Returns

0 on success, negative value on error

7.15.3.7 int INV_EXPORT inv_icm20602_resume_akm (struct inv_icm20602 * *s*)

Starts the compass.

Returns

0 in case of success, -1 for any error

7.15.3.8 int INV_EXPORT inv_icm20602_setup_compass_akm (struct inv_icm20602 * s)

Initializes the compass.

Returns

0 in case of success, -1 for any error

7.15.3.9 int INV_EXPORT inv_icm20602_suspend_akm (struct inv_icm20602 * s)

Stops the compass.

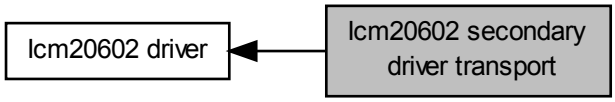
Returns

0 in case of success, -1 for any error

7.16 Icm20602 secondary driver transport

Low-level Icm20602 secondary interface access.

Collaboration diagram for Icm20602 secondary driver transport:



Macros

- #define COMPASS_I2C_SLV_READ 0

  Secondary I2C channel usage :

Functions

- void inv_icm20602_init_secondary (struct inv_icm20602 ∗s)

  Initializes the register for the i2c communication.
- int inv_icm20602_read_secondary (struct inv_icm20602 ∗s, int index, unsigned char addr, unsigned char reg, char len)

  Reads data in i2c a secondary device.
- int inv_icm20602_execute_read_secondary (struct inv_icm20602 ∗s, int index, unsigned char addr, int reg, int len, unsigned char ∗d)

  Reads data in i2c a secondary device directly.
- int inv_icm20602_write_secondary (struct inv_icm20602 ∗s, int index, unsigned char addr, unsigned char reg, char v)

  Writes data in i2c a secondary device.
- int inv_icm20602_execute_write_secondary (struct inv_icm20602 ∗s, int index, unsigned char addr, int reg, unsigned char v)

  Writes data in i2c a secondary device directly.
- int inv_icm20602_secondary_stop_channel (struct inv_icm20602 ∗s, int index)

  Stop one secondary I2C channel by writing 0 in its control register.
- int inv_icm20602_secondary_enable_i2c (struct inv_icm20602 ∗s)

  Enable secondary I2C interface.
- int inv_icm20602_secondary_disable_i2c (struct inv_icm20602 ∗s)

  Stop secondary I2C interface.

7.16.1 Detailed Description

Low-level Icm20602 secondary interface access.

7.16.2 Macro Definition Documentation

7.16.2.1 #define COMPASS_I2C_SLV_READ 0

Secondary I2C channel usage :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register

7.16.3 Function Documentation

7.16.3.1 int inv_icm20602_execute_read_secondary (struct inv_icm20602 * *s,* int *index,* unsigned char *addr,* int *reg,* int *len,* unsigned char * *d*)

Reads data in i2c a secondary device directly.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be read on the secondary device
in	*len*	Size of data to be read
out	*d*	pointer to the data to be read

Returns

0 in case of success, -1 for any error

7.16.3.2 int inv_icm20602_execute_write_secondary (struct inv_icm20602 * *s,* int *index,* unsigned char *addr,* int *reg,* unsigned char *v*)

Writes data in i2c a secondary device directly.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be write on the secondary device
in	*v*	the data to be written

Returns

0 in case of success, -1 for any error

7.16.3.3 int inv_icm20602_read_secondary (struct inv_icm20602 * s, int index, unsigned char addr, unsigned char reg, char len)

Reads data in i2c a secondary device.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	I2c address slave of the secondary slave
in	*reg*	the register to be read on the secondary device
in	*len*	Size of data to be read

Returns

0 in case of success, -1 for any error

7.16.3.4 int inv_icm20602_secondary_disable_i2c (struct inv_icm20602 * s)

Stop secondary I2C interface.

Returns

0 in case of success, -1 for any error

Warning

It stops all I2C transactions, whatever the channel status

7.16.3.5 int inv_icm20602_secondary_enable_i2c (struct inv_icm20602 * s)

Enable secondary I2C interface.

Returns

0 in case of success, -1 for any error

7.16.3.6 int inv_icm20602_secondary_stop_channel (struct inv_icm20602 * s, int index)

Stop one secondary I2C channel by writing 0 in its control register.

Parameters

in	*index*	the channel id to be stopped

Returns

0 in case of success, -1 for any error

Warning

It does not stop I2C secondary interface, just one channel

7.16.3.7 int inv_icm20602_write_secondary (struct inv_icm20602 * *s*, int *index*, unsigned char *addr*, unsigned char *reg*, char *v*)

Writes data in i2c a secondary device.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be write on the secondary device
in	*v*	the data to be written

Returns

0 in case of success, -1 for any error

7.17 Icm20602 control

Low-level function to control a Icm20602 device.

Collaboration diagram for Icm20602 control:

Classes

- struct inv_icm20602_fifo_states

  Check and retrieve for new data.

Enumerations

Functions

- int INV_EXPORT inv_icm20602_enable_mems (struct inv_icm20602 *s, int bit_mask, uint16_t smplrt_divider)

  Enables accel and/or gyro and/or pressure if integrated with gyro and accel.
- int INV_EXPORT inv_icm20602_set_sensor_period (struct inv_icm20602 *s, enum inv_icm20602_sensor sensor, uint16_t delayInMs)

  Sets the odr for a sensor.
- int INV_EXPORT inv_icm20602_is_sensor_enabled (struct inv_icm20602 *s, enum inv_icm20602_sensor sensor)

  Knows if the sensor is enabled or disbaled.
- int INV_EXPORT inv_icm20602_enable_sensor (struct inv_icm20602 *s, enum inv_icm20602_sensor sensor, uint8_t enable)

  Enables / disables a sensor.
- int INV_EXPORT inv_icm20602_configure_accel_wom (struct inv_icm20602 *s, uint8_t wom_threshold)

  Configuress accel WOM.
- int INV_EXPORT inv_icm20602_poll_sensor_data_reg (struct inv_icm20602 *s, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3])

  Check and retrieve for new data.
- int INV_EXPORT inv_icm20602_poll_fifo_data_setup (struct inv_icm20602 *s, struct inv_icm20602_fifo_states *states, uint8_t int_status)

  Initializes states for FIFO reading and parsing.
- int INV_EXPORT inv_icm20602_poll_fifo_data (struct inv_icm20602 *s, struct inv_icm20602_fifo_states *states, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3], int16_t compass_data[3])

  Read one packet from the FIFO (packet corresponding to data for all current active sensor)
- int INV_EXPORT inv_icm20602_has_data_ready (struct inv_icm20602 *s)

  Read interrupt status and check for DDRY flag.
- int INV_EXPORT inv_icm20602_get_int_status (struct inv_icm20602 *s, uint8_t *int_status)

Read interrupt status and return its value.

- int INV_EXPORT inv_icm20602_check_drdy (struct inv_icm20602 *s, uint8_t int_status)

  Check for DRDY flag in INT register value.
- int INV_EXPORT inv_icm20602_check_wom_status (struct inv_icm20602 *s, uint8_t int_status)

  Check for WOM bits in INT register value.
- int INV_EXPORT inv_icm20602_disable_fifo (struct inv_icm20602 *s)

  Disable FIFO.
- int INV_EXPORT inv_icm20602_enable_fifo (struct inv_icm20602 *s, int bit_mask)

  Enable FIFO.
- int INV_EXPORT inv_icm20602_reset_fifo (struct inv_icm20602 *s)

  Reset FIFO.
- int INV_EXPORT inv_icm20602_set_slave_compass_id (struct inv_icm20602 *s)

  Initalize AUX I2C and Compass.
- int INV_EXPORT inv_icm20602_poll_ois_gyro_data (struct inv_icm20602 *s, int16_t gyro_data[3])

  Poll OIS gyro data.
- int INV_EXPORT inv_icm20602_poll_delay_count (struct inv_icm20602 *s, int16_t *delay_count)

  Poll EIS flag and delay counter.
- int INV_EXPORT inv_icm20602_all_sensors_off (struct inv_icm20602 *s)

  test if all sensors are off
- int inv_icm20602_run_selftest (struct inv_icm20602 *s)

  Perform hardware self-test for Accel, Gyro and Compass.
- void INV_EXPORT inv_icm20602_get_st_bias (struct inv_icm20602 *s, int *st_bias)

  Retrieve bias collected by self-test.
- void inv_icm20602_set_st_bias (struct inv_icm20602 *s, int *st_bias)

  Apply bias.

7.17.1 Detailed Description

Low-level function to control a Icm20602 device.

7.17.2 Function Documentation

7.17.2.1 int INV_EXPORT inv_icm20602_all_sensors_off (struct inv_icm20602 * *s*)

test if all sensors are off

Returns

true if all sensors are off, false otherwise

7.17.2.2 int INV_EXPORT inv_icm20602_check_drdy (struct inv_icm20602 * *s*, uint8_t *int_status*)

Check for DRDY flag in INT register value.

Parameters

in	*int_status*	INT status register value

Returns

1 if DDRDY is set, 0 otherwise

7.17.2.3 int INV_EXPORT inv_icm20602_check_wom_status (struct inv_icm20602 * s, uint8_t int_status)

Check for WOM bits in INT register value.

Parameters

in	*int_status*	INT status register value

Returns

bit mask corresponding to x, y, z axis that caused WOM (0x01: x axis, 0x02: y axis, 0x01: z axis)

7.17.2.4 int INV_EXPORT inv_icm20602_configure_accel_wom (struct inv_icm20602 * s, uint8_t wom_threshold)

Configuress accel WOM.

Parameters

in	*wom_threshold*	threshold value for X,Y,Z axis that should trigger a WOM interrupt (0 to diable WOM)

Returns

0 on success, negative value on error.

7.17.2.5 int INV_EXPORT inv_icm20602_disable_fifo (struct inv_icm20602 * s)

Disable FIFO.

Returns

0 on success, negative value on error.

7.17.2.6 int INV_EXPORT inv_icm20602_enable_fifo (struct inv_icm20602 * s, int bit_mask)

Enable FIFO.

Parameters

in	*bit_mask*	A mask of sensor to push to FIFO.

Returns

0 on success, negative value on error.

7.17.2.7 int INV_EXPORT inv_icm20602_enable_mems (struct inv_icm20602 * *s*, int *bit_mask*, uint16_t *smplrt_divider*)

Enables accel and/or gyro and/or pressure if integrated with gyro and accel.

Parameters

in	*bit_mask*	A mask where 2 means turn on accel, 1 means turn on gyro, 4 is for pressure. By default, this only turns on a sensor if all sensors are off otherwise the DMP controls this register including turning off a sensor. To override this behavior add in a mask of 128.
in	*smplrt_divider*	The divider which was applied to internal sample rate based on field sample_rate from base_driver_t to get minimum ODR for accel and gyro

Returns

0 on success, negative value on error

7.17.2.8 int INV_EXPORT inv_icm20602_enable_sensor (struct inv_icm20602 * *s*, enum inv_icm20602_sensor *sensor*, uint8_t *enable*)

Enables / disables a sensor.

Parameters

in	*androidSensor*	Sensor Identity
in	*enable*	0=off, 1=on

Returns

0 in case of success, negative value on error

7.17.2.9 int INV_EXPORT inv_icm20602_get_int_status (struct inv_icm20602 * *s*, uint8_t * *int_status*)

Read interrupt status and return its value.

Parameters

out	*int_status*	INT status register value

Returns

0 on success, negative value on error

7.17.2.10 void INV_EXPORT inv_icm20602_get_st_bias (struct inv_icm20602 * s, int * st_bias)

Retrieve bias collected by self-test.

Parameters

out	*st_bias*	bias scaled by 2^16, accel is gee and gyro is dps. The buffer will be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z

7.17.2.11 int INV_EXPORT inv_icm20602_has_data_ready (struct inv_icm20602 * s)

Read interrupt status and check for DDRY flag.

Returns

1 if DDRY inteript is set, 0 otherwise and negative value on error

7.17.2.12 int INV_EXPORT inv_icm20602_is_sensor_enabled (struct inv_icm20602 * s, enum inv_icm20602_sensor sensor)

Knows if the sensor is enabled or disbaled.

Parameters

in	*sensor*	the sensor is enabled or not

Returns

1 if active, 0 otherwise

7.17.2.13 int INV_EXPORT inv_icm20602_poll_delay_count (struct inv_icm20602 * s, int16_t * delay_count)

Poll EIS flag and delay counter.

Parameters

out	*delay_count*	delay time in us between the FSYNC event (before the gyro data event) and the gyro data event

Returns

0x8 if FSYNC event reported, 0 otherwise, negative value on error.

7.17.2.14 int INV_EXPORT inv_icm20602_poll_fifo_data (struct inv_icm20602 * s, struct inv_icm20602_fifo_states * states, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3], int16_t compass_data[3])

Read one packet from the FIFO (packet corresponding to data for all current active sensor)

Parameters

in	*states*	placeholder to FIFO states
out	*acc_data*	raw gyro data
out	*temp_data*	raw temp data
out	*gyro_data*	raw gyro data
out	*compass_data*	raw compass data

Returns

0 : FIFO empty bit0 : gyro data output set bit1 : acc data output set bit2 : compass output set $<$ 0 : error

7.17.2.15 int INV_EXPORT inv_icm20602_poll_fifo_data_setup (struct inv_icm20602 * s, struct inv_icm20602_fifo_states * states, uint8_t int_status)

Initializes states for FIFO reading and parsing.

Parameters

in	*states*	placeholder to FIFO states
in	*int_status*	int status register value (used to check for overflow)

Returns

0 for success, 1 for overlfow detected, negative value for other errors

7.17.2.16 int INV_EXPORT inv_icm20602_poll_ois_gyro_data (struct inv_icm20602 * s, int16_t gyro_data[3])

Poll OIS gyro data.

Returns

0x10 if OIS data reported, 0 otherwise, negative value on error.

7.17.2.17 int INV_EXPORT inv_icm20602_poll_sensor_data_reg (struct inv_icm20602 * s, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3])

Check and retrieve for new data.

Parameters

out	*acc_data*	raw gyro data
out	*temp_data*	raw temp data
out	*gyro_data*	raw gyro data

Returns

0 : not data ready 0x01 : gyro data output set 0x02 : acc data output set 0x03 : gyro and acc data output set $<$ 0 : error

7.17.2.18 int INV_EXPORT inv_icm20602_reset_fifo (struct inv_icm20602 * *s*)

Reset FIFO.

Returns

0 on success, negative value on error.

7.17.2.19 int inv_icm20602_run_selftest (struct inv_icm20602 * *s*)

Perform hardware self-test for Accel, Gyro and Compass.

Parameters

in	*None*	

Returns

COMPASS_SUCCESS$<<$2 | ACCEL_SUCCESS$<<$1 | GYRO_SUCCESS so 3

7.17.2.20 int INV_EXPORT inv_icm20602_set_sensor_period (struct inv_icm20602 * *s*, enum inv_icm20602_sensor *sensor*, uint16_t *delayInMs*)

Sets the odr for a sensor.

Parameters

in	*sensor*	Sensor Identity
in	*delayInMs*	the delay between two values in ms

Returns

0 in case of success, -1 for any error

7.17.2.21 int INV_EXPORT inv_icm20602_set_slave_compass_id (struct inv_icm20602 * s)

Initalize AUX I2C and Compass.

Returns

0 on success, negative value on error.

7.17.2.22 void inv_icm20602_set_st_bias (struct inv_icm20602 * s, int * st_bias)

Apply bias.

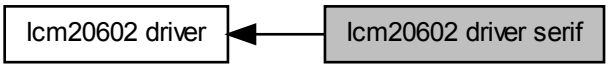
Parameters

in	*st_bias*	bias scaled by 2^16, accel is gee and gyro is dps. The buffer should be be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z

7.18 Icm20602 driver serif

Interface for low-level serial (I2C/SPI) access.

Collaboration diagram for Icm20602 driver serif:



Classes

- struct inv_icm20602_serif

  basesensor serial interface

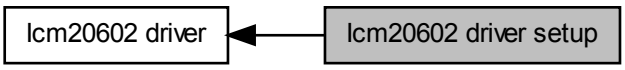
7.18.1 Detailed Description

Interface for low-level serial (I2C/SPI) access.

7.19 Icm20602 driver setup

Low-level function to setup an Icm20602 device.

Collaboration diagram for Icm20602 driver setup:



Functions

- int INV_EXPORT inv_icm20602_get_whoami (struct inv_icm20602 *s, uint8_t *whoami)
  return WHOAMI value
- int INV_EXPORT inv_icm20602_get_chip_info (struct inv_icm20602 *s, uint8_t chip_info[3])
  return WHOAMI value
- int INV_EXPORT inv_icm20602_initialize (struct inv_icm20602 *s)
  Initialize the device.
- int INV_EXPORT inv_icm20602_soft_reset (struct inv_icm20602 *s)
  Perform a soft reset of the device.
- int INV_EXPORT inv_icm20602_set_divider (struct inv_icm20602 *s, uint8_t idiv)
  Set the mpu sample rate.
- int INV_EXPORT inv_icm20602_set_chip_power_state (struct inv_icm20602 *s, uint8_t func, uint8_t on_off)
  Sets the power state of the Ivory chip loop.
- uint8_t INV_EXPORT inv_icm20602_get_chip_power_state (struct inv_icm20602 *s)
  Current wake status of the Mems chip.
- uint16_t INV_EXPORT inv_icm20602_get_chip_base_sample_rate (struct inv_icm20602 *s)
  Get internal sample rate currently configured.
- int INV_EXPORT inv_icm20602_accel_fsr_2_reg (int32_t fsr)
  Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.
- int INV_EXPORT inv_icm20602_reg_2_accel_fsr (uint8_t reg)
  Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.
- int INV_EXPORT inv_icm20602_set_accel_fullscale (struct inv_icm20602 *s, int level)
  Sets fullscale range of accel in hardware.
- int INV_EXPORT inv_icm20602_set_accel_ois_fullscale (struct inv_icm20602 *s, int level)
  Sets fullscale range of OIS accel in hardware.
- int INV_EXPORT inv_icm20602_set_accel_bandwidth (struct inv_icm20602 *s, int level)
  Sets bandwidth range of accel in hardware.
- uint8_t INV_EXPORT inv_icm20602_get_accel_fullscale (struct inv_icm20602 *s)
  Returns fullscale range of accelerometer in hardware.
- uint8_t INV_EXPORT inv_icm20602_get_accel_ois_fullscale (struct inv_icm20602 *s)
  Returns fullscale range of OIS accelerometer in hardware.
- uint16_t INV_EXPORT inv_icm20602_get_accel_bandwidth (struct inv_icm20602 *s)

Returns bandwidth range of accelerometer in hardware.

- int INV_EXPORT inv_icm20602_gyro_fsr_2_reg (int32_t fsr)

  Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)
- int INV_EXPORT inv_icm20602_reg_2_gyro_fsr (uint8_t reg)

  Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)
- int INV_EXPORT inv_icm20602_set_gyro_fullscale (struct inv_icm20602 *s, int level)

  Sets fullscale range of gyro in hardware.
- int INV_EXPORT inv_icm20602_set_gyro_ois_fullscale (struct inv_icm20602 *s, int level)

  Sets fullscale range of OIS gyro in hardware.
- int INV_EXPORT inv_icm20602_set_gyro_bandwidth (struct inv_icm20602 *s, int level)

  Sets bandwidth range of gyro in hardware.
- uint8_t INV_EXPORT inv_icm20602_get_gyro_fullscale (struct inv_icm20602 *s)

  Returns fullscale range of gyroscope in hardware.
- uint8_t INV_EXPORT inv_icm20602_get_gyro_ois_fullscale (struct inv_icm20602 *s)

  Returns fullscale range of OIS gyroscope in hardware.
- uint16_t INV_EXPORT inv_icm20602_get_gyro_bandwidth (struct inv_icm20602 *s)

  Returns bandwidth range of gyroscope in hardware.
- int INV_EXPORT inv_icm20602_is_advanced_features_supported (void)

  Get the available features according the base sensor device.
- int INV_EXPORT inv_icm20602_set_fsync_bit_location (struct inv_icm20602 *s, int bit_location)

  Set FSYNC bit location.

7.19.1 Detailed Description

Low-level function to setup an Icm20602 device.

7.19.2 Function Documentation

7.19.2.1 int INV_EXPORT inv_icm20602_accel_fsr_2_reg (int32_t *fsr*)

Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.

Returns

internal register value for FSR confiugration

7.19.2.2 uint16_t INV_EXPORT inv_icm20602_get_accel_bandwidth (struct inv_icm20602 * *s*)

Returns bandwidth range of accelerometer in hardware.

Returns

the bandwidth range

7.19.2.3 uint8_t INV_EXPORT inv_icm20602_get_accel_fullscale (struct inv_icm20602 * s)

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.19.2.4 uint8_t INV_EXPORT inv_icm20602_get_accel_ois_fullscale (struct inv_icm20602 * s)

Returns fullscale range of OIS accelerometer in hardware.

Returns

the fullscale range

7.19.2.5 uint16_t INV_EXPORT inv_icm20602_get_chip_base_sample_rate (struct inv_icm20602 * s)

Get internal sample rate currently configured.

Returns

Internal sample rate currently configured in MEMS registers in Hz

7.19.2.6 int INV_EXPORT inv_icm20602_get_chip_info (struct inv_icm20602 * s, uint8_t chip_info[3])

return WHOAMI value

Parameters

out	*chip_info[0]*	WHOAMI for device
out	*chip_info[1]*	MANUFACTURER_ID for device
out	*chip_info[2]*	CHIP_ID for device

Returns

0 on success, negative value on error

7.19.2.7 uint8_t INV_EXPORT inv_icm20602_get_chip_power_state (struct inv_icm20602 * s)

Current wake status of the Mems chip.

Returns

the wake status

7.19.2.8 uint16_t INV_EXPORT inv_icm20602_get_gyro_bandwidth (struct inv_icm20602 * s)

Returns bandwidth range of gyroscope in hardware.

Returns

the bandwidth range

7.19.2.9 uint8_t INV_EXPORT inv_icm20602_get_gyro_fullscale (struct inv_icm20602 * s)

Returns fullscale range of gyroscope in hardware.

Returns

the fullscale range

7.19.2.10 uint8_t INV_EXPORT inv_icm20602_get_gyro_ois_fullscale (struct inv_icm20602 * s)

Returns fullscale range of OIS gyroscope in hardware.

Returns

the fullscale range

7.19.2.11 int INV_EXPORT inv_icm20602_get_whoami (struct inv_icm20602 * s, uint8_t * whoami)

return WHOAMI value

Parameters

out	*whoami*	WHOAMI for device

Returns

0 on success, negative value on error

7.19.2.12 int INV_EXPORT inv_icm20602_gyro_fsr_2_reg (int32_t fsr)

Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)

Returns

internal register value for FSR confiugration

7.19.2.13 int INV_EXPORT inv_icm20602_initialize (struct inv_icm20602 * s)

Initialize the device.

Returns

0 on success, negative value on error.

7.19.2.14 int INV_EXPORT inv_icm20602_is_advanced_features_supported (void)

Get the available features according the base sensor device.

Parameters

in	*in*	0: No advanced features supported 1: OIS sensor and FSYNC behavior supported (for icm20690 only)

7.19.2.15 int INV_EXPORT inv_icm20602_reg_2_accel_fsr (uint8_t *reg*)

Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.

Returns

FSR value in mg

7.19.2.16 int INV_EXPORT inv_icm20602_reg_2_gyro_fsr (uint8_t *reg*)

Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)

Returns

FSR value in dps

7.19.2.17 int INV_EXPORT inv_icm20602_set_accel_bandwidth (struct inv_icm20602 * *s*, int *level*)

Sets bandwidth range of accel in hardware.

Parameters

in	*level*	See mpu_accel_bw.

Returns

0 on success, negative value on error.

7.19.2.18 int INV_EXPORT inv_icm20602_set_accel_fullscale (struct inv_icm20602 * *s*, int *level*)

Sets fullscale range of accel in hardware.

Parameters

in	*level*	See mpu_accel_fs.

Returns

0 on success, negative value on error.

7.19.2.19 int INV_EXPORT inv_icm20602_set_accel_ois_fullscale (struct inv_icm20602 * *s*, int *level*)

Sets fullscale range of OIS accel in hardware.

Parameters

in	*level*	See mpu_accel_ois_fs.

Returns

0 on success, negative value on error.

7.19.2.20 int INV_EXPORT inv_icm20602_set_chip_power_state (struct inv_icm20602 * *s*, uint8_t *func*, uint8_t *on_off*)

Sets the power state of the Ivory chip loop.

Parameters

in	*func*	CHIP_AWAKE, CHIP_LP_ENABLE
in	*on_off*	The functions are enabled if previously disabled and disabled if previously enabled based on the value of On/Off.

Returns

0 on success, negative value on error.

7.19.2.21 int INV_EXPORT inv_icm20602_set_divider (struct inv_icm20602 * *s*, uint8_t *idiv*)

Set the mpu sample rate.

Returns

Value written to MPUREG_SMPLRT_DIV register.

7.19.2.22 int INV_EXPORT inv_icm20602_set_fsync_bit_location (struct inv_icm20602 * *s*, int *bit_location*)

Set FSYNC bit location.

Parameters

in	*in*	0: Disable FSYNC pin data to be sampled 1: Enable FSYNC pin data to be sampled at TEMP_OUT_L[0]

7.19.2.23 int INV_EXPORT inv_icm20602_set_gyro_bandwidth (struct inv_icm20602 * *s*, int *level*)

Sets bandwidth range of gyro in hardware.

Parameters

in	*level*	See mpu_gyro_bw.

Returns

0 on success, negative value on error.

7.19.2.24 int INV_EXPORT inv_icm20602_set_gyro_fullscale (struct inv_icm20602 * *s*, int *level*)

Sets fullscale range of gyro in hardware.

Parameters

in	*level*	See mpu_gyro_fs.

Returns

0 on success, negative value on error.

7.19.2.25 int INV_EXPORT inv_icm20602_set_gyro_ois_fullscale (struct inv_icm20602 * *s*, int *level*)

Sets fullscale range of OIS gyro in hardware.

Parameters

in	*level*	See mpu_gyro_fs.

Returns

0 on success, negative value on error.

7.19.2.26 int INV_EXPORT inv_icm20602_soft_reset (struct inv_icm20602 * s)

Perform a soft reset of the device.

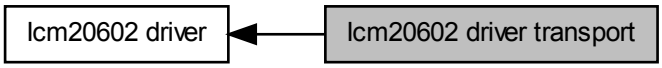
Returns

0 on success, negative value on error.

7.20 Icm20602 driver transport

Low-level Icm20602 register access.

Collaboration diagram for Icm20602 driver transport:



Functions

- int INV_EXPORT inv_icm20602_mems_write_reg (struct inv_icm20602 ∗s, uint16_t reg, uint32_t length, const uint8_t ∗data)

  Writes data from a register on mems.
- int INV_EXPORT inv_icm20602_mems_write_reg_one (struct inv_icm20602 ∗s, uint16_t reg, uint8_t data)

  Writes a single byte of data from a register on mems with no power control.
- int INV_EXPORT inv_icm20602_mems_read_reg (struct inv_icm20602 ∗s, uint16_t reg, uint32_t length, uint8_t ∗data)

  Reads data from a register on mems.

7.20.1 Detailed Description

Low-level Icm20602 register access.

7.20.2 Function Documentation

7.20.2.1 int INV_EXPORT inv_icm20602_mems_read_reg (struct inv_icm20602 ∗ *s*, uint16_t *reg*, uint32_t *length*, uint8_t ∗ *data*)

Reads data from a register on mems.

Parameters

in	*reg*	register address
in	*length*	length of data
out	*data*	output data from the register

Returns

0 in case of success, -1 for any error

7.20.2.2 int INV_EXPORT inv_icm20602_mems_write_reg (struct inv_icm20602 * *s*, uint16_t *reg*, uint32_t *length*, const uint8_t * *data*)

Writes data from a register on mems.

Parameters

in	*length*	number of byte to be written
out	*data*	output data from the register

Returns

0 in case of success, -1 for any error

7.20.2.3 int INV_EXPORT inv_icm20602_mems_write_reg_one (struct inv_icm20602 * *s*, uint16_t *reg*, uint8_t *data*)

Writes a single byte of data from a register on mems with no power control.

Parameters

in	*reg*	Register address
out	*data*	Data to be written

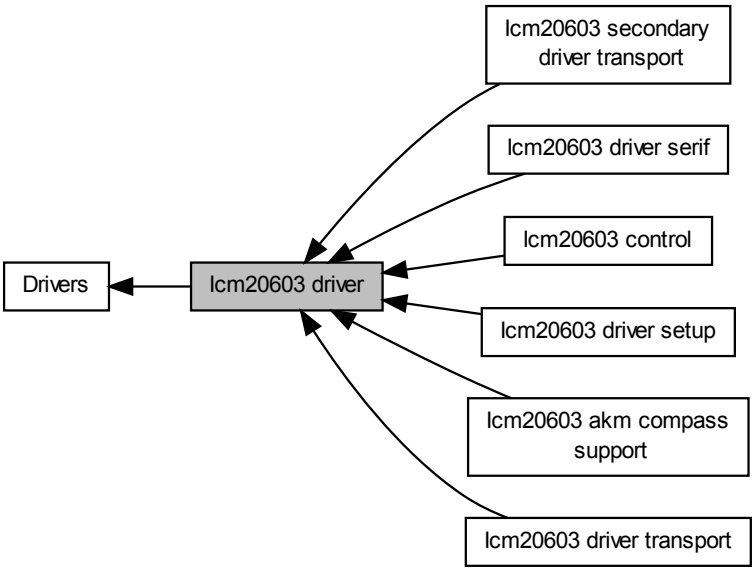
Returns

0 in case of success, -1 for any error

7.21 Icm20603 driver

Low-level driver for Icm20603 devices.

Collaboration diagram for Icm20603 driver:



Modules

- Icm20603 akm compass support

  Low-level Icm20603 aux sensor access.
- Icm20603 secondary driver transport

  Low-level Icm20603 secondary interface access.
- Icm20603 control

  Low-level function to control a Icm20603 device.
- Icm20603 driver serif

  Interface for low-level serial (I2C/SPI) access.
- Icm20603 driver setup

  Low-level function to setup an Icm20603 device.
- Icm20603 driver transport

  Low-level Icm20603 register access.

Classes

- struct inv_icm20603

  Icm20603 driver states definition.

Typedefs

- typedef struct inv_icm20603 inv_icm20603_t

  Icm20603 driver states definition.

Functions

- void inv_icm20603_sleep (int ms)

  Hook for low-level system sleep() function to be implemented by upper layer.
- void inv_icm20603_sleep_us (int us)

  Hook for low-level high res system sleep() function to be implemented by upper layer ∼100us resolution is sufficient.
- uint64_t inv_icm20603_get_time_us (void)

  Hook for low-level system time() function to be implemented by upper layer.
- uint64_t inv_icm20603_get_dataready_interrupt_time_us (void)

  Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get_time_us()
- static void inv_icm20603_reset_states (struct inv_icm20603 ∗s, const struct inv_icm20603_serif ∗serif)

  Reset and initialize driver states.
- static void inv_icm20603_reset_states_serif_ois (struct inv_icm20603 ∗s, const struct inv_icm20603_serif ∗serif_ois)

  Register secondary SERIF object for OIS access Must be called after inv_icm20603_reset_states()

7.21.1 Detailed Description

Low-level driver for Icm20603 devices.

7.21.2 Function Documentation

7.21.2.1 uint64_t inv_icm20603_get_dataready_interrupt_time_us (void)

Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get_time_us()

Returns

data ready interrupt timestamp in us

7.21.2.2 uint64_t inv_icm20603_get_time_us (void)

Hook for low-level system time() function to be implemented by upper layer.

Returns

monotonic timestamp in us

7.21.2.3 static void inv_icm20603_reset_states (struct inv_icm20603 ∗ *s*, const struct inv_icm20603_serif ∗ *serif*) [inline], [static]

Reset and initialize driver states.

Parameters

in	*s*	handle to driver states structure
in	*serif*	handle to SERIF object for underlying register access

7.21.2.4 static void inv_icm20603_reset_states_serif_ois (struct inv_icm20603 * *s*, const struct inv_icm20603_serif * *serif_ois*) [inline], [static]

Register secondary SERIF object for OIS access Must be called after inv_icm20603_reset_states()

Parameters

in	*s*	handle to driver states structure
in	*serif*	handle to SERIF object for underlying register access to OIS

7.21.2.5 void inv_icm20603_sleep (int *ms*)

Hook for low-level system sleep() function to be implemented by upper layer.

Parameters

in	*ms*	number of millisecond the calling thread should sleep

7.21.2.6 void inv_icm20603_sleep_us (int *us*)

Hook for low-level high res system sleep() function to be implemented by upper layer ∼100us resolution is sufficient.

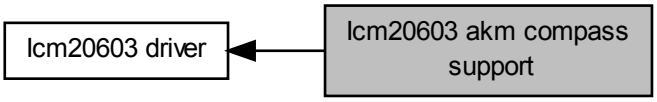
Parameters

in	*us*	number of us the calling thread should sleep

7.22 Icm20603 akm compass support

Low-level Icm20603 aux sensor access.

Collaboration diagram for Icm20603 akm compass support:



Enumerations

Functions

- void INV_EXPORT inv_icm20603_register_aux_compass (struct inv_icm20603 *s, enum inv_icm20603_↩ compass_id compass_id, uint8_t compass_i2c_addr)

  Register AUX compass.
- int INV_EXPORT inv_icm20603_is_compass_registered (struct inv_icm20603 *s)

  Return non-zero value is AUX compass was regitered, 0 if not If compass was registered but setup failed, will also return 0.
- int INV_EXPORT inv_icm20603_setup_compass_akm (struct inv_icm20603 *s)

  Initializes the compass.
- int INV_EXPORT inv_icm20603_check_akm_self_test (struct inv_icm20603 *s)

  Self test for the compass.
- int INV_EXPORT inv_icm20603_read_akm_scale (struct inv_icm20603 *s, int *scale)

  Reads the scale of the compass.
- int INV_EXPORT inv_icm20603_suspend_akm (struct inv_icm20603 *s)

  Stops the compass.
- int INV_EXPORT inv_icm20603_resume_akm (struct inv_icm20603 *s)

  Starts the compass.
- char INV_EXPORT inv_icm20603_compass_getstate (struct inv_icm20603 *s)

  Get compass power status.
- int INV_EXPORT inv_icm20603_get_compass_data (struct inv_icm20603 *s, const unsigned char *packet, unsigned char *raw_compass)

  Parse compass data packet.
- int INV_EXPORT inv_icm20603_get_compass_bytes (struct inv_icm20603 *s)

  Get data packet size according to comass type.

7.22.1 Detailed Description

Low-level Icm20603 aux sensor access.

7.22.2 Enumeration Type Documentation

7.22.2.1 enum inv_icm20603_compass_id

Supported auxiliary compass identifer.

Enumerator

INV_ICM20603_COMPASS_ID_NONE no compass

INV_ICM20603_COMPASS_ID_AK09911 AKM AK09911.

INV_ICM20603_COMPASS_ID_AK09912 AKM AK09912.

7.22.3 Function Documentation

7.22.3.1 int INV_EXPORT inv_icm20603_check_akm_self_test (struct inv_icm20603 * *s*)

Self test for the compass.

Returns

0 in case of success, -1 for any error

7.22.3.2 char INV_EXPORT inv_icm20603_compass_getstate (struct inv_icm20603 * *s*)

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

7.22.3.3 int INV_EXPORT inv_icm20603_get_compass_bytes (struct inv_icm20603 * *s*)

Get data packet size according to comass type.

Returns

size in bytes

7.22.3.4 int INV_EXPORT inv_icm20603_get_compass_data (struct inv_icm20603 * *s*, const unsigned char * *packet*, unsigned char * *raw_compass*)

Parse compass data packet.

Parameters

in	*data*	packet pointer to compass data packet
in	*raw_compass*	pointer to raw compass. big endian 2 bytes for each axis.

Returns

0 in case of success, -1 for any error

7.22.3.5 int INV_EXPORT inv_icm20603_read_akm_scale (struct inv_icm20603 * *s*, int * *scale*)

Reads the scale of the compass.

Parameters

out	*scale*	pointer to recuperate the scale

Returns

0 in case of success, -1 for any error

7.22.3.6 void INV_EXPORT inv_icm20603_register_aux_compass (struct inv_icm20603 * *s*, enum inv_icm20603_compass_id *compass_id*, uint8_t *compass_i2c_addr*)

Register AUX compass.

Will only set internal states and won't perform any transaction on the bus. Must be called before inv_icm20603_initialize().

Parameters

in	*compass_id*	Compass ID
in	*compass_i2c_addr*	Compass I2C address

Returns

0 on success, negative value on error

7.22.3.7 int INV_EXPORT inv_icm20603_resume_akm (struct inv_icm20603 * *s*)

Starts the compass.

Returns

0 in case of success, -1 for any error

7.22.3.8 int INV_EXPORT inv_icm20603_setup_compass_akm (struct inv_icm20603 * s)

Initializes the compass.

Returns

0 in case of success, -1 for any error

7.22.3.9 int INV_EXPORT inv_icm20603_suspend_akm (struct inv_icm20603 * s)

Stops the compass.

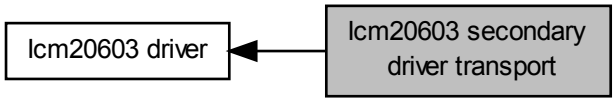
Returns

0 in case of success, -1 for any error

7.23 Icm20603 secondary driver transport

Low-level Icm20603 secondary interface access.

Collaboration diagram for Icm20603 secondary driver transport:



Macros

- #define COMPASS_I2C_SLV_READ 0

  Secondary I2C channel usage :

Functions

- void inv_icm20603_init_secondary (struct inv_icm20603 ∗s)

  Initializes the register for the i2c communication.
- int inv_icm20603_read_secondary (struct inv_icm20603 ∗s, int index, unsigned char addr, unsigned char reg, char len)

  Reads data in i2c a secondary device.
- int inv_icm20603_execute_read_secondary (struct inv_icm20603 ∗s, int index, unsigned char addr, int reg, int len, unsigned char ∗d)

  Reads data in i2c a secondary device directly.
- int inv_icm20603_write_secondary (struct inv_icm20603 ∗s, int index, unsigned char addr, unsigned char reg, char v)

  Writes data in i2c a secondary device.
- int inv_icm20603_execute_write_secondary (struct inv_icm20603 ∗s, int index, unsigned char addr, int reg, unsigned char v)

  Writes data in i2c a secondary device directly.
- int inv_icm20603_secondary_stop_channel (struct inv_icm20603 ∗s, int index)

  Stop one secondary I2C channel by writing 0 in its control register.
- int inv_icm20603_secondary_enable_i2c (struct inv_icm20603 ∗s)

  Enable secondary I2C interface.
- int inv_icm20603_secondary_disable_i2c (struct inv_icm20603 ∗s)

  Stop secondary I2C interface.

7.23.1 Detailed Description

Low-level Icm20603 secondary interface access.

7.23.2 Macro Definition Documentation

7.23.2.1 #define COMPASS_I2C_SLV_READ 0

Secondary I2C channel usage :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register

7.23.3 Function Documentation

7.23.3.1 int inv_icm20603_execute_read_secondary (struct inv_icm20603 * *s,* int *index,* unsigned char *addr,* int *reg,* int *len,* unsigned char * *d*)

Reads data in i2c a secondary device directly.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be read on the secondary device
in	*len*	Size of data to be read
out	*d*	pointer to the data to be read

Returns

0 in case of success, -1 for any error

7.23.3.2 int inv_icm20603_execute_write_secondary (struct inv_icm20603 * *s,* int *index,* unsigned char *addr,* int *reg,* unsigned char *v*)

Writes data in i2c a secondary device directly.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be write on the secondary device
in	*v*	the data to be written

Returns

0 in case of success, -1 for any error

7.23.3.3 int inv_icm20603_read_secondary (struct inv_icm20603 * s, int index, unsigned char addr, unsigned char reg, char len)

Reads data in i2c a secondary device.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	I2c address slave of the secondary slave
in	*reg*	the register to be read on the secondary device
in	*len*	Size of data to be read

Returns

0 in case of success, -1 for any error

7.23.3.4 int inv_icm20603_secondary_disable_i2c (struct inv_icm20603 * s)

Stop secondary I2C interface.

Returns

0 in case of success, -1 for any error

Warning

It stops all I2C transactions, whatever the channel status

7.23.3.5 int inv_icm20603_secondary_enable_i2c (struct inv_icm20603 * s)

Enable secondary I2C interface.

Returns

0 in case of success, -1 for any error

7.23.3.6 int inv_icm20603_secondary_stop_channel (struct inv_icm20603 * s, int index)

Stop one secondary I2C channel by writing 0 in its control register.

Parameters

in	*index*	the channel id to be stopped

Returns

0 in case of success, -1 for any error

Warning

It does not stop I2C secondary interface, just one channel

7.23.3.7 int inv_icm20603_write_secondary (struct inv_icm20603 * *s*, int *index*, unsigned char *addr*, unsigned char *reg*, char *v*)

Writes data in i2c a secondary device.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be write on the secondary device
in	*v*	the data to be written

Returns

0 in case of success, -1 for any error

7.24 Icm20603 control

Low-level function to control a Icm20603 device.

Collaboration diagram for Icm20603 control:

Classes

- struct inv_icm20603_fifo_states

  Check and retrieve for new data.

Enumerations

Functions

- int INV_EXPORT inv_icm20603_enable_mems (struct inv_icm20603 *s, int bit_mask, uint16_t smplrt_divider)

  Enables accel and/or gyro and/or pressure if integrated with gyro and accel.
- int INV_EXPORT inv_icm20603_set_sensor_period (struct inv_icm20603 *s, enum inv_icm20603_sensor sensor, uint16_t delayInMs)

  Sets the odr for a sensor.
- int INV_EXPORT inv_icm20603_is_sensor_enabled (struct inv_icm20603 *s, enum inv_icm20603_sensor sensor)

  Knows if the sensor is enabled or disbaled.
- int INV_EXPORT inv_icm20603_enable_sensor (struct inv_icm20603 *s, enum inv_icm20603_sensor sensor, uint8_t enable)

  Enables / disables a sensor.
- int INV_EXPORT inv_icm20603_configure_accel_wom (struct inv_icm20603 *s, uint8_t wom_threshold)

  Configuress accel WOM.
- int INV_EXPORT inv_icm20603_poll_sensor_data_reg (struct inv_icm20603 *s, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3])

  Check and retrieve for new data.
- int INV_EXPORT inv_icm20603_poll_fifo_data_setup (struct inv_icm20603 *s, struct inv_icm20603_fifo_states *states, uint8_t int_status)

  Initializes states for FIFO reading and parsing.
- int INV_EXPORT inv_icm20603_poll_fifo_data (struct inv_icm20603 *s, struct inv_icm20603_fifo_states *states, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3], int16_t compass_data[3])

  Read one packet from the FIFO (packet corresponding to data for all current active sensor)
- int INV_EXPORT inv_icm20603_has_data_ready (struct inv_icm20603 *s)

  Read interrupt status and check for DDRY flag.
- int INV_EXPORT inv_icm20603_get_int_status (struct inv_icm20603 *s, uint8_t *int_status)

Read interrupt status and return its value.

- int INV_EXPORT inv_icm20603_check_drdy (struct inv_icm20603 *s, uint8_t int_status)

  Check for DRDY flag in INT register value.
- int INV_EXPORT inv_icm20603_check_wom_status (struct inv_icm20603 *s, uint8_t int_status)

  Check for WOM bits in INT register value.
- int INV_EXPORT inv_icm20603_disable_fifo (struct inv_icm20603 *s)

  Disable FIFO.
- int INV_EXPORT inv_icm20603_enable_fifo (struct inv_icm20603 *s, int bit_mask)

  Enable FIFO.
- int INV_EXPORT inv_icm20603_reset_fifo (struct inv_icm20603 *s)

  Reset FIFO.
- int INV_EXPORT inv_icm20603_set_slave_compass_id (struct inv_icm20603 *s)

  Initalize AUX I2C and Compass.
- int INV_EXPORT inv_icm20603_poll_ois_gyro_data (struct inv_icm20603 *s, int16_t gyro_data[3])

  Poll OIS gyro data.
- int INV_EXPORT inv_icm20603_poll_delay_count (struct inv_icm20603 *s, int16_t *delay_count)

  Poll EIS flag and delay counter.
- int INV_EXPORT inv_icm20603_all_sensors_off (struct inv_icm20603 *s)

  test if all sensors are off
- int inv_icm20603_run_selftest (struct inv_icm20603 *s)

  Perform hardware self-test for Accel, Gyro and Compass.
- void INV_EXPORT inv_icm20603_get_st_bias (struct inv_icm20603 *s, int *st_bias)

  Retrieve bias collected by self-test.
- void inv_icm20603_set_st_bias (struct inv_icm20603 *s, int *st_bias)

  Apply bias.

7.24.1 Detailed Description

Low-level function to control a Icm20603 device.

7.24.2 Function Documentation

7.24.2.1 int INV_EXPORT inv_icm20603_all_sensors_off (struct inv_icm20603 * *s*)

test if all sensors are off

Returns

true if all sensors are off, false otherwise

7.24.2.2 int INV_EXPORT inv_icm20603_check_drdy (struct inv_icm20603 * *s*, uint8_t *int_status*)

Check for DRDY flag in INT register value.

Parameters

in	*int_status*	INT status register value

Returns

1 if DDRDY is set, 0 otherwise

7.24.2.3 int INV_EXPORT inv_icm20603_check_wom_status (struct inv_icm20603 * s, uint8_t int_status)

Check for WOM bits in INT register value.

Parameters

in	*int_status*	INT status register value

Returns

bit mask corresponding to x, y, z axis that caused WOM (0x01: x axis, 0x02: y axis, 0x01: z axis)

7.24.2.4 int INV_EXPORT inv_icm20603_configure_accel_wom (struct inv_icm20603 * s, uint8_t wom_threshold)

Configuress accel WOM.

Parameters

in	*wom_threshold*	threshold value for X,Y,Z axis that should trigger a WOM interrupt (0 to diable WOM)

Returns

0 on success, negative value on error.

7.24.2.5 int INV_EXPORT inv_icm20603_disable_fifo (struct inv_icm20603 * s)

Disable FIFO.

Returns

0 on success, negative value on error.

7.24.2.6 int INV_EXPORT inv_icm20603_enable_fifo (struct inv_icm20603 * s, int bit_mask)

Enable FIFO.

Parameters

in	*bit_mask*	A mask of sensor to push to FIFO.

Returns

0 on success, negative value on error.

7.24.2.7 int INV_EXPORT inv_icm20603_enable_mems (struct inv_icm20603 * s, int bit_mask, uint16_t smplrt_divider)

Enables accel and/or gyro and/or pressure if integrated with gyro and accel.

Parameters

in	*bit_mask*	A mask where 2 means turn on accel, 1 means turn on gyro, 4 is for pressure. By default, this only turns on a sensor if all sensors are off otherwise the DMP controls this register including turning off a sensor. To override this behavior add in a mask of 128.
in	*smplrt_divider*	The divider which was applied to internal sample rate based on field sample_rate from base_driver_t to get minimum ODR for accel and gyro

Returns

0 on success, negative value on error

7.24.2.8 int INV_EXPORT inv_icm20603_enable_sensor (struct inv_icm20603 * s, enum inv_icm20603_sensor sensor, uint8_t enable)

Enables / disables a sensor.

Parameters

in	*androidSensor*	Sensor Identity
in	*enable*	0=off, 1=on

Returns

0 in case of success, negative value on error

7.24.2.9 int INV_EXPORT inv_icm20603_get_int_status (struct inv_icm20603 * s, uint8_t * int_status)

Read interrupt status and return its value.

Parameters

out	*int_status*	INT status register value

Returns

0 on success, negative value on error

7.24.2.10 void INV_EXPORT inv_icm20603_get_st_bias (struct inv_icm20603 * s, int * st_bias)

Retrieve bias collected by self-test.

Parameters

out	*st_bias*	bias scaled by 2^16, accel is gee and gyro is dps. The buffer will be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z

7.24.2.11 int INV_EXPORT inv_icm20603_has_data_ready (struct inv_icm20603 * s)

Read interrupt status and check for DDRY flag.

Returns

1 if DDRY inteript is set, 0 otherwise and negative value on error

7.24.2.12 int INV_EXPORT inv_icm20603_is_sensor_enabled (struct inv_icm20603 * s, enum inv_icm20603_sensor sensor)

Knows if the sensor is enabled or disbaled.

Parameters

in	*sensor*	the sensor is enabled or not

Returns

1 if active, 0 otherwise

7.24.2.13 int INV_EXPORT inv_icm20603_poll_delay_count (struct inv_icm20603 * s, int16_t * delay_count)

Poll EIS flag and delay counter.

Parameters

out	*delay_count*	delay time in us between the FSYNC event (before the gyro data event) and the gyro data event

Returns

0x8 if FSYNC event reported, 0 otherwise, negative value on error.

7.24.2.14 int INV_EXPORT inv_icm20603_poll_fifo_data (struct inv_icm20603 * s, struct inv_icm20603_fifo_states * states, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3], int16_t compass_data[3])

Read one packet from the FIFO (packet corresponding to data for all current active sensor)

Parameters

in	*states*	placeholder to FIFO states
out	*acc_data*	raw gyro data
out	*temp_data*	raw temp data
out	*gyro_data*	raw gyro data
out	*compass_data*	raw compass data

Returns

0 : FIFO empty bit0 : gyro data output set bit1 : acc data output set bit2 : compass output set $<$ 0 : error

7.24.2.15 int INV_EXPORT inv_icm20603_poll_fifo_data_setup (struct inv_icm20603 * s, struct inv_icm20603_fifo_states * states, uint8_t int_status)

Initializes states for FIFO reading and parsing.

Parameters

in	*states*	placeholder to FIFO states
in	*int_status*	int status register value (used to check for overflow)

Returns

0 for success, 1 for overlfow detected, negative value for other errors

7.24.2.16 int INV_EXPORT inv_icm20603_poll_ois_gyro_data (struct inv_icm20603 * s, int16_t gyro_data[3])

Poll OIS gyro data.

Returns

0x10 if OIS data reported, 0 otherwise, negative value on error.

7.24.2.17 int INV_EXPORT inv_icm20603_poll_sensor_data_reg (struct inv_icm20603 * s, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3])

Check and retrieve for new data.

Parameters

out	*acc_data*	raw gyro data
out	*temp_data*	raw temp data
out	*gyro_data*	raw gyro data

Returns

0 : not data ready 0x01 : gyro data output set 0x02 : acc data output set 0x03 : gyro and acc data output set $<$ 0 : error

7.24.2.18 int INV_EXPORT inv_icm20603_reset_fifo (struct inv_icm20603 * *s*)

Reset FIFO.

Returns

0 on success, negative value on error.

7.24.2.19 int inv_icm20603_run_selftest (struct inv_icm20603 * *s*)

Perform hardware self-test for Accel, Gyro and Compass.

Parameters

in	*None*	

Returns

COMPASS_SUCCESS$<<$2 | ACCEL_SUCCESS$<<$1 | GYRO_SUCCESS so 3

7.24.2.20 int INV_EXPORT inv_icm20603_set_sensor_period (struct inv_icm20603 * *s*, enum inv_icm20603_sensor *sensor*, uint16_t *delayInMs*)

Sets the odr for a sensor.

Parameters

in	*sensor*	Sensor Identity
in	*delayInMs*	the delay between two values in ms

Returns

0 in case of success, -1 for any error

7.24.2.21 int INV_EXPORT inv_icm20603_set_slave_compass_id (struct inv_icm20603 * s)

Initalize AUX I2C and Compass.

Returns

0 on success, negative value on error.

7.24.2.22 void inv_icm20603_set_st_bias (struct inv_icm20603 * s, int * st_bias)

Apply bias.

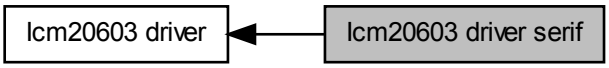
Parameters

in	*st_bias*	bias scaled by 2^16, accel is gee and gyro is dps. The buffer should be be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z

7.25 Icm20603 driver serif

Interface for low-level serial (I2C/SPI) access.

Collaboration diagram for Icm20603 driver serif:



Classes

- struct inv_icm20603_serif

  basesensor serial interface

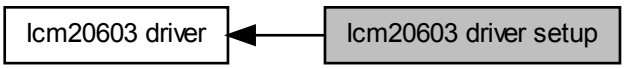
7.25.1 Detailed Description

Interface for low-level serial (I2C/SPI) access.

7.26 Icm20603 driver setup

Low-level function to setup an Icm20603 device.

Collaboration diagram for Icm20603 driver setup:



Functions

- int INV_EXPORT inv_icm20603_get_whoami (struct inv_icm20603 ∗s, uint8_t ∗whoami)

  return WHOAMI value
- int INV_EXPORT inv_icm20603_get_chip_info (struct inv_icm20603 ∗s, uint8_t chip_info[3])

  return WHOAMI value
- int INV_EXPORT inv_icm20603_initialize (struct inv_icm20603 ∗s)

  Initialize the device.
- int INV_EXPORT inv_icm20603_soft_reset (struct inv_icm20603 ∗s)

  Perform a soft reset of the device.
- int INV_EXPORT inv_icm20603_set_divider (struct inv_icm20603 ∗s, uint8_t idiv)

  Set the mpu sample rate.
- int INV_EXPORT inv_icm20603_set_chip_power_state (struct inv_icm20603 ∗s, uint8_t func, uint8_t on_off)

  Sets the power state of the Ivory chip loop.
- uint8_t INV_EXPORT inv_icm20603_get_chip_power_state (struct inv_icm20603 ∗s)

  Current wake status of the Mems chip.
- uint16_t INV_EXPORT inv_icm20603_get_chip_base_sample_rate (struct inv_icm20603 ∗s)

  Get internal sample rate currently configured.
- int INV_EXPORT inv_icm20603_accel_fsr_2_reg (int32_t fsr)

  Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.
- int INV_EXPORT inv_icm20603_reg_2_accel_fsr (uint8_t reg)

  Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.
- int INV_EXPORT inv_icm20603_set_accel_fullscale (struct inv_icm20603 ∗s, int level)

  Sets fullscale range of accel in hardware.
- int INV_EXPORT inv_icm20603_set_accel_ois_fullscale (struct inv_icm20603 ∗s, int level)

  Sets fullscale range of OIS accel in hardware.
- int INV_EXPORT inv_icm20603_set_accel_bandwidth (struct inv_icm20603 ∗s, int level)

  Sets bandwidth range of accel in hardware.
- uint8_t INV_EXPORT inv_icm20603_get_accel_fullscale (struct inv_icm20603 ∗s)

  Returns fullscale range of accelerometer in hardware.
- uint8_t INV_EXPORT inv_icm20603_get_accel_ois_fullscale (struct inv_icm20603 ∗s)

  Returns fullscale range of OIS accelerometer in hardware.
- uint16_t INV_EXPORT inv_icm20603_get_accel_bandwidth (struct inv_icm20603 ∗s)

Returns bandwidth range of accelerometer in hardware.

- int INV_EXPORT inv_icm20603_gyro_fsr_2_reg (int32_t fsr)

  Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)
- int INV_EXPORT inv_icm20603_reg_2_gyro_fsr (uint8_t reg)

  Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)
- int INV_EXPORT inv_icm20603_set_gyro_fullscale (struct inv_icm20603 *s, int level)

  Sets fullscale range of gyro in hardware.
- int INV_EXPORT inv_icm20603_set_gyro_ois_fullscale (struct inv_icm20603 *s, int level)

  Sets fullscale range of OIS gyro in hardware.
- int INV_EXPORT inv_icm20603_set_gyro_bandwidth (struct inv_icm20603 *s, int level)

  Sets bandwidth range of gyro in hardware.
- uint8_t INV_EXPORT inv_icm20603_get_gyro_fullscale (struct inv_icm20603 *s)

  Returns fullscale range of gyroscope in hardware.
- uint8_t INV_EXPORT inv_icm20603_get_gyro_ois_fullscale (struct inv_icm20603 *s)

  Returns fullscale range of OIS gyroscope in hardware.
- uint16_t INV_EXPORT inv_icm20603_get_gyro_bandwidth (struct inv_icm20603 *s)

  Returns bandwidth range of gyroscope in hardware.
- int INV_EXPORT inv_icm20603_is_advanced_features_supported (void)

  Get the available features according the base sensor device.
- int INV_EXPORT inv_icm20603_set_fsync_bit_location (struct inv_icm20603 *s, int bit_location)

  Set FSYNC bit location.

7.26.1 Detailed Description

Low-level function to setup an Icm20603 device.

7.26.2 Function Documentation

7.26.2.1 int INV_EXPORT inv_icm20603_accel_fsr_2_reg (int32_t *fsr*)

Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.

Returns

internal register value for FSR confiugration

7.26.2.2 uint16_t INV_EXPORT inv_icm20603_get_accel_bandwidth (struct inv_icm20603 * *s*)

Returns bandwidth range of accelerometer in hardware.

Returns

the bandwidth range

7.26.2.3 uint8_t INV_EXPORT inv_icm20603_get_accel_fullscale (struct inv_icm20603 * s)

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.26.2.4 uint8_t INV_EXPORT inv_icm20603_get_accel_ois_fullscale (struct inv_icm20603 * s)

Returns fullscale range of OIS accelerometer in hardware.

Returns

the fullscale range

7.26.2.5 uint16_t INV_EXPORT inv_icm20603_get_chip_base_sample_rate (struct inv_icm20603 * s)

Get internal sample rate currently configured.

Returns

Internal sample rate currently configured in MEMS registers in Hz

7.26.2.6 int INV_EXPORT inv_icm20603_get_chip_info (struct inv_icm20603 * s, uint8_t chip_info[3])

return WHOAMI value

Parameters

out	*chip_info[0]*	WHOAMI for device
out	*chip_info[1]*	MANUFACTURER_ID for device
out	*chip_info[2]*	CHIP_ID for device

Returns

0 on success, negative value on error

7.26.2.7 uint8_t INV_EXPORT inv_icm20603_get_chip_power_state (struct inv_icm20603 * s)

Current wake status of the Mems chip.

Returns

the wake status

7.26.2.8 uint16_t INV_EXPORT inv_icm20603_get_gyro_bandwidth (struct inv_icm20603 * s)

Returns bandwidth range of gyroscope in hardware.

Returns

the bandwidth range

7.26.2.9 uint8_t INV_EXPORT inv_icm20603_get_gyro_fullscale (struct inv_icm20603 * s)

Returns fullscale range of gyroscope in hardware.

Returns

the fullscale range

7.26.2.10 uint8_t INV_EXPORT inv_icm20603_get_gyro_ois_fullscale (struct inv_icm20603 * s)

Returns fullscale range of OIS gyroscope in hardware.

Returns

the fullscale range

7.26.2.11 int INV_EXPORT inv_icm20603_get_whoami (struct inv_icm20603 * s, uint8_t * whoami)

return WHOAMI value

Parameters

out	*whoami*	WHOAMI for device

Returns

0 on success, negative value on error

7.26.2.12 int INV_EXPORT inv_icm20603_gyro_fsr_2_reg (int32_t fsr)

Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)

Returns

internal register value for FSR confiugration

7.26.2.13 int INV_EXPORT inv_icm20603_initialize (struct inv_icm20603 * s)

Initialize the device.

Returns

0 on success, negative value on error.

7.26.2.14 int INV_EXPORT inv_icm20603_is_advanced_features_supported (void)

Get the available features according the base sensor device.

Parameters

in	*in*	0: No advanced features supported 1: OIS sensor and FSYNC behavior supported (for icm20690 only)

7.26.2.15 int INV_EXPORT inv_icm20603_reg_2_accel_fsr (uint8_t *reg*)

Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.

Returns

FSR value in mg

7.26.2.16 int INV_EXPORT inv_icm20603_reg_2_gyro_fsr (uint8_t *reg*)

Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)

Returns

FSR value in dps

7.26.2.17 int INV_EXPORT inv_icm20603_set_accel_bandwidth (struct inv_icm20603 * *s*, int *level*)

Sets bandwidth range of accel in hardware.

Parameters

in	*level*	See mpu_accel_bw.

Returns

0 on success, negative value on error.

7.26.2.18 int INV_EXPORT inv_icm20603_set_accel_fullscale (struct inv_icm20603 * *s*, int *level*)

Sets fullscale range of accel in hardware.

Parameters

in	*level*	See mpu_accel_fs.

Returns

0 on success, negative value on error.

7.26.2.19 int INV_EXPORT inv_icm20603_set_accel_ois_fullscale (struct inv_icm20603 * *s*, int *level*)

Sets fullscale range of OIS accel in hardware.

Parameters

in	*level*	See mpu_accel_ois_fs.

Returns

0 on success, negative value on error.

7.26.2.20 int INV_EXPORT inv_icm20603_set_chip_power_state (struct inv_icm20603 * *s*, uint8_t *func*, uint8_t *on_off*)

Sets the power state of the Ivory chip loop.

Parameters

in	*func*	CHIP_AWAKE, CHIP_LP_ENABLE
in	*on_off*	The functions are enabled if previously disabled and disabled if previously enabled based on the value of On/Off.

Returns

0 on success, negative value on error.

7.26.2.21 int INV_EXPORT inv_icm20603_set_divider (struct inv_icm20603 * *s*, uint8_t *idiv*)

Set the mpu sample rate.

Returns

Value written to MPUREG_SMPLRT_DIV register.

7.26.2.22 int INV_EXPORT inv_icm20603_set_fsync_bit_location (struct inv_icm20603 * s, int bit_location)

Set FSYNC bit location.

Parameters

in	*in*	0: Disable FSYNC pin data to be sampled 1: Enable FSYNC pin data to be sampled at TEMP_OUT_L[0]

7.26.2.23 int INV_EXPORT inv_icm20603_set_gyro_bandwidth (struct inv_icm20603 * s, int level)

Sets bandwidth range of gyro in hardware.

Parameters

in	*level*	See mpu_gyro_bw.

Returns

0 on success, negative value on error.

7.26.2.24 int INV_EXPORT inv_icm20603_set_gyro_fullscale (struct inv_icm20603 * s, int level)

Sets fullscale range of gyro in hardware.

Parameters

in	*level*	See mpu_gyro_fs.

Returns

0 on success, negative value on error.

7.26.2.25 int INV_EXPORT inv_icm20603_set_gyro_ois_fullscale (struct inv_icm20603 * s, int level)

Sets fullscale range of OIS gyro in hardware.

Parameters

in	*level*	See mpu_gyro_fs.

Returns

0 on success, negative value on error.

7.26.2.26 int INV_EXPORT inv_icm20603_soft_reset (struct inv_icm20603 * s)

Perform a soft reset of the device.

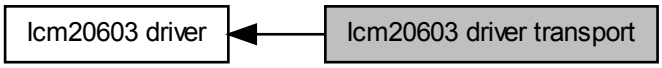
Returns

0 on success, negative value on error.

7.27 Icm20603 driver transport

Low-level Icm20603 register access.

Collaboration diagram for Icm20603 driver transport:



Functions

- int INV_EXPORT inv_icm20603_mems_write_reg (struct inv_icm20603 ∗s, uint16_t reg, uint32_t length, const uint8_t ∗data)

  Writes data from a register on mems.
- int INV_EXPORT inv_icm20603_mems_write_reg_one (struct inv_icm20603 ∗s, uint16_t reg, uint8_t data)

  Writes a single byte of data from a register on mems with no power control.
- int INV_EXPORT inv_icm20603_mems_read_reg (struct inv_icm20603 ∗s, uint16_t reg, uint32_t length, uint8_t ∗data)

  Reads data from a register on mems.

7.27.1 Detailed Description

Low-level Icm20603 register access.

7.27.2 Function Documentation

7.27.2.1 int INV_EXPORT inv_icm20603_mems_read_reg (struct inv_icm20603 ∗ *s*, uint16_t *reg*, uint32_t *length*, uint8_t ∗ *data*)

Reads data from a register on mems.

Parameters

in	*reg*	register address
in	*length*	length of data
out	*data*	output data from the register

Returns

0 in case of success, -1 for any error

7.27.2.2 int INV_EXPORT inv_icm20603_mems_write_reg (struct inv_icm20603 * *s*, uint16_t *reg*, uint32_t *length*, const uint8_t * *data*)

Writes data from a register on mems.

Parameters

in	*length*	number of byte to be written
out	*data*	output data from the register

Returns

0 in case of success, -1 for any error

7.27.2.3 int INV_EXPORT inv_icm20603_mems_write_reg_one (struct inv_icm20603 * *s*, uint16_t *reg*, uint8_t *data*)

Writes a single byte of data from a register on mems with no power control.

Parameters

in	*reg*	Register address
out	*data*	Data to be written

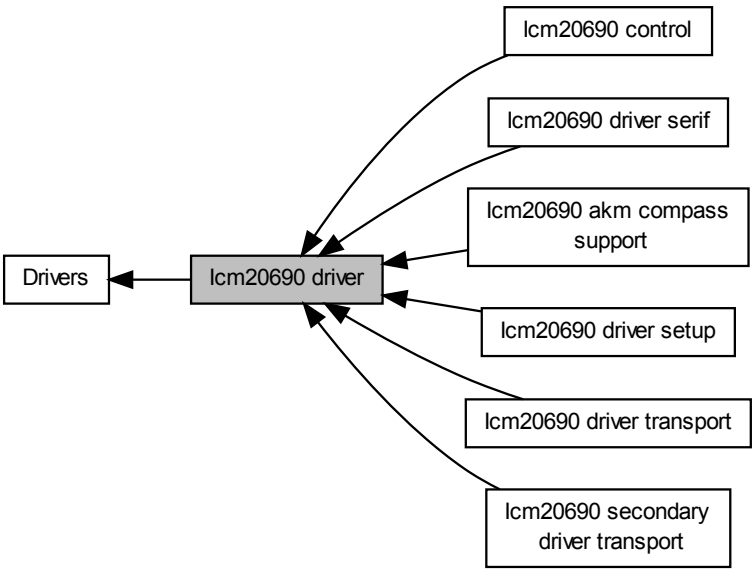
Returns

0 in case of success, -1 for any error

7.28 Icm20690 driver

Low-level driver for Icm20690 devices.

Collaboration diagram for Icm20690 driver:



Modules

- Icm20690 akm compass support

  Low-level Icm20690 aux sensor access.
- Icm20690 secondary driver transport

  Low-level Icm20690 secondary interface access.
- Icm20690 control

  Low-level function to control a Icm20690 device.
- Icm20690 driver serif

  Interface for low-level serial (I2C/SPI) access.
- Icm20690 driver setup

  Low-level function to setup an Icm20690 device.
- Icm20690 driver transport

  Low-level Icm20690 register access.

Classes

- struct inv_icm20690

  Icm20690 driver states definition.

Typedefs

- typedef struct inv_icm20690 inv_icm20690_t

  Icm20690 driver states definition.

Functions

- void inv_icm20690_sleep (int ms)

  Hook for low-level system sleep() function to be implemented by upper layer.
- void inv_icm20690_sleep_us (int us)

  Hook for low-level high res system sleep() function to be implemented by upper layer ∼100us resolution is sufficient.
- uint64_t inv_icm20690_get_time_us (void)

  Hook for low-level system time() function to be implemented by upper layer.
- uint64_t inv_icm20690_get_dataready_interrupt_time_us (void)

  Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get_time_us()
- static void inv_icm20690_reset_states (struct inv_icm20690 ∗s, const struct inv_icm20690_serif ∗serif)

  Reset and initialize driver states.
- static void inv_icm20690_reset_states_serif_ois (struct inv_icm20690 ∗s, const struct inv_icm20690_serif ∗serif_ois)

  Register secondary SERIF object for OIS access Must be called after inv_icm20690_reset_states()

7.28.1 Detailed Description

Low-level driver for Icm20690 devices.

7.28.2 Function Documentation

7.28.2.1 uint64_t inv_icm20690_get_dataready_interrupt_time_us (void)

Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get_time_us()

Returns

data ready interrupt timestamp in us

7.28.2.2 uint64_t inv_icm20690_get_time_us (void)

Hook for low-level system time() function to be implemented by upper layer.

Returns

monotonic timestamp in us

7.28.2.3 static void inv_icm20690_reset_states (struct inv_icm20690 ∗ *s*, const struct inv_icm20690_serif ∗ *serif*) [inline], [static]

Reset and initialize driver states.

Parameters

in	*s*	handle to driver states structure
in	*serif*	handle to SERIF object for underlying register access

7.28.2.4 static void inv_icm20690_reset_states_serif_ois (struct inv_icm20690 * *s*, const struct inv_icm20690_serif * *serif_ois*) [inline], [static]

Register secondary SERIF object for OIS access Must be called after inv_icm20690_reset_states()

Parameters

in	*s*	handle to driver states structure
in	*serif*	handle to SERIF object for underlying register access to OIS

7.28.2.5 void inv_icm20690_sleep (int *ms*)

Hook for low-level system sleep() function to be implemented by upper layer.

Parameters

in	*ms*	number of millisecond the calling thread should sleep

7.28.2.6 void inv_icm20690_sleep_us (int *us*)

Hook for low-level high res system sleep() function to be implemented by upper layer ∼100us resolution is sufficient.

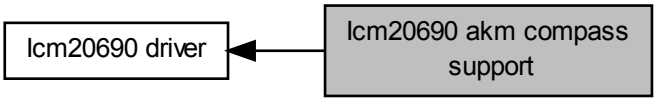
Parameters

in	*us*	number of us the calling thread should sleep

7.29 Icm20690 akm compass support

Low-level Icm20690 aux sensor access.

Collaboration diagram for Icm20690 akm compass support:



Enumerations

Functions

- void INV_EXPORT inv_icm20690_register_aux_compass (struct inv_icm20690 *s, enum inv_icm20690_↩ compass_id compass_id, uint8_t compass_i2c_addr)

  Register AUX compass.
- int INV_EXPORT inv_icm20690_is_compass_registered (struct inv_icm20690 *s)

  Return non-zero value is AUX compass was regitered, 0 if not If compass was registered but setup failed, will also return 0.
- int INV_EXPORT inv_icm20690_setup_compass_akm (struct inv_icm20690 *s)

  Initializes the compass.
- int INV_EXPORT inv_icm20690_check_akm_self_test (struct inv_icm20690 *s)

  Self test for the compass.
- int INV_EXPORT inv_icm20690_read_akm_scale (struct inv_icm20690 *s, int *scale)

  Reads the scale of the compass.
- int INV_EXPORT inv_icm20690_suspend_akm (struct inv_icm20690 *s)

  Stops the compass.
- int INV_EXPORT inv_icm20690_resume_akm (struct inv_icm20690 *s)

  Starts the compass.
- char INV_EXPORT inv_icm20690_compass_getstate (struct inv_icm20690 *s)

  Get compass power status.
- int INV_EXPORT inv_icm20690_get_compass_data (struct inv_icm20690 *s, const unsigned char *packet, unsigned char *raw_compass)

  Parse compass data packet.
- int INV_EXPORT inv_icm20690_get_compass_bytes (struct inv_icm20690 *s)

  Get data packet size according to comass type.

7.29.1 Detailed Description

Low-level Icm20690 aux sensor access.

7.29.2 Enumeration Type Documentation

7.29.2.1 enum inv_icm20690_compass_id

Supported auxiliary compass identifer.

Enumerator

INV_ICM20690_COMPASS_ID_NONE no compass

INV_ICM20690_COMPASS_ID_AK09911 AKM AK09911.

INV_ICM20690_COMPASS_ID_AK09912 AKM AK09912.

7.29.3 Function Documentation

7.29.3.1 int INV_EXPORT inv_icm20690_check_akm_self_test (struct inv_icm20690 * *s*)

Self test for the compass.

Returns

0 in case of success, -1 for any error

7.29.3.2 char INV_EXPORT inv_icm20690_compass_getstate (struct inv_icm20690 * *s*)

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

7.29.3.3 int INV_EXPORT inv_icm20690_get_compass_bytes (struct inv_icm20690 * *s*)

Get data packet size according to comass type.

Returns

size in bytes

7.29.3.4 int INV_EXPORT inv_icm20690_get_compass_data (struct inv_icm20690 * *s*, const unsigned char * *packet*, unsigned char * *raw_compass*)

Parse compass data packet.

Parameters

in	*data*	packet pointer to compass data packet
in	*raw_compass*	pointer to raw compass. big endian 2 bytes for each axis.

Returns

0 in case of success, -1 for any error

7.29.3.5 int INV_EXPORT inv_icm20690_read_akm_scale (struct inv_icm20690 * *s*, int * *scale*)

Reads the scale of the compass.

Parameters

out	*scale*	pointer to recuperate the scale

Returns

0 in case of success, -1 for any error

7.29.3.6 void INV_EXPORT inv_icm20690_register_aux_compass (struct inv_icm20690 * *s*, enum inv_icm20690_compass_id *compass_id*, uint8_t *compass_i2c_addr*)

Register AUX compass.

Will only set internal states and won't perform any transaction on the bus. Must be called before inv_icm20690_↩ initialize().

Parameters

in	*compass_id*	Compass ID
in	*compass_i2c_addr*	Compass I2C address

Returns

0 on success, negative value on error

7.29.3.7 int INV_EXPORT inv_icm20690_resume_akm (struct inv_icm20690 * *s*)

Starts the compass.

Returns

0 in case of success, -1 for any error

7.29.3.8 int INV_EXPORT inv_icm20690_setup_compass_akm (struct inv_icm20690 * s)

Initializes the compass.

Returns

0 in case of success, -1 for any error

7.29.3.9 int INV_EXPORT inv_icm20690_suspend_akm (struct inv_icm20690 * s)

Stops the compass.

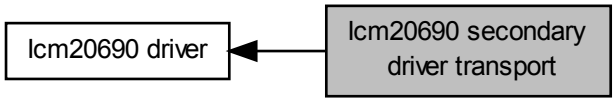
Returns

0 in case of success, -1 for any error

7.30 Icm20690 secondary driver transport

Low-level Icm20690 secondary interface access.

Collaboration diagram for Icm20690 secondary driver transport:



Macros

- #define COMPASS_I2C_SLV_READ 0

  Secondary I2C channel usage :

Functions

- void inv_icm20690_init_secondary (struct inv_icm20690 ∗s)

  Initializes the register for the i2c communication.
- int inv_icm20690_read_secondary (struct inv_icm20690 ∗s, int index, unsigned char addr, unsigned char reg, char len)

  Reads data in i2c a secondary device.
- int inv_icm20690_execute_read_secondary (struct inv_icm20690 ∗s, int index, unsigned char addr, int reg, int len, unsigned char ∗d)

  Reads data in i2c a secondary device directly.
- int inv_icm20690_write_secondary (struct inv_icm20690 ∗s, int index, unsigned char addr, unsigned char reg, char v)

  Writes data in i2c a secondary device.
- int inv_icm20690_execute_write_secondary (struct inv_icm20690 ∗s, int index, unsigned char addr, int reg, unsigned char v)

  Writes data in i2c a secondary device directly.
- int inv_icm20690_secondary_stop_channel (struct inv_icm20690 ∗s, int index)

  Stop one secondary I2C channel by writing 0 in its control register.
- int inv_icm20690_secondary_enable_i2c (struct inv_icm20690 ∗s)

  Enable secondary I2C interface.
- int inv_icm20690_secondary_disable_i2c (struct inv_icm20690 ∗s)

  Stop secondary I2C interface.

7.30.1 Detailed Description

Low-level Icm20690 secondary interface access.

7.30.2 Macro Definition Documentation

7.30.2.1 #define COMPASS_I2C_SLV_READ 0

Secondary I2C channel usage :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register

7.30.3 Function Documentation

7.30.3.1 int inv_icm20690_execute_read_secondary (struct inv_icm20690 * *s,* int *index,* unsigned char *addr,* int *reg,* int *len,* unsigned char * *d*)

Reads data in i2c a secondary device directly.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be read on the secondary device
in	*len*	Size of data to be read
out	*d*	pointer to the data to be read

Returns

0 in case of success, -1 for any error

7.30.3.2 int inv_icm20690_execute_write_secondary (struct inv_icm20690 * *s,* int *index,* unsigned char *addr,* int *reg,* unsigned char *v*)

Writes data in i2c a secondary device directly.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be write on the secondary device
in	*v*	the data to be written

Returns

0 in case of success, -1 for any error

7.30.3.3 int inv_icm20690_read_secondary (struct inv_icm20690 * s, int index, unsigned char addr, unsigned char reg, char len)

Reads data in i2c a secondary device.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	I2c address slave of the secondary slave
in	*reg*	the register to be read on the secondary device
in	*len*	Size of data to be read

Returns

0 in case of success, -1 for any error

7.30.3.4 int inv_icm20690_secondary_disable_i2c (struct inv_icm20690 * s)

Stop secondary I2C interface.

Returns

0 in case of success, -1 for any error

Warning

It stops all I2C transactions, whatever the channel status

7.30.3.5 int inv_icm20690_secondary_enable_i2c (struct inv_icm20690 * s)

Enable secondary I2C interface.

Returns

0 in case of success, -1 for any error

7.30.3.6 int inv_icm20690_secondary_stop_channel (struct inv_icm20690 * s, int index)

Stop one secondary I2C channel by writing 0 in its control register.

Parameters

in	*index*	the channel id to be stopped

Returns

0 in case of success, -1 for any error

Warning

It does not stop I2C secondary interface, just one channel

7.30.3.7 int inv_icm20690_write_secondary (struct inv_icm20690 * *s*, int *index*, unsigned char *addr*, unsigned char *reg*, char *v*)

Writes data in i2c a secondary device.

Parameters

in	*index*	The i2c slave what you would use
in	*addr*	i2c address slave of the secondary slave
in	*reg*	the register to be write on the secondary device
in	*v*	the data to be written

Returns

0 in case of success, -1 for any error

7.31 Icm20690 control

Low-level function to control a Icm20690 device.

Collaboration diagram for Icm20690 control:

Classes

- struct inv_icm20690_fifo_states

  Check and retrieve for new data.

Enumerations

Functions

- int INV_EXPORT inv_icm20690_enable_mems (struct inv_icm20690 *s, int bit_mask, uint16_t smplrt_divider)

  Enables accel and/or gyro and/or pressure if integrated with gyro and accel.
- int INV_EXPORT inv_icm20690_set_sensor_period (struct inv_icm20690 *s, enum inv_icm20690_sensor sensor, uint16_t delayInMs)

  Sets the odr for a sensor.
- int INV_EXPORT inv_icm20690_is_sensor_enabled (struct inv_icm20690 *s, enum inv_icm20690_sensor sensor)

  Knows if the sensor is enabled or disbaled.
- int INV_EXPORT inv_icm20690_enable_sensor (struct inv_icm20690 *s, enum inv_icm20690_sensor sensor, uint8_t enable)

  Enables / disables a sensor.
- int INV_EXPORT inv_icm20690_configure_accel_wom (struct inv_icm20690 *s, uint8_t wom_threshold)

  Configuress accel WOM.
- int INV_EXPORT inv_icm20690_poll_sensor_data_reg (struct inv_icm20690 *s, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3])

  Check and retrieve for new data.
- int INV_EXPORT inv_icm20690_poll_fifo_data_setup (struct inv_icm20690 *s, struct inv_icm20690_fifo_states *states, uint8_t int_status)

  Initializes states for FIFO reading and parsing.
- int INV_EXPORT inv_icm20690_poll_fifo_data (struct inv_icm20690 *s, struct inv_icm20690_fifo_states *states, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3], int16_t compass_data[3])

  Read one packet from the FIFO (packet corresponding to data for all current active sensor)
- int INV_EXPORT inv_icm20690_has_data_ready (struct inv_icm20690 *s)

  Read interrupt status and check for DDRY flag.
- int INV_EXPORT inv_icm20690_get_int_status (struct inv_icm20690 *s, uint8_t *int_status)

Read interrupt status and return its value.

- int INV_EXPORT inv_icm20690_check_drdy (struct inv_icm20690 *s, uint8_t int_status)

  Check for DRDY flag in INT register value.
- int INV_EXPORT inv_icm20690_check_wom_status (struct inv_icm20690 *s, uint8_t int_status)

  Check for WOM bits in INT register value.
- int INV_EXPORT inv_icm20690_disable_fifo (struct inv_icm20690 *s)

  Disable FIFO.
- int INV_EXPORT inv_icm20690_enable_fifo (struct inv_icm20690 *s, int bit_mask)

  Enable FIFO.
- int INV_EXPORT inv_icm20690_reset_fifo (struct inv_icm20690 *s)

  Reset FIFO.
- int INV_EXPORT inv_icm20690_set_slave_compass_id (struct inv_icm20690 *s)

  Initalize AUX I2C and Compass.
- int INV_EXPORT inv_icm20690_poll_ois_gyro_data (struct inv_icm20690 *s, int16_t gyro_data[3])

  Poll OIS gyro data.
- int INV_EXPORT inv_icm20690_poll_delay_count (struct inv_icm20690 *s, int16_t *delay_count)

  Poll EIS flag and delay counter.
- int INV_EXPORT inv_icm20690_all_sensors_off (struct inv_icm20690 *s)

  test if all sensors are off
- int inv_icm20690_run_selftest (struct inv_icm20690 *s)

  Perform hardware self-test for Accel, Gyro and Compass.
- void INV_EXPORT inv_icm20690_get_st_bias (struct inv_icm20690 *s, int *st_bias)

  Retrieve bias collected by self-test.
- void inv_icm20690_set_st_bias (struct inv_icm20690 *s, int *st_bias)

  Apply bias.

7.31.1 Detailed Description

Low-level function to control a Icm20690 device.

7.31.2 Function Documentation

7.31.2.1 int INV_EXPORT inv_icm20690_all_sensors_off (struct inv_icm20690 * *s*)

test if all sensors are off

Returns

true if all sensors are off, false otherwise

7.31.2.2 int INV_EXPORT inv_icm20690_check_drdy (struct inv_icm20690 * *s*, uint8_t *int_status*)

Check for DRDY flag in INT register value.

Parameters

in	*int_status*	INT status register value

Returns

1 if DDRDY is set, 0 otherwise

7.31.2.3 int INV_EXPORT inv_icm20690_check_wom_status (struct inv_icm20690 * s, uint8_t int_status)

Check for WOM bits in INT register value.

Parameters

in	*int_status*	INT status register value

Returns

bit mask corresponding to x, y, z axis that caused WOM (0x01: x axis, 0x02: y axis, 0x01: z axis)

7.31.2.4 int INV_EXPORT inv_icm20690_configure_accel_wom (struct inv_icm20690 * s, uint8_t wom_threshold)

Configuress accel WOM.

Parameters

in	*wom_threshold*	threshold value for X,Y,Z axis that should trigger a WOM interrupt (0 to diable WOM)

Returns

0 on success, negative value on error.

7.31.2.5 int INV_EXPORT inv_icm20690_disable_fifo (struct inv_icm20690 * s)

Disable FIFO.

Returns

0 on success, negative value on error.

7.31.2.6 int INV_EXPORT inv_icm20690_enable_fifo (struct inv_icm20690 * s, int bit_mask)

Enable FIFO.

Parameters

in	*bit_mask*	A mask of sensor to push to FIFO.

Returns

0 on success, negative value on error.

7.31.2.7 int INV_EXPORT inv_icm20690_enable_mems (struct inv_icm20690 * *s*, int *bit_mask*, uint16_t *smplrt_divider*)

Enables accel and/or gyro and/or pressure if integrated with gyro and accel.

Parameters

in	*bit_mask*	A mask where 2 means turn on accel, 1 means turn on gyro, 4 is for pressure. By default, this only turns on a sensor if all sensors are off otherwise the DMP controls this register including turning off a sensor. To override this behavior add in a mask of 128.
in	*smplrt_divider*	The divider which was applied to internal sample rate based on field sample_rate from base_driver_t to get minimum ODR for accel and gyro

Returns

0 on success, negative value on error

7.31.2.8 int INV_EXPORT inv_icm20690_enable_sensor (struct inv_icm20690 * *s*, enum inv_icm20690_sensor *sensor*, uint8_t *enable*)

Enables / disables a sensor.

Parameters

in	*androidSensor*	Sensor Identity
in	*enable*	0=off, 1=on

Returns

0 in case of success, negative value on error

7.31.2.9 int INV_EXPORT inv_icm20690_get_int_status (struct inv_icm20690 * *s*, uint8_t * *int_status*)

Read interrupt status and return its value.

Parameters

out	*int_status*	INT status register value

Returns

0 on success, negative value on error

7.31.2.10 void INV_EXPORT inv_icm20690_get_st_bias (struct inv_icm20690 * s, int * st_bias)

Retrieve bias collected by self-test.

Parameters

out	*st_bias*	bias scaled by 2^16, accel is gee and gyro is dps. The buffer will be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z

7.31.2.11 int INV_EXPORT inv_icm20690_has_data_ready (struct inv_icm20690 * s)

Read interrupt status and check for DDRY flag.

Returns

1 if DDRY inteript is set, 0 otherwise and negative value on error

7.31.2.12 int INV_EXPORT inv_icm20690_is_sensor_enabled (struct inv_icm20690 * s, enum inv_icm20690_sensor sensor)

Knows if the sensor is enabled or disbaled.

Parameters

in	*sensor*	the sensor is enabled or not

Returns

1 if active, 0 otherwise

7.31.2.13 int INV_EXPORT inv_icm20690_poll_delay_count (struct inv_icm20690 * s, int16_t * delay_count)

Poll EIS flag and delay counter.

Parameters

out	*delay_count*	delay time in us between the FSYNC event (before the gyro data event) and the gyro data event

Returns

0x8 if FSYNC event reported, 0 otherwise, negative value on error.

7.31.2.14 int INV_EXPORT inv_icm20690_poll_fifo_data (struct inv_icm20690 * s, struct inv_icm20690_fifo_states * states, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3], int16_t compass_data[3])

Read one packet from the FIFO (packet corresponding to data for all current active sensor)

Parameters

in	*states*	placeholder to FIFO states
out	*acc_data*	raw gyro data
out	*temp_data*	raw temp data
out	*gyro_data*	raw gyro data
out	*compass_data*	raw compass data

Returns

0 : FIFO empty bit0 : gyro data output set bit1 : acc data output set bit2 : compass output set $<$ 0 : error

7.31.2.15 int INV_EXPORT inv_icm20690_poll_fifo_data_setup (struct inv_icm20690 * s, struct inv_icm20690_fifo_states * states, uint8_t int_status)

Initializes states for FIFO reading and parsing.

Parameters

in	*states*	placeholder to FIFO states
in	*int_status*	int status register value (used to check for overflow)

Returns

0 for success, 1 for overlfow detected, negative value for other errors

7.31.2.16 int INV_EXPORT inv_icm20690_poll_ois_gyro_data (struct inv_icm20690 * s, int16_t gyro_data[3])

Poll OIS gyro data.

Returns

0x10 if OIS data reported, 0 otherwise, negative value on error.

7.31.2.17 int INV_EXPORT inv_icm20690_poll_sensor_data_reg (struct inv_icm20690 * s, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3])

Check and retrieve for new data.

Parameters

out	*acc_data*	raw gyro data
out	*temp_data*	raw temp data
out	*gyro_data*	raw gyro data

Returns

0 : not data ready 0x01 : gyro data output set 0x02 : acc data output set 0x03 : gyro and acc data output set $<$ 0 : error

7.31.2.18 int INV_EXPORT inv_icm20690_reset_fifo (struct inv_icm20690 * *s*)

Reset FIFO.

Returns

0 on success, negative value on error.

7.31.2.19 int inv_icm20690_run_selftest (struct inv_icm20690 * *s*)

Perform hardware self-test for Accel, Gyro and Compass.

Parameters

in	*None*	

Returns

COMPASS_SUCCESS$<<$2 | ACCEL_SUCCESS$<<$1 | GYRO_SUCCESS so 3

7.31.2.20 int INV_EXPORT inv_icm20690_set_sensor_period (struct inv_icm20690 * *s*, enum inv_icm20690_sensor *sensor*, uint16_t *delayInMs*)

Sets the odr for a sensor.

Parameters

in	*sensor*	Sensor Identity
in	*delayInMs*	the delay between two values in ms

Returns

0 in case of success, -1 for any error

7.31.2.21 int INV_EXPORT inv_icm20690_set_slave_compass_id (struct inv_icm20690 * s)

Initalize AUX I2C and Compass.

Returns

0 on success, negative value on error.

7.31.2.22 void inv_icm20690_set_st_bias (struct inv_icm20690 * s, int * st_bias)

Apply bias.

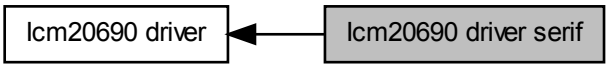
Parameters

in	*st_bias*	bias scaled by 2^16, accel is gee and gyro is dps. The buffer should be be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z

7.32 Icm20690 driver serif

Interface for low-level serial (I2C/SPI) access.

Collaboration diagram for Icm20690 driver serif:



Classes

- struct inv_icm20690_serif
  basesensor serial interface

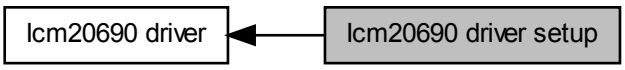
7.32.1 Detailed Description

Interface for low-level serial (I2C/SPI) access.

7.33 Icm20690 driver setup

Low-level function to setup an Icm20690 device.

Collaboration diagram for Icm20690 driver setup:



Functions

- int INV_EXPORT inv_icm20690_get_whoami (struct inv_icm20690 *s, uint8_t *whoami)
  return WHOAMI value
- int INV_EXPORT inv_icm20690_get_chip_info (struct inv_icm20690 *s, uint8_t chip_info[3])
  return WHOAMI value
- int INV_EXPORT inv_icm20690_initialize (struct inv_icm20690 *s)
  Initialize the device.
- int INV_EXPORT inv_icm20690_soft_reset (struct inv_icm20690 *s)
  Perform a soft reset of the device.
- int INV_EXPORT inv_icm20690_set_divider (struct inv_icm20690 *s, uint8_t idiv)
  Set the mpu sample rate.
- int INV_EXPORT inv_icm20690_set_chip_power_state (struct inv_icm20690 *s, uint8_t func, uint8_t on_off)
  Sets the power state of the Ivory chip loop.
- uint8_t INV_EXPORT inv_icm20690_get_chip_power_state (struct inv_icm20690 *s)
  Current wake status of the Mems chip.
- uint16_t INV_EXPORT inv_icm20690_get_chip_base_sample_rate (struct inv_icm20690 *s)
  Get internal sample rate currently configured.
- int INV_EXPORT inv_icm20690_accel_fsr_2_reg (int32_t fsr)
  Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.
- int INV_EXPORT inv_icm20690_reg_2_accel_fsr (uint8_t reg)
  Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.
- int INV_EXPORT inv_icm20690_set_accel_fullscale (struct inv_icm20690 *s, int level)
  Sets fullscale range of accel in hardware.
- int INV_EXPORT inv_icm20690_set_accel_ois_fullscale (struct inv_icm20690 *s, int level)
  Sets fullscale range of OIS accel in hardware.
- int INV_EXPORT inv_icm20690_set_accel_bandwidth (struct inv_icm20690 *s, int level)
  Sets bandwidth range of accel in hardware.
- uint8_t INV_EXPORT inv_icm20690_get_accel_fullscale (struct inv_icm20690 *s)
  Returns fullscale range of accelerometer in hardware.
- uint8_t INV_EXPORT inv_icm20690_get_accel_ois_fullscale (struct inv_icm20690 *s)
  Returns fullscale range of OIS accelerometer in hardware.
- uint16_t INV_EXPORT inv_icm20690_get_accel_bandwidth (struct inv_icm20690 *s)

Returns bandwidth range of accelerometer in hardware.

- int INV_EXPORT inv_icm20690_gyro_fsr_2_reg (int32_t fsr)

  Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)
- int INV_EXPORT inv_icm20690_reg_2_gyro_fsr (uint8_t reg)

  Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)
- int INV_EXPORT inv_icm20690_set_gyro_fullscale (struct inv_icm20690 *s, int level)

  Sets fullscale range of gyro in hardware.
- int INV_EXPORT inv_icm20690_set_gyro_ois_fullscale (struct inv_icm20690 *s, int level)

  Sets fullscale range of OIS gyro in hardware.
- int INV_EXPORT inv_icm20690_set_gyro_bandwidth (struct inv_icm20690 *s, int level)

  Sets bandwidth range of gyro in hardware.
- uint8_t INV_EXPORT inv_icm20690_get_gyro_fullscale (struct inv_icm20690 *s)

  Returns fullscale range of gyroscope in hardware.
- uint8_t INV_EXPORT inv_icm20690_get_gyro_ois_fullscale (struct inv_icm20690 *s)

  Returns fullscale range of OIS gyroscope in hardware.
- uint16_t INV_EXPORT inv_icm20690_get_gyro_bandwidth (struct inv_icm20690 *s)

  Returns bandwidth range of gyroscope in hardware.
- int INV_EXPORT inv_icm20690_is_advanced_features_supported (void)

  Get the available features according the base sensor device.
- int INV_EXPORT inv_icm20690_set_fsync_bit_location (struct inv_icm20690 *s, int bit_location)

  Set FSYNC bit location.

7.33.1 Detailed Description

Low-level function to setup an Icm20690 device.

7.33.2 Function Documentation

7.33.2.1 int INV_EXPORT inv_icm20690_accel_fsr_2_reg (int32_t *fsr*)

Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.

Returns

internal register value for FSR confiugration

7.33.2.2 uint16_t INV_EXPORT inv_icm20690_get_accel_bandwidth (struct inv_icm20690 * *s*)

Returns bandwidth range of accelerometer in hardware.

Returns

the bandwidth range

7.33.2.3 uint8_t INV_EXPORT inv_icm20690_get_accel_fullscale (struct inv_icm20690 * s)

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.33.2.4 uint8_t INV_EXPORT inv_icm20690_get_accel_ois_fullscale (struct inv_icm20690 * s)

Returns fullscale range of OIS accelerometer in hardware.

Returns

the fullscale range

7.33.2.5 uint16_t INV_EXPORT inv_icm20690_get_chip_base_sample_rate (struct inv_icm20690 * s)

Get internal sample rate currently configured.

Returns

Internal sample rate currently configured in MEMS registers in Hz

7.33.2.6 int INV_EXPORT inv_icm20690_get_chip_info (struct inv_icm20690 * s, uint8_t chip_info[3])

return WHOAMI value

Parameters

out	*chip_info[0]*	WHOAMI for device
out	*chip_info[1]*	MANUFACTURER_ID for device
out	*chip_info[2]*	CHIP_ID for device

Returns

0 on success, negative value on error

7.33.2.7 uint8_t INV_EXPORT inv_icm20690_get_chip_power_state (struct inv_icm20690 * s)

Current wake status of the Mems chip.

Returns

the wake status

7.33.2.8 uint16_t INV_EXPORT inv_icm20690_get_gyro_bandwidth (struct inv_icm20690 * s)

Returns bandwidth range of gyroscope in hardware.

Returns

the bandwidth range

7.33.2.9 uint8_t INV_EXPORT inv_icm20690_get_gyro_fullscale (struct inv_icm20690 * s)

Returns fullscale range of gyroscope in hardware.

Returns

the fullscale range

7.33.2.10 uint8_t INV_EXPORT inv_icm20690_get_gyro_ois_fullscale (struct inv_icm20690 * s)

Returns fullscale range of OIS gyroscope in hardware.

Returns

the fullscale range

7.33.2.11 int INV_EXPORT inv_icm20690_get_whoami (struct inv_icm20690 * s, uint8_t * whoami)

return WHOAMI value

Parameters

out	*whoami*	WHOAMI for device

Returns

0 on success, negative value on error

7.33.2.12 int INV_EXPORT inv_icm20690_gyro_fsr_2_reg (int32_t fsr)

Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)

Returns

internal register value for FSR confiugration

7.33.2.13 int INV_EXPORT inv_icm20690_initialize (struct inv_icm20690 * s)

Initialize the device.

Returns

0 on success, negative value on error.

7.33.2.14 int INV_EXPORT inv_icm20690_is_advanced_features_supported (void)

Get the available features according the base sensor device.

Parameters

in	*in*	0: No advanced features supported 1: OIS sensor and FSYNC behavior supported (for icm20690 only)

7.33.2.15 int INV_EXPORT inv_icm20690_reg_2_accel_fsr (uint8_t *reg*)

Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.

Returns

FSR value in mg

7.33.2.16 int INV_EXPORT inv_icm20690_reg_2_gyro_fsr (uint8_t *reg*)

Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)

Returns

FSR value in dps

7.33.2.17 int INV_EXPORT inv_icm20690_set_accel_bandwidth (struct inv_icm20690 * *s*, int *level*)

Sets bandwidth range of accel in hardware.

Parameters

in	*level*	See mpu_accel_bw.

Returns

0 on success, negative value on error.

7.33.2.18 int INV_EXPORT inv_icm20690_set_accel_fullscale (struct inv_icm20690 * *s*, int *level*)

Sets fullscale range of accel in hardware.

Parameters

in	*level*	See mpu_accel_fs.

Returns

0 on success, negative value on error.

7.33.2.19 int INV_EXPORT inv_icm20690_set_accel_ois_fullscale (struct inv_icm20690 * *s*, int *level*)

Sets fullscale range of OIS accel in hardware.

Parameters

in	*level*	See mpu_accel_ois_fs.

Returns

0 on success, negative value on error.

7.33.2.20 int INV_EXPORT inv_icm20690_set_chip_power_state (struct inv_icm20690 * *s*, uint8_t *func*, uint8_t *on_off*)

Sets the power state of the Ivory chip loop.

Parameters

in	*func*	CHIP_AWAKE, CHIP_LP_ENABLE
in	*on_off*	The functions are enabled if previously disabled and disabled if previously enabled based on the value of On/Off.

Returns

0 on success, negative value on error.

7.33.2.21 int INV_EXPORT inv_icm20690_set_divider (struct inv_icm20690 * *s*, uint8_t *idiv*)

Set the mpu sample rate.

Returns

Value written to MPUREG_SMPLRT_DIV register.

7.33.2.22 int INV_EXPORT inv_icm20690_set_fsync_bit_location (struct inv_icm20690 * *s*, int *bit_location*)

Set FSYNC bit location.

Parameters

in	*in*	0: Disable FSYNC pin data to be sampled 1: Enable FSYNC pin data to be sampled at TEMP_OUT_L[0]

7.33.2.23 int INV_EXPORT inv_icm20690_set_gyro_bandwidth (struct inv_icm20690 * *s*, int *level*)

Sets bandwidth range of gyro in hardware.

Parameters

in	*level*	See mpu_gyro_bw.

Returns

0 on success, negative value on error.

7.33.2.24 int INV_EXPORT inv_icm20690_set_gyro_fullscale (struct inv_icm20690 * *s*, int *level*)

Sets fullscale range of gyro in hardware.

Parameters

in	*level*	See mpu_gyro_fs.

Returns

0 on success, negative value on error.

7.33.2.25 int INV_EXPORT inv_icm20690_set_gyro_ois_fullscale (struct inv_icm20690 * *s*, int *level*)

Sets fullscale range of OIS gyro in hardware.

Parameters

in	*level*	See mpu_gyro_fs.

Returns

0 on success, negative value on error.

7.33.2.26 int INV_EXPORT inv_icm20690_soft_reset (struct inv_icm20690 * s)

Perform a soft reset of the device.

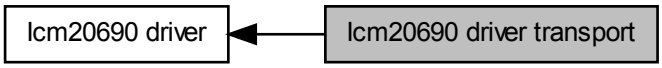
Returns

0 on success, negative value on error.

7.34 Icm20690 driver transport

Low-level Icm20690 register access.

Collaboration diagram for Icm20690 driver transport:



Functions

- int INV_EXPORT inv_icm20690_mems_write_reg (struct inv_icm20690 ∗s, uint16_t reg, uint32_t length, const uint8_t ∗data)

  Writes data from a register on mems.
- int INV_EXPORT inv_icm20690_mems_write_reg_one (struct inv_icm20690 ∗s, uint16_t reg, uint8_t data)

  Writes a single byte of data from a register on mems with no power control.
- int INV_EXPORT inv_icm20690_mems_read_reg (struct inv_icm20690 ∗s, uint16_t reg, uint32_t length, uint8_t ∗data)

  Reads data from a register on mems.

7.34.1 Detailed Description

Low-level Icm20690 register access.

7.34.2 Function Documentation

7.34.2.1 int INV_EXPORT inv_icm20690_mems_read_reg (struct inv_icm20690 ∗ *s*, uint16_t *reg*, uint32_t *length*, uint8_t ∗ *data*)

Reads data from a register on mems.

Parameters

in	*reg*	register address
in	*length*	length of data
out	*data*	output data from the register

Returns

0 in case of success, -1 for any error

7.34.2.2 int INV_EXPORT inv_icm20690_mems_write_reg (struct inv_icm20690 * *s,* uint16_t *reg,* uint32_t *length,* const uint8_t * *data*)

Writes data from a register on mems.

Parameters

in	*length*	number of byte to be written
out	*data*	output data from the register

Returns

0 in case of success, -1 for any error

7.34.2.3 int INV_EXPORT inv_icm20690_mems_write_reg_one (struct inv_icm20690 * *s,* uint16_t *reg,* uint8_t *data*)

Writes a single byte of data from a register on mems with no power control.

Parameters

in	*reg*	Register address
out	*data*	Data to be written

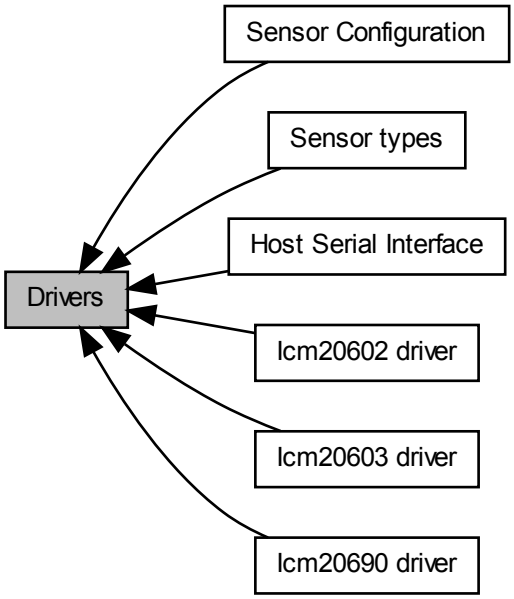
Returns

0 in case of success, -1 for any error

7.35 Drivers

Low-level drivers for InvenSense devices.

Collaboration diagram for Drivers:



Modules

- Sensor types

  Sensor related types definitions.
- Sensor Configuration

  General sensor configuration types definitions.
- Host Serial Interface

  Virtual abstraction of host adapter for serial interface.
- Icm20602 driver

  Low-level driver for Icm20602 devices.
- Icm20603 driver

  Low-level driver for Icm20603 devices.
- Icm20690 driver

  Low-level driver for Icm20690 devices.

7.35.1 Detailed Description

Low-level drivers for InvenSense devices.

Those drivers are intended to be portable and used in embedded context. They can be used directly but this is not advised as they may not be user-friendly. The proper way to access a device from the application is through the Device API.

7.36 Utils

Utility functions.

Utility functions.

Chapter 8

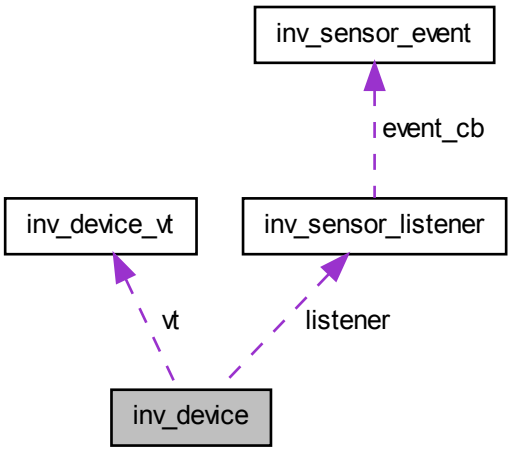
Class Documentation

8.1 inv_device Struct Reference

Abtract device object definition.

`#include <Device.h>`

Collaboration diagram for inv_device:



Public Attributes

- void * instance
  pointer to object instance
- const struct inv_device_vt * vt
  pointer to object virtual table
- const inv_sensor_listener_t * listener
  pointer to listener instance

8.1.1 Detailed Description

Abtract device object definition.

Examples:

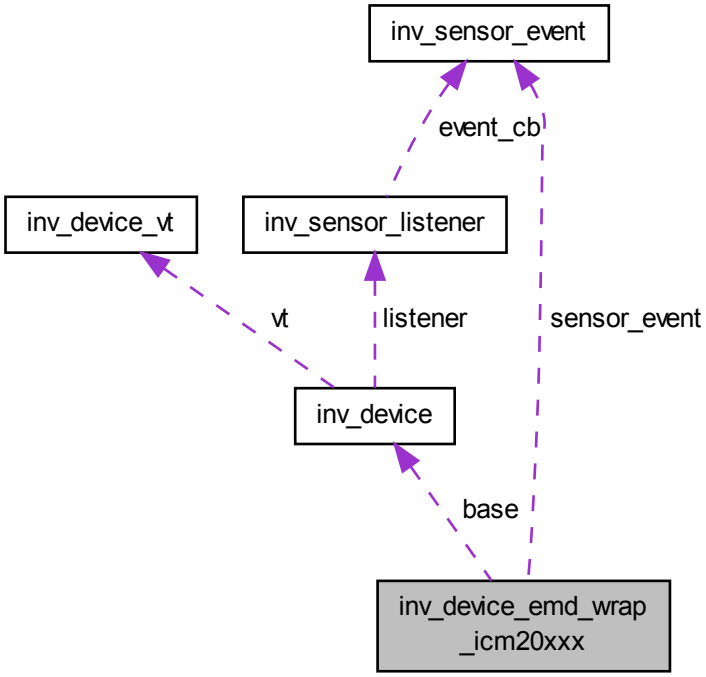
ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

The documentation for this struct was generated from the following file:

- Device.h

8.2 inv_device_emd_wrap_icm20xxx Struct Reference

Collaboration diagram for inv_device_emd_wrap_icm20xxx:



The documentation for this struct was generated from the following file:

- DeviceEmdWrapIcm20xxx.h

8.3 inv_device_emd_wrap_icm20xxx_serial Struct Reference

The documentation for this struct was generated from the following file:

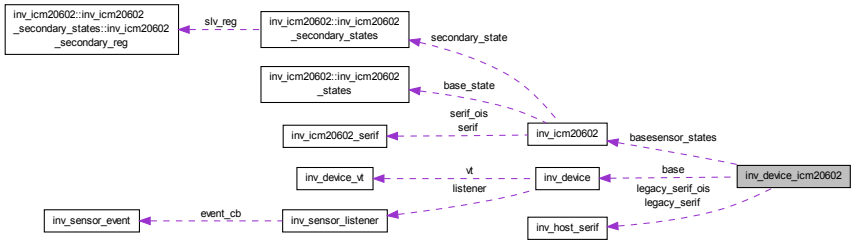
- DeviceEmdWrapIcm20xxx.h

8.4 inv_device_icm20602 Struct Reference

States for Icm20602 device.

`#include <DeviceIcm20602.h>`

Collaboration diagram for inv_device_icm20602:



8.4.1 Detailed Description

States for Icm20602 device.

Examples:

ExampleDeviceIcm20602EMD.c.

The documentation for this struct was generated from the following file:

- DeviceIcm20602.h

8.5 inv_device_icm20602_config_bias_st Struct Reference

Bias collected during self-test (config INV_DEVICE_ICM20602_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

`#include <DeviceIcm20602.h>`

Public Attributes

- int32_t gyr_bias_lp [3]

  Gyro LP mode X,Y,Z.
- int32_t gyr_bias_nl [3]

  Gyro normal mode X,Y,Z.
- int32_t acc_bias_lp [3]

  Accel LP mode X,Y,Z.
- int32_t acc_bias_nl [3]

  Accel normal mode X,Y,Z.

8.5.1 Detailed Description

Bias collected during self-test (config INV_DEVICE_ICM20602_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

The documentation for this struct was generated from the following file:

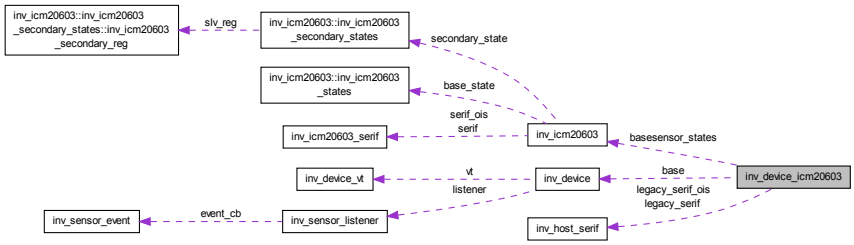
- DeviceIcm20602.h

8.6 inv_device_icm20603 Struct Reference

States for Icm20603 device.

`#include <DeviceIcm20603.h>`

Collaboration diagram for inv_device_icm20603:



8.6.1 Detailed Description

States for Icm20603 device.

Examples:

ExampleDeviceIcm20603EMD.c.

The documentation for this struct was generated from the following file:

- DeviceIcm20603.h

8.7 inv_device_icm20603_config_bias_st Struct Reference

Bias collected during self-test (config INV_DEVICE_ICM20603_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

`#include <DeviceIcm20603.h>`

Public Attributes

- int32_t gyr_bias_lp [3]

  Gyro LP mode X,Y,Z.
- int32_t gyr_bias_nl [3]

  Gyro normal mode X,Y,Z.
- int32_t acc_bias_lp [3]

  Accel LP mode X,Y,Z.
- int32_t acc_bias_nl [3]

  Accel normal mode X,Y,Z.

8.7.1 Detailed Description

Bias collected during self-test (config INV_DEVICE_ICM20603_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

The documentation for this struct was generated from the following file:

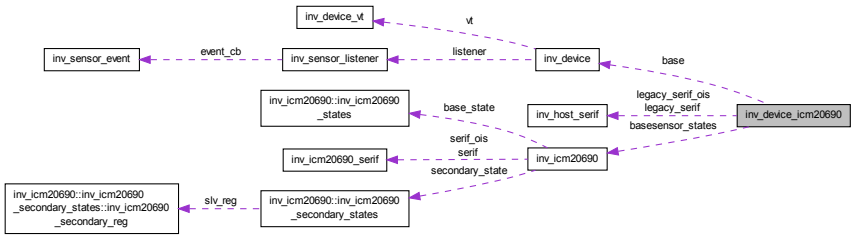
- DeviceIcm20603.h

8.8 inv_device_icm20690 Struct Reference

States for Icm20690 device.

`#include <DeviceIcm20690.h>`

Collaboration diagram for inv_device_icm20690:



8.8.1 Detailed Description

States for Icm20690 device.

Examples:

ExampleDeviceIcm20690EMD.c.

The documentation for this struct was generated from the following file:

- DeviceIcm20690.h

8.9 inv_device_icm20690_config_bias_st Struct Reference

Bias collected during self-test (config INV_DEVICE_ICM20690_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

```
#include <DeviceIcm20690.h>
```

Public Attributes

- int32_t gyr_bias_lp [3]

  Gyro LP mode X,Y,Z.
- int32_t gyr_bias_nl [3]

  Gyro normal mode X,Y,Z.
- int32_t acc_bias_lp [3]

  Accel LP mode X,Y,Z.
- int32_t acc_bias_nl [3]

  Accel normal mode X,Y,Z.

8.9.1 Detailed Description

Bias collected during self-test (config INV_DEVICE_ICM20690_CONFIG_BIAS_ST) Value is scaled by 2^16, accel is gee and gyro is dps.

The documentation for this struct was generated from the following file:

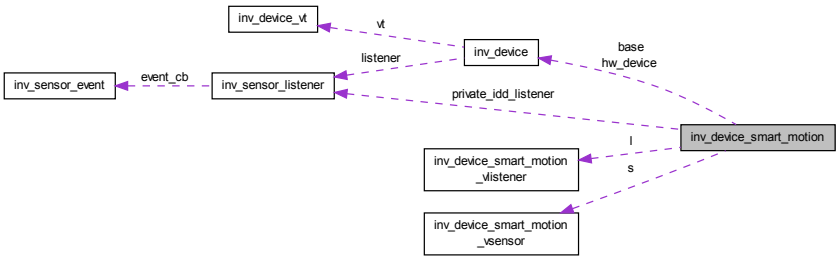
- DeviceIcm20690.h

8.10 inv_device_smart_motion Struct Reference

States for SmartMotion device implementation.

`#include <DeviceSmartMotion.h>`

Collaboration diagram for inv_device_smart_motion:



Public Attributes

- inv_device_t base

  base inv_device_t states
- inv_device_t * hw_device

  reference to inv_device_t that will provide sensor data for base sensor
- uint8_t base_chip_info [3]

  parameter given at init time to identify underlying ICM
- inv_sensor_listener_t private_idd_listener

  internal IDD listener to catch events from HW device
- InvList leaves_list

  list of graph leaves (corresponds to sensors not present at hardware level and that are emulated)
- struct {
  struct inv_device_smart_motion_vsensor inv_device_smart_motion::s
  placeholder for roots VSensor
  struct inv_device_smart_motion_vlistener inv_device_smart_motion::l
  placeholder for leaves corresponding to root VSensor
  } graph_roots [5]

  placehold holder for graph roots objects
- struct {
  struct inv_device_smart_motion_vlistener inv_device_smart_motion::l
  placeholder for leaves VSensorListener
  } graph_leaves [24]

  placeholder for leaves corresponding to emulated sensors
- uint64_t hw_sensor_available_mask

  mask to keep track of HW available sensor from underlying device

8.10.1 Detailed Description

States for SmartMotion device implementation.

The documentation for this struct was generated from the following file:

- DeviceSmartMotion.h

8.11 inv_device_smart_motion_vlistener Struct Reference

Internal definition for graph leaves (based on VSensorListener)

`#include <DeviceSmartMotion.h>`

Public Attributes

- VSensorListener vlistener
  base VSensorListener states
- void(∗ build_event)(const void ∗, inv_sensor_event_t ∗)
  callback to convert VSensorData to IDD sensor event
- uint8_t idd_type
  corresponding IDD sensor type
- InvList node
  pointer to next element when stored in a list

8.11.1 Detailed Description

Internal definition for graph leaves (based on VSensorListener)

The documentation for this struct was generated from the following file:

- DeviceSmartMotion.h

8.12 inv_device_smart_motion_vsensor Struct Reference

Internal definition for graph roots (based on VSensor)

`#include <DeviceSmartMotion.h>`

Public Attributes

- VSensor vsensor
  base VSensor states
- void(∗ build_vsensor_data)(const inv_sensor_event_t ∗, void ∗)
  callback to convert IDD sensor event to VSensorData
- uint8_t idd_type
  corresponding IDD sensor type

8.12.1 Detailed Description

Internal definition for graph roots (based on VSensor)

The documentation for this struct was generated from the following file:

- DeviceSmartMotion.h

8.13 inv_device_vt Struct Reference

Device virtual table definition.

`#include <Device.h>`

8.13.1 Detailed Description

Device virtual table definition.

The documentation for this struct was generated from the following file:

- Device.h

8.14 inv_fw_version Struct Reference

FW version structure definition.

`#include <Device.h>`

Public Attributes

- uint8_t patch
  major, minor, patch version number
- char suffix [16]
  version suffix string (always terminated by '\0')
- uint32_t crc
  FW checksum.

8.14.1 Detailed Description

FW version structure definition.

The documentation for this struct was generated from the following file:

- Device.h

8.15 inv_host_serif Struct Reference

Serial Interface interface definition.

`#include <HostSerif.h>`

Public Attributes

- int(∗ open)(void)
  Open connection to and initialize Serial Interface adapter.
- int(∗ close)(void)
  Close connection to Serial Interface adapter.
- int(∗ read_reg)(uint8_t reg, uint8_t ∗data, uint32_t len)
  Perform a read register transaction over the serial interface.
- int(∗ write_reg)(uint8_t reg, const uint8_t ∗data, uint32_t len)
  Perform a write register transaction over the serial interface.
- int(∗ register_interrupt_callback)(void(∗interrupt_cb)(void ∗context, int int_num), void ∗context)
  Register a callback to the adapter.
- uint32_t max_read_size
  Maximum number of bytes allowed per serial read.
- uint32_t max_write_size
  Maximum number of bytes allowed per serial write.
- int serif_type
  Type of underlying serial interface.

8.15.1 Detailed Description

Serial Interface interface definition.

8.15.2 Member Data Documentation

8.15.2.1 int(∗ inv_host_serif::close) (void)

Close connection to Serial Interface adapter.

Returns

0 on sucess, negative value on error

8.15.2.2 int(∗ inv_host_serif::open) (void)

Open connection to and initialize Serial Interface adapter.

Returns

0 on sucess, negative value on error

8.15.2.3 int(∗ inv_host_serif::read_reg) (uint8_t reg, uint8_t ∗data, uint32_t len)

Perform a read register transaction over the serial interface.

Parameters

in	*reg*	register
out	*data*	pointer to output buffer
in	*len*	number of byte to read (should not exceed MAX_TRANSACTION_SIZE)

Returns

0 on sucess, negative value on error

8.15.2.4 int(∗ inv_host_serif::register_interrupt_callback) (void(∗interrupt_cb)(void ∗context, int int_num), void ∗context)

Register a callback to the adapter.

Parameters

in	*interrupt_cb*	callback to call on interrupt
in	*context*	context passed to callback

Returns

0 on sucess, negative value on error

8.15.2.5 int(∗ inv_host_serif::write_reg) (uint8_t reg, const uint8_t ∗data, uint32_t len)

Perform a write register transaction over the serial interface.

Parameters

in	*reg*	register
out	*data*	pointer to output buffer
in	*len*	number of byte to read (should not exceed MAX_TRANSACTION_SIZE)

Returns

0 on sucess, negative value on error

The documentation for this struct was generated from the following file:

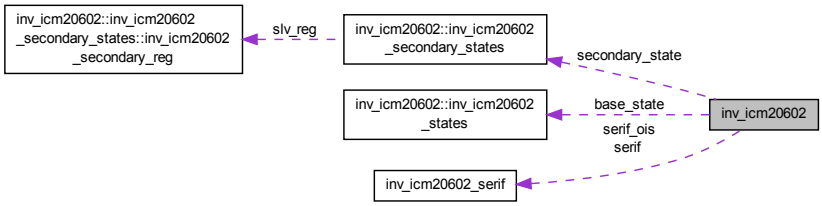
- HostSerif.h

8.16 inv_icm20602 Struct Reference

Icm20602 driver states definition.

`#include <Icm20602.h>`

Collaboration diagram for inv_icm20602:



Classes

- struct inv_icm20602_secondary_states
  secondary device support
- struct inv_icm20602_states
  icm20602 internal state structure

8.16.1 Detailed Description

Icm20602 driver states definition.

The documentation for this struct was generated from the following file:

- Icm20602.h

8.17 inv_icm20602_fifo_states Struct Reference

Check and retrieve for new data.

`#include <Icm20602Ctrl.h>`

Public Attributes

- uint8_t overflow:1
  indicates FIFO overflow: user should restart all sensors if this occurs

8.17.1 Detailed Description

Check and retrieve for new data.

The documentation for this struct was generated from the following file:

- Icm20602Ctrl.h

8.18 inv_icm20602::inv_icm20602_secondary_states::inv_icm20602_secondary_reg Struct Reference

secondary register mapping

`#include <Icm20602.h>`

8.18.1 Detailed Description

secondary register mapping

The documentation for this struct was generated from the following file:

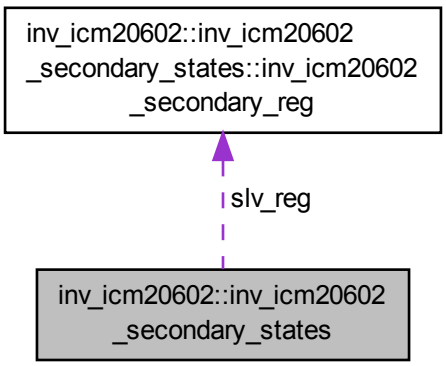
- Icm20602.h

8.19 inv_icm20602::inv_icm20602_secondary_states Struct Reference

secondary device support

`#include <Icm20602.h>`

Collaboration diagram for inv_icm20602::inv_icm20602_secondary_states:



Classes

- struct inv_icm20602_secondary_reg
  secondary register mapping

8.19.1 Detailed Description

secondary device support

The documentation for this struct was generated from the following file:

- Icm20602.h

8.20 inv_icm20602_serif Struct Reference

basesensor serial interface

`#include <Icm20602Serif.h>`

8.20.1 Detailed Description

basesensor serial interface

The documentation for this struct was generated from the following file:

- Icm20602Serif.h

8.21 inv_icm20602::inv_icm20602_states Struct Reference

icm20602 internal state structure

`#include <Icm20602.h>`

8.21.1 Detailed Description

icm20602 internal state structure

The documentation for this struct was generated from the following file:

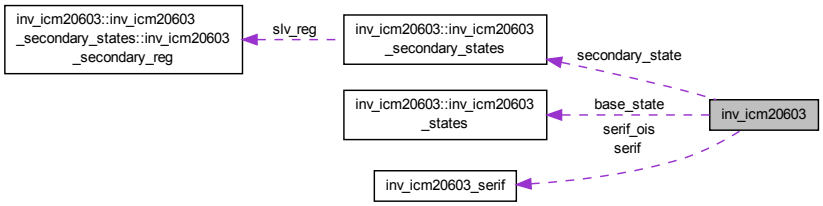
- Icm20602.h

8.22 inv_icm20603 Struct Reference

Icm20603 driver states definition.

`#include <Icm20603.h>`

Collaboration diagram for inv_icm20603:



Classes

- struct inv_icm20603_secondary_states

  secondary device support
- struct inv_icm20603_states

  icm20603 internal state structure

8.22.1 Detailed Description

Icm20603 driver states definition.

The documentation for this struct was generated from the following file:

- Icm20603.h

8.23 inv_icm20603_fifo_states Struct Reference

Check and retrieve for new data.

`#include <Icm20603Ctrl.h>`

Public Attributes

- uint8_t overflow:1

  indicates FIFO overflow: user should restart all sensors if this occurs

8.23.1 Detailed Description

Check and retrieve for new data.

The documentation for this struct was generated from the following file:

- Icm20603Ctrl.h

8.24 inv_icm20603::inv_icm20603_secondary_states::inv_icm20603_secondary_reg Struct Reference

secondary register mapping

`#include <Icm20603.h>`

8.24.1 Detailed Description

secondary register mapping

The documentation for this struct was generated from the following file:

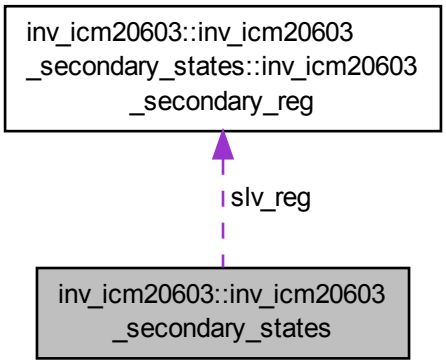
- Icm20603.h

8.25 inv_icm20603::inv_icm20603_secondary_states Struct Reference

secondary device support

`#include <Icm20603.h>`

Collaboration diagram for inv_icm20603::inv_icm20603_secondary_states:



Classes

- struct inv_icm20603_secondary_reg
  secondary register mapping

8.25.1 Detailed Description

secondary device support

The documentation for this struct was generated from the following file:

- Icm20603.h

8.26 inv_icm20603_serif Struct Reference

basesensor serial interface

```
#include <Icm20603Serif.h>
```

8.26.1 Detailed Description

basesensor serial interface

The documentation for this struct was generated from the following file:

- Icm20603Serif.h

8.27 inv_icm20603::inv_icm20603_states Struct Reference

icm20603 internal state structure

```
#include <Icm20603.h>
```

8.27.1 Detailed Description

icm20603 internal state structure

The documentation for this struct was generated from the following file:

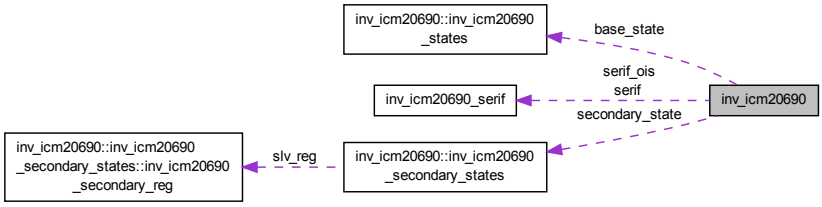
- Icm20603.h

8.28 inv_icm20690 Struct Reference

Icm20690 driver states definition.

`#include <Icm20690.h>`

Collaboration diagram for inv_icm20690:



Classes

- struct inv_icm20690_secondary_states

  secondary device support
- struct inv_icm20690_states

  icm20690 internal state structure

8.28.1 Detailed Description

Icm20690 driver states definition.

The documentation for this struct was generated from the following file:

- Icm20690.h

8.29 inv_icm20690_fifo_states Struct Reference

Check and retrieve for new data.

`#include <Icm20690Ctrl.h>`

Public Attributes

- uint8_t overflow:1

  indicates FIFO overflow: user should restart all sensors if this occurs

8.29.1 Detailed Description

Check and retrieve for new data.

The documentation for this struct was generated from the following file:

- Icm20690Ctrl.h

8.30 inv_icm20690::inv_icm20690_secondary_states::inv_icm20690_secondary_reg Struct Reference

secondary register mapping

`#include <Icm20690.h>`

8.30.1 Detailed Description

secondary register mapping

The documentation for this struct was generated from the following file:

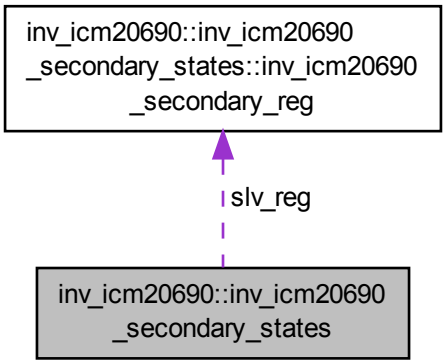
- Icm20690.h

8.31 inv_icm20690::inv_icm20690_secondary_states Struct Reference

secondary device support

`#include <Icm20690.h>`

Collaboration diagram for inv_icm20690::inv_icm20690_secondary_states:



Classes

- struct inv_icm20690_secondary_reg
  secondary register mapping

8.31.1 Detailed Description

secondary device support

The documentation for this struct was generated from the following file:

- Icm20690.h

8.32 inv_icm20690_serif Struct Reference

basesensor serial interface

`#include <Icm20690Serif.h>`

8.32.1 Detailed Description

basesensor serial interface

The documentation for this struct was generated from the following file:

- Icm20690Serif.h

8.33 inv_icm20690::inv_icm20690_states Struct Reference

icm20690 internal state structure

`#include <Icm20690.h>`

8.33.1 Detailed Description

icm20690 internal state structure

The documentation for this struct was generated from the following file:

- Icm20690.h

8.34 inv_sensor_config_bac Struct Reference

Define the configuration for BAC.

`#include <SensorConfig.h>`

8.34.1 Detailed Description

Define the configuration for BAC.

Parameters

enableNotify	enable disable notify

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.35 inv_sensor_config_BSCD Struct Reference

Define the configuration for the BSCD virtual sensor.

`#include <SensorConfig.h>`

8.35.1 Detailed Description

Define the configuration for the BSCD virtual sensor.

Parameters

Age	age in year; Range is (0;100). Default is 35.
Gender	gender is 0 for men, 1 for female. Default is 0
Height	height in centimeter; Range is (50;250). Default is 175.
Weight	weight in kg; Range is (3;300). Default is 75
enableNotify	bitmask to enable/disable notify on a a specific sensor event bit 0 (1): enable/disable notify on BAC event bit 1 (2): enable/disable notify on step counter event bit 2 (4): enable/disable notify on energy expenditure event bit 3 (8): enable/disable notify on distance event

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.36 inv_sensor_config_context Struct Reference

Define configuration context value (associated with INV_SENSOR_CONFIG_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implemetation.

`#include <SensorConfig.h>`

8.36.1 Detailed Description

Define configuration context value (associated with INV_SENSOR_CONFIG_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implemetation.

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.37 inv_sensor_config_distance Struct Reference

Define the configuration for the distance's algorithm.

`#include <SensorConfig.h>`

8.37.1 Detailed Description

Define the configuration for the distance's algorithm.

Parameters

user_height	height of the user in cm
enableNotify	enable disable notify

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.38 inv_sensor_config_double_tap Struct Reference

Define the configuration for the double tap's algorithm.

`#include <SensorConfig.h>`

8.38.1 Detailed Description

Define the configuration for the double tap's algorithm.

Parameters

minimum_threshold	This parameter sets the minimum threshold to reach in order to start a Tap detection. Default value is 2000, recommended range [500 ; 2500]
t_max	This parameter sets the maximum time after a Tap event in [sample]. Default value is 100, recommended range [30 ; 200].

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.39 inv_sensor_config_energy_expenditure Struct Reference

Define the configuration for the energy expenditure's algorithm.

`#include <SensorConfig.h>`

8.39.1 Detailed Description

Define the configuration for the energy expenditure's algorithm.

Parameters

age	age in year; Range is (0;100).
gender	gender is 0 for men, 1 for female.
height	height in centimeter; Range is (50;250)
weight	weight in kg; Range is (3;300)
enableNotify	enable disable notify

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.40 inv_sensor_config_fsr Struct Reference

Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV_SENSOR_↩ CONFIG_FSR config ID) Value is expetcted to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.

`#include <SensorConfig.h>`

8.40.1 Detailed Description

Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV_SENSOR_↩ CONFIG_FSR config ID) Value is expetcted to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.41 inv_sensor_config_gain Struct Reference

Define gain matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.

`#include <SensorConfig.h>`

8.41.1 Detailed Description

Define gain matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.42 inv_sensor_config_mounting_mtx Struct Reference

Define mounting matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_MOUNTING_MATR↩ IX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.

`#include <SensorConfig.h>`

8.42.1 Detailed Description

Define mounting matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_MOUNTING_MATR↩ IX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.

Value is expetcted to be a rotation matrix.

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.43 inv_sensor_config_offset Struct Reference

Define offset vector value for 3-axis sensors (associated with INV_SENSOR_CONFIG_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.

`#include <SensorConfig.h>`

8.43.1 Detailed Description

Define offset vector value for 3-axis sensors (associated with INV_SENSOR_CONFIG_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.

If applied to RAW sensor, value is expected to be in lsb. If applied to other sensor, value is expected to be in sensor unit (g, uT or dps).

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.44 inv_sensor_config_powermode Struct Reference

Define chip power mode (associated with INV_SENSOR_CONFIG_POWER_MODE config ID) Value is expetcted to be 0 for low power or 1 for low noise.

`#include <SensorConfig.h>`

8.44.1 Detailed Description

Define chip power mode (associated with INV_SENSOR_CONFIG_POWER_MODE config ID) Value is expetcted to be 0 for low power or 1 for low noise.

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.45 inv_sensor_config_shake_wrist Struct Reference

Define the configuration for the shake wrist's algorithm.

`#include <SensorConfig.h>`

8.45.1 Detailed Description

Define the configuration for the shake wrist's algorithm.

Parameters

max_period	This parameter sets the maximal duration for half oscillation to detect a Shake wrist. The default value is 20, recommend range [15 ; 40], 15 for the lower sensitivity and 40 for the higher sensitivity. Notice that increasing the sensitivity will increase the number of false detection, and also slightly increase response time.
dummy_padding	Dummy byte for padding. Set it to 0.

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.46 inv_sensor_config_stepc Struct Reference

Define the configuration for steps counter.

`#include <SensorConfig.h>`

8.46.1 Detailed Description

Define the configuration for steps counter.

Parameters

enableNotify	enable disable notify

The documentation for this struct was generated from the following file:

- SensorConfig.h

8.47 inv_sensor_event Struct Reference

Sensor event definition.

`#include <SensorTypes.h>`

Public Attributes

- unsigned int sensor
  sensor type
- int status
  sensor data status as of enum inv_sensor_status
- uint64_t timestamp
  sensor data timestamp in us
- union {
  struct {
  float vect [3]
  x,y,z vector data
  float bias [3]
  x,y,z bias vector data
  uint8_t accuracy_flag
  accuracy flag
  } acc
  3d accelerometer data in g
  struct {
  float vect [3]
  x,y,z vector data
  float bias [3]
  x,y,z bias vector data (for uncal sensor variant)
  uint8_t accuracy_flag
  accuracy flag
  } mag
  3d magnetometer data in uT
  struct {
  float vect [3]
  x,y,z vector data
  float bias [3]
  x,y,z bias vector data (for uncal sensor variant)
  uint8_t accuracy_flag
  accuracy flag
  } gyr
  3d gyroscope data in deg/s
  struct {
  float quat [4]
  w,x,y,z quaternion data
  float accuracy
  heading accuracy in deg
  uint8_t accuracy_flag
  accuracy flag specific for GRV

} quaternion
quaternion data
struct {
float z
x,y,z angles in deg as defined by Google Orientation sensor
uint8_t accuracy_flag
heading accuracy in deg
} orientation
orientation data
struct {
float bpm
beat per minute
uint8_t confidence
confidence level
uint8_t sqi
signal quality as seen by the the HRM engine
} hrm
heart rate monitor data
struct {
int32_t acc [3]
accel data used by hrm algorithm
int32_t gyr [3]
gyro data used by hrm algorithm
uint32_t ppg_value
ppg value read from HRM sensor
float ppm
beat per minute
uint8_t confidence
confidence level
uint8_t sqi
signal quality as seen by the the HRM engine
uint8_t touch_status
touch status, detected or not by the PPG
uint8_t gyrEnable
1 gyro is enable else 0
} hrmlogger
heart rate monitor logger data
struct {
int16_t rr_interval [4]
beat-to-beat(RR) interval
} hrv
heart rate variability data
struct {
uint32_t ppg_value
ppg value read from HRM sensor
uint8_t touch_status
touch status, detected or not
} rawppg
raw heart rate monitor data
struct {
uint8_t sleep_phase
state of sleep phases: 0 not defined, 1 restless sleep, 2 light sleep, 3 deep sleep
uint32_t timestamp
time stamp of the sleep phase transition (seconds)
int32_t sleep_onset
time until first period of 20 min sleep without more than 1 min wake
int32_t sleep_latency
time until first sleep phase

uint32_t time_in_bed
time in bed (seconds)
uint32_t total_sleep_time
total sleep time (seconds)
uint8_t sleep_efficiency
ratio between total sleep time and time in bed
} sleepanalysis
sleep analysis data
struct {
int event
BAC extended data begin/end event as of enum inv_sensor_bac_ext_event.
} bacext
activity classifier (BAC) extended data
struct {
uint32_t durationWalk
ms
uint32_t durationRun
ms
uint32_t durationTransportSit
ms
uint32_t durationTransportStand
ms
uint32_t durationBiking
ms
uint32_t durationStillSit
ms
uint32_t durationStillStand
ms
uint32_t durationTotalSit
Still-Sit + Transport-Sit + Biking (ms)
uint32_t durationTotalStand
Still-Stand + Transport-Stand (ms)
uint32_t stepWalk
walk step count
uint32_t stepRun
run step count
} bacstat
activity classifier (BAC) statistics data
struct {
int32_t floorsUp
number of floors climbed Up on foot by user.
int32_t floorsDown
number of floors climbed Down on foot by user.
} floorclimb
floor climbed data
struct {
int32_t instantEEkcal
energy expenditure in kilocalorie/min since last output.
int32_t instantEEmets
energy expenditure in METs(Metabolic Equivalent of Task) since last output.
int32_t cumulativeEEkcal
cumulative energy expenditure since the last reset in kilocalorie.
int32_t cumulativeEEmets
cumulative energy expenditure since the last reset in METs (Metabolic Equivalent of Task).
} energyexp
energy expenditure data
struct {
int32_t distanceWalk

distance in meters
int32_t distanceRun
distance in meters
} distance
distance data
struct {
float tmp
temperature in deg celcius
} temperature
temperature data
struct {
float percent
relative humidity in %
} humidity
humidity data
struct {
uint64_t count
number of steps
} step
step-counter data
struct {
uint32_t level
light level in lux
} light
light data
struct {
uint32_t distance
distance in mm
} proximity
proximity data
struct {
uint32_t pressure
pressure in Pa
} pressure
pressure data
struct {
int event
BAC data begin/end event as of enum inv_sensor_bac_event.
} bac
BAC data.
struct {
uint32_t fxdata [12]
PDR data in fixpoint.
} pdr
PDR data.
struct {
float vect [3]
x,y,z vector data
float bias [3]
x,y,z bias vector data (for uncal sensor variant)
int16_t delta_ts
timestamp delta between standard gyro and EIS gyro
} eis
EIS data.
struct {
int32_t vect [3]
x,y,z vector data
uint32_t fsr

full scale range
} raw3d
3d raw acc, mag or gyr
struct {
int32_t raw
raw temperature value
} rawtemp
Raw temperature data.
struct {
uint8_t status [6]
raw temperature value
} tsimu_status
TSIMU status data.
inv_bool_t event
event state for gesture-like sensor (SMD, B2S, Step-detector, Tilt-detector, Wake, Glance, Pick-Up, Shake, Double-tap,
struct {
} wom
< FSYNC tag (EIS sensor)
struct {
uint8_t * buffer
pointer to buffer
uint32_t size
current buffer size
} audio_buffer
Wake-up on motion data.
struct {
struct {
int event
BAC data begin/end event as of enum inv_sensor_bac_event.
} bac
BAC data.
struct {
uint64_t count
number of steps
} step
step-counter data
int32_t cumulativeEEkcal
cumulative energy expenditure since the last reset in kilocalorie.
int32_t distance
sum of walk and run distance in meters
} bscd
buffer of custom BSCD
struct {
int32_t raw_pressure
raw pressure
float pressure
pressure in Pa
int32_t raw_temperature
raw temperature
float temperature
temperature in deg C
} custom_pressure
pressure data
uint8_t reserved [INV_SENSOR_EVENT_DATA_SIZE]
reserved sensor data for future sensor
} data

sensor data

- int32_t table [7]

  data encrypted table
- int16_t delay_count

  delay counter in us between FSYNC tag and previous gyro data

8.47.1 Detailed Description

Sensor event definition.

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

8.47.2 Member Data Documentation

8.47.2.1 uint8_t inv_sensor_event::accuracy_flag

accuracy flag

heading accuracy in deg

accuracy flag specific for GRV

8.47.2.2 struct { ... } inv_sensor_event::audio_buffer

Wake-up on motion data.

buffer of audio data

8.47.2.3 float inv_sensor_event::bias[3]

x,y,z bias vector data

x,y,z bias vector data (for uncal sensor variant)

8.47.2.4 int32_t inv_sensor_event::cumulativeEEkcal

cumulative energy expenditure since the last reset in kilocalorie.

Format is q0: 1 = 1 kcal

8.47.2.5 int32_t inv_sensor_event::cumulativeEEmets

cumulative energy expenditure since the last reset in METs (Metabolic Equivalent of Task).

Format is q0: 1 = 1 METs

8.47.2.6 struct { ... } inv_sensor_event::eis

EIS data.

Warning

experimental: structure is likely to change in near future

8.47.2.7 int inv_sensor_event::event

BAC extended data begin/end event as of enum inv_sensor_bac_ext_event.

BAC data begin/end event as of enum inv_sensor_bac_event.

8.47.2.8 int32_t inv_sensor_event::floorsDown

number of floors climbed Down on foot by user.

8.47.2.9 int32_t inv_sensor_event::floorsUp

number of floors climbed Up on foot by user.

8.47.2.10 struct { ... } inv_sensor_event::hrm

heart rate monitor data

heart rate monitor data

8.47.2.11 int32_t inv_sensor_event::instantEEkcal

energy expenditure in kilocalorie/min since last output.

Format is q15: 2^15 = 1 kcal/min

8.47.2.12 int32_t inv_sensor_event::instantEEmets

energy expenditure in METs(Metabolic Equivalent of Task) since last output.

Format is q15: 2^15 = 1 METs

8.47.2.13 uint8_t inv_sensor_event::touch_status

touch status, detected or not by the PPG

touch status, detected or not

The documentation for this struct was generated from the following file:

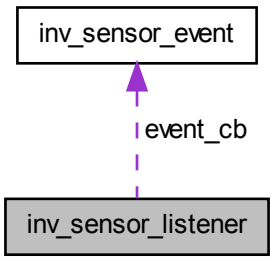
- SensorTypes.h

8.48 inv_sensor_listener Struct Reference

Sensor event listener definition.

`#include <SensorTypes.h>`

Collaboration diagram for inv_sensor_listener:



Public Attributes

- inv_sensor_listener_event_cb_t event_cb
  sensor event callback
- void * context
  listener context

8.48.1 Detailed Description

Sensor event listener definition.

Examples:

ExampleDeviceIcm20602EMD.c, ExampleDeviceIcm20603EMD.c, and ExampleDeviceIcm20690EMD.c.

The documentation for this struct was generated from the following file:

- SensorTypes.h

Chapter 9

Example Documentation

9.1 ExampleDeviceIcm20602EMD.c

This example shows how to use the Device API for ICM20602 device in embedded context

```
/*
 * ________________________________________________________________________________________________________
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 * This software, related documentation and any modifications thereto (collectively “Software”) is subject
 * to InvenSense and its licensors’ intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
       Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
       agreement
 * from InvenSense is strictly prohibited.
 *
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
 * PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
 * INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
 * NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THE SOFTWARE.
 * ________________________________________________________________________________________________________
 */

#include "Invn/Devices/DeviceIcm20602.h"

/*
 * Sleep implementation for Icm20602.
 */
void inv_icm20602_sleep(int ms)
{
        /*
         * You may provide a sleep function that blocks the current programm
         * execution for the specified amount of ms
         */
        (void)ms;
}

/*
 * High resolution sleep implementation for Icm20602.
 * Used at initilization stage. ~100us is sufficient.
 */
void inv_icm20602_sleep_us(int us)
{
        /*
         * You may provide a sleep function that blocks the current programm
         * execution for the specified amount of us
         */
        (void)us;
}

/*
```

```
 * Time implementation for Icm20602.
 */
uint64_t inv_icm20602_get_time_us(void)
{
        /*
         * You may provide a time function that return a monotonic timestamp in us
         */
        return 0;
}

/*
 * Callback called upon sensor event reception
 * This function is called in the same function than inv_device_poll()
 */
static void sensor_event_cb(const inv_sensor_event_t * event, void * arg)
{
        /* arg will contained the value provided at init time */
        (void)arg;

        (void)event;
        /* ... do something with event */
}


/*
 * A listener onject will handle sensor events
 */
static inv_sensor_listener_t sensor_listener = {
        sensor_event_cb, /* callback that will receive sensor events */
        (void *)0xDEAD   /* some pointer passed to the callback */
};


/*
 * States for icm20602 device object
 */
static inv_device_icm20602_t device_icm20602;

/*
 * serif_hal object that abstract low level serial interface between host and device
 */
extern int my_serif_read_reg(void * context, uint8_t reg, uint8_t * data, uint32_t len);
extern int my_serif_write_reg(void * context, uint8_t reg, const uint8_t * data, uint32_t len);

const inv_serif_hal_t serif_instance = {
    my_serif_read_reg,      /* user read_register() function that reads a register over the serial
       interface */
    my_serif_write_reg,     /* user write_register() function that writes a register over the serial
       interface */
    128,                    /* maximum number of bytes allowed per read transaction */
    128,                    /* maximum number of bytes allowed per write transaction */
    INV_SERIF_HAL_TYPE_SPI, /* type of the serial interface used between SPI or I2C */
    (void *)0xDEADBEEF      /* some context pointer passed to read/write callbacks */
};

/*
 * Example main function
 */
int example(void);
int example(void)
{
        int rc = 0;

        inv_device_t * device; /* just a handy variable to keep the handle to device object */
        uint8_t whoami;

        /*
         * Open serial interface (SPI or I2C) before playing with the device
         */
        /* call low level drive initialization here... */

        /*
         * Create ICM20602 Device
         * Pass to the driver:
         * - reference to serial interface object,
         * - reference to listener that will catch sensor events,
         */
        inv_device_icm20602_init2(&device_icm20602, &serif_instance, &
      sensor_listener);

        /*
         * Simply get generic device handle from Icm20602 Device
         */
        device = inv_device_icm20602_get_base(&device_icm20602);

        /*
         * Just get the whoami
```

```
         */
        rc += inv_device_whoami(device, &whoami);
        /* ... do something with whoami */

        /*
         * Configure and initialize the Icm20602 device
         */
        rc += inv_device_setup(device);


        /*
         * Now that device is ready, you must call inv_device_poll() function
         * periodically or upon interrupt.
         * The poll function will check for sensor events, and notify, if any,
         * by means of the callback from the listener that was provided on device init.
         */
        rc += inv_device_poll(device);

        /*
         * Check if Accelerometer is available sensor
         * if rc value is 0, it means sensor is available,
         * if rc value is INV_ERROR or INV_ERROR_BAD_ARG, sensor is NA
         */
        rc += inv_device_ping_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_ping_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /* ... */

        /*
         * Start RAW accelerometer and gyroscope sensor at 20 Hz
         */
        rc += inv_device_set_sensor_period(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER, 50);
        rc += inv_device_start_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_set_sensor_period(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE, 50);
        rc += inv_device_start_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /* ... */

        /*
         * Stop accelerometer sensor
         */
        rc += inv_device_stop_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_stop_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /*
         * Shutdown everything.
         */
        rc += inv_device_cleanup(device);

        /*
         * Close serial interface link
         */
        /* call low level drive de-initialization here... */

        return rc;
}
```

9.2 ExampleDeviceIcm20603EMD.c

This example shows how to use the Device API for ICM20603 device in embedded context

```
/*
 * ________________________________________________________________________________________________________
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 * This software, related documentation and any modifications thereto (collectively “Software”) is subject
 * to InvenSense and its licensors’ intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
       Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
       agreement
```

```
 * from InvenSense is strictly prohibited.
 *
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
 * PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
 * INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
 * NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THE SOFTWARE.
 * ____________________________________________________________________________________________________
 */

#include "Invn/Devices/DeviceIcm20603.h"

/*
 * Sleep implementation for Icm20603.
 */
void inv_icm20603_sleep(int ms)
{
        /*
         * You may provide a sleep function that blocks the current programm
         * execution for the specified amount of ms
         */
        (void)ms;
}

/*
 * High resolution sleep implementation for Icm20603.
 * Used at initilization stage. ~100us is sufficient.
 */
void inv_icm20603_sleep_us(int us)
{
        /*
         * You may provide a sleep function that blocks the current programm
         * execution for the specified amount of us
         */
        (void)us;
}

/*
 * Time implementation for Icm20603.
 */
uint64_t inv_icm20603_get_time_us(void)
{
        /*
         * You may provide a time function that return a monotonic timestamp in us
         */
        return 0;
}

/*
 * Callback called upon sensor event reception
 * This function is called in the same function than inv_device_poll()
 */
static void sensor_event_cb(const inv_sensor_event_t * event, void * arg)
{
        /* arg will contained the value provided at init time */
        (void)arg;

        (void)event;
        /* ... do something with event */
}


/*
 * A listener onject will handle sensor events
 */
static inv_sensor_listener_t sensor_listener = {
        sensor_event_cb, /* callback that will receive sensor events */
        (void *)0xDEAD   /* some pointer passed to the callback */
};


/*
 * States for icm20603 device object
 */
static inv_device_icm20603_t device_icm20603;

/*
 * serif_hal object that abstract low level serial interface between host and device
 */
extern int my_serif_read_reg(void * context, uint8_t reg, uint8_t * data, uint32_t len);
extern int my_serif_write_reg(void * context, uint8_t reg, const uint8_t * data, uint32_t len);

const inv_serif_hal_t serif_instance = {
    my_serif_read_reg,      /* user read_register() function that reads a register over the serial
```

```
       interface */
    my_serif_write_reg,     /* user write_register() function that writes a register over the serial
       interface */
    128,                    /* maximum number of bytes allowed per read transaction */
    128,                    /* maximum number of bytes allowed per write transaction */
    INV_SERIF_HAL_TYPE_SPI, /* type of the serial interface used between SPI or I2C */
    (void *)0xDEADBEEF      /* some context pointer passed to read/write callbacks */
};

/*
 * Example main function
 */
int example(void);
int example(void)
{
        int rc = 0;

        inv_device_t * device; /* just a handy variable to keep the handle to device object */
        uint8_t whoami;

        /*
         * Open serial interface (SPI or I2C) before playing with the device
         */
        /* call low level drive initialization here... */

        /*
         * Create ICM20603 Device
         * Pass to the driver:
         * - reference to serial interface object,
         * - reference to listener that will catch sensor events,
         */
        inv_device_icm20603_init2(&device_icm20603, &serif_instance, &
      sensor_listener);

        /*
         * Simply get generic device handle from Icm20603 Device
         */
        device = inv_device_icm20603_get_base(&device_icm20603);

        /*
         * Just get the whoami
         */
        rc += inv_device_whoami(device, &whoami);
        /* ... do something with whoami */

        /*
         * Configure and initialize the Icm20603 device
         */
        rc += inv_device_setup(device);


        /*
         * Now that device is ready, you must call inv_device_poll() function
         * periodically or upon interrupt.
         * The poll function will check for sensor events, and notify, if any,
         * by means of the callback from the listener that was provided on device init.
         */
        rc += inv_device_poll(device);

        /*
         * Check if Accelerometer is available sensor
         * if rc value is 0, it means sensor is available,
         * if rc value is INV_ERROR or INV_ERROR_BAD_ARG, sensor is NA
         */
        rc += inv_device_ping_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_ping_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /* ... */

        /*
         * Start RAW accelerometer and gyroscope sensor at 20 Hz
         */
        rc += inv_device_set_sensor_period(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER, 50);
        rc += inv_device_start_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_set_sensor_period(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE, 50);
        rc += inv_device_start_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /* ... */

        /*
         * Stop accelerometer sensor
```

```
         */
        rc += inv_device_stop_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_stop_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /*
         * Shutdown everything.
         */
        rc += inv_device_cleanup(device);

        /*
         * Close serial interface link
         */
        /* call low level drive de-initialization here... */

        return rc;
}
```

9.3 ExampleDeviceIcm20690EMD.c

This example shows how to use the Device API for ICM20690 device in embedded context

```
/*
 * ________________________________________________________________________________________________________
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 * This software, related documentation and any modifications thereto (collectively “Software”) is subject
 * to InvenSense and its licensors’ intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
      Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
      agreement
 * from InvenSense is strictly prohibited.
 *
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
 * PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
 * INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
 * NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THE SOFTWARE.
 * ________________________________________________________________________________________________________
 */

#include "Invn/Devices/DeviceIcm20690.h"

/*
 * Sleep implementation for Icm20690.
 */
void inv_icm20690_sleep(int ms)
{
        /*
         * You may provide a sleep function that blocks the current programm
         * execution for the specified amount of ms
         */
        (void)ms;
}

/*
 * High resolution sleep implementation for Icm20690.
 * Used at initilization stage. ~100us is sufficient.
 */
void inv_icm20690_sleep_us(int us)
{
        /*
         * You may provide a sleep function that blocks the current programm
         * execution for the specified amount of us
         */
        (void)us;
}

/*
 * Time implementation for Icm20690.
 */
uint64_t inv_icm20690_get_time_us(void)
{
        /*
```

```
         * You may provide a time function that return a monotonic timestamp in us
         */
        return 0;
}

/*
 * Callback called upon sensor event reception
 * This function is called in the same function than inv_device_poll()
 */
static void sensor_event_cb(const inv_sensor_event_t * event, void * arg)
{
        /* arg will contained the value provided at init time */
        (void)arg;

        (void)event;
        /* ... do something with event */
}


/*
 * A listener onject will handle sensor events
 */
static inv_sensor_listener_t sensor_listener = {
        sensor_event_cb, /* callback that will receive sensor events */
        (void *)0xDEAD   /* some pointer passed to the callback */
};


/*
 * States for icm20690 device object
 */
static inv_device_icm20690_t device_icm20690;

/*
 * serif_hal object that abstract low level serial interface between host and device
 */
extern int my_serif_read_reg(void * context, uint8_t reg, uint8_t * data, uint32_t len);
extern int my_serif_write_reg(void * context, uint8_t reg, const uint8_t * data, uint32_t len);

const inv_serif_hal_t serif_instance = {
        my_serif_read_reg,      /* user read_register() function that reads a register over the serial
       interface */
        my_serif_write_reg,     /* user write_register() function that writes a register over the serial
       interface */
        128,                    /* maximum number of bytes allowed per read transaction */
        128,                    /* maximum number of bytes allowed per write transaction */
        INV_SERIF_HAL_TYPE_SPI, /* type of the serial interface used between SPI or I2C */
        (void *)0xDEADBEEF      /* some context pointer passed to read/write callbacks */
};

/*
 * Example main function
 */
int example(void);
int example(void)
{
        int rc = 0;

        inv_device_t * device; /* just a handy variable to keep the handle to device object */
        uint8_t whoami;

        /*
         * Open serial interface (SPI or I2C) before playing with the device
         */
        /* call low level drive initialization here... */

        /*
         * Create ICM20690 Device
         * Pass to the driver:
         * - reference to serial interface object,
         * - reference to listener that will catch sensor events,
         */
        inv_device_icm20690_init2(&device_icm20690, &serif_instance, &
      sensor_listener);

        /*
         * Simply get generic device handle from Icm20690 Device
         */
        device = inv_device_icm20690_get_base(&device_icm20690);

        /*
         * Just get the whoami
         */
        rc += inv_device_whoami(device, &whoami);
        /* ... do something with whoami */

        /*
```

```
         * Configure and initialize the Icm20690 device
         */
        rc += inv_device_setup(device);


        /*
         * Now that device is ready, you must call inv_device_poll() function
         * periodically or upon interrupt.
         * The poll function will check for sensor events, and notify, if any,
         * by means of the callback from the listener that was provided on device init.
         */
        rc += inv_device_poll(device);

        /*
         * Check if Accelerometer is available sensor
         * if rc value is 0, it means sensor is available,
         * if rc value is INV_ERROR or INV_ERROR_BAD_ARG, sensor is NA
         */
        rc += inv_device_ping_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_ping_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /* ... */

        /*
         * Start RAW accelerometer and gyroscope sensor at 20 Hz
         */
        rc += inv_device_set_sensor_period(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER, 50);
        rc += inv_device_start_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_set_sensor_period(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE, 50);
        rc += inv_device_start_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /* ... */

        /*
         * Stop accelerometer sensor
         */
        rc += inv_device_stop_sensor(device,
      INV_SENSOR_TYPE_RAW_ACCELEROMETER);
        rc += inv_device_stop_sensor(device,
      INV_SENSOR_TYPE_RAW_GYROSCOPE);

        /*
         * Shutdown everything.
         */
        rc += inv_device_cleanup(device);

        /*
         * Close serial interface link
         */
        /* call low level drive de-initialization here... */

        return rc;
}
```

9.4 ExampleSerifHal.c

Basic template for SerifHal implementation.

```
/*
 * ________________________________________________________________________________________________________
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 *
 * This software, related documentation and any modifications thereto (collectively “Software”) is subject
 * to InvenSense and its licensors’ intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
      Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
      agreement
 * from InvenSense is strictly prohibited.
 *
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
 * PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
```

```
 * TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
 * INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
 * NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THE SOFTWARE.
 * ____________________________________________________________________________________________________
 */

#include "Invn/Devices/SerifHal.h"

// include to low level system driver
// #include "MyTarget/SPI.h"

// forward declarations
int my_serif_open_read_reg(void * context, uint8_t reg, uint8_t * rbuffer, uint32_t rlen);
int my_serif_open_write_reg(void * context, uint8_t reg, const uint8_t * wbuffer, uint32_t wlen);

// Exported instance of SerifHal object:
// A pointer to this structure shall be passed to the Device API,
// for the device driver to access to the SPI/I2C bus.
// The device will not modify the object, so it can be declared const
// The underlying HW serial interface must be up and running before calling any
// device methods
const inv_serif_hal_t my_serif_instance = {
        my_serif_open_read_reg,  /* callback to read_reg low level method */
        my_serif_open_write_reg, /* callback to read_reg low level method */
        128,                     /* maximum number of bytes allowed per read transaction,
                                    (limitation can come from internal buffer in the system driver) */
        128,                     /* maximum number of bytes allowed per write transaction,
                                    (limitation can come from internal buffer in the system driver) */
        INV_SERIF_HAL_TYPE_SPI,  /* type of the serial interface (between SPI or I2C) */
        (void *)0xDEAD           /* some context pointer passed to read_reg/write_reg callbacks */
};


int my_serif_open_read_reg(void * context, uint8_t reg, uint8_t * rbuffer, uint32_t rlen)
{
        (void)context, (void)reg, (void)rbuffer, (void)rlen;
        // MyTarget_SPI_do_read_reg(&reg, 1, rbuffer, rlen);
        return 0; // shall return a negative value on error
}

int my_serif_open_write_reg(void * context, uint8_t reg, const uint8_t * wbuffer, uint32_t wlen)
{
        (void)context, (void)reg, (void)wbuffer, (void)wlen;
        // MyTarget_SPI_do_write_reg(&reg, 1, wbuffer, wlen);
        return 0; // shall return a negative value on error
}
```

Index