InvenSense Device Driver library

3.8.9

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.

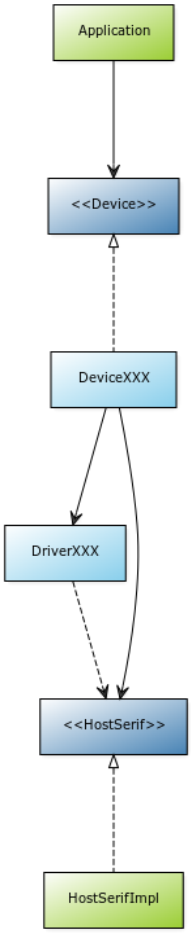


Figure 1.1 High level class diagram of libIDD

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 :

- ExampleDeviceIcm20648EMD.c
- ExampleDeviceIcm20948EMD.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 ICM20648, a 6-axis based MEMS embedding a dmp, Android L sensors are supported and auxiliary.
- The ICM20948, a 6-axis based MEMS embedding a dmp and an akm9916 magnetometer. Android L sensors are supported and auxiliary.

3.1 ICM20x48

The ICM20648 and ICM20948 gives you access to all Android L sensors.

Primary interface:

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

For an example on how to use ICM20648 device with libIDD please see ExampleDeviceIcm20648EMD.c. For an example on how to use ICM20948 device with libIDD please see ExampleDeviceIcm20948EMD.c.

Chapter 4

Deprecated List

Module HostSerif
Use SerifHal.h instead

Member inv_device_enable
use inv_device_enable_sensor

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.

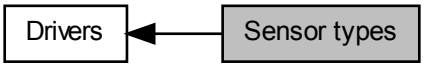
Enumerator

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.

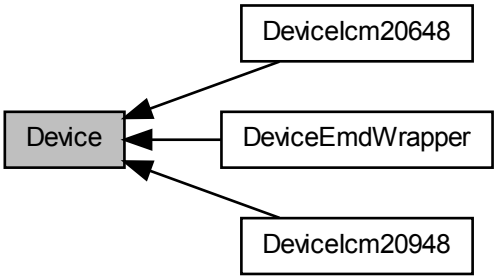
INV_SENSOR_CONFIG_CUSTOM Configuration ID above this value are device specific.

INV_SENSOR_CONFIG_MAX Absolute maximum value for sensor config.

7.4 Device

Abstract device interface definition.

Collaboration diagram for Device:



Modules

- DeviceEmdWrapper

  emd wrapper implementation for device interface
- DeviceIcm20648

  Concrete implementation of the 'Device' interface for Icm20648 devices.
- DeviceIcm20948

  Concrete implementation of the 'Device' interface for Icm20948 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:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.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

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:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.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:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.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:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.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:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.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:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.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:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.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 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

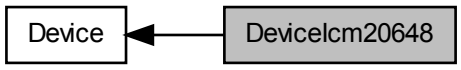
7.5.1 Detailed Description

emd wrapper implementation for device interface

7.6 DeviceIcm20648

Concrete implementation of the 'Device' interface for Icm20648 devices.

Collaboration diagram for DeviceIcm20648:



Classes

- struct inv_device_icm20648
  States for Icm20648 device.

Typedefs

- typedef struct inv_device_icm20648 inv_device_icm20648_t
  States for Icm20648 device.

Enumerations

Functions

- static struct inv_icm20648 * inv_device_icm20648_get_driver_handle (inv_device_icm20648_t *self)
  Return handle to underlying driver states.
- void INV_EXPORT inv_device_icm20648_init (inv_device_icm20648_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener, const uint8_t *dmp3_image, uint32_t dmp3_image_size)
  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20648_init2 (inv_device_icm20648_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener, const uint8_t *dmp3_image, uint32_t dmp3_image_size)
  constructor-like function for Icm20648 device
- static inv_device_t * inv_device_icm20648_get_base (inv_device_icm20648_t *self)
  Helper function to get handle to base object.

7.6.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20648 devices.

See ExampleDeviceIcm20648.c example.

7.6.2 Function Documentation

7.6.2.1 static struct inv_icm20648* inv_device_icm20648_get_driver_handle (inv_device_icm20648_t * *self*) [static]

Return handle to underlying driver states.

Parameters

in	*self*	handle to device

Returns

pointer to underlying driver states

7.6.2.2 void INV_EXPORT inv_device_icm20648_init (inv_device_icm20648_t * *self*, const inv_host_serif_t * *serif*, const inv_sensor_listener_t * *listener*, const uint8_t * *dmp3_image*, uint32_t *dmp3_image_size*)

constructor-like function for basesensor device

Will initialize inv_device_icm20648_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

7.6.2.3 void INV_EXPORT inv_device_icm20648_init2 (inv_device_icm20648_t * *self*, const inv_serif_hal_t * *serif*, const inv_sensor_listener_t * *listener*, const uint8_t * *dmp3_image*, uint32_t *dmp3_image_size*)

constructor-like function for Icm20648 device

Will initialize inv_device_icm20648_t object states to default value for ICM20648.

Parameters

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

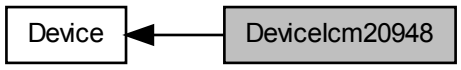
Examples:

ExampleDeviceIcm20648EMD.c.

7.7 DeviceIcm20948

Concrete implementation of the 'Device' interface for Icm20948 devices.

Collaboration diagram for DeviceIcm20948:



Classes

- struct inv_device_icm20948
  States for Icm20948 device.

Typedefs

- typedef struct inv_device_icm20948 inv_device_icm20948_t
  States for Icm20948 device.

Enumerations

Functions

- static struct inv_icm20948 * inv_device_icm20948_get_driver_handle (inv_device_icm20948_t *self)
  Return handle to underlying driver states.
- void INV_EXPORT inv_device_icm20948_init (inv_device_icm20948_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener, const uint8_t *dmp3_image, uint32_t dmp3_image_size)
  constructor-like function for basesensor device
- void INV_EXPORT inv_device_icm20948_init2 (inv_device_icm20948_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener, const uint8_t *dmp3_image, uint32_t dmp3_image_size)
  constructor-like function for Icm20948 device
- static inv_device_t * inv_device_icm20948_get_base (inv_device_icm20948_t *self)
  Helper function to get handle to base object.

7.7.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20948 devices.

See ExampleDeviceIcm20948.c example.

7.7.2 Function Documentation

7.7.2.1 static struct inv_icm20948* inv_device_icm20948_get_driver_handle (inv_device_icm20948_t * *self*) [static]

Return handle to underlying driver states.

Parameters

in	*self*	handle to device

Returns

pointer to underlying driver states

7.7.2.2 void INV_EXPORT inv_device_icm20948_init (inv_device_icm20948_t * *self*, const inv_host_serif_t * *serif*, const inv_sensor_listener_t * *listener*, const uint8_t * *dmp3_image*, uint32_t *dmp3_image_size*)

constructor-like function for basesensor device

Will initialize inv_device_icm20948_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

7.7.2.3 void INV_EXPORT inv_device_icm20948_init2 (inv_device_icm20948_t * *self*, const inv_serif_hal_t * *serif*, const inv_sensor_listener_t * *listener*, const uint8_t * *dmp3_image*, uint32_t *dmp3_image_size*)

constructor-like function for Icm20948 device

Will initialize inv_device_icm20948_t object states to default value for ICM20948.

Parameters

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

Examples:

ExampleDeviceIcm20948EMD.c.

7.8 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.

7.8.1 Detailed Description

Virtual abstraction of host adapter for serial interface.

Deprecated Use SerifHal.h instead

7.9 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.

7.9.1 Detailed Description

Helper functions to convert integer.

7.9.2 Function Documentation

7.9.2.1 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

7.9.2.2 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

7.10 Error Helper

Helper functions realted to error code.

Functions

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

7.10.1 Detailed Description

Helper functions realted to error code.

7.10.2 Function Documentation

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

Returns string describing error number.

See also

enum inv_error

7.11 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)

7.11.1 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.

7.11.2 Macro Definition Documentation

7.11.2.1 #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

7.11.2.2 #define INV_MSG_LEVEL _INV_MSG_LEVEL

Return current level.

Warning

This macro may expand as a function call

7.11.3 Function Documentation

7.11.3.1 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

7.11.3.2 int INV_EXPORT inv_msg_get_level (void)

Return current message level.

Returns

current message level

7.11.3.3 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

7.11.3.4 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

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

Set default message level and printer.

Returns

none

7.11.3.6 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

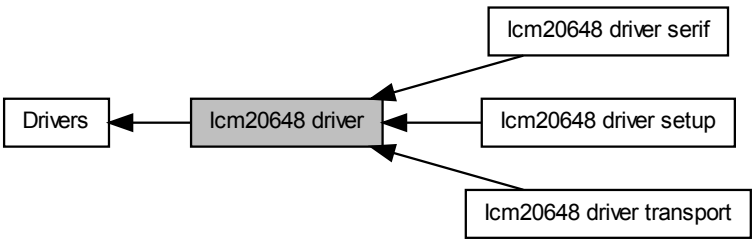
Returns

none

7.12 Icm20648 driver

Low-level driver for ICM20648 devices.

Collaboration diagram for Icm20648 driver:



Modules

- Icm20648 driver serif

  Interface for low-level serial (I2C/SPI) access.
- Icm20648 driver setup

  Low-level function to setup an Icm20648 device.
- Icm20648 driver transport

  Low-level ICM20648 register access.

Classes

- struct sensor_type_icm20648

  ICM20648 driver states definition.
- struct inv_icm20648

Typedefs

- typedef enum inv_icm20648_compass_state inv_icm20648_compass_state_t

  States for the secondary device.
- typedef struct sensor_type_icm20648 sensor_type_icm20648_t

  ICM20648 driver states definition.

Enumerations

Functions

- void inv_icm20648_sleep_us (int us)

  Hook for low-level system sleep() function to be implemented by upper layer.
- uint64_t inv_icm20648_get_time_us (void)

  Hook for low-level system time() function to be implemented by upper layer.
- static void inv_icm20648_reset_states (struct inv_icm20648 *s, const struct inv_icm20648_serif *serif)

  Reset and initialize driver states.

Variables

- struct inv_icm20648 * icm20648_instance

  ICM20648 driver states singleton declaration Because of Low-level driver limitation only one insance of the driver is allowed.

7.12.1 Detailed Description

Low-level driver for ICM20648 devices.

7.12.2 Function Documentation

7.12.2.1 uint64_t inv_icm20648_get_time_us (void)

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

Returns

monotonic timestamp in us

7.12.2.2 static void inv_icm20648_reset_states (struct inv_icm20648 * *s*, const struct inv_icm20648_serif * *serif*) [inline], [static]

Reset and initialize driver states.

Parameters

in	*s*	handle to driver states structure

7.12.2.3 void inv_icm20648_sleep_us (int *us*)

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

Parameters

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

7.13 augmented_sensors

Functions

- int INV_EXPORT inv_icm20648_augmented_init (struct inv_icm20648 *s)

  Initialize structure values.
- int INV_EXPORT inv_icm20648_augmented_sensors_get_gravity (struct inv_icm20648 *s, long gravity[3], const long quat6axis_3e[3])

  Gets the 3 axis gravity value based on GRV quaternion.
- int INV_EXPORT inv_icm20648_augmented_sensors_get_linearacceleration (long linacc[3], const long gravity[3], const long accel[3])

  Gets the 3 axis linear acceleration value based on gravity and accelerometer values.
- int INV_EXPORT inv_icm20648_augmented_sensors_get_orientation (long orientation[3], const long quat9axis_3e[4])

  Gets the 3 axis orientation value based on RV quaternion.
- unsigned short INV_EXPORT inv_icm20648_augmented_sensors_set_odr (struct inv_icm20648 *s, unsigned char androidSensor, unsigned short delayInMs)

  Set ODR for one of the augmented sensor-related Android sensor.
- void INV_EXPORT inv_icm20648_augmented_sensors_update_odr (struct inv_icm20648 *s, unsigned char androidSensor, unsigned short *updatedDelayPtr)

  Update ODR when an augmented sensor-related Android sensor was enabled or disable with ODR unchanged.

7.13.1 Detailed Description

7.13.2 Function Documentation

7.13.2.1 int INV_EXPORT inv_icm20648_augmented_init (struct inv_icm20648 * *s*)

Initialize structure values.

Parameters

in	*base*	state structre

7.13.2.2 int INV_EXPORT inv_icm20648_augmented_sensors_get_gravity (struct inv_icm20648 * *s*, long *gravity[3]*, const long *quat6axis_3e[3]*)

Gets the 3 axis gravity value based on GRV quaternion.

Parameters

out	*gravity*	3 components resulting gravity in Q16 in m/s2
in	*quat*	3 components input AG-based quaternion in Q30

Returns

0 in case of success, -1 for any error

7.13.2.3 int INV_EXPORT inv_icm20648_augmented_sensors_get_linearacceleration (long *linacc[3],* const long *gravity[3],* const long *accel[3]*)

Gets the 3 axis linear acceleration value based on gravity and accelerometer values.

Parameters

out	*linacc*	3 components resulting linear acceleration in Q16 in m/s2
in	*gravity*	3 components gravity in Q16 in m/s2
in	*accel*	3 components acceleration in Q16 in m/s2

Returns

0 in case of success, -1 for any error

7.13.2.4 int INV_EXPORT inv_icm20648_augmented_sensors_get_orientation (long *orientation[3],* const long *quat9axis_3e[4]*)

Gets the 3 axis orientation value based on RV quaternion.

Parameters

out	*orientation*	3 components resulting orientation in Q16 in degrees The x field is azimuth, the angle between the magnetic north direction and the y axis around the the z axis. The y field is pitch, the rotation arounf x axis, positive when the z axis moves toward the y axis. The z field is roll, the rotation arount the y axis, positive when the x axis moves toward the z axis.
in	*quat9axis_3e*	3 components input AGM-based quaternion in Q30

Returns

0 in case of success, -1 for any error

7.13.2.5 unsigned short INV_EXPORT inv_icm20648_augmented_sensors_set_odr (struct inv_icm20648 * *s,* unsigned char *androidSensor,* unsigned short *delayInMs*)

Set ODR for one of the augmented sensor-related Android sensor.

Parameters

in	*androidSensor*	Android sensor ID for which a new delay in to be applied
in	*delayInMs*	the new delay in ms requested for androidSensor

Returns

the delay in ms to be applied to quat6 output

7.13.2.6 void INV_EXPORT inv_icm20648_augmented_sensors_update_odr (struct inv_icm20648 * *s*, unsigned char *androidSensor*, unsigned short * *updatedDelayPtr*)

Update ODR when an augmented sensor-related Android sensor was enabled or disable with ODR unchanged.

Parameters

in	*androidSensor*	Android sensor ID for which status was updated
out	*updatedDelayPtr*	Handler where should be written new delay to be applied

Returns

None

7.14 inv_slave_compass

Enumerations

Functions

- void INV_EXPORT inv_icm20648_register_aux_compass (struct inv_icm20648 *s, enum inv_icm20648_compass_id compass_id, uint8_t compass_i2c_addr)

  Register AUX compass.
- int INV_EXPORT inv_icm20648_setup_compass_akm (struct inv_icm20648 *s)

  Initializes the compass.
- int INV_EXPORT inv_icm20648_check_akm_self_test (struct inv_icm20648 *s)

  Self test for the compass.
- int INV_EXPORT inv_icm20648_write_akm_scale (struct inv_icm20648 *s, int data)

  Changes the scale of the compass.
- int INV_EXPORT inv_icm20648_read_akm_scale (struct inv_icm20648 *s, int *scale)

  Reads the scale of the compass.
- int INV_EXPORT inv_icm20648_suspend_akm (struct inv_icm20648 *s)

  Stops the compass.
- int INV_EXPORT inv_icm20648_resume_akm (struct inv_icm20648 *s)

  Starts the compass.
- char INV_EXPORT inv_icm20648_compass_getstate (struct inv_icm20648 *s)

  Get compass power status.
- int INV_EXPORT inv_icm20648_compass_isconnected (struct inv_icm20648 *s)

  detects if the compass is connected
- int INV_EXPORT inv_icm20648_compass_dmp_cal (struct inv_icm20648 *s, const signed char *m, const signed char *compass_m)

  Calibrates the data.
- int INV_EXPORT inv_icm20648_apply_raw_compass_matrix (struct inv_icm20648 *s, short *raw_data, long *compensated_out)

  Applies mounting matrix and scaling to raw compass data.

7.14.1 Detailed Description

7.14.2 Enumeration Type Documentation

7.14.2.1 enum inv_icm20648_compass_id

Supported auxiliary compass identifer.

Enumerator

INV_ICM20648_COMPASS_ID_NONE	no compass
INV_ICM20648_COMPASS_ID_AK09911	AKM AK09911.
INV_ICM20648_COMPASS_ID_AK09912	AKM AK09912.
INV_ICM20648_COMPASS_ID_AK09916	AKM AK09916.
INV_ICM20648_COMPASS_ID_AK08963	AKM AK08963.

7.14.3 Function Documentation

7.14.3.1 int INV_EXPORT inv_icm20648_apply_raw_compass_matrix (struct inv_icm20648 * *s*, short * *raw_data*, long * *compensated_out*)

Applies mounting matrix and scaling to raw compass data.

Parameters

in	*raw_data*	Raw compass data
in	*compensated_out*	Compensated compass data

Returns

0 in case of success, -1 for any error

7.14.3.2 int INV_EXPORT inv_icm20648_check_akm_self_test (struct inv_icm20648 * *s*)

Self test for the compass.

Returns

0 in case of success, -1 for any error

7.14.3.3 int INV_EXPORT inv_icm20648_compass_dmp_cal (struct inv_icm20648 * *s*, const signed char * *m*, const signed char * *compass_m*)

Calibrates the data.

Parameters

in	*m*	pointer to the raw compass data
out	*compass_m*	pointer to the calibrated compass data

Returns

0 in case of success, -1 for any error

7.14.3.4 char INV_EXPORT inv_icm20648_compass_getstate (struct inv_icm20648 * *s*)

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

7.14.3.5 int INV_EXPORT inv_icm20648_compass_isconnected (struct inv_icm20648 * *s*)

detects if the compass is connected

Returns

1 if the compass is connected, 0 otherwise

7.14.3.6 int INV_EXPORT inv_icm20648_read_akm_scale (struct inv_icm20648 * 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

7.14.3.7 void INV_EXPORT inv_icm20648_register_aux_compass (struct inv_icm20648 * s, enum inv_icm20648_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_icm20648_↩ initialize().

Parameters

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

Returns

0 on success, negative value on error

7.14.3.8 int INV_EXPORT inv_icm20648_resume_akm (struct inv_icm20648 * s)

Starts the compass.

Returns

0 in case of success, -1 for any error

7.14.3.9 int INV_EXPORT inv_icm20648_setup_compass_akm (struct inv_icm20648 * s)

Initializes the compass.

Returns

0 in case of success, -1 for any error

7.14.3.10 int INV_EXPORT inv_icm20648_suspend_akm (struct inv_icm20648 * *s*)

Stops the compass.

Returns

0 in case of success, -1 for any error

7.14.3.11 int INV_EXPORT inv_icm20648_write_akm_scale (struct inv_icm20648 * *s*, int *data*)

Changes the scale of the compass.

Parameters

in	*data*	new scale for the compass

Returns

0 in case of success, -1 for any error

7.15 inv_secondary_transport

Macros

- #define COMPASS_I2C_SLV_READ 0

  I2C from secondary device can stand on up to 4 channels.

Functions

- void INV_EXPORT inv_icm20648_init_secondary (struct inv_icm20648 *s)

  Initializes the register for the i2c communication.
- int INV_EXPORT inv_icm20648_read_secondary (struct inv_icm20648 *s, int index, unsigned char addr, unsigned char reg, char len)

  Reads data in i2c a secondary device.
- int INV_EXPORT inv_icm20648_execute_read_secondary (struct inv_icm20648 *s, int index, unsigned char addr, int reg, int len, uint8_t *d)

  Reads data in i2c a secondary device directly.
- int INV_EXPORT inv_icm20648_write_secondary (struct inv_icm20648 *s, int index, unsigned char addr, unsigned char reg, char v)

  Writes data in i2c a secondary device.
- int INV_EXPORT inv_icm20648_execute_write_secondary (struct inv_icm20648 *s, int index, unsigned char addr, int reg, uint8_t v)

  Writes data in i2c a secondary device directly.
- void INV_EXPORT inv_icm20648_secondary_saveI2cOdr (struct inv_icm20648 *s)

  Save current secondary I2C ODR configured.
- void INV_EXPORT inv_icm20648_secondary_restoreI2cOdr (struct inv_icm20648 *s)

  Restore secondary I2C ODR configured based on the one saved with inv_icm20648_secondary_saveI2cOdr()
- int INV_EXPORT inv_icm20648_secondary_stop_channel (struct inv_icm20648 *s, int index)

  Stop one secondary I2C channel by writing 0 in its control register.
- int INV_EXPORT inv_icm20648_secondary_enable_i2c (struct inv_icm20648 *s)

  Enable secondary I2C interface.
- int INV_EXPORT inv_icm20648_secondary_disable_i2c (struct inv_icm20648 *s)

  Stop secondary I2C interface.
- int INV_EXPORT inv_icm20648_secondary_set_odr (struct inv_icm20648 *s, int divider, unsigned int *effectiveDivider)

  Changes the odr of the I2C master.

7.15.1 Detailed Description

7.15.2 Macro Definition Documentation

7.15.2.1 #define COMPASS_I2C_SLV_READ 0

I2C from secondary device can stand on up to 4 channels.

To perform automatic read and feed DMP :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register
- channel 2 is reserved for als reading data

7.15.3 Function Documentation

7.15.3.1 int INV_EXPORT inv_icm20648_execute_read_secondary (struct inv_icm20648 * s, int index, unsigned char addr, int reg, int len, uint8_t * 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

7.15.3.2 int INV_EXPORT inv_icm20648_execute_write_secondary (struct inv_icm20648 * s, int index, unsigned char addr, int reg, uint8_t 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

7.15.3.3 int INV_EXPORT inv_icm20648_read_secondary (struct inv_icm20648 * 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

7.15.3.4 int INV_EXPORT inv_icm20648_secondary_disable_i2c (struct inv_icm20648 * *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

7.15.3.5 int INV_EXPORT inv_icm20648_secondary_enable_i2c (struct inv_icm20648 * *s*)

Enable secondary I2C interface.

Returns

0 in case of success, -1 for any error

7.15.3.6 int INV_EXPORT inv_icm20648_secondary_set_odr (struct inv_icm20648 * *s*, int *divider*, unsigned int * *effectiveDivider*)

Changes the odr of the I2C master.

Parameters

in	*divider*	frequency divider to BASE_SAMPLE_RATE
out	*effectiveDivider*	divider finally applied to base sample rate, at which data will be actually read on I2C bus

Returns

0 in case of success, -1 for any error

7.15.3.7 int INV_EXPORT inv_icm20648_secondary_stop_channel (struct inv_icm20648 * *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

7.15.3.8 int INV_EXPORT inv_icm20648_write_secondary (struct inv_icm20648 * *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

7.16 base_control

Enumerations

Functions

- int INV_EXPORT inv_icm20648_base_control_init (struct inv_icm20648 *s)

  Initialize structure values.
- int INV_EXPORT inv_icm20648_set_odr (struct inv_icm20648 *s, unsigned char androidSensor, unsigned short delayInMs)

  Sets the odr for a sensor.
- int INV_EXPORT inv_icm20648_ctrl_enable_sensor (struct inv_icm20648 *s, unsigned char androidSensor, unsigned char enable)

  Enables / disables a sensor.
- int INV_EXPORT inv_icm20648_ctrl_enable_batch (struct inv_icm20648 *s, unsigned char enable)

  Enables / disables batch for the sensors.
- void INV_EXPORT inv_icm20648_ctrl_set_batch_mode_status (struct inv_icm20648 *s, unsigned char enable)

  Set batch mode status.
- unsigned char INV_EXPORT inv_icm20648_ctrl_get_batch_mode_status (struct inv_icm20648 *s)

  Get batch mode status.
- int INV_EXPORT inv_icm20648_ctrl_set_batch_timeout (struct inv_icm20648 *s, unsigned short batch_↩ time_in_seconds)

  Sets the timeout for the batch in second.
- int INV_EXPORT inv_icm20648_ctrl_set_batch_timeout_ms (struct inv_icm20648 *s, unsigned short batch↩ _time_in_ms)

  Sets the timeout for the batch in millisecond.
- void INV_EXPORT inv_icm20648_ctrl_enable_activity_classifier (struct inv_icm20648 *s, unsigned char enable)

  Enables / disables BAC.
- void INV_EXPORT inv_icm20648_ctrl_enable_tilt (struct inv_icm20648 *s, unsigned char enable)

  Enables / disables tilt.
- void INV_EXPORT inv_icm20648_ctrl_enable_b2s (unsigned char enable)

  Enables / disables bring to see.
- unsigned long INV_EXPORT * inv_icm20648_ctrl_get_androidSensorsOn_mask (struct inv_icm20648 *s)

  Returns the mask for the different sensors enabled.
- unsigned long INV_EXPORT inv_icm20648_ctrl_androidSensor_enabled (struct inv_icm20648 *s, unsigned char androidSensor)

  Check if a sensor is enabled.
- unsigned short INV_EXPORT inv_icm20648_ctrl_get_activitiy_classifier_on_flag (struct inv_icm20648 *s)

  Returns a flag to know if the BAC is running.
- int INV_EXPORT inv_icm20648_ctrl_get_odr (struct inv_icm20648 *s, unsigned char SensorId, uint32_↩ t *odr, enum INV_ODR_TYPE odr_units)

  Gets the odr for a sensor.
- int INV_EXPORT inv_icm20648_ctrl_set_accel_quaternion_gain (struct inv_icm20648 *s, unsigned short hw_smplrt_divider)

  Sets accel quaternion gain according to accel engine rate.
- int INV_EXPORT inv_icm20648_ctrl_set_accel_cal_params (struct inv_icm20648 *s, unsigned short hw_↩ smplrt_divider)

  Sets accel cal parameters according to accel engine rate.
- int INV_EXPORT inv_icm20648_ctrl_enable_pickup (struct inv_icm20648 *s, unsigned char enable)

Enables / disables pickup gesture.

- int inv_icm20648_ctrl_get_acc_bias (struct inv_icm20648 *s, int *acc_bias)

  get acc bias from dmp driver
- int inv_icm20648_ctrl_get_gyr_bias (struct inv_icm20648 *s, int *gyr_bias)

  get gyr bias from dmp driver
- int INV_EXPORT inv_icm20648_ctrl_get_mag_bias (struct inv_icm20648 *s, int *mag_bias)

  get mag bias from dmp driver
- int inv_icm20648_ctrl_set_acc_bias (struct inv_icm20648 *s, int *acc_bias)

  set acc bias from dmp driver
- int inv_icm20648_ctrl_set_gyr_bias (struct inv_icm20648 *s, int *gyr_bias)

  set gyr bias from dmp driver
- int INV_EXPORT inv_icm20648_ctrl_set_mag_bias (struct inv_icm20648 *s, int *mag_bias)

  set mag bias from dmp driver

7.16.1 Detailed Description

7.16.2 Function Documentation

7.16.2.1 int INV_EXPORT inv_icm20648_base_control_init (struct inv_icm20648 * *s*)

Initialize structure values.

Parameters

in	*base*	state structre

7.16.2.2 unsigned long INV_EXPORT inv_icm20648_ctrl_androidSensor_enabled (struct inv_icm20648 * *s*, unsigned char *androidSensor*)

Check if a sensor is enabled.

Returns

1 if sensor is enabled

7.16.2.3 void INV_EXPORT inv_icm20648_ctrl_enable_activity_classifier (struct inv_icm20648 * *s*, unsigned char *enable*)

Enables / disables BAC.

Parameters

in	*enable*	0=off, 1=on

7.16.2.4 void INV_EXPORT inv_icm20648_ctrl_enable_b2s (unsigned char *enable*)

Enables / disables bring to see.

Parameters

in	*enable*	0=off, 1=on

7.16.2.5 int INV_EXPORT inv_icm20648_ctrl_enable_batch (struct inv_icm20648 * *s,* unsigned char *enable*)

Enables / disables batch for the sensors.

Parameters

in	*enable*	0=off, 1=on

Returns

0 in case of success, -1 for any error

7.16.2.6 int INV_EXPORT inv_icm20648_ctrl_enable_pickup (struct inv_icm20648 * *s,* unsigned char *enable*)

Enables / disables pickup gesture.

Parameters

7.16.2.7 int INV_EXPORT inv_icm20648_ctrl_enable_sensor (struct inv_icm20648 * *s,* unsigned char *androidSensor,* unsigned char *enable*)

Enables / disables a sensor.

Parameters

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

Returns

0 in case of success, -1 for any error

7.16.2.8 void INV_EXPORT inv_icm20648_ctrl_enable_tilt (struct inv_icm20648 * *s,* unsigned char *enable*)

Enables / disables tilt.

Parameters

in	*enable*	0=off, 1=on

7.16.2.9 int inv_icm20648_ctrl_get_acc_bias (struct inv_icm20648 * *s*, int * *acc_bias*)

get acc bias from dmp driver

Parameters

7.16.2.10 unsigned short INV_EXPORT inv_icm20648_ctrl_get_activitiy_classifier_on_flag (struct inv_icm20648 * *s*)

Returns a flag to know if the BAC is running.

Returns

1 if started, 0 if stopped

7.16.2.11 unsigned long INV_EXPORT* inv_icm20648_ctrl_get_androidSensorsOn_mask (struct inv_icm20648 * *s*)

Returns the mask for the different sensors enabled.

Returns

the mask

7.16.2.12 unsigned char INV_EXPORT inv_icm20648_ctrl_get_batch_mode_status (struct inv_icm20648 * *s*)

Get batch mode status.

Returns

0=batch mode disable, 1=batch mode enable

7.16.2.13 int inv_icm20648_ctrl_get_gyr_bias (struct inv_icm20648 * *s*, int * *gyr_bias*)

get gyr bias from dmp driver

Parameters

7.16.2.14 int INV_EXPORT inv_icm20648_ctrl_get_mag_bias (struct inv_icm20648 * *s*, int * *mag_bias*)

get mag bias from dmp driver

Parameters

7.16.2.15 int INV_EXPORT inv_icm20648_ctrl_get_odr (struct inv_icm20648 * *s*, unsigned char *SensorId*, uint32_t * *odr*, enum INV_ODR_TYPE *odr_units*)

Gets the odr for a sensor.

Parameters

in	*SensorId*	Sensor Identity
out	*odr*	pointer to the ODR for this sensor
in	*odr_units*	unit expected for odr, one of INV_ODR_TYPE

Returns

0 in case of success, -1 for any error

7.16.2.16 int inv_icm20648_ctrl_set_acc_bias (struct inv_icm20648 * *s*, int * *acc_bias*)

set acc bias from dmp driver

Parameters

7.16.2.17 int INV_EXPORT inv_icm20648_ctrl_set_accel_cal_params (struct inv_icm20648 * *s*, unsigned short *hw_smplrt_divider*)

Sets accel cal parameters according to accel engine rate.

Parameters

in	*hw_smplrt_divider*	hardware sample rate divider such that accel engine rate = 1125Hz/hw_smplrt_divider

Returns

0 in case of success, -1 for any error

7.16.2.18 int INV_EXPORT inv_icm20648_ctrl_set_accel_quaternion_gain (struct inv_icm20648 * s, unsigned short hw_smplrt_divider)

Sets accel quaternion gain according to accel engine rate.

Parameters

in	*hw_smplrt_divider*	hardware sample rate divider such that accel engine rate = 1125Hz/hw_smplrt_divider

Returns

0 in case of success, -1 for any error

7.16.2.19 void INV_EXPORT inv_icm20648_ctrl_set_batch_mode_status (struct inv_icm20648 * s, unsigned char enable)

Set batch mode status.

Parameters

in	*enable*	0=off, 1=on

7.16.2.20 int INV_EXPORT inv_icm20648_ctrl_set_batch_timeout (struct inv_icm20648 * s, unsigned short batch_time_in_seconds)

Sets the timeout for the batch in second.

Parameters

in	*batch_time_in_seconds*	time in second

Returns

0 in case of success, -1 for any error

7.16.2.21 int INV_EXPORT inv_icm20648_ctrl_set_batch_timeout_ms (struct inv_icm20648 * s, unsigned short batch_time_in_ms)

Sets the timeout for the batch in millisecond.

Parameters

in	*batch_time_in_ms*	time in millisecond

Returns

0 in case of success, -1 for any error

7.16.2.22 int inv_icm20648_ctrl_set_gyr_bias (struct inv_icm20648 * *s,* int * *gyr_bias*)

set gyr bias from dmp driver

Parameters

7.16.2.23 int INV_EXPORT inv_icm20648_ctrl_set_mag_bias (struct inv_icm20648 * *s,* int * *mag_bias*)

set mag bias from dmp driver

Parameters

7.16.2.24 int INV_EXPORT inv_icm20648_set_odr (struct inv_icm20648 * *s,* unsigned char *androidSensor,* unsigned short *delayInMs*)

Sets the odr for a sensor.

Parameters

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

Returns

0 in case of success, -1 for any error

7.17 base_driver

Functions

- int INV_EXPORT inv_icm20648_initialize_lower_driver (struct inv_icm20648 *s, enum SMARTSENSOR_↩ SERIAL_INTERFACE type, const uint8_t *dmp3_image, uint32_t dmp3_image_size)

  Initializes the platform.
- int INV_EXPORT inv_icm20648_set_slave_compass_id (struct inv_icm20648 *s, int id)

  Initializes the compass and the id address.
- int INV_EXPORT inv_icm20648_set_serial_comm (struct inv_icm20648 *s, enum SMARTSENSOR_SER↩ IAL_INTERFACE type)

  Selects the interface of communication with the board.
- int INV_EXPORT inv_icm20648_wakeup_mems (struct inv_icm20648 *s)

  Wakes up mems platform.
- int INV_EXPORT inv_icm20648_sleep_mems (struct inv_icm20648 *s)

  Sleeps up mems platform.
- int INV_EXPORT inv_icm20648_set_chip_power_state (struct inv_icm20648 *s, unsigned char func, unsigned char on_off)

  Sets the power state of the Ivory chip loop.
- uint8_t INV_EXPORT inv_icm20648_get_chip_power_state (struct inv_icm20648 *s)

  Current wake status of the Mems chip.
- int INV_EXPORT inv_icm20648_set_dmp_address (struct inv_icm20648 *s)

  Sets up dmp start address and firmware.
- int INV_EXPORT inv_icm20648_set_secondary (struct inv_icm20648 *s)

  Sets up the secondary i2c bus.
- int INV_EXPORT inv_icm20648_enable_hw_sensors (struct inv_icm20648 *s, int bit_mask)

  Enables accel and/or gyro and/or pressure if integrated with gyro and accel.
- int INV_EXPORT inv_icm20648_set_gyro_sf (struct inv_icm20648 *s, unsigned char div, int gyro_level)

  Sets the dmp for a particular gyro configuration.
- int INV_EXPORT inv_icm20648_set_gyro_divider (struct inv_icm20648 *s, unsigned char div)

  Sets the gyro sample rate.
- unsigned char INV_EXPORT inv_icm20648_get_gyro_divider (struct inv_icm20648 *s)

  Returns the gyro sample rate.
- uint32_t INV_EXPORT inv_icm20648_get_odr_in_units (struct inv_icm20648 *s, unsigned short odrIn↩ Divider, unsigned char odr_units)

  Returns the real odr in Milliseconds, Micro Seconds or Ticks.
- int INV_EXPORT inv_icm20648_set_accel_divider (struct inv_icm20648 *s, short div)

  Sets the accel sample rate.
- short INV_EXPORT inv_icm20648_get_accel_divider (struct inv_icm20648 *s)

  Returns the accel sample rate.
- int INV_EXPORT inv_icm20648_set_secondary_divider (struct inv_icm20648 *s, unsigned char div)

  Sets the I2C secondary device sample rate.
- unsigned short INV_EXPORT inv_icm20648_get_secondary_divider (struct inv_icm20648 *s)

  Returns the I2C secondary device sample rate.
- int INV_EXPORT inv_icm20648_set_gyro_fullscale (struct inv_icm20648 *s, int level)

  Sets fullscale range of gyro in hardware.
- uint8_t INV_EXPORT inv_icm20648_get_gyro_fullscale (struct inv_icm20648 *s)

  Returns fullscale range of gyrometer in hardware.
- int INV_EXPORT inv_icm20648_set_icm20648_gyro_fullscale (struct inv_icm20648 *s, int level)

  Sets fullscale range of gyro in hardware.
- int INV_EXPORT inv_icm20648_set_accel_fullscale (struct inv_icm20648 *s, int level)

Sets fullscale range of accel in hardware.

- uint8_t INV_EXPORT inv_icm20648_get_accel_fullscale (struct inv_icm20648 *s)

  Returns fullscale range of accelerometer in hardware.
- int INV_EXPORT inv_icm20648_set_icm20648_accel_fullscale (struct inv_icm20648 *s, int level)

  Sets fullscale range of accel in hardware.
- int INV_EXPORT inv_icm20648_set_int1_assertion (struct inv_icm20648 *s, int enable)

  Asserts int1 interrupt when DMP execute INT1 cmd.
- int INV_EXPORT inv_icm20648_accel_read_hw_reg_data (struct inv_icm20648 *s, short accel_hw_reg_↩ data[3])

  Reads accelerometer data stored in hardware register.
- void INV_EXPORT inv_icm20648_prevent_lpen_control (struct inv_icm20648 *s)

  Prevent LP_EN from being set to 1 again, this speeds up transaction.
- void INV_EXPORT inv_icm20648_allow_lpen_control (struct inv_icm20648 *s)

  Allow LP_EN to be set to 1 again and sets it to 1 again if supported by chip.
- int INV_EXPORT inv_icm20648_get_compass_availability (struct inv_icm20648 *s)

  Determine if compass could be successfully found and inited on board.
- int INV_EXPORT inv_icm20648_get_pressure_availability (struct inv_icm20648 *s)

  Determine if pressure could be successfully found and inited on board.
- int INV_EXPORT inv_icm20648_get_proximity_availability (struct inv_icm20648 *s)

  Determine if proximity could be successfully found and inited on board.
- int INV_EXPORT inv_icm20648_enter_duty_cycle_mode (struct inv_icm20648 *s)

  Have the chip to enter stand duty cycled mode, also called low-power mode where max reporting frequency is 562Hz.
- int INV_EXPORT inv_icm20648_enter_low_noise_mode (struct inv_icm20648 *s)

  Have the chip to enter low-noise mode.

7.17.1 Detailed Description

7.17.2 Function Documentation

7.17.2.1 int INV_EXPORT inv_icm20648_accel_read_hw_reg_data (struct inv_icm20648 * *s*, short *accel_hw_reg_data[3]*)

Reads accelerometer data stored in hardware register.

Parameters

in	*accel_hw_reg_data*	variable to be recuperated the accelerometer data

Returns

0 on success, negative value on error.

7.17.2.2 int INV_EXPORT inv_icm20648_enable_hw_sensors (struct inv_icm20648 * *s*, int *bit_mask*)

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.

Returns

0 on success, negative value on error.

7.17.2.3 short INV_EXPORT inv_icm20648_get_accel_divider (struct inv_icm20648 * *s*)

Returns the accel sample rate.

Returns

the divider for the accel

7.17.2.4 uint8_t INV_EXPORT inv_icm20648_get_accel_fullscale (struct inv_icm20648 * *s*)

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.17.2.5 uint8_t INV_EXPORT inv_icm20648_get_chip_power_state (struct inv_icm20648 * *s*)

Current wake status of the Mems chip.

Returns

the wake status

7.17.2.6 int INV_EXPORT inv_icm20648_get_compass_availability (struct inv_icm20648 * *s*)

Determine if compass could be successfully found and inited on board.

Returns

1 on success, 0 if not available.

7.17.2.7 unsigned char INV_EXPORT inv_icm20648_get_gyro_divider (struct inv_icm20648 * s)

Returns the gyro sample rate.

Returns

Value written to GYRO_SMPLRT_DIV register.

7.17.2.8 uint8_t INV_EXPORT inv_icm20648_get_gyro_fullscale (struct inv_icm20648 * s)

Returns fullscale range of gyrometer in hardware.

Returns

the fullscale range

7.17.2.9 uint32_t INV_EXPORT inv_icm20648_get_odr_in_units (struct inv_icm20648 * s, unsigned short *odrInDivider*, unsigned char *odr_units*)

Returns the real odr in Milliseconds, Micro Seconds or Ticks.

Parameters

in	*odrInDivider*	Odr In divider
in	*odr_units*	Use the enum values: ODR_IN_Ms, ODR_IN_Us or ODR_IN_Ticks

Returns

Odr in fucntion of enum.

7.17.2.10 int INV_EXPORT inv_icm20648_get_pressure_availability (struct inv_icm20648 * s)

Determine if pressure could be successfully found and inited on board.

Returns

1 on success, 0 if not available.

7.17.2.11 int INV_EXPORT inv_icm20648_get_proximity_availability (struct inv_icm20648 * s)

Determine if proximity could be successfully found and inited on board.

Returns

1 on success, 0 if not available.

7.17.2.12 unsigned short INV_EXPORT inv_icm20648_get_secondary_divider (struct inv_icm20648 * *s*)

Returns the I2C secondary device sample rate.

Returns

the divider for the I2C secondary device interface

7.17.2.13 int INV_EXPORT inv_icm20648_initialize_lower_driver (struct inv_icm20648 * *s*, enum SMARTSENSOR_SERIAL_INTERFACE *type*, const uint8_t * *dmp3_image*, uint32_t *dmp3_image_size*)

Initializes the platform.

Parameters

in	*type*	Define the interface for communicate : SERIAL_INTERFACE_I2C or SERIAL_INTERFACE_SPI
out	*dmp_image_sram*	4 The image to be load

Returns

0 on success, negative value on error.

7.17.2.14 int INV_EXPORT inv_icm20648_set_accel_divider (struct inv_icm20648 * *s*, short *div*)

Sets the accel sample rate.

Parameters

in	*div*	Value written to ACCEL_SMPLRT_DIV register

Returns

0 on success, negative value on error.

7.17.2.15 int INV_EXPORT inv_icm20648_set_accel_fullscale (struct inv_icm20648 * *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.

7.17.2.16 int INV_EXPORT inv_icm20648_set_chip_power_state (struct inv_icm20648 * *s*, unsigned char *func*, unsigned char *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.

7.17.2.17 int INV_EXPORT inv_icm20648_set_dmp_address (struct inv_icm20648 * *s*)

Sets up dmp start address and firmware.

Returns

0 on success, negative value on error.

7.17.2.18 int INV_EXPORT inv_icm20648_set_gyro_divider (struct inv_icm20648 * *s*, unsigned char *div*)

Sets the gyro sample rate.

Parameters

in	*div*	Value written to GYRO_SMPLRT_DIV register

Returns

0 on success, negative value on error.

7.17.2.19 int INV_EXPORT inv_icm20648_set_gyro_fullscale (struct inv_icm20648 * *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.

7.17.2.20 int INV_EXPORT inv_icm20648_set_gyro_sf (struct inv_icm20648 * s, unsigned char div, int gyro_level)

Sets the dmp for a particular gyro configuration.

Parameters

in	*gyro_div*	Value written to GYRO_SMPLRT_DIV register, where 0=1125Hz sample rate, 1=562.5Hz sample rate, ... 4=225Hz sample rate, ... 10=102.2727Hz sample rate, ... etc.
in	*gyro_level*	0=250 dps, 1=500 dps, 2=1000 dps, 3=2000 dps

Returns

0 on success, negative value on error.

7.17.2.21 int INV_EXPORT inv_icm20648_set_icm20648_accel_fullscale (struct inv_icm20648 * 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.

7.17.2.22 int INV_EXPORT inv_icm20648_set_icm20648_gyro_fullscale (struct inv_icm20648 * 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.

7.17.2.23 int INV_EXPORT inv_icm20648_set_int1_assertion (struct inv_icm20648 * s, int enable)

Asserts int1 interrupt when DMP execute INT1 cmd.

Parameters

in	*enable*	0=off, 1=on

Returns

0 on success, negative value on error.

7.17.2.24 int INV_EXPORT inv_icm20648_set_secondary (struct inv_icm20648 * *s*)

Sets up the secondary i2c bus.

Returns

0 on success, negative value on error.

7.17.2.25 int INV_EXPORT inv_icm20648_set_secondary_divider (struct inv_icm20648 * *s*, unsigned char *div*)

Sets the I2C secondary device sample rate.

Parameters

in	*div*	Value written to REG_I2C_MST_ODR_CONFIG register

Returns

0 on success, negative value on error.

7.17.2.26 int INV_EXPORT inv_icm20648_set_serial_comm (struct inv_icm20648 * *s*, enum SMARTSENSOR_SERIAL_INTERFACE *type*)

Selects the interface of communication with the board.

Parameters

in	*type*	Define the interface for communicate : SERIAL_INTERFACE_I2C or SERIAL_INTERFACE_SPI

Returns

0 on success, negative value on error.

7.17.2.27 int INV_EXPORT inv_icm20648_set_slave_compass_id (struct inv_icm20648 * *s*, int *id*)

Initializes the compass and the id address.

Parameters

in	*id*	address of compass component

Returns

0 on success, negative value on error.

7.17.2.28 int INV_EXPORT inv_icm20648_sleep_mems (struct inv_icm20648 * *s*)

Sleeps up mems platform.

Returns

0 on success, negative value on error.

7.17.2.29 int INV_EXPORT inv_icm20648_wakeup_mems (struct inv_icm20648 * *s*)

Wakes up mems platform.

Returns

0 on success, negative value on error.

7.18 data_converter

Macros

- #define ABS(x) (((x)>=0)?(x):-(x))

  Computes the absolute value of its argument x.
- #define MAX(x, y) (((x)>(y))?(x):(y))

  Computes the maximum of x and y.
- #define MIN(x, y) (((x)<(y))?(x):(y))

  Computes the minimum of x and y.
- #define INVN_FLT_TO_FXP(value, shift) ((int32_t) ((float)(value)*(1ULL << (shift)) + ((value>=0)-0.5f)))

  Convert the value from float to QN value.
- #define INVN_CONVERT_FLT_TO_FXP(fltptr, fixptr, length, shift) { int i; for(i=0; i<(length); ++i) (fixptr)[i] = INVN_FLT_TO_FXP((fltptr)[i], shift); }

  Macro to convert float values from an address into QN values, and copy them to another address.

Functions

- void INV_EXPORT inv_icm20648_q_mult_q_qi (const long *q1, const long *q2, long *qProd)

  *Performs a fixed point quaternion multiply with inverse on second element q1*q2'.*
- void INV_EXPORT inv_icm20648_set_chip_to_body (struct inv_icm20648 *s, long *quat)

  Sets the transformation used for chip to body frame.
- void INV_EXPORT inv_icm20648_convert_rotation_vector (struct inv_icm20648 *s, const long *quat, float *values)

  Converts fixed point DMP rotation vector to floating point android notation.
- void INV_EXPORT inv_icm20648_convert_rotation_vector_2 (struct inv_icm20648 *s, const long *quat, long *quat4_world)

  Converts 3 element fixed point DMP rotation vector to 4 element rotation vector in world frame.
- void INV_EXPORT inv_icm20648_convert_rotation_vector_3 (const long *quat4_world, float *values)

  Converts 4 element rotation vector in world frame to floating point android notation.
- void INV_EXPORT inv_icm20648_convert_dmp3_to_body (struct inv_icm20648 *s, const long *vec3, float scale, float *values)

  Converts the data in android values.
- void INV_EXPORT inv_icm20648_set_chip_to_body_axis_quaternion (struct inv_icm20648 *s, signed char *accel_gyro_matrix, float angle)

  Converts the data in android quaternion values.
- unsigned char INV_EXPORT * inv_icm20648_int32_to_little8 (long x, unsigned char *little8)

  Converts a 32-bit long to a little endian byte stream.
- float INV_EXPORT inv_icm20648_convert_deg_to_rad (float deg_val)

  Converts degree angle to radian.
- long INV_EXPORT inv_icm20648_convert_mult_q30_fxp (long a_q30, long b_q30)

  Performs a multiply and shift by 30.
- int INV_EXPORT inv_icm20648_convert_compute_scalar_part_fxp (const long *inQuat_q30, long *out↩ Quat_q30)

  Compute real part of quaternion, element[0].
- long INV_EXPORT inv_icm20648_convert_fast_sqrt_fxp (long x0_q30)

  Calculates square-root of a fixed-point number (30 bit mantissa, positive)
- int INV_EXPORT inv_icm20648_convert_test_limits_and_scale_fxp (long *x0_q30, int *pow)

  Auxiliary function used by inv_OneOverX(), inv_fastSquareRoot(), inv_inverseSqrt().
- int16_t INV_EXPORT inv_icm20648_convert_get_highest_bit_position (uint32_t *value)

Auxiliary function used by testLimitsAndScale() Find the highest nonzero bit in an unsigned 32 bit integer:

- void INV_EXPORT inv_icm20648_convert_matrix_to_quat_fxp (long *Rcb_q30, long *Qcb_q30)
  Converts a rotation matrix to a quaternion.
- long INV_EXPORT inv_icm20648_convert_sqrt_q30_fxp (long x_q30)
  Calculates square-root of a fixed-point number.
- long INV_EXPORT inv_icm20648_convert_inv_sqrt_q30_fxp (long x_q30, int *pow2)
  Calculates 1/square-root of a fixed-point number (30 bit mantissa, positive): Q1.30.
- long INV_EXPORT inv_icm20648_convert_inverse_q30_fxp (long x_q30, int *pow2)
  Inverse function based on Newton-Raphson 1/sqrt(x) calculation.
- void INV_EXPORT inv_icm20648_convert_matrix_to_quat_flt (float *R, float *q)
  Converts a rotation matrix to a quaternion in floating point.
- long INV_EXPORT inv_icm20648_convert_mult_qfix_fxp (long a, long b, unsigned char qfix)
  Performs a multiply and shift by shift.
- void INV_EXPORT inv_icm20648_convert_quat_to_col_major_matrix_fxp (const long *quat_q30, long *rot↩ _q30)
  Converts a quaternion to a rotation matrix in column major convention.
- long INV_EXPORT inv_icm20648_math_atan2_q15_fxp (long y_q15, long x_q15)
  Seventh order Chebychev polynomial approximation in Q15.
- uint8_t INV_EXPORT * inv_icm20648_convert_int16_to_big8 (int16_t x, uint8_t *big8)
  Converts a 16-bit short to a big endian byte stream.
- uint8_t INV_EXPORT * inv_icm20648_convert_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_icm20648_convert_big8_to_int32 (const uint8_t *big8)
  Converts a big endian byte stream into a 32-bit long.
- void INV_EXPORT inv_icm20648_convert_quat_rotate_fxp (const long *quat_q30, const long *in, long *out)
  Converts long values according to quat_30 matrix.

7.18.1 Detailed Description

7.18.2 Macro Definition Documentation

7.18.2.1 #define ABS(*x*) (((x)>=0)?(x):-(x))

Computes the absolute value of its argument *x*.

7.18.2.2 #define INVN_FLT_TO_FXP(*value, shift*) ((int32_t) ((float)(value)*(1ULL << (shift)) + ((value>=0)-0.5f)))

Convert the *value* from float to QN value.

7.18.2.3 #define MAX(*x, y*) (((x)>(y))?(x):(y))

Computes the maximum of *x* and *y*.

7.18.2.4 #define MIN(*x, y*) (((x)<(y))?(x):(y))

Computes the minimum of *x* and *y*.

7.18.3 Function Documentation

7.18.3.1 int32_t INV_EXPORT inv_icm20648_convert_big8_to_int32 (const uint8_t * *big8*)

Converts a big endian byte stream into a 32-bit long.

Parameters

in	*big8*	big endian byte stream

Returns

corresponding 32-bit integer.

7.18.3.2 int INV_EXPORT inv_icm20648_convert_compute_scalar_part_fxp (const long * *inQuat_q30,* long * *outQuat_q30*)

Compute real part of quaternion, element[0].

Parameters

in	*inQuat_q30*	3 elements gyro quaternion. Dimension is 3.
out	*outQuat_q30*	Quaternion. Dimension is 4. 4 elements gyro quaternion

Returns

0

7.18.3.3 float INV_EXPORT inv_icm20648_convert_deg_to_rad (float *deg_val*)

Converts degree angle to radian.

Parameters

in	*deg_val*	the angle in degree

Returns

the angle in radian

7.18.3.4 void INV_EXPORT inv_icm20648_convert_dmp3_to_body (struct inv_icm20648 * *s,* const long * *vec3,* float *scale,* float * *values*)

Converts the data in android values.

Parameters

in	*vec3*	vector of the DMP
in	*scale*	scale calculated
out	*values*	in Android format

7.18.3.5 long INV_EXPORT inv_icm20648_convert_fast_sqrt_fxp (long *x0_q30*)

Calculates square-root of a fixed-point number (30 bit mantissa, positive)

Input must be a positive scaled (2^{30}) integer The number is scaled to lie between a range in which a Newton-↩ Raphson iteration works best.

Parameters

in	*x0_q30*	length 1. Fixed point format is Q30

Returns

scaled square root if succeed else 0.

7.18.3.6 int16_t INV_EXPORT inv_icm20648_convert_get_highest_bit_position (uint32_t * *value*)

Auxiliary function used by testLimitsAndScale() Find the highest nonzero bit in an unsigned 32 bit integer:

Parameters

in	*value*	operand Dimension is 1.

Returns

highest bit position.

Note

This function performs the log2 of an interger as well.

7.18.3.7 uint8_t INV_EXPORT* inv_icm20648_convert_int16_to_big8 (int16_t *x,* uint8_t * *big8*)

Converts a 16-bit short to a big endian byte stream.

Parameters

in	*x*	operand
out	*big8*	big endian byte stream

Returns

big8 pointer

7.18.3.8 uint8_t INV_EXPORT* inv_icm20648_convert_int32_to_big8 (int32_t *x*, uint8_t * *big8*)

Converts a 32-bit long to a big endian byte stream.

Parameters

in	*x*	operand
out	*big8*	big endian byte stream

Returns

big8 pointer

7.18.3.9 long INV_EXPORT inv_icm20648_convert_inv_sqrt_q30_fxp (long *x_q30,* int * *pow2*)

Calculates 1/square-root of a fixed-point number (30 bit mantissa, positive): Q1.30.

The number is scaled to lie between a range in which a Newton-Raphson iteration works best. Caller must scale final result by 2^rempow (while avoiding overflow).

Parameters

in	*x_q30*	Input. The input must be positive. Fixed point format is Q30.
out	*pow2*	Corresponding square root of the power of two is returned. length 1

Returns

square root of x in Q30.

7.18.3.10 long INV_EXPORT inv_icm20648_convert_inverse_q30_fxp (long *x_q30,* int * *pow2*)

Inverse function based on Newton-Raphson 1/sqrt(x) calculation.

Note that upshifting c (the result) by pow2 right away will overflow q30 if b<0.5 in q30 (=536870912).
So if you are doing some multiplication later on (like a/b), then it might be better to do `q30_mult(a,c)` first and then shift it up by pow2: `q30_mult(a,c)<<pow2`
The result might still overflow in some cases (large a, small b: a=1073741824, b=1 but precise limits of the overflow are tbd).

Parameters

in	*x_q30*	the operand. Fixed point format is Q30
in	*pow2*	a power of 2 by which 1/b is downshifted to fit in q30.

Returns

the 1/x result in Q30 downshifted by pow2.

7.18.3.11 void INV_EXPORT inv_icm20648_convert_matrix_to_quat_flt (float * *R,* float * *q*)

Converts a rotation matrix to a quaternion in floating point.

Parameters

in	*R*	Rotation matrix in floating point. The First 3 elements of the rotation matrix, represent the first row of the matrix.
out	*q*	4-element quaternion in floating point.

Warning

This functions does not retrieve fixed point quaternion anymore. Use a conversion flt_to_fxp.

7.18.3.12 void INV_EXPORT inv_icm20648_convert_matrix_to_quat_fxp (long * *Rcb_q30,* long * *Qcb_q30*)

Converts a rotation matrix to a quaternion.

Parameters

in	*Rcb_q30*	Rotation matrix. Fixed point format is Q30.
out	*Qcb_q30*	quaternion related to provided rotation matrix. Vector size is 4. Fixed point format is Q30.

7.18.3.13 long INV_EXPORT inv_icm20648_convert_mult_q30_fxp (long *a_q30,* long *b_q30*)

Performs a multiply and shift by 30.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

Parameters

in	*a*	
in	*b*	

Returns

((long long)a*b)>>30

7.18.3.14 long INV_EXPORT inv_icm20648_convert_mult_qfix_fxp (long *a,* long *b,* unsigned char *qfix*)

Performs a multiply and shift by shift.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

Parameters

in	*a*	First multicand
in	*b*	Second multicand
in	*shift*	Shift amount after multiplying

Returns

((long long)a∗b)>>shift

Warning

Same function that invn_math_mult_qfix_fxp.

7.18.3.15 void INV_EXPORT inv_icm20648_convert_quat_rotate_fxp (const long ∗ *quat_q30,* const long ∗ *in,* long ∗ *out*)

Converts long values according to quat_30 matrix.

Parameters

in	*quat_30*	mounting matrix to apply
in	*in*	long values to be converted
out	*out*	long values converted

Returns

void

7.18.3.16 void INV_EXPORT inv_icm20648_convert_quat_to_col_major_matrix_fxp (const long ∗ *quat_q30,* long ∗ *rot_q30*)

Converts a quaternion to a rotation matrix in column major convention.

Parameters

in	*quat_q30*	4-element quaternion in fixed point. Fixed point format is Q30.
out	*rot_q30*	Rotation matrix in fixed point. One is 2^30. The Rotation matrix multiplied by a 3 element column vector transforms a vector from Body to World.

Warning

output matrix storage is column major. colmajor_convention

7.18.3.17 void INV_EXPORT inv_icm20648_convert_rotation_vector (struct inv_icm20648 ∗ *s,* const long ∗ *quat,* float ∗ *values*)

Converts fixed point DMP rotation vector to floating point android notation.

Parameters

in	*quat*	3 element rotation vector from DMP, missing the scalar part. Converts from Chip frame to World frame
out	*values*	4 element quaternion in Android format

7.18.3.18 void INV_EXPORT inv_icm20648_convert_rotation_vector_2 (struct inv_icm20648 * *s,* const long * *quat,* long * *quat4_world*)

Converts 3 element fixed point DMP rotation vector to 4 element rotation vector in world frame.

Parameters

in	*quat*	3 element rotation vector from DMP, missing the scalar part. Converts from Chip frame to World frame
out	*quat4_world*	4 element quaternion

7.18.3.19 void INV_EXPORT inv_icm20648_convert_rotation_vector_3 (const long * *quat4_world,* float * *values*)

Converts 4 element rotation vector in world frame to floating point android notation.

Parameters

in	*quat4_world*	4 element rotation vector in World frame
out	*values*	in Android format

7.18.3.20 long INV_EXPORT inv_icm20648_convert_sqrt_q30_fxp (long *x_q30*)

Calculates square-root of a fixed-point number.

This code calls 1/sqrt(x) and multiplies result with x, i.e. $\sqrt{x} = x * (1/\sqrt{x})$.

Parameters

in	*x_q30*	Input. Fixed point format is Q30

Returns

square root of x0 in Q30.

7.18.3.21 int INV_EXPORT inv_icm20648_convert_test_limits_and_scale_fxp (long * *x0_q30,* int * *pow*)

Auxiliary function used by inv_OneOverX(), inv_fastSquareRoot(), inv_inverseSqrt().

Finds the range of the argument, determines the optimal number of Newton-Raphson iterations and . Restrictions: Number is represented as Q1.30. Number is betweeen the range 2<x<=0

Parameters

in	*x0_q30*	Input length 1. Number is represented as Q30. Number is betweeen the range 2<x<=0
out	*pow*	Corresponding square root of the power of two is returned. length 1

Returns

number of Newton Raphson iterations, x0 scaled between log(2) and log(4) and 2^N scaling (N=pow)

7.18.3.22 unsigned char INV_EXPORT* inv_icm20648_int32_to_little8 (long *x*, unsigned char * *little8*)

Converts a 32-bit long to a little endian byte stream.

Parameters

in	*x*	the long to be converted
in	*little8*	little endian byte converted

Returns

0 on success, negative value on error.

7.18.3.23 long INV_EXPORT inv_icm20648_math_atan2_q15_fxp (long *y_q15*, long *x_q15*)

Seventh order Chebychev polynomial approximation in Q15.

Chebychev 7th order polynomial approximation :

- in fixed point : $constA7 = int32(2^{15} * [0.999133448222780 - 0.3205332923816640.144982490144465, -0.038254464970299$
- in float : $A = [0.999133 - 0.3205330.144982 - 0.0382544);$

The related formula is :
$\xi = \{ \mid y|/|x|in(0, \pi/4]$
$|x|/|y|in(\pi/4, \pi/2), \quad Cheb = A(1) * \xi + A(2) * \xi^3 + A(3) * \xi^5 + A(4) * \xi^7$

7th Order Accuracy is +/-0.02 degrees (worst case) through entire range (accomplished with scaling).
This code depends on: reciprocal_fun_q15 , inverse_sqrt_q15 , inv_q15_mult

Parameters

in	*y_q15*	first operand of atan2(y, x). Fixed point format is Q15.
in	*x_q15*	second operand of atan2(y, x). Fixed point format is Q15.

Returns

output angle in radians. Fixed point format is Q15.

7.18.3.24 void INV_EXPORT inv_icm20648_q_mult_q_qi (const long * *q1,* const long * *q2,* long * *qProd*)

Performs a fixed point quaternion multiply with inverse on second element q1∗q2'.

Parameters

in	*q1*	First Quaternion Multicand, length 4. 1.0 scaled to 2^30
in	*q2*	Second Quaternion Multicand, length 4. 1.0 scaled to 2^30. Inverse will be take before multiply
out	*qProd*	Product after quaternion multiply q1∗q2'. Length 4. 1.0 scaled to 2^30.

7.18.3.25 void INV_EXPORT inv_icm20648_set_chip_to_body (struct inv_icm20648 * *s,* long * *quat*)

Sets the transformation used for chip to body frame.

Parameters

in	*quat*	the quaternion used for the transformation

7.18.3.26 void INV_EXPORT inv_icm20648_set_chip_to_body_axis_quaternion (struct inv_icm20648 * *s,* signed char * *accel_gyro_matrix,* float *angle*)

Converts the data in android quaternion values.

Parameters

in	*accel_gyro_matrix*	vector of the DMP
out	*angle*	angle calculated

7.19 load_firmware

Functions

- int INV_EXPORT inv_icm20648_firmware_load (struct inv_icm20648 *s, const unsigned char *data, unsigned short size, unsigned short load_addr)

  Loads the DMP firmware from SRAM.

7.19.1 Detailed Description

7.19.2 Function Documentation

7.19.2.1 int INV_EXPORT inv_icm20648_firmware_load (struct inv_icm20648 * *s,* const unsigned char * *data,* unsigned short *size,* unsigned short *load_addr*)

Loads the DMP firmware from SRAM.

Parameters

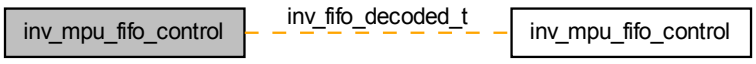
in	*data*	pointer where the image
in	*size*	size if the image
in	*load_addr*	address to loading the image

Returns

0 in case of success, -1 for any error

7.20 inv_mpu_fifo_control

Collaboration diagram for inv_mpu_fifo_control:



Classes

- struct inv_fifo_decoded_t

  Struct for the fifo.

Functions

- int INV_EXPORT inv_icm20648_identify_interrupt (struct inv_icm20648 *s, short *int_read)

  Identify the interrupt.
- int INV_EXPORT inv_icm20648_dmp_process_fifo (struct inv_icm20648 *s, int *left_in_fifo, unsigned short *user_header, unsigned short *user_header2, long long *time_stamp)

  Process the fifo.
- int INV_EXPORT inv_icm20648_dmp_get_accel (long acl[3])

  Gets the accelerometer data.
- int INV_EXPORT inv_icm20648_dmp_get_raw_gyro (short raw_gyro[3])

  Gets the raw gyrometer data.
- int INV_EXPORT inv_icm20648_dmp_get_gyro_bias (short gyro_bias[3])

  Gets gyro bias.
- int INV_EXPORT inv_icm20648_dmp_get_calibrated_gyro (signed long calibratedData[3], signed long raw[3], signed long bias[3])

  Gets calibrated gyro value based on raw gyro and gyro bias.
- int INV_EXPORT inv_icm20648_dmp_get_6quaternion (long quat[3])

  Gets the quaternion 6 axis data.
- int INV_EXPORT inv_icm20648_dmp_get_9quaternion (long quat[3])

  Gets the quaternion 9 axis data.
- int INV_EXPORT inv_icm20648_dmp_get_gmrvquaternion (long quat[3])

  Gets the quaternion GMRV data.
- int INV_EXPORT inv_icm20648_dmp_get_raw_compass (long raw_compass[3])

  Gets the raw compass data.
- int INV_EXPORT inv_icm20648_dmp_get_calibrated_compass (long cal_compass[3])

  Gets the calibrated compass data.
- int INV_EXPORT inv_icm20648_inv_decode_one_ivory_fifo_packet (struct inv_icm20648 *s, struct inv_fifo↩_decoded_t *fd, const unsigned char *fifo_ptr)

  Decodes the fifo packet.
- int INV_EXPORT inv_icm20648_dmp_get_bac_state (uint16_t *bac_state)

  Gets the state of the BAC sensor.
- int INV_EXPORT inv_icm20648_dmp_get_bac_ts (long *bac_ts)

Gets the timestamp of the BAC sensor.

- int INV_EXPORT inv_icm20648_dmp_get_flip_pickup_state (uint16_t *flip_pickup)

  Gets the state of the pick up sensor.
- int INV_EXPORT inv_icm20648_get_accel_accuracy (void)

  Returns the accelerometer accuracy.
- int INV_EXPORT inv_icm20648_get_gyro_accuracy (void)

  Returns the gyrometer accuracy.
- int INV_EXPORT inv_icm20648_get_mag_accuracy (void)

  Returns the magnetometer accuracy.
- int INV_EXPORT inv_icm20648_get_gmrv_accuracy (void)

  Returns the geomagnetic rotation vector accuracy.
- int INV_EXPORT inv_icm20648_get_rv_accuracy (void)

  Returns the rotation vector accuracy.
- int INV_EXPORT inv_icm20648_mpu_set_FIFO_RST_Diamond (struct inv_icm20648 *s, unsigned char value)

  Resets the fifo.
- int INV_EXPORT inv_icm20648_fifo_swmirror (struct inv_icm20648 *s, int *left_in_fifo, unsigned short *total_sample_cnt, unsigned short *sample_cnt_array)

  Mirror DMP HW FIFO into SW FIFO.
- int INV_EXPORT inv_icm20648_fifo_pop (struct inv_icm20648 *s, unsigned short *user_header, unsigned short *user_header2, int *left_in_fifo)

  Pop one sample out of SW FIFO.

7.20.1 Detailed Description

7.20.2 Function Documentation

7.20.2.1 int INV_EXPORT inv_icm20648_dmp_get_6quaternion (long *quat[3]*)

Gets the quaternion 6 axis data.

Parameters

out	*quat[3]*	the quaternion 6 axis data

Returns

0 on success, negative value on error.

7.20.2.2 int INV_EXPORT inv_icm20648_dmp_get_9quaternion (long *quat[3]*)

Gets the quaternion 9 axis data.

Parameters

out	*quat[3]*	the quaternion 9 axis data

Returns

0 on success, negative value on error.

7.20.2.3 int INV_EXPORT inv_icm20648_dmp_get_accel (long *acl[3]*)

Gets the accelerometer data.

Parameters

out	*acl[3]*	the accelerometer data

Returns

0 on success, negative value on error.

7.20.2.4 int INV_EXPORT inv_icm20648_dmp_get_bac_state (uint16_t * *bac_state*)

Gets the state of the BAC sensor.

Parameters

in	*bac_state*	pointer for recuperate the state of BAC

Returns

0 on success, negative value on error.

7.20.2.5 int INV_EXPORT inv_icm20648_dmp_get_bac_ts (long * *bac_ts*)

Gets the timestamp of the BAC sensor.

Parameters

in	*bac_ts*	pointer for recuperate the timestamp of BAC

Returns

0 on success, negative value on error.

7.20.2.6 int INV_EXPORT inv_icm20648_dmp_get_calibrated_compass (long *cal_compass[3]*)

Gets the calibrated compass data.

Parameters

out	*cal_compass[3]*	the calibrated compass data

Returns

0 on success, negative value on error.

7.20.2.7 int INV_EXPORT inv_icm20648_dmp_get_calibrated_gyro (signed long *calibratedData[3],* signed long *raw[3],* signed long *bias[3]*)

Gets calibrated gyro value based on raw gyro and gyro bias.

Parameters

out	*calibratedData[3]*	Calibred Gyro x,y,z
in	*raw[3]*	Gyro raw data x,y,z
in	*bias[3]*	Gyro bias x,y,z

Returns

0 on success, negative value on error.

7.20.2.8 int INV_EXPORT inv_icm20648_dmp_get_flip_pickup_state (uint16_t * *flip_pickup*)

Gets the state of the pick up sensor.

Parameters

in	*flip_pickup*	pointer for recuperate the state of pickup

Returns

0 on success, negative value on error.

7.20.2.9 int INV_EXPORT inv_icm20648_dmp_get_gmrvquaternion (long *quat[3]*)

Gets the quaternion GMRV data.

Parameters

out	*quat[3]*	the quaternion GMRV 6 axis data

Returns

0 on success, negative value on error.

7.20.2.10 int INV_EXPORT inv_icm20648_dmp_get_gyro_bias (short *gyro_bias[3]*)

Gets gyro bias.

Parameters

out	*quat[3]*	Gyro bias x,y,z

Returns

0 on success, negative value on error.

7.20.2.11 int INV_EXPORT inv_icm20648_dmp_get_raw_compass (long *raw_compass[3]*)

Gets the raw compass data.

Parameters

out	*cal_compass[3]*	the raw compass data

Returns

0 on success, negative value on error.

7.20.2.12 int INV_EXPORT inv_icm20648_dmp_get_raw_gyro (short *raw_gyro[3]*)

Gets the raw gyrometer data.

Parameters

out	*raw_gyro[3]*	the raw gyrometer data

Returns

0 on success, negative value on error.

7.20.2.13 int INV_EXPORT inv_icm20648_dmp_process_fifo (struct inv_icm20648 * *s,* int * *left_in_fifo,* unsigned short * *user_header,* unsigned short * *user_header2,* long long * *time_stamp*)

Process the fifo.

Parameters

in	*left_in_fifo*	pointer for the fifo to be processed
in	*user_header*	pointer for the user header
in	*user_header2*	pointer for the user header 2
in	*time_stamp*	pointer for the timestamp

Returns

0 on success, negative value on error.

7.20.2.14 int INV_EXPORT inv_icm20648_fifo_pop (struct inv_icm20648 * *s,* unsigned short * *user_header,* unsigned short * *user_header2,* int * *left_in_fifo*)

Pop one sample out of SW FIFO.

Parameters

out	*user_header*	Header value read from SW FIFO
out	*user_header2*	Header2 value read from SW FIFO
in,out	*left_in_fifo*	Contains number of bytes still be parsed from SW FIFO

Returns

0 on success, negative value on error.

7.20.2.15 int INV_EXPORT inv_icm20648_fifo_swmirror (struct inv_icm20648 * *s,* int * *left_in_fifo,* unsigned short * *total_sample_cnt,* unsigned short * *sample_cnt_array*)

Mirror DMP HW FIFO into SW FIFO.

Parameters

in,out	*left_in_fifo*	pointer to number of bytes in SW FIFO : before function is called, must contain number of bytes still present in FIFO which must not be overwritten after function is called, will contain number of bytes present in SW FIFO to be analyzed
out	*total_sample_cnt*	number of total sensor samples present in SW FIFO
out	*sample_cnt_array*	array of number of sensor samples present in SW FIFO for each sensor, should be inited to 0 before being called

Returns

0 on success, negative value on error.

7.20.2.16 int INV_EXPORT inv_icm20648_get_accel_accuracy (void)

Returns the accelerometer accuracy.

Returns

the accelerometer accuracy value

7.20.2.17 int INV_EXPORT inv_icm20648_get_gmrv_accuracy (void)

Returns the geomagnetic rotation vector accuracy.

Returns

the geomagnetic rotation vector accuracy in Q29

7.20.2.18 int INV_EXPORT inv_icm20648_get_gyro_accuracy (void)

Returns the gyrometer accuracy.

Returns

the gyrometer accuracy value

7.20.2.19 int INV_EXPORT inv_icm20648_get_mag_accuracy (void)

Returns the magnetometer accuracy.

Returns

the magnetometer accuracy value

7.20.2.20 int INV_EXPORT inv_icm20648_get_rv_accuracy (void)

Returns the rotation vector accuracy.

Returns

the rotation vector accuracy value in Q29

7.20.2.21 int INV_EXPORT inv_icm20648_identify_interrupt (struct inv_icm20648 * *s*, short * *int_read*)

Identify the interrupt.

Parameters

in	*int_read*	pointer to the DMP interrupt status

Returns

0 on success, negative value on error.

7.20.2.22 int INV_EXPORT inv_icm20648_inv_decode_one_ivory_fifo_packet (struct inv_icm20648 * *s*, struct inv_fifo_decoded_t * *fd*, const unsigned char * *fifo_ptr*)

Decodes the fifo packet.

Parameters

in	*fifo_ptr*	pointer to the fifo data
in	*fd*	pointer to the fifo what contains the sensor data

Returns

0 on success, negative value on error.

7.20.2.23 int INV_EXPORT inv_icm20648_mpu_set_FIFO_RST_Diamond (struct inv_icm20648 * *s*, unsigned char *value*)

Resets the fifo.

Parameters

in	*value*	0=no, 1=yes

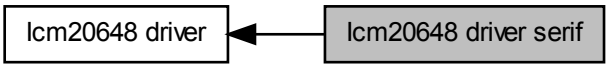
Returns

0 on success, negative value on error.

7.21 Icm20648 driver serif

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

Collaboration diagram for Icm20648 driver serif:



Classes

- struct inv_icm20648_serif

  ICM20648 serial interface.

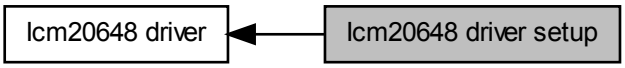
7.21.1 Detailed Description

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

7.22 Icm20648 driver setup

Low-level function to setup an Icm20648 device.

Collaboration diagram for Icm20648 driver setup:



Enumerations

Functions

- int INV_EXPORT inv_icm20648_set_lowpower_or_highperformance (struct inv_icm20648 *s, uint8_t lowpower_or_highperformance)

  Have the chip to enter low-power or low-noise mode.

7.22.1 Detailed Description

Low-level function to setup an Icm20648 device.

7.22.2 Function Documentation

7.22.2.1 int INV_EXPORT inv_icm20648_set_lowpower_or_highperformance (struct inv_icm20648 * *s*, uint8_t *lowpower_or_highperformance*)

Have the chip to enter low-power or low-noise mode.

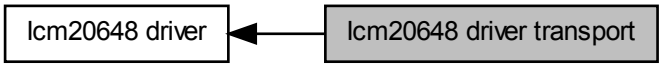
Parameters

in	*lowpower_or_highperformance*	0=low-power, 1=low-noise

7.23 Icm20648 driver transport

Low-level ICM20648 register access.

Collaboration diagram for Icm20648 driver transport:



Macros

- #define INV_MAX_SERIAL_READ 16

  Max size that can be read across I2C or SPI data lines.
- #define INV_MAX_SERIAL_WRITE 16

  Max size that can be written across I2C or SPI data lines.

Functions

- int INV_EXPORT inv_icm20648_write_mems_reg (struct inv_icm20648 ∗s, uint16_t reg, unsigned int length, const unsigned char ∗data)

  Write data to a register on MEMs.
- int INV_EXPORT inv_icm20648_write_single_mems_reg (struct inv_icm20648 ∗s, uint16_t reg, const unsigned char data)

  Write single byte of data to a register on MEMs.
- int INV_EXPORT inv_icm20648_read_mems_reg (struct inv_icm20648 ∗s, uint16_t reg, unsigned int length, unsigned char ∗data)

  Read data from a register on MEMs.
- int INV_EXPORT inv_icm20648_read_mems (struct inv_icm20648 ∗s, unsigned short reg, unsigned int length, unsigned char ∗data)

  Read data from a register in DMP memory.
- int INV_EXPORT inv_icm20648_write_mems (struct inv_icm20648 ∗s, unsigned short reg, unsigned int length, const unsigned char ∗data)

  Write data to a register in DMP memory.
- int INV_EXPORT inv_icm20648_write_single_mems_reg_core (struct inv_icm20648 ∗s, uint16_t reg, const uint8_t data)

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

7.23.1 Detailed Description

Low-level ICM20648 register access.

7.23.2 Function Documentation

7.23.2.1 int INV_EXPORT inv_icm20648_read_mems (struct inv_icm20648 ∗ *s*, unsigned short *reg*, unsigned int *length*, unsigned char ∗ *data*)

Read data from a register in DMP memory.

Parameters

in	*DMP*	memory address
in	*number*	of byte to be read
in	*input*	data from the register

Returns

0 if successful.

7.23.2.2 int INV_EXPORT inv_icm20648_read_mems_reg (struct inv_icm20648 * *s*, uint16_t *reg*, unsigned int *length*, unsigned char * *data*)

Read data from a register on MEMs.

Parameters

in	*Register*	address
in	*Length*	of data
in	*Data*	to be written

Returns

0 if successful.

7.23.2.3 int INV_EXPORT inv_icm20648_write_mems (struct inv_icm20648 * *s*, unsigned short *reg*, unsigned int *length*, const unsigned char * *data*)

Write data to a register in DMP memory.

Parameters

in	*DMP*	memory address
in	*number*	of byte to be written
out	*output*	data from the register

Returns

0 if successful.

7.23.2.4 int INV_EXPORT inv_icm20648_write_mems_reg (struct inv_icm20648 * *s*, uint16_t *reg*, unsigned int *length*, const unsigned char * *data*)

Write data to a register on MEMs.

Parameters

in	*Register*	address
in	*Length*	of data
in	*Data*	to be written

Returns

0 if successful.

7.23.2.5 int INV_EXPORT inv_icm20648_write_single_mems_reg (struct inv_icm20648 * *s*, uint16_t *reg*, const unsigned char *data*)

Write single byte of data to a register on MEMs.

Parameters

in	*Register*	address
in	*Data*	to be written

Returns

0 if successful.

7.23.2.6 int INV_EXPORT inv_icm20648_write_single_mems_reg_core (struct inv_icm20648 * *s*, uint16_t *reg*, const uint8_t *data*)

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

Parameters

in	*reg*	DMP memory address
out	*data*	Data to be written

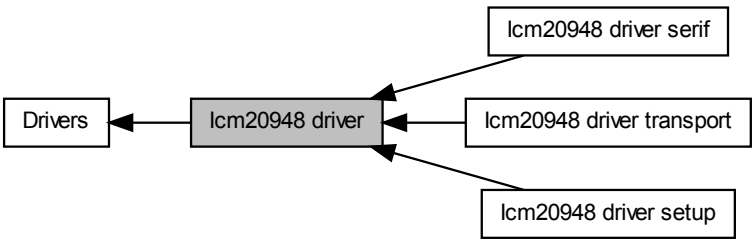
Returns

0 in case of success, -1 for any error

7.24 Icm20948 driver

Low-level driver for ICM20948 devices.

Collaboration diagram for Icm20948 driver:



Modules

- Icm20948 driver serif

  Interface for low-level serial (I2C/SPI) access.
- Icm20948 driver setup

  Low-level function to setup an Icm20948 device.
- Icm20948 driver transport

  Low-level ICM20948 register access.

Classes

- struct sensor_type_icm20948

  ICM20948 driver states definition.
- struct inv_icm20948

Typedefs

- typedef enum inv_icm20948_compass_state inv_icm20948_compass_state_t

  States for the secondary device.
- typedef struct sensor_type_icm20948 sensor_type_icm20948_t

  ICM20948 driver states definition.

Enumerations

Functions

- void inv_icm20948_sleep_us (int us)

  Hook for low-level system sleep() function to be implemented by upper layer.
- uint64_t inv_icm20948_get_time_us (void)

  Hook for low-level system time() function to be implemented by upper layer.
- static void inv_icm20948_reset_states (struct inv_icm20948 *s, const struct inv_icm20948_serif *serif)

  Reset and initialize driver states.

Variables

- struct inv_icm20948 * icm20948_instance

  ICM20948 driver states singleton declaration Because of Low-level driver limitation only one insance of the driver is allowed.

7.24.1 Detailed Description

Low-level driver for ICM20948 devices.

7.24.2 Function Documentation

7.24.2.1 uint64_t inv_icm20948_get_time_us (void)

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

Returns

monotonic timestamp in us

7.24.2.2 static void inv_icm20948_reset_states (struct inv_icm20948 * *s*, const struct inv_icm20948_serif * *serif*) [inline], [static]

Reset and initialize driver states.

Parameters

in	*s*	handle to driver states structure

7.24.2.3 void inv_icm20948_sleep_us (int *us*)

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

Parameters

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

7.25 augmented_sensors

Functions

- int INV_EXPORT inv_icm20948_augmented_init (struct inv_icm20948 *s)

  Initialize structure values.
- int INV_EXPORT inv_icm20948_augmented_sensors_get_gravity (struct inv_icm20948 *s, long gravity[3], const long quat6axis_3e[3])

  Gets the 3 axis gravity value based on GRV quaternion.
- int INV_EXPORT inv_icm20948_augmented_sensors_get_linearacceleration (long linacc[3], const long gravity[3], const long accel[3])

  Gets the 3 axis linear acceleration value based on gravity and accelerometer values.
- int INV_EXPORT inv_icm20948_augmented_sensors_get_orientation (long orientation[3], const long quat9axis_3e[4])

  Gets the 3 axis orientation value based on RV quaternion.
- unsigned short INV_EXPORT inv_icm20948_augmented_sensors_set_odr (struct inv_icm20948 *s, unsigned char androidSensor, unsigned short delayInMs)

  Set ODR for one of the augmented sensor-related Android sensor.
- void INV_EXPORT inv_icm20948_augmented_sensors_update_odr (struct inv_icm20948 *s, unsigned char androidSensor, unsigned short *updatedDelayPtr)

  Update ODR when an augmented sensor-related Android sensor was enabled or disable with ODR unchanged.

7.25.1 Detailed Description

7.25.2 Function Documentation

7.25.2.1 int INV_EXPORT inv_icm20948_augmented_init (struct inv_icm20948 * *s*)

Initialize structure values.

Parameters

in	*base*	state structre

7.25.2.2 int INV_EXPORT inv_icm20948_augmented_sensors_get_gravity (struct inv_icm20948 * *s*, long *gravity[3]*, const long *quat6axis_3e[3]*)

Gets the 3 axis gravity value based on GRV quaternion.

Parameters

out	*gravity*	3 components resulting gravity in Q16 in m/s2
in	*quat*	3 components input AG-based quaternion in Q30

Returns

0 in case of success, -1 for any error

7.25.2.3 int INV_EXPORT inv_icm20948_augmented_sensors_get_linearacceleration (long *linacc[3],* const long *gravity[3],* const long *accel[3]*)

Gets the 3 axis linear acceleration value based on gravity and accelerometer values.

Parameters

out	*linacc*	3 components resulting linear acceleration in Q16 in m/s2
in	*gravity*	3 components gravity in Q16 in m/s2
in	*accel*	3 components acceleration in Q16 in m/s2

Returns

0 in case of success, -1 for any error

7.25.2.4 int INV_EXPORT inv_icm20948_augmented_sensors_get_orientation (long *orientation[3],* const long *quat9axis_3e[4]*)

Gets the 3 axis orientation value based on RV quaternion.

Parameters

out	*orientation*	3 components resulting orientation in Q16 in degrees The x field is azimuth, the angle between the magnetic north direction and the y axis around the the z axis. The y field is pitch, the rotation arounf x axis, positive when the z axis moves toward the y axis. The z field is roll, the rotation arount the y axis, positive when the x axis moves toward the z axis.
in	*quat9axis_3e*	3 components input AGM-based quaternion in Q30

Returns

0 in case of success, -1 for any error

7.25.2.5 unsigned short INV_EXPORT inv_icm20948_augmented_sensors_set_odr (struct inv_icm20948 * *s,* unsigned char *androidSensor,* unsigned short *delayInMs*)

Set ODR for one of the augmented sensor-related Android sensor.

Parameters

in	*androidSensor*	Android sensor ID for which a new delay in to be applied
in	*delayInMs*	the new delay in ms requested for androidSensor

Returns

the delay in ms to be applied to quat6 output

7.25.2.6 void INV_EXPORT inv_icm20948_augmented_sensors_update_odr (struct inv_icm20948 * *s*, unsigned char *androidSensor*, unsigned short * *updatedDelayPtr*)

Update ODR when an augmented sensor-related Android sensor was enabled or disable with ODR unchanged.

Parameters

in	*androidSensor*	Android sensor ID for which status was updated
out	*updatedDelayPtr*	Handler where should be written new delay to be applied

Returns

None

7.26 inv_slave_compass

Enumerations

Functions

- void INV_EXPORT inv_icm20948_register_aux_compass (struct inv_icm20948 *s, enum inv_icm20948_compass_id compass_id, uint8_t compass_i2c_addr)

  Register AUX compass.
- int INV_EXPORT inv_icm20948_setup_compass_akm (struct inv_icm20948 *s)

  Initializes the compass.
- int INV_EXPORT inv_icm20948_check_akm_self_test (struct inv_icm20948 *s)

  Self test for the compass.
- int INV_EXPORT inv_icm20948_write_akm_scale (struct inv_icm20948 *s, int data)

  Changes the scale of the compass.
- int INV_EXPORT inv_icm20948_read_akm_scale (struct inv_icm20948 *s, int *scale)

  Reads the scale of the compass.
- int INV_EXPORT inv_icm20948_suspend_akm (struct inv_icm20948 *s)

  Stops the compass.
- int INV_EXPORT inv_icm20948_resume_akm (struct inv_icm20948 *s)

  Starts the compass.
- char INV_EXPORT inv_icm20948_compass_getstate (struct inv_icm20948 *s)

  Get compass power status.
- int INV_EXPORT inv_icm20948_compass_isconnected (struct inv_icm20948 *s)

  detects if the compass is connected
- int INV_EXPORT inv_icm20948_compass_dmp_cal (struct inv_icm20948 *s, const signed char *m, const signed char *compass_m)

  Calibrates the data.
- int INV_EXPORT inv_icm20948_apply_raw_compass_matrix (struct inv_icm20948 *s, short *raw_data, long *compensated_out)

  Applies mounting matrix and scaling to raw compass data.

7.26.1 Detailed Description

7.26.2 Enumeration Type Documentation

7.26.2.1 enum inv_icm20948_compass_id

Supported auxiliary compass identifer.

Enumerator

INV_ICM20948_COMPASS_ID_NONE	no compass
INV_ICM20948_COMPASS_ID_AK09911	AKM AK09911.
INV_ICM20948_COMPASS_ID_AK09912	AKM AK09912.
INV_ICM20948_COMPASS_ID_AK09916	AKM AK09916.
INV_ICM20948_COMPASS_ID_AK08963	AKM AK08963.

7.26.3 Function Documentation

7.26.3.1 int INV_EXPORT inv_icm20948_apply_raw_compass_matrix (struct inv_icm20948 * *s*, short * *raw_data*, long * *compensated_out*)

Applies mounting matrix and scaling to raw compass data.

Parameters

in	*raw_data*	Raw compass data
in	*compensated_out*	Compensated compass data

Returns

0 in case of success, -1 for any error

7.26.3.2 int INV_EXPORT inv_icm20948_check_akm_self_test (struct inv_icm20948 * s)

Self test for the compass.

Returns

0 in case of success, -1 for any error

7.26.3.3 int INV_EXPORT inv_icm20948_compass_dmp_cal (struct inv_icm20948 * s, const signed char * m, const signed char * compass_m)

Calibrates the data.

Parameters

in	*m*	pointer to the raw compass data
out	*compass_m*	pointer to the calibrated compass data

Returns

0 in case of success, -1 for any error

7.26.3.4 char INV_EXPORT inv_icm20948_compass_getstate (struct inv_icm20948 * s)

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

7.26.3.5 int INV_EXPORT inv_icm20948_compass_isconnected (struct inv_icm20948 * s)

detects if the compass is connected

Returns

1 if the compass is connected, 0 otherwise

7.26.3.6 int INV_EXPORT inv_icm20948_read_akm_scale (struct inv_icm20948 * 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

7.26.3.7 void INV_EXPORT inv_icm20948_register_aux_compass (struct inv_icm20948 * s, enum inv_icm20948_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_icm20948_↩ initialize().

Parameters

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

Returns

0 on success, negative value on error

7.26.3.8 int INV_EXPORT inv_icm20948_resume_akm (struct inv_icm20948 * s)

Starts the compass.

Returns

0 in case of success, -1 for any error

7.26.3.9 int INV_EXPORT inv_icm20948_setup_compass_akm (struct inv_icm20948 * s)

Initializes the compass.

Returns

0 in case of success, -1 for any error

7.26.3.10 int INV_EXPORT inv_icm20948_suspend_akm (struct inv_icm20948 * *s*)

Stops the compass.

Returns

0 in case of success, -1 for any error

7.26.3.11 int INV_EXPORT inv_icm20948_write_akm_scale (struct inv_icm20948 * *s*, int *data*)

Changes the scale of the compass.

Parameters

in	*data*	new scale for the compass

Returns

0 in case of success, -1 for any error

7.27 inv_secondary_transport

Macros

- #define COMPASS_I2C_SLV_READ 0

  I2C from secondary device can stand on up to 4 channels.

Functions

- void INV_EXPORT inv_icm20948_init_secondary (struct inv_icm20948 *s)

  Initializes the register for the i2c communication.
- int INV_EXPORT inv_icm20948_read_secondary (struct inv_icm20948 *s, int index, unsigned char addr, unsigned char reg, char len)

  Reads data in i2c a secondary device.
- int INV_EXPORT inv_icm20948_execute_read_secondary (struct inv_icm20948 *s, int index, unsigned char addr, int reg, int len, uint8_t *d)

  Reads data in i2c a secondary device directly.
- int INV_EXPORT inv_icm20948_write_secondary (struct inv_icm20948 *s, int index, unsigned char addr, unsigned char reg, char v)

  Writes data in i2c a secondary device.
- int INV_EXPORT inv_icm20948_execute_write_secondary (struct inv_icm20948 *s, int index, unsigned char addr, int reg, uint8_t v)

  Writes data in i2c a secondary device directly.
- void INV_EXPORT inv_icm20948_secondary_saveI2cOdr (struct inv_icm20948 *s)

  Save current secondary I2C ODR configured.
- void INV_EXPORT inv_icm20948_secondary_restoreI2cOdr (struct inv_icm20948 *s)

  Restore secondary I2C ODR configured based on the one saved with inv_icm20948_secondary_saveI2cOdr()
- int INV_EXPORT inv_icm20948_secondary_stop_channel (struct inv_icm20948 *s, int index)

  Stop one secondary I2C channel by writing 0 in its control register.
- int INV_EXPORT inv_icm20948_secondary_enable_i2c (struct inv_icm20948 *s)

  Enable secondary I2C interface.
- int INV_EXPORT inv_icm20948_secondary_disable_i2c (struct inv_icm20948 *s)

  Stop secondary I2C interface.
- int INV_EXPORT inv_icm20948_secondary_set_odr (struct inv_icm20948 *s, int divider, unsigned int *effectiveDivider)

  Changes the odr of the I2C master.

7.27.1 Detailed Description

7.27.2 Macro Definition Documentation

7.27.2.1 #define COMPASS_I2C_SLV_READ 0

I2C from secondary device can stand on up to 4 channels.

To perform automatic read and feed DMP :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register
- channel 2 is reserved for als reading data

7.27.3 Function Documentation

7.27.3.1 int INV_EXPORT inv_icm20948_execute_read_secondary (struct inv_icm20948 * s, int index, unsigned char addr, int reg, int len, uint8_t * 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

7.27.3.2 int INV_EXPORT inv_icm20948_execute_write_secondary (struct inv_icm20948 * s, int index, unsigned char addr, int reg, uint8_t 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

7.27.3.3 int INV_EXPORT inv_icm20948_read_secondary (struct inv_icm20948 * 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

7.27.3.4 int INV_EXPORT inv_icm20948_secondary_disable_i2c (struct inv_icm20948 * *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

7.27.3.5 int INV_EXPORT inv_icm20948_secondary_enable_i2c (struct inv_icm20948 * *s*)

Enable secondary I2C interface.

Returns

0 in case of success, -1 for any error

7.27.3.6 int INV_EXPORT inv_icm20948_secondary_set_odr (struct inv_icm20948 * *s*, int *divider*, unsigned int * *effectiveDivider*)

Changes the odr of the I2C master.

Parameters

in	*divider*	frequency divider to BASE_SAMPLE_RATE
out	*effectiveDivider*	divider finally applied to base sample rate, at which data will be actually read on I2C bus

Returns

0 in case of success, -1 for any error

7.27.3.7 int INV_EXPORT inv_icm20948_secondary_stop_channel (struct inv_icm20948 * *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

7.27.3.8 int INV_EXPORT inv_icm20948_write_secondary (struct inv_icm20948 * *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

7.28 base_control

Enumerations

Functions

- int INV_EXPORT inv_icm20948_base_control_init (struct inv_icm20948 *s)

  Initialize structure values.
- int INV_EXPORT inv_icm20948_set_odr (struct inv_icm20948 *s, unsigned char androidSensor, unsigned short delayInMs)

  Sets the odr for a sensor.
- int INV_EXPORT inv_icm20948_ctrl_enable_sensor (struct inv_icm20948 *s, unsigned char androidSensor, unsigned char enable)

  Enables / disables a sensor.
- int INV_EXPORT inv_icm20948_ctrl_enable_batch (struct inv_icm20948 *s, unsigned char enable)

  Enables / disables batch for the sensors.
- void INV_EXPORT inv_icm20948_ctrl_set_batch_mode_status (struct inv_icm20948 *s, unsigned char enable)

  Set batch mode status.
- unsigned char INV_EXPORT inv_icm20948_ctrl_get_batch_mode_status (struct inv_icm20948 *s)

  Get batch mode status.
- int INV_EXPORT inv_icm20948_ctrl_set_batch_timeout (struct inv_icm20948 *s, unsigned short batch_↩ time_in_seconds)

  Sets the timeout for the batch in second.
- int INV_EXPORT inv_icm20948_ctrl_set_batch_timeout_ms (struct inv_icm20948 *s, unsigned short batch↩ _time_in_ms)

  Sets the timeout for the batch in millisecond.
- void INV_EXPORT inv_icm20948_ctrl_enable_activity_classifier (struct inv_icm20948 *s, unsigned char enable)

  Enables / disables BAC.
- void INV_EXPORT inv_icm20948_ctrl_enable_tilt (struct inv_icm20948 *s, unsigned char enable)

  Enables / disables tilt.
- void INV_EXPORT inv_icm20948_ctrl_enable_b2s (unsigned char enable)

  Enables / disables bring to see.
- unsigned long INV_EXPORT * inv_icm20948_ctrl_get_androidSensorsOn_mask (struct inv_icm20948 *s)

  Returns the mask for the different sensors enabled.
- unsigned long INV_EXPORT inv_icm20948_ctrl_androidSensor_enabled (struct inv_icm20948 *s, unsigned char androidSensor)

  Check if a sensor is enabled.
- unsigned short INV_EXPORT inv_icm20948_ctrl_get_activitiy_classifier_on_flag (struct inv_icm20948 *s)

  Returns a flag to know if the BAC is running.
- int INV_EXPORT inv_icm20948_ctrl_get_odr (struct inv_icm20948 *s, unsigned char SensorId, uint32_↩ t *odr, enum INV_ODR_TYPE odr_units)

  Gets the odr for a sensor.
- int INV_EXPORT inv_icm20948_ctrl_set_accel_quaternion_gain (struct inv_icm20948 *s, unsigned short hw_smplrt_divider)

  Sets accel quaternion gain according to accel engine rate.
- int INV_EXPORT inv_icm20948_ctrl_set_accel_cal_params (struct inv_icm20948 *s, unsigned short hw_↩ smplrt_divider)

  Sets accel cal parameters according to accel engine rate.
- int INV_EXPORT inv_icm20948_ctrl_enable_pickup (struct inv_icm20948 *s, unsigned char enable)

Enables / disables pickup gesture.

- int inv_icm20948_ctrl_get_acc_bias (struct inv_icm20948 *s, int *acc_bias)

  get acc bias from dmp driver
- int inv_icm20948_ctrl_get_gyr_bias (struct inv_icm20948 *s, int *gyr_bias)

  get gyr bias from dmp driver
- int INV_EXPORT inv_icm20948_ctrl_get_mag_bias (struct inv_icm20948 *s, int *mag_bias)

  get mag bias from dmp driver
- int inv_icm20948_ctrl_set_acc_bias (struct inv_icm20948 *s, int *acc_bias)

  set acc bias from dmp driver
- int inv_icm20948_ctrl_set_gyr_bias (struct inv_icm20948 *s, int *gyr_bias)

  set gyr bias from dmp driver
- int INV_EXPORT inv_icm20948_ctrl_set_mag_bias (struct inv_icm20948 *s, int *mag_bias)

  set mag bias from dmp driver

7.28.1 Detailed Description

7.28.2 Function Documentation

7.28.2.1 int INV_EXPORT inv_icm20948_base_control_init (struct inv_icm20948 * *s*)

Initialize structure values.

Parameters

in	*base*	state structre

7.28.2.2 unsigned long INV_EXPORT inv_icm20948_ctrl_androidSensor_enabled (struct inv_icm20948 * *s*, unsigned char *androidSensor*)

Check if a sensor is enabled.

Returns

1 if sensor is enabled

7.28.2.3 void INV_EXPORT inv_icm20948_ctrl_enable_activity_classifier (struct inv_icm20948 * *s*, unsigned char *enable*)

Enables / disables BAC.

Parameters

in	*enable*	0=off, 1=on

7.28.2.4 void INV_EXPORT inv_icm20948_ctrl_enable_b2s (unsigned char *enable*)

Enables / disables bring to see.

Parameters

in	*enable*	0=off, 1=on

7.28.2.5 int INV_EXPORT inv_icm20948_ctrl_enable_batch (struct inv_icm20948 * *s,* unsigned char *enable*)

Enables / disables batch for the sensors.

Parameters

in	*enable*	0=off, 1=on

Returns

0 in case of success, -1 for any error

7.28.2.6 int INV_EXPORT inv_icm20948_ctrl_enable_pickup (struct inv_icm20948 * *s,* unsigned char *enable*)

Enables / disables pickup gesture.

Parameters

7.28.2.7 int INV_EXPORT inv_icm20948_ctrl_enable_sensor (struct inv_icm20948 * *s,* unsigned char *androidSensor,* unsigned char *enable*)

Enables / disables a sensor.

Parameters

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

Returns

0 in case of success, -1 for any error

7.28.2.8 void INV_EXPORT inv_icm20948_ctrl_enable_tilt (struct inv_icm20948 * *s,* unsigned char *enable*)

Enables / disables tilt.

Parameters

in	*enable*	0=off, 1=on

7.28.2.9 int inv_icm20948_ctrl_get_acc_bias (struct inv_icm20948 * *s*, int * *acc_bias*)

get acc bias from dmp driver

Parameters

7.28.2.10 unsigned short INV_EXPORT inv_icm20948_ctrl_get_activitiy_classifier_on_flag (struct inv_icm20948 * *s*)

Returns a flag to know if the BAC is running.

Returns

1 if started, 0 if stopped

7.28.2.11 unsigned long INV_EXPORT* inv_icm20948_ctrl_get_androidSensorsOn_mask (struct inv_icm20948 * *s*)

Returns the mask for the different sensors enabled.

Returns

the mask

7.28.2.12 unsigned char INV_EXPORT inv_icm20948_ctrl_get_batch_mode_status (struct inv_icm20948 * *s*)

Get batch mode status.

Returns

0=batch mode disable, 1=batch mode enable

7.28.2.13 int inv_icm20948_ctrl_get_gyr_bias (struct inv_icm20948 * *s*, int * *gyr_bias*)

get gyr bias from dmp driver

Parameters

7.28.2.14 int INV_EXPORT inv_icm20948_ctrl_get_mag_bias (struct inv_icm20948 * s, int * mag_bias)

get mag bias from dmp driver

Parameters

7.28.2.15 int INV_EXPORT inv_icm20948_ctrl_get_odr (struct inv_icm20948 * s, unsigned char SensorId, uint32_t * odr, enum INV_ODR_TYPE odr_units)

Gets the odr for a sensor.

Parameters

in	*SensorId*	Sensor Identity
out	*odr*	pointer to the ODR for this sensor
in	*odr_units*	unit expected for odr, one of INV_ODR_TYPE

Returns

0 in case of success, -1 for any error

7.28.2.16 int inv_icm20948_ctrl_set_acc_bias (struct inv_icm20948 * s, int * acc_bias)

set acc bias from dmp driver

Parameters

7.28.2.17 int INV_EXPORT inv_icm20948_ctrl_set_accel_cal_params (struct inv_icm20948 * s, unsigned short hw_smplrt_divider)

Sets accel cal parameters according to accel engine rate.

Parameters

in	*hw_smplrt_divider*	hardware sample rate divider such that accel engine rate = 1125Hz/hw_smplrt_divider

Returns

0 in case of success, -1 for any error

7.28.2.18 int INV_EXPORT inv_icm20948_ctrl_set_accel_quaternion_gain (struct inv_icm20948 * s, unsigned short hw_smplrt_divider)

Sets accel quaternion gain according to accel engine rate.

Parameters

in	*hw_smplrt_divider*	hardware sample rate divider such that accel engine rate = 1125Hz/hw_smplrt_divider

Returns

0 in case of success, -1 for any error

7.28.2.19 void INV_EXPORT inv_icm20948_ctrl_set_batch_mode_status (struct inv_icm20948 * s, unsigned char enable)

Set batch mode status.

Parameters

in	*enable*	0=off, 1=on

7.28.2.20 int INV_EXPORT inv_icm20948_ctrl_set_batch_timeout (struct inv_icm20948 * s, unsigned short batch_time_in_seconds)

Sets the timeout for the batch in second.

Parameters

in	*batch_time_in_seconds*	time in second

Returns

0 in case of success, -1 for any error

7.28.2.21 int INV_EXPORT inv_icm20948_ctrl_set_batch_timeout_ms (struct inv_icm20948 * s, unsigned short batch_time_in_ms)

Sets the timeout for the batch in millisecond.

Parameters

in	*batch_time_in_ms*	time in millisecond

Returns

0 in case of success, -1 for any error

7.28.2.22 int inv_icm20948_ctrl_set_gyr_bias (struct inv_icm20948 * *s,* int * *gyr_bias*)

set gyr bias from dmp driver

Parameters

7.28.2.23 int INV_EXPORT inv_icm20948_ctrl_set_mag_bias (struct inv_icm20948 * *s,* int * *mag_bias*)

set mag bias from dmp driver

Parameters

7.28.2.24 int INV_EXPORT inv_icm20948_set_odr (struct inv_icm20948 * *s,* unsigned char *androidSensor,* unsigned short *delayInMs*)

Sets the odr for a sensor.

Parameters

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

Returns

0 in case of success, -1 for any error

7.29 base_driver

Functions

- int INV_EXPORT inv_icm20948_initialize_lower_driver (struct inv_icm20948 *s, enum SMARTSENSOR_↩ SERIAL_INTERFACE type, const uint8_t *dmp3_image, uint32_t dmp3_image_size)

  Initializes the platform.
- int INV_EXPORT inv_icm20948_set_slave_compass_id (struct inv_icm20948 *s, int id)

  Initializes the compass and the id address.
- int INV_EXPORT inv_icm20948_set_serial_comm (struct inv_icm20948 *s, enum SMARTSENSOR_SER↩ IAL_INTERFACE type)

  Selects the interface of communication with the board.
- int INV_EXPORT inv_icm20948_wakeup_mems (struct inv_icm20948 *s)

  Wakes up mems platform.
- int INV_EXPORT inv_icm20948_sleep_mems (struct inv_icm20948 *s)

  Sleeps up mems platform.
- int INV_EXPORT inv_icm20948_set_chip_power_state (struct inv_icm20948 *s, unsigned char func, unsigned char on_off)

  Sets the power state of the Ivory chip loop.
- uint8_t INV_EXPORT inv_icm20948_get_chip_power_state (struct inv_icm20948 *s)

  Current wake status of the Mems chip.
- int INV_EXPORT inv_icm20948_set_dmp_address (struct inv_icm20948 *s)

  Sets up dmp start address and firmware.
- int INV_EXPORT inv_icm20948_set_secondary (struct inv_icm20948 *s)

  Sets up the secondary i2c bus.
- int INV_EXPORT inv_icm20948_enable_hw_sensors (struct inv_icm20948 *s, int bit_mask)

  Enables accel and/or gyro and/or pressure if integrated with gyro and accel.
- int INV_EXPORT inv_icm20948_set_gyro_sf (struct inv_icm20948 *s, unsigned char div, int gyro_level)

  Sets the dmp for a particular gyro configuration.
- int INV_EXPORT inv_icm20948_set_gyro_divider (struct inv_icm20948 *s, unsigned char div)

  Sets the gyro sample rate.
- unsigned char INV_EXPORT inv_icm20948_get_gyro_divider (struct inv_icm20948 *s)

  Returns the gyro sample rate.
- uint32_t INV_EXPORT inv_icm20948_get_odr_in_units (struct inv_icm20948 *s, unsigned short odrIn↩ Divider, unsigned char odr_units)

  Returns the real odr in Milliseconds, Micro Seconds or Ticks.
- int INV_EXPORT inv_icm20948_set_accel_divider (struct inv_icm20948 *s, short div)

  Sets the accel sample rate.
- short INV_EXPORT inv_icm20948_get_accel_divider (struct inv_icm20948 *s)

  Returns the accel sample rate.
- int INV_EXPORT inv_icm20948_set_secondary_divider (struct inv_icm20948 *s, unsigned char div)

  Sets the I2C secondary device sample rate.
- unsigned short INV_EXPORT inv_icm20948_get_secondary_divider (struct inv_icm20948 *s)

  Returns the I2C secondary device sample rate.
- int INV_EXPORT inv_icm20948_set_gyro_fullscale (struct inv_icm20948 *s, int level)

  Sets fullscale range of gyro in hardware.
- uint8_t INV_EXPORT inv_icm20948_get_gyro_fullscale (struct inv_icm20948 *s)

  Returns fullscale range of gyrometer in hardware.
- int INV_EXPORT inv_icm20948_set_icm20948_gyro_fullscale (struct inv_icm20948 *s, int level)

  Sets fullscale range of gyro in hardware.
- int INV_EXPORT inv_icm20948_set_accel_fullscale (struct inv_icm20948 *s, int level)

Sets fullscale range of accel in hardware.

- uint8_t INV_EXPORT inv_icm20948_get_accel_fullscale (struct inv_icm20948 *s)

  Returns fullscale range of accelerometer in hardware.
- int INV_EXPORT inv_icm20948_set_icm20948_accel_fullscale (struct inv_icm20948 *s, int level)

  Sets fullscale range of accel in hardware.
- int INV_EXPORT inv_icm20948_set_int1_assertion (struct inv_icm20948 *s, int enable)

  Asserts int1 interrupt when DMP execute INT1 cmd.
- int INV_EXPORT inv_icm20948_accel_read_hw_reg_data (struct inv_icm20948 *s, short accel_hw_reg_↩ data[3])

  Reads accelerometer data stored in hardware register.
- void INV_EXPORT inv_icm20948_prevent_lpen_control (struct inv_icm20948 *s)

  Prevent LP_EN from being set to 1 again, this speeds up transaction.
- void INV_EXPORT inv_icm20948_allow_lpen_control (struct inv_icm20948 *s)

  Allow LP_EN to be set to 1 again and sets it to 1 again if supported by chip.
- int INV_EXPORT inv_icm20948_get_compass_availability (struct inv_icm20948 *s)

  Determine if compass could be successfully found and inited on board.
- int INV_EXPORT inv_icm20948_get_pressure_availability (struct inv_icm20948 *s)

  Determine if pressure could be successfully found and inited on board.
- int INV_EXPORT inv_icm20948_get_proximity_availability (struct inv_icm20948 *s)

  Determine if proximity could be successfully found and inited on board.
- int INV_EXPORT inv_icm20948_enter_duty_cycle_mode (struct inv_icm20948 *s)

  Have the chip to enter stand duty cycled mode, also called low-power mode where max reporting frequency is 562Hz.
- int INV_EXPORT inv_icm20948_enter_low_noise_mode (struct inv_icm20948 *s)

  Have the chip to enter low-noise mode.

7.29.1 Detailed Description

7.29.2 Function Documentation

7.29.2.1 int INV_EXPORT inv_icm20948_accel_read_hw_reg_data (struct inv_icm20948 * *s*, short *accel_hw_reg_data[3]*)

Reads accelerometer data stored in hardware register.

Parameters

in	*accel_hw_reg_data*	variable to be recuperated the accelerometer data

Returns

0 on success, negative value on error.

7.29.2.2 int INV_EXPORT inv_icm20948_enable_hw_sensors (struct inv_icm20948 * *s*, int *bit_mask*)

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.

Returns

0 on success, negative value on error.

7.29.2.3 short INV_EXPORT inv_icm20948_get_accel_divider (struct inv_icm20948 * *s*)

Returns the accel sample rate.

Returns

the divider for the accel

7.29.2.4 uint8_t INV_EXPORT inv_icm20948_get_accel_fullscale (struct inv_icm20948 * *s*)

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.29.2.5 uint8_t INV_EXPORT inv_icm20948_get_chip_power_state (struct inv_icm20948 * *s*)

Current wake status of the Mems chip.

Returns

the wake status

7.29.2.6 int INV_EXPORT inv_icm20948_get_compass_availability (struct inv_icm20948 * *s*)

Determine if compass could be successfully found and inited on board.

Returns

1 on success, 0 if not available.

7.29.2.7 unsigned char INV_EXPORT inv_icm20948_get_gyro_divider (struct inv_icm20948 * s)

Returns the gyro sample rate.

Returns

Value written to GYRO_SMPLRT_DIV register.

7.29.2.8 uint8_t INV_EXPORT inv_icm20948_get_gyro_fullscale (struct inv_icm20948 * s)

Returns fullscale range of gyrometer in hardware.

Returns

the fullscale range

7.29.2.9 uint32_t INV_EXPORT inv_icm20948_get_odr_in_units (struct inv_icm20948 * s, unsigned short *odrInDivider*, unsigned char *odr_units*)

Returns the real odr in Milliseconds, Micro Seconds or Ticks.

Parameters

in	*odrInDivider*	Odr In divider
in	*odr_units*	Use the enum values: ODR_IN_Ms, ODR_IN_Us or ODR_IN_Ticks

Returns

Odr in fucntion of enum.

7.29.2.10 int INV_EXPORT inv_icm20948_get_pressure_availability (struct inv_icm20948 * s)

Determine if pressure could be successfully found and inited on board.

Returns

1 on success, 0 if not available.

7.29.2.11 int INV_EXPORT inv_icm20948_get_proximity_availability (struct inv_icm20948 * s)

Determine if proximity could be successfully found and inited on board.

Returns

1 on success, 0 if not available.

7.29.2.12 unsigned short INV_EXPORT inv_icm20948_get_secondary_divider (struct inv_icm20948 * s)

Returns the I2C secondary device sample rate.

Returns

the divider for the I2C secondary device interface

7.29.2.13 int INV_EXPORT inv_icm20948_initialize_lower_driver (struct inv_icm20948 * s, enum SMARTSENSOR_SERIAL_INTERFACE type, const uint8_t * dmp3_image, uint32_t dmp3_image_size)

Initializes the platform.

Parameters

in	*type*	Define the interface for communicate : SERIAL_INTERFACE_I2C or SERIAL_INTERFACE_SPI
out	*dmp_image_sram*	4 The image to be load

Returns

0 on success, negative value on error.

7.29.2.14 int INV_EXPORT inv_icm20948_set_accel_divider (struct inv_icm20948 * s, short div)

Sets the accel sample rate.

Parameters

in	*div*	Value written to ACCEL_SMPLRT_DIV register

Returns

0 on success, negative value on error.

7.29.2.15 int INV_EXPORT inv_icm20948_set_accel_fullscale (struct inv_icm20948 * 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.

7.29.2.16 int INV_EXPORT inv_icm20948_set_chip_power_state (struct inv_icm20948 * s, unsigned char func, unsigned char 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.

7.29.2.17 int INV_EXPORT inv_icm20948_set_dmp_address (struct inv_icm20948 * s)

Sets up dmp start address and firmware.

Returns

0 on success, negative value on error.

7.29.2.18 int INV_EXPORT inv_icm20948_set_gyro_divider (struct inv_icm20948 * s, unsigned char div)

Sets the gyro sample rate.

Parameters

in	*div*	Value written to GYRO_SMPLRT_DIV register

Returns

0 on success, negative value on error.

7.29.2.19 int INV_EXPORT inv_icm20948_set_gyro_fullscale (struct inv_icm20948 * 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.

7.29.2.20 int INV_EXPORT inv_icm20948_set_gyro_sf (struct inv_icm20948 * s, unsigned char div, int gyro_level)

Sets the dmp for a particular gyro configuration.

Parameters

in	*gyro_div*	Value written to GYRO_SMPLRT_DIV register, where 0=1125Hz sample rate, 1=562.5Hz sample rate, ... 4=225Hz sample rate, ... 10=102.2727Hz sample rate, ... etc.
in	*gyro_level*	0=250 dps, 1=500 dps, 2=1000 dps, 3=2000 dps

Returns

0 on success, negative value on error.

7.29.2.21 int INV_EXPORT inv_icm20948_set_icm20948_accel_fullscale (struct inv_icm20948 * 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.

7.29.2.22 int INV_EXPORT inv_icm20948_set_icm20948_gyro_fullscale (struct inv_icm20948 * 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.

7.29.2.23 int INV_EXPORT inv_icm20948_set_int1_assertion (struct inv_icm20948 * s, int enable)

Asserts int1 interrupt when DMP execute INT1 cmd.

Parameters

in	*enable*	0=off, 1=on

Returns

0 on success, negative value on error.

7.29.2.24 int INV_EXPORT inv_icm20948_set_secondary (struct inv_icm20948 * *s*)

Sets up the secondary i2c bus.

Returns

0 on success, negative value on error.

7.29.2.25 int INV_EXPORT inv_icm20948_set_secondary_divider (struct inv_icm20948 * *s*, unsigned char *div*)

Sets the I2C secondary device sample rate.

Parameters

in	*div*	Value written to REG_I2C_MST_ODR_CONFIG register

Returns

0 on success, negative value on error.

7.29.2.26 int INV_EXPORT inv_icm20948_set_serial_comm (struct inv_icm20948 * *s*, enum SMARTSENSOR_SERIAL_INTERFACE *type*)

Selects the interface of communication with the board.

Parameters

in	*type*	Define the interface for communicate : SERIAL_INTERFACE_I2C or SERIAL_INTERFACE_SPI

Returns

0 on success, negative value on error.

7.29.2.27 int INV_EXPORT inv_icm20948_set_slave_compass_id (struct inv_icm20948 * *s*, int *id*)

Initializes the compass and the id address.

Parameters

in	*id*	address of compass component

Returns

0 on success, negative value on error.

7.29.2.28 int INV_EXPORT inv_icm20948_sleep_mems (struct inv_icm20948 * *s*)

Sleeps up mems platform.

Returns

0 on success, negative value on error.

7.29.2.29 int INV_EXPORT inv_icm20948_wakeup_mems (struct inv_icm20948 * *s*)

Wakes up mems platform.

Returns

0 on success, negative value on error.

7.30 data_converter

Macros

- #define ABS(x) (((x)>=0)?(x):-(x))

  Computes the absolute value of its argument x.
- #define MAX(x, y) (((x)>(y))?(x):(y))

  Computes the maximum of x and y.
- #define MIN(x, y) (((x)<(y))?(x):(y))

  Computes the minimum of x and y.
- #define INVN_FLT_TO_FXP(value, shift) ((int32_t) ((float)(value)*(1ULL << (shift)) + ((value>=0)-0.5f)))

  Convert the value from float to QN value.
- #define INVN_CONVERT_FLT_TO_FXP(fltptr, fixptr, length, shift) { int i; for(i=0; i<(length); ++i) (fixptr)[i] = INVN_FLT_TO_FXP((fltptr)[i], shift); }

  Macro to convert float values from an address into QN values, and copy them to another address.

Functions

- void INV_EXPORT inv_icm20948_q_mult_q_qi (const long *q1, const long *q2, long *qProd)

  *Performs a fixed point quaternion multiply with inverse on second element q1*q2'.*
- void INV_EXPORT inv_icm20948_set_chip_to_body (struct inv_icm20948 *s, long *quat)

  Sets the transformation used for chip to body frame.
- void INV_EXPORT inv_icm20948_convert_rotation_vector (struct inv_icm20948 *s, const long *quat, float *values)

  Converts fixed point DMP rotation vector to floating point android notation.
- void INV_EXPORT inv_icm20948_convert_rotation_vector_2 (struct inv_icm20948 *s, const long *quat, long *quat4_world)

  Converts 3 element fixed point DMP rotation vector to 4 element rotation vector in world frame.
- void INV_EXPORT inv_icm20948_convert_rotation_vector_3 (const long *quat4_world, float *values)

  Converts 4 element rotation vector in world frame to floating point android notation.
- void INV_EXPORT inv_icm20948_convert_dmp3_to_body (struct inv_icm20948 *s, const long *vec3, float scale, float *values)

  Converts the data in android values.
- void INV_EXPORT inv_icm20948_set_chip_to_body_axis_quaternion (struct inv_icm20948 *s, signed char *accel_gyro_matrix, float angle)

  Converts the data in android quaternion values.
- unsigned char INV_EXPORT * inv_icm20948_int32_to_little8 (long x, unsigned char *little8)

  Converts a 32-bit long to a little endian byte stream.
- float INV_EXPORT inv_icm20948_convert_deg_to_rad (float deg_val)

  Converts degree angle to radian.
- long INV_EXPORT inv_icm20948_convert_mult_q30_fxp (long a_q30, long b_q30)

  Performs a multiply and shift by 30.
- int INV_EXPORT inv_icm20948_convert_compute_scalar_part_fxp (const long *inQuat_q30, long *out↩ Quat_q30)

  Compute real part of quaternion, element[0].
- long INV_EXPORT inv_icm20948_convert_fast_sqrt_fxp (long x0_q30)

  Calculates square-root of a fixed-point number (30 bit mantissa, positive)
- int INV_EXPORT inv_icm20948_convert_test_limits_and_scale_fxp (long *x0_q30, int *pow)

  Auxiliary function used by inv_OneOverX(), inv_fastSquareRoot(), inv_inverseSqrt().
- int16_t INV_EXPORT inv_icm20948_convert_get_highest_bit_position (uint32_t *value)

Auxiliary function used by testLimitsAndScale() Find the highest nonzero bit in an unsigned 32 bit integer:

- void INV_EXPORT inv_icm20948_convert_matrix_to_quat_fxp (long *Rcb_q30, long *Qcb_q30)
  Converts a rotation matrix to a quaternion.
- long INV_EXPORT inv_icm20948_convert_sqrt_q30_fxp (long x_q30)
  Calculates square-root of a fixed-point number.
- long INV_EXPORT inv_icm20948_convert_inv_sqrt_q30_fxp (long x_q30, int *pow2)
  Calculates 1/square-root of a fixed-point number (30 bit mantissa, positive): Q1.30.
- long INV_EXPORT inv_icm20948_convert_inverse_q30_fxp (long x_q30, int *pow2)
  Inverse function based on Newton-Raphson 1/sqrt(x) calculation.
- void INV_EXPORT inv_icm20948_convert_matrix_to_quat_flt (float *R, float *q)
  Converts a rotation matrix to a quaternion in floating point.
- long INV_EXPORT inv_icm20948_convert_mult_qfix_fxp (long a, long b, unsigned char qfix)
  Performs a multiply and shift by shift.
- void INV_EXPORT inv_icm20948_convert_quat_to_col_major_matrix_fxp (const long *quat_q30, long *rot↩ _q30)
  Converts a quaternion to a rotation matrix in column major convention.
- long INV_EXPORT inv_icm20948_math_atan2_q15_fxp (long y_q15, long x_q15)
  Seventh order Chebychev polynomial approximation in Q15.
- uint8_t INV_EXPORT * inv_icm20948_convert_int16_to_big8 (int16_t x, uint8_t *big8)
  Converts a 16-bit short to a big endian byte stream.
- uint8_t INV_EXPORT * inv_icm20948_convert_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_icm20948_convert_big8_to_int32 (const uint8_t *big8)
  Converts a big endian byte stream into a 32-bit long.
- void INV_EXPORT inv_icm20948_convert_quat_rotate_fxp (const long *quat_q30, const long *in, long *out)
  Converts long values according to quat_30 matrix.

7.30.1 Detailed Description

7.30.2 Macro Definition Documentation

7.30.2.1 #define ABS(*x*) (((x)>=0)?(x):-(x))

Computes the absolute value of its argument *x*.

7.30.2.2 #define INVN_FLT_TO_FXP(*value, shift*) ((int32_t) ((float)(value)*(1ULL << (shift)) + ((value>=0)-0.5f)))

Convert the *value* from float to QN value.

7.30.2.3 #define MAX(*x, y*) (((x)>(y))?(x):(y))

Computes the maximum of *x* and *y*.

7.30.2.4 #define MIN(*x, y*) (((x)<(y))?(x):(y))

Computes the minimum of *x* and *y*.

7.30.3 Function Documentation

7.30.3.1 int32_t INV_EXPORT inv_icm20948_convert_big8_to_int32 (const uint8_t * *big8*)

Converts a big endian byte stream into a 32-bit long.

Parameters

in	*big8*	big endian byte stream

Returns

corresponding 32-bit integer.

7.30.3.2 int INV_EXPORT inv_icm20948_convert_compute_scalar_part_fxp (const long * *inQuat_q30,* long * *outQuat_q30*)

Compute real part of quaternion, element[0].

Parameters

in	*inQuat_q30*	3 elements gyro quaternion. Dimension is 3.
out	*outQuat_q30*	Quaternion. Dimension is 4. 4 elements gyro quaternion

Returns

0

7.30.3.3 float INV_EXPORT inv_icm20948_convert_deg_to_rad (float *deg_val*)

Converts degree angle to radian.

Parameters

in	*deg_val*	the angle in degree

Returns

the angle in radian

7.30.3.4 void INV_EXPORT inv_icm20948_convert_dmp3_to_body (struct inv_icm20948 * *s,* const long * *vec3,* float *scale,* float * *values*)

Converts the data in android values.

Parameters

in	*vec3*	vector of the DMP
in	*scale*	scale calculated
out	*values*	in Android format

7.30.3.5 long INV_EXPORT inv_icm20948_convert_fast_sqrt_fxp (long *x0_q30*)

Calculates square-root of a fixed-point number (30 bit mantissa, positive)

Input must be a positive scaled (2^{30}) integer The number is scaled to lie between a range in which a Newton-↩ Raphson iteration works best.

Parameters

in	*x0_q30*	length 1. Fixed point format is Q30

Returns

scaled square root if succeed else 0.

7.30.3.6 int16_t INV_EXPORT inv_icm20948_convert_get_highest_bit_position (uint32_t * *value*)

Auxiliary function used by testLimitsAndScale() Find the highest nonzero bit in an unsigned 32 bit integer:

Parameters

in	*value*	operand Dimension is 1.

Returns

highest bit position.

Note

This function performs the log2 of an interger as well.

7.30.3.7 uint8_t INV_EXPORT* inv_icm20948_convert_int16_to_big8 (int16_t *x,* uint8_t * *big8*)

Converts a 16-bit short to a big endian byte stream.

Parameters

in	*x*	operand
out	*big8*	big endian byte stream

Returns

big8 pointer

7.30.3.8 uint8_t INV_EXPORT* inv_icm20948_convert_int32_to_big8 (int32_t *x*, uint8_t * *big8*)

Converts a 32-bit long to a big endian byte stream.

Parameters

in	*x*	operand
out	*big8*	big endian byte stream

Returns

big8 pointer

7.30.3.9 long INV_EXPORT inv_icm20948_convert_inv_sqrt_q30_fxp (long *x_q30,* int * *pow2*)

Calculates 1/square-root of a fixed-point number (30 bit mantissa, positive): Q1.30.

The number is scaled to lie between a range in which a Newton-Raphson iteration works best. Caller must scale final result by 2^rempow (while avoiding overflow).

Parameters

in	*x_q30*	Input. The input must be positive. Fixed point format is Q30.
out	*pow2*	Corresponding square root of the power of two is returned. length 1

Returns

square root of x in Q30.

7.30.3.10 long INV_EXPORT inv_icm20948_convert_inverse_q30_fxp (long *x_q30,* int * *pow2*)

Inverse function based on Newton-Raphson 1/sqrt(x) calculation.

Note that upshifting c (the result) by pow2 right away will overflow q30 if b<0.5 in q30 (=536870912).
So if you are doing some multiplication later on (like a/b), then it might be better to do `q30_mult(a,c)` first and then shift it up by pow2: `q30_mult(a,c)<<pow2`
The result might still overflow in some cases (large a, small b: a=1073741824, b=1 but precise limits of the overflow are tbd).

Parameters

in	*x_q30*	the operand. Fixed point format is Q30
in	*pow2*	a power of 2 by which 1/b is downshifted to fit in q30.

Returns

the 1/x result in Q30 downshifted by pow2.

7.30.3.11 void INV_EXPORT inv_icm20948_convert_matrix_to_quat_flt (float * *R,* float * *q*)

Converts a rotation matrix to a quaternion in floating point.

Parameters

in	*R*	Rotation matrix in floating point. The First 3 elements of the rotation matrix, represent the first row of the matrix.
out	*q*	4-element quaternion in floating point.

Warning

This functions does not retrieve fixed point quaternion anymore. Use a conversion flt_to_fxp.

7.30.3.12 void INV_EXPORT inv_icm20948_convert_matrix_to_quat_fxp (long * *Rcb_q30,* long * *Qcb_q30*)

Converts a rotation matrix to a quaternion.

Parameters

in	*Rcb_q30*	Rotation matrix. Fixed point format is Q30.
out	*Qcb_q30*	quaternion related to provided rotation matrix. Vector size is 4. Fixed point format is Q30.

7.30.3.13 long INV_EXPORT inv_icm20948_convert_mult_q30_fxp (long *a_q30,* long *b_q30*)

Performs a multiply and shift by 30.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

Parameters

in	*a*	
in	*b*	

Returns

((long long)a*b)>>30

7.30.3.14 long INV_EXPORT inv_icm20948_convert_mult_qfix_fxp (long *a,* long *b,* unsigned char *qfix*)

Performs a multiply and shift by shift.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

Parameters

in	*a*	First multicand
in	*b*	Second multicand
in	*shift*	Shift amount after multiplying

Returns

((long long)a∗b)>>shift

Warning

Same function that invn_math_mult_qfix_fxp.

7.30.3.15 void INV_EXPORT inv_icm20948_convert_quat_rotate_fxp (const long ∗ *quat_q30,* const long ∗ *in,* long ∗ *out*)

Converts long values according to quat_30 matrix.

Parameters

in	*quat_30*	mounting matrix to apply
in	*in*	long values to be converted
out	*out*	long values converted

Returns

void

7.30.3.16 void INV_EXPORT inv_icm20948_convert_quat_to_col_major_matrix_fxp (const long ∗ *quat_q30,* long ∗ *rot_q30*)

Converts a quaternion to a rotation matrix in column major convention.

Parameters

in	*quat_q30*	4-element quaternion in fixed point. Fixed point format is Q30.
out	*rot_q30*	Rotation matrix in fixed point. One is 2^30. The Rotation matrix multiplied by a 3 element column vector transforms a vector from Body to World.

Warning

output matrix storage is column major. colmajor_convention

7.30.3.17 void INV_EXPORT inv_icm20948_convert_rotation_vector (struct inv_icm20948 ∗ *s,* const long ∗ *quat,* float ∗ *values*)

Converts fixed point DMP rotation vector to floating point android notation.

Parameters

in	*quat*	3 element rotation vector from DMP, missing the scalar part. Converts from Chip frame to World frame
out	*values*	4 element quaternion in Android format

7.30.3.18 void INV_EXPORT inv_icm20948_convert_rotation_vector_2 (struct inv_icm20948 * *s,* const long * *quat,* long * *quat4_world*)

Converts 3 element fixed point DMP rotation vector to 4 element rotation vector in world frame.

Parameters

in	*quat*	3 element rotation vector from DMP, missing the scalar part. Converts from Chip frame to World frame
out	*quat4_world*	4 element quaternion

7.30.3.19 void INV_EXPORT inv_icm20948_convert_rotation_vector_3 (const long * *quat4_world,* float * *values*)

Converts 4 element rotation vector in world frame to floating point android notation.

Parameters

in	*quat4_world*	4 element rotation vector in World frame
out	*values*	in Android format

7.30.3.20 long INV_EXPORT inv_icm20948_convert_sqrt_q30_fxp (long *x_q30*)

Calculates square-root of a fixed-point number.

This code calls 1/sqrt(x) and multiplies result with x, i.e. $\sqrt{x} = x * (1/\sqrt{x})$.

Parameters

in	*x_q30*	Input. Fixed point format is Q30

Returns

square root of x0 in Q30.

7.30.3.21 int INV_EXPORT inv_icm20948_convert_test_limits_and_scale_fxp (long * *x0_q30,* int * *pow*)

Auxiliary function used by inv_OneOverX(), inv_fastSquareRoot(), inv_inverseSqrt().

Finds the range of the argument, determines the optimal number of Newton-Raphson iterations and . Restrictions: Number is represented as Q1.30. Number is betweeen the range 2<x<=0

Parameters

in	*x0_q30*	Input length 1. Number is represented as Q30. Number is betweeen the range 2<x<=0
out	*pow*	Corresponding square root of the power of two is returned. length 1

Returns

number of Newton Raphson iterations, x0 scaled between log(2) and log(4) and 2^N scaling (N=pow)

7.30.3.22 unsigned char INV_EXPORT* inv_icm20948_int32_to_little8 (long *x*, unsigned char * *little8*)

Converts a 32-bit long to a little endian byte stream.

Parameters

in	*x*	the long to be converted
in	*little8*	little endian byte converted

Returns

0 on success, negative value on error.

7.30.3.23 long INV_EXPORT inv_icm20948_math_atan2_q15_fxp (long *y_q15*, long *x_q15*)

Seventh order Chebychev polynomial approximation in Q15.

Chebychev 7th order polynomial approximation :

- in fixed point : $constA7 = int32(2^{15} * [0.999133448222780 - 0.3205332923816640.144982490144465, -0.038254464970299$
- in float : $A = [0.999133 - 0.3205330.144982 - 0.0382544);$

The related formula is :
$\xi = \{ \mid y|/|x|in(0, \pi/4]$
$|x|/|y|in(\pi/4, \pi/2), \quad Cheb = A(1) * \xi + A(2) * \xi^3 + A(3) * \xi^5 + A(4) * \xi^7$

7th Order Accuracy is +/-0.02 degrees (worst case) through entire range (accomplished with scaling).
This code depends on: reciprocal_fun_q15 , inverse_sqrt_q15 , inv_q15_mult

Parameters

in	*y_q15*	first operand of atan2(y, x). Fixed point format is Q15.
in	*x_q15*	second operand of atan2(y, x). Fixed point format is Q15.

Returns

output angle in radians. Fixed point format is Q15.

7.30.3.24 void INV_EXPORT inv_icm20948_q_mult_q_qi (const long * *q1,* const long * *q2,* long * *qProd*)

Performs a fixed point quaternion multiply with inverse on second element q1∗q2'.

Parameters

in	*q1*	First Quaternion Multicand, length 4. 1.0 scaled to 2^30
in	*q2*	Second Quaternion Multicand, length 4. 1.0 scaled to 2^30. Inverse will be take before multiply
out	*qProd*	Product after quaternion multiply q1∗q2'. Length 4. 1.0 scaled to 2^30.

7.30.3.25 void INV_EXPORT inv_icm20948_set_chip_to_body (struct inv_icm20948 * *s,* long * *quat*)

Sets the transformation used for chip to body frame.

Parameters

in	*quat*	the quaternion used for the transformation

7.30.3.26 void INV_EXPORT inv_icm20948_set_chip_to_body_axis_quaternion (struct inv_icm20948 * *s,* signed char * *accel_gyro_matrix,* float *angle*)

Converts the data in android quaternion values.

Parameters

in	*accel_gyro_matrix*	vector of the DMP
out	*angle*	angle calculated

7.31 load_firmware

Functions

- int INV_EXPORT inv_icm20948_firmware_load (struct inv_icm20948 *s, const unsigned char *data, unsigned short size, unsigned short load_addr)

  Loads the DMP firmware from SRAM.

7.31.1 Detailed Description

7.31.2 Function Documentation

7.31.2.1 int INV_EXPORT inv_icm20948_firmware_load (struct inv_icm20948 * *s,* const unsigned char * *data,* unsigned short *size,* unsigned short *load_addr*)

Loads the DMP firmware from SRAM.

Parameters

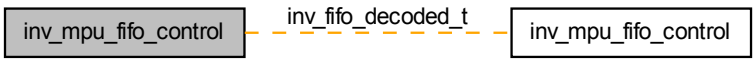
in	*data*	pointer where the image
in	*size*	size if the image
in	*load_addr*	address to loading the image

Returns

0 in case of success, -1 for any error

7.32 inv_mpu_fifo_control

Collaboration diagram for inv_mpu_fifo_control:



Classes

- struct inv_fifo_decoded_t

  Struct for the fifo.

Functions

- int INV_EXPORT inv_icm20948_identify_interrupt (struct inv_icm20948 *s, short *int_read)

  Identify the interrupt.
- int INV_EXPORT inv_icm20948_dmp_process_fifo (struct inv_icm20948 *s, int *left_in_fifo, unsigned short *user_header, unsigned short *user_header2, long long *time_stamp)

  Process the fifo.
- int INV_EXPORT inv_icm20948_dmp_get_accel (long acl[3])

  Gets the accelerometer data.
- int INV_EXPORT inv_icm20948_dmp_get_raw_gyro (short raw_gyro[3])

  Gets the raw gyrometer data.
- int INV_EXPORT inv_icm20948_dmp_get_gyro_bias (short gyro_bias[3])

  Gets gyro bias.
- int INV_EXPORT inv_icm20948_dmp_get_calibrated_gyro (signed long calibratedData[3], signed long raw[3], signed long bias[3])

  Gets calibrated gyro value based on raw gyro and gyro bias.
- int INV_EXPORT inv_icm20948_dmp_get_6quaternion (long quat[3])

  Gets the quaternion 6 axis data.
- int INV_EXPORT inv_icm20948_dmp_get_9quaternion (long quat[3])

  Gets the quaternion 9 axis data.
- int INV_EXPORT inv_icm20948_dmp_get_gmrvquaternion (long quat[3])

  Gets the quaternion GMRV data.
- int INV_EXPORT inv_icm20948_dmp_get_raw_compass (long raw_compass[3])

  Gets the raw compass data.
- int INV_EXPORT inv_icm20948_dmp_get_calibrated_compass (long cal_compass[3])

  Gets the calibrated compass data.
- int INV_EXPORT inv_icm20948_inv_decode_one_ivory_fifo_packet (struct inv_icm20948 *s, struct inv_fifo↩_decoded_t *fd, const unsigned char *fifo_ptr)

  Decodes the fifo packet.
- int INV_EXPORT inv_icm20948_dmp_get_bac_state (uint16_t *bac_state)

  Gets the state of the BAC sensor.
- int INV_EXPORT inv_icm20948_dmp_get_bac_ts (long *bac_ts)

Gets the timestamp of the BAC sensor.

- int INV_EXPORT inv_icm20948_dmp_get_flip_pickup_state (uint16_t *flip_pickup)

  Gets the state of the pick up sensor.
- int INV_EXPORT inv_icm20948_get_accel_accuracy (void)

  Returns the accelerometer accuracy.
- int INV_EXPORT inv_icm20948_get_gyro_accuracy (void)

  Returns the gyrometer accuracy.
- int INV_EXPORT inv_icm20948_get_mag_accuracy (void)

  Returns the magnetometer accuracy.
- int INV_EXPORT inv_icm20948_get_gmrv_accuracy (void)

  Returns the geomagnetic rotation vector accuracy.
- int INV_EXPORT inv_icm20948_get_rv_accuracy (void)

  Returns the rotation vector accuracy.
- int INV_EXPORT inv_icm20948_mpu_set_FIFO_RST_Diamond (struct inv_icm20948 *s, unsigned char value)

  Resets the fifo.
- int INV_EXPORT inv_icm20948_fifo_swmirror (struct inv_icm20948 *s, int *left_in_fifo, unsigned short *total_sample_cnt, unsigned short *sample_cnt_array)

  Mirror DMP HW FIFO into SW FIFO.
- int INV_EXPORT inv_icm20948_fifo_pop (struct inv_icm20948 *s, unsigned short *user_header, unsigned short *user_header2, int *left_in_fifo)

  Pop one sample out of SW FIFO.

7.32.1 Detailed Description

7.32.2 Function Documentation

7.32.2.1 int INV_EXPORT inv_icm20948_dmp_get_6quaternion (long *quat[3]*)

Gets the quaternion 6 axis data.

Parameters

out	*quat[3]*	the quaternion 6 axis data

Returns

0 on success, negative value on error.

7.32.2.2 int INV_EXPORT inv_icm20948_dmp_get_9quaternion (long *quat[3]*)

Gets the quaternion 9 axis data.

Parameters

out	*quat[3]*	the quaternion 9 axis data

Returns

0 on success, negative value on error.

7.32.2.3 int INV_EXPORT inv_icm20948_dmp_get_accel (long *acl[3]*)

Gets the accelerometer data.

Parameters

out	*acl[3]*	the accelerometer data

Returns

0 on success, negative value on error.

7.32.2.4 int INV_EXPORT inv_icm20948_dmp_get_bac_state (uint16_t * *bac_state*)

Gets the state of the BAC sensor.

Parameters

in	*bac_state*	pointer for recuperate the state of BAC

Returns

0 on success, negative value on error.

7.32.2.5 int INV_EXPORT inv_icm20948_dmp_get_bac_ts (long * *bac_ts*)

Gets the timestamp of the BAC sensor.

Parameters

in	*bac_ts*	pointer for recuperate the timestamp of BAC

Returns

0 on success, negative value on error.

7.32.2.6 int INV_EXPORT inv_icm20948_dmp_get_calibrated_compass (long *cal_compass[3]*)

Gets the calibrated compass data.

Parameters

out	*cal_compass[3]*	the calibrated compass data

Returns

0 on success, negative value on error.

7.32.2.7 int INV_EXPORT inv_icm20948_dmp_get_calibrated_gyro (signed long *calibratedData[3],* signed long *raw[3],* signed long *bias[3]*)

Gets calibrated gyro value based on raw gyro and gyro bias.

Parameters

out	*calibratedData[3]*	Calibred Gyro x,y,z
in	*raw[3]*	Gyro raw data x,y,z
in	*bias[3]*	Gyro bias x,y,z

Returns

0 on success, negative value on error.

7.32.2.8 int INV_EXPORT inv_icm20948_dmp_get_flip_pickup_state (uint16_t * *flip_pickup*)

Gets the state of the pick up sensor.

Parameters

in	*flip_pickup*	pointer for recuperate the state of pickup

Returns

0 on success, negative value on error.

7.32.2.9 int INV_EXPORT inv_icm20948_dmp_get_gmrvquaternion (long *quat[3]*)

Gets the quaternion GMRV data.

Parameters

out	*quat[3]*	the quaternion GMRV 6 axis data

Returns

0 on success, negative value on error.

7.32.2.10 int INV_EXPORT inv_icm20948_dmp_get_gyro_bias (short *gyro_bias[3]*)

Gets gyro bias.

Parameters

out	*quat[3]*	Gyro bias x,y,z

Returns

0 on success, negative value on error.

7.32.2.11 int INV_EXPORT inv_icm20948_dmp_get_raw_compass (long *raw_compass[3]*)

Gets the raw compass data.

Parameters

out	*cal_compass[3]*	the raw compass data

Returns

0 on success, negative value on error.

7.32.2.12 int INV_EXPORT inv_icm20948_dmp_get_raw_gyro (short *raw_gyro[3]*)

Gets the raw gyrometer data.

Parameters

out	*raw_gyro[3]*	the raw gyrometer data

Returns

0 on success, negative value on error.

7.32.2.13 int INV_EXPORT inv_icm20948_dmp_process_fifo (struct inv_icm20948 * *s,* int * *left_in_fifo,* unsigned short * *user_header,* unsigned short * *user_header2,* long long * *time_stamp*)

Process the fifo.

Parameters

in	*left_in_fifo*	pointer for the fifo to be processed
in	*user_header*	pointer for the user header
in	*user_header2*	pointer for the user header 2
in	*time_stamp*	pointer for the timestamp

Returns

0 on success, negative value on error.

7.32.2.14 int INV_EXPORT inv_icm20948_fifo_pop (struct inv_icm20948 * *s,* unsigned short * *user_header,* unsigned short * *user_header2,* int * *left_in_fifo*)

Pop one sample out of SW FIFO.

Parameters

out	*user_header*	Header value read from SW FIFO
out	*user_header2*	Header2 value read from SW FIFO
in,out	*left_in_fifo*	Contains number of bytes still be parsed from SW FIFO

Returns

0 on success, negative value on error.

7.32.2.15 int INV_EXPORT inv_icm20948_fifo_swmirror (struct inv_icm20948 * *s,* int * *left_in_fifo,* unsigned short * *total_sample_cnt,* unsigned short * *sample_cnt_array*)

Mirror DMP HW FIFO into SW FIFO.

Parameters

in,out	*left_in_fifo*	pointer to number of bytes in SW FIFO : before function is called, must contain number of bytes still present in FIFO which must not be overwritten after function is called, will contain number of bytes present in SW FIFO to be analyzed
out	*total_sample_cnt*	number of total sensor samples present in SW FIFO
out	*sample_cnt_array*	array of number of sensor samples present in SW FIFO for each sensor, should be inited to 0 before being called

Returns

0 on success, negative value on error.

7.32.2.16 int INV_EXPORT inv_icm20948_get_accel_accuracy (void)

Returns the accelerometer accuracy.

Returns

the accelerometer accuracy value

7.32.2.17 int INV_EXPORT inv_icm20948_get_gmrv_accuracy (void)

Returns the geomagnetic rotation vector accuracy.

Returns

the geomagnetic rotation vector accuracy in Q29

7.32.2.18 int INV_EXPORT inv_icm20948_get_gyro_accuracy (void)

Returns the gyrometer accuracy.

Returns

the gyrometer accuracy value

7.32.2.19 int INV_EXPORT inv_icm20948_get_mag_accuracy (void)

Returns the magnetometer accuracy.

Returns

the magnetometer accuracy value

7.32.2.20 int INV_EXPORT inv_icm20948_get_rv_accuracy (void)

Returns the rotation vector accuracy.

Returns

the rotation vector accuracy value in Q29

7.32.2.21 int INV_EXPORT inv_icm20948_identify_interrupt (struct inv_icm20948 * *s*, short * *int_read*)

Identify the interrupt.

Parameters

in	*int_read*	pointer to the DMP interrupt status

Returns

0 on success, negative value on error.

7.32.2.22 int INV_EXPORT inv_icm20948_inv_decode_one_ivory_fifo_packet (struct inv_icm20948 * *s*, struct inv_fifo_decoded_t * *fd*, const unsigned char * *fifo_ptr*)

Decodes the fifo packet.

Parameters

in	*fifo_ptr*	pointer to the fifo data
in	*fd*	pointer to the fifo what contains the sensor data

Returns

0 on success, negative value on error.

7.32.2.23 int INV_EXPORT inv_icm20948_mpu_set_FIFO_RST_Diamond (struct inv_icm20948 * *s*, unsigned char *value*)

Resets the fifo.

Parameters

in	*value*	0=no, 1=yes

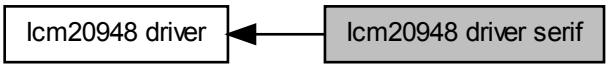
Returns

0 on success, negative value on error.

7.33 Icm20948 driver serif

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

Collaboration diagram for Icm20948 driver serif:



Classes

- struct inv_icm20948_serif

  ICM20948 serial interface.

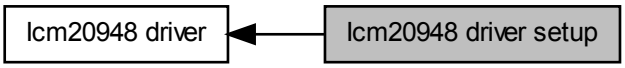
7.33.1 Detailed Description

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

7.34 Icm20948 driver setup

Low-level function to setup an Icm20948 device.

Collaboration diagram for Icm20948 driver setup:



Enumerations

Functions

- int INV_EXPORT inv_icm20948_set_lowpower_or_highperformance (struct inv_icm20948 *s, uint8_t lowpower_or_highperformance)

  Have the chip to enter low-power or low-noise mode.

7.34.1 Detailed Description

Low-level function to setup an Icm20948 device.

7.34.2 Function Documentation

7.34.2.1 int INV_EXPORT inv_icm20948_set_lowpower_or_highperformance (struct inv_icm20948 * *s*, uint8_t *lowpower_or_highperformance*)

Have the chip to enter low-power or low-noise mode.

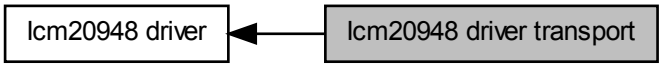
Parameters

in	*lowpower_or_highperformance*	0=low-power, 1=low-noise

7.35 Icm20948 driver transport

Low-level ICM20948 register access.

Collaboration diagram for Icm20948 driver transport:



Macros

- #define INV_MAX_SERIAL_READ 16

  Max size that can be read across I2C or SPI data lines.
- #define INV_MAX_SERIAL_WRITE 16

  Max size that can be written across I2C or SPI data lines.

Functions

- int INV_EXPORT inv_icm20948_write_mems_reg (struct inv_icm20948 ∗s, uint16_t reg, unsigned int length, const unsigned char ∗data)

  Write data to a register on MEMs.
- int INV_EXPORT inv_icm20948_write_single_mems_reg (struct inv_icm20948 ∗s, uint16_t reg, const unsigned char data)

  Write single byte of data to a register on MEMs.
- int INV_EXPORT inv_icm20948_read_mems_reg (struct inv_icm20948 ∗s, uint16_t reg, unsigned int length, unsigned char ∗data)

  Read data from a register on MEMs.
- int INV_EXPORT inv_icm20948_read_mems (struct inv_icm20948 ∗s, unsigned short reg, unsigned int length, unsigned char ∗data)

  Read data from a register in DMP memory.
- int INV_EXPORT inv_icm20948_write_mems (struct inv_icm20948 ∗s, unsigned short reg, unsigned int length, const unsigned char ∗data)

  Write data to a register in DMP memory.
- int INV_EXPORT inv_icm20948_write_single_mems_reg_core (struct inv_icm20948 ∗s, uint16_t reg, const uint8_t data)

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

7.35.1 Detailed Description

Low-level ICM20948 register access.

7.35.2 Function Documentation

7.35.2.1 int INV_EXPORT inv_icm20948_read_mems (struct inv_icm20948 ∗ *s*, unsigned short *reg*, unsigned int *length*, unsigned char ∗ *data*)

Read data from a register in DMP memory.

Parameters

in	*DMP*	memory address
in	*number*	of byte to be read
in	*input*	data from the register

Returns

0 if successful.

7.35.2.2 int INV_EXPORT inv_icm20948_read_mems_reg (struct inv_icm20948 * *s*, uint16_t *reg*, unsigned int *length*, unsigned char * *data*)

Read data from a register on MEMs.

Parameters

in	*Register*	address
in	*Length*	of data
in	*Data*	to be written

Returns

0 if successful.

7.35.2.3 int INV_EXPORT inv_icm20948_write_mems (struct inv_icm20948 * *s*, unsigned short *reg*, unsigned int *length*, const unsigned char * *data*)

Write data to a register in DMP memory.

Parameters

in	*DMP*	memory address
in	*number*	of byte to be written
out	*output*	data from the register

Returns

0 if successful.

7.35.2.4 int INV_EXPORT inv_icm20948_write_mems_reg (struct inv_icm20948 * *s*, uint16_t *reg*, unsigned int *length*, const unsigned char * *data*)

Write data to a register on MEMs.

Parameters

in	*Register*	address
in	*Length*	of data
in	*Data*	to be written

Returns

0 if successful.

7.35.2.5 int INV_EXPORT inv_icm20948_write_single_mems_reg (struct inv_icm20948 * *s*, uint16_t *reg*, const unsigned char *data*)

Write single byte of data to a register on MEMs.

Parameters

in	*Register*	address
in	*Data*	to be written

Returns

0 if successful.

7.35.2.6 int INV_EXPORT inv_icm20948_write_single_mems_reg_core (struct inv_icm20948 * *s*, uint16_t *reg*, const uint8_t *data*)

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

Parameters

in	*reg*	DMP memory address
out	*data*	Data to be written

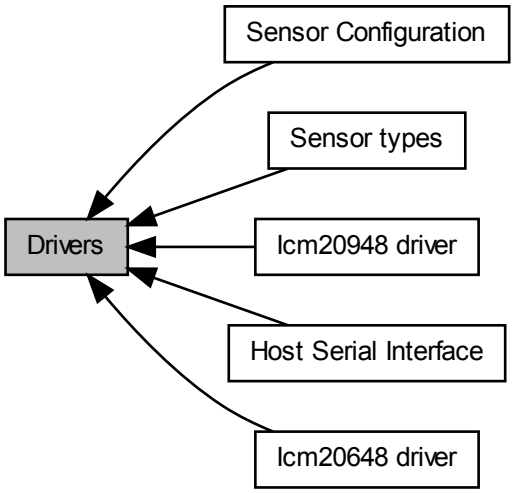
Returns

0 in case of success, -1 for any error

7.36 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.
- Icm20648 driver

  Low-level driver for ICM20648 devices.
- Icm20948 driver

  Low-level driver for ICM20948 devices.

7.36.1 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.

7.37 Utils

Utility functions.

Utility functions.

Chapter 8

Class Documentation

8.1 inv_icm20648::base_driver_t Struct Reference

struct for the base_driver : this contains the Mems information

`#include <Icm20648.h>`

8.1.1 Detailed Description

struct for the base_driver : this contains the Mems information

The documentation for this struct was generated from the following file:

- Icm20648.h

8.2 inv_icm20948::base_driver_t Struct Reference

struct for the base_driver : this contains the Mems information

`#include <Icm20948.h>`

8.2.1 Detailed Description

struct for the base_driver : this contains the Mems information

The documentation for this struct was generated from the following file:

- Icm20948.h

8.3 inv_icm20948::fifo_info_t Struct Reference

The documentation for this struct was generated from the following file:

- Icm20948.h

8.4 inv_icm20648::fifo_info_t Struct Reference

The documentation for this struct was generated from the following file:

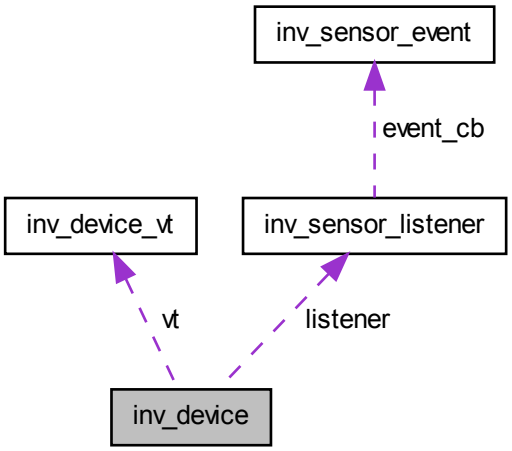
- Icm20648.h

8.5 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

8.5.1 Detailed Description

Abtract device object definition.

Examples:

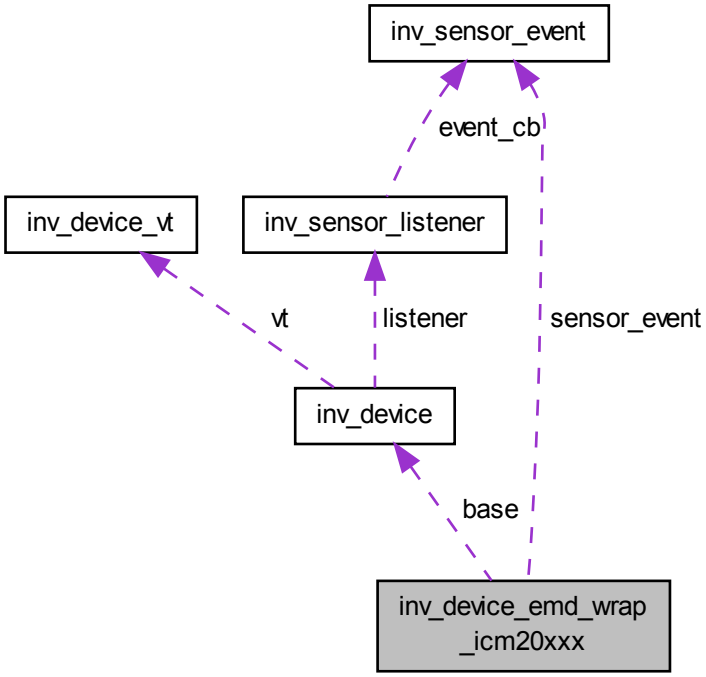
ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.c.

The documentation for this struct was generated from the following file:

- Device.h

8.6 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

8.7 inv_device_emd_wrap_icm20xxx_serial Struct Reference

The documentation for this struct was generated from the following file:

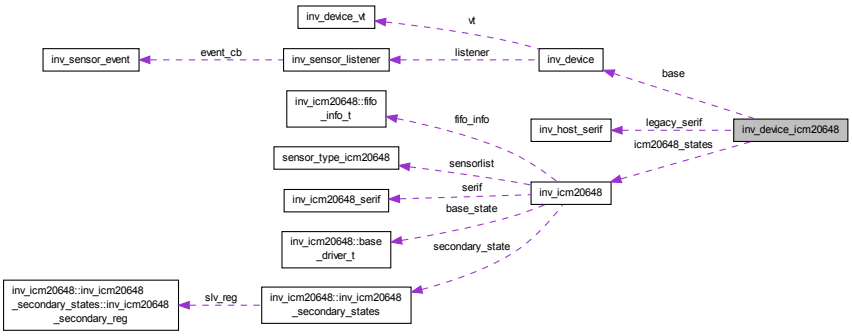
- DeviceEmdWrapIcm20xxx.h

8.8 inv_device_icm20648 Struct Reference

States for Icm20648 device.

`#include <DeviceIcm20648.h>`

Collaboration diagram for inv_device_icm20648:



8.8.1 Detailed Description

States for Icm20648 device.

Examples:

ExampleDeviceIcm20648EMD.c.

The documentation for this struct was generated from the following file:

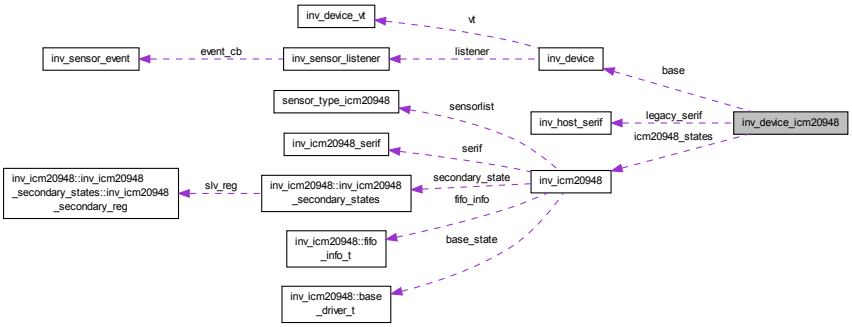
- DeviceIcm20648.h

8.9 inv_device_icm20948 Struct Reference

States for Icm20948 device.

`#include <DeviceIcm20948.h>`

Collaboration diagram for inv_device_icm20948:



8.9.1 Detailed Description

States for Icm20948 device.

Examples:

ExampleDeviceIcm20948EMD.c.

The documentation for this struct was generated from the following file:

- DeviceIcm20948.h

8.10 inv_device_vt Struct Reference

Device virtual table definition.

`#include <Device.h>`

8.10.1 Detailed Description

Device virtual table definition.

The documentation for this struct was generated from the following file:

- Device.h

8.11 inv_fifo_decoded_t Struct Reference

Struct for the fifo.

`#include <Icm20648MPUFifoControl.h>`

8.11.1 Detailed Description

Struct for the fifo.

this contains the sensor data

The documentation for this struct was generated from the following files:

- Icm20648MPUFifoControl.h
- Icm20948MPUFifoControl.h

8.12 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.

8.12.1 Detailed Description

FW version structure definition.

The documentation for this struct was generated from the following file:

- Device.h

8.13 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.

8.13.1 Detailed Description

Serial Interface interface definition.

8.13.2 Member Data Documentation

8.13.2.1 int(∗ inv_host_serif::close) (void)

Close connection to Serial Interface adapter.

Returns

0 on sucess, negative value on error

8.13.2.2 int(∗ inv_host_serif::open) (void)

Open connection to and initialize Serial Interface adapter.

Returns

0 on sucess, negative value on error

8.13.2.3 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

8.13.2.4 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

8.13.2.5 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

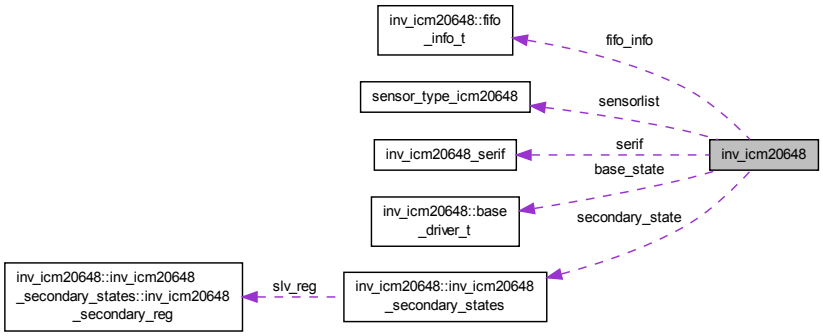
0 on sucess, negative value on error

The documentation for this struct was generated from the following file:

- HostSerif.h

8.14 inv_icm20648 Struct Reference

Collaboration diagram for inv_icm20648:



Classes

- struct base_driver_t

  struct for the base_driver : this contains the Mems information
- struct fifo_info_t
- struct inv_icm20648_secondary_states

The documentation for this struct was generated from the following file:

- Icm20648.h

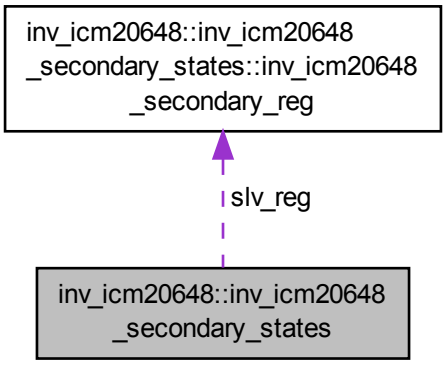
8.15 inv_icm20648::inv_icm20648_secondary_states::inv_icm20648_secondary_reg Struct Reference

The documentation for this struct was generated from the following file:

- Icm20648.h

8.16 inv_icm20648::inv_icm20648_secondary_states Struct Reference

Collaboration diagram for inv_icm20648::inv_icm20648_secondary_states:



Classes

- struct inv_icm20648_secondary_reg

The documentation for this struct was generated from the following file:

- Icm20648.h

8.17 inv_icm20648_serif Struct Reference

ICM20648 serial interface.

`#include <Icm20648Serif.h>`

8.17.1 Detailed Description

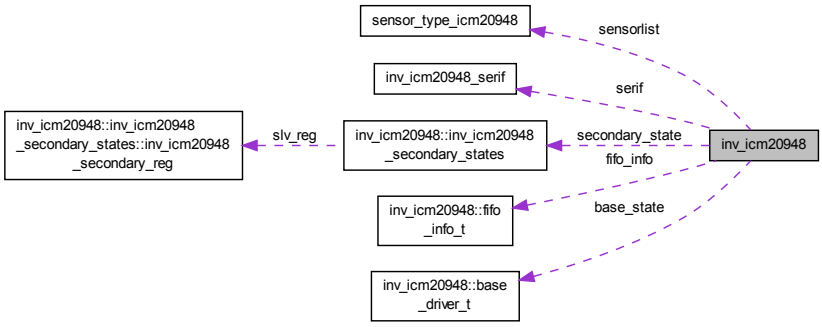
ICM20648 serial interface.

The documentation for this struct was generated from the following file:

- Icm20648Serif.h

8.18 inv_icm20948 Struct Reference

Collaboration diagram for inv_icm20948:



Classes

- struct base_driver_t

  struct for the base_driver : this contains the Mems information
- struct fifo_info_t
- struct inv_icm20948_secondary_states

The documentation for this struct was generated from the following file:

- Icm20948.h

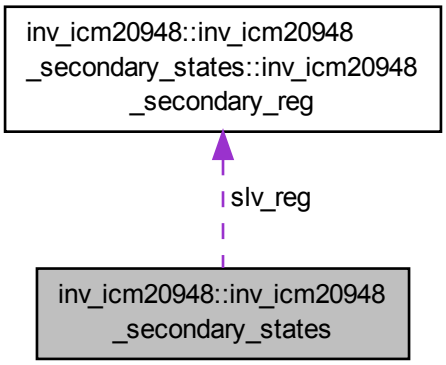
8.19 inv_icm20948::inv_icm20948_secondary_states::inv_icm20948_secondary_reg Struct Reference

The documentation for this struct was generated from the following file:

- Icm20948.h

8.20 inv_icm20948::inv_icm20948_secondary_states Struct Reference

Collaboration diagram for inv_icm20948::inv_icm20948_secondary_states:



Classes

- struct inv_icm20948_secondary_reg

The documentation for this struct was generated from the following file:

- Icm20948.h

8.21 inv_icm20948_serif Struct Reference

ICM20948 serial interface.

`#include <Icm20948Serif.h>`

8.21.1 Detailed Description

ICM20948 serial interface.

The documentation for this struct was generated from the following file:

- Icm20948Serif.h

8.22 inv_sensor_config_bac Struct Reference

Define the configuration for BAC.

`#include <SensorConfig.h>`

8.22.1 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

8.23 inv_sensor_config_BSCD Struct Reference

Define the configuration for the BSCD virtual sensor.

`#include <SensorConfig.h>`

8.23.1 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

8.24 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>`

8.24.1 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

8.25 inv_sensor_config_distance Struct Reference

Define the configuration for the distance's algorithm.

`#include <SensorConfig.h>`

8.25.1 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

8.26 inv_sensor_config_double_tap Struct Reference

Define the configuration for the double tap's algorithm.

`#include <SensorConfig.h>`

8.26.1 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

8.27 inv_sensor_config_energy_expenditure Struct Reference

Define the configuration for the energy expenditure's algorithm.

`#include <SensorConfig.h>`

8.27.1 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

8.28 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>`

8.28.1 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

8.29 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>`

8.29.1 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

8.30 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>`

8.30.1 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

8.31 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>`

8.31.1 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

8.32 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>`

8.32.1 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

8.33 inv_sensor_config_shake_wrist Struct Reference

Define the configuration for the shake wrist's algorithm.

`#include <SensorConfig.h>`

8.33.1 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

8.34 inv_sensor_config_stepc Struct Reference

Define the configuration for steps counter.

`#include <SensorConfig.h>`

8.34.1 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

8.35 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

8.35.1 Detailed Description

Sensor event definition.

Examples:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.c.

8.35.2 Member Data Documentation

8.35.2.1 uint8_t inv_sensor_event::accuracy_flag

accuracy flag

heading accuracy in deg

accuracy flag specific for GRV

8.35.2.2 struct { ... } inv_sensor_event::audio_buffer

Wake-up on motion data.

buffer of audio data

8.35.2.3 float inv_sensor_event::bias[3]

x,y,z bias vector data

x,y,z bias vector data (for uncal sensor variant)

8.35.2.4 int32_t inv_sensor_event::cumulativeEEkcal

cumulative energy expenditure since the last reset in kilocalorie.

Format is q0: 1 = 1 kcal

8.35.2.5 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

8.35.2.6 struct { ... } inv_sensor_event::eis

EIS data.

Warning

experimental: structure is likely to change in near future

8.35.2.7 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.

8.35.2.8 int32_t inv_sensor_event::floorsDown

number of floors climbed Down on foot by user.

8.35.2.9 int32_t inv_sensor_event::floorsUp

number of floors climbed Up on foot by user.

8.35.2.10 struct { ... } inv_sensor_event::hrm

heart rate monitor data

heart rate monitor data

8.35.2.11 int32_t inv_sensor_event::instantEEkcal

energy expenditure in kilocalorie/min since last output.

Format is q15: 2^15 = 1 kcal/min

8.35.2.12 int32_t inv_sensor_event::instantEEmets

energy expenditure in METs(Metabolic Equivalent of Task) since last output.

Format is q15: 2^15 = 1 METs

8.35.2.13 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:

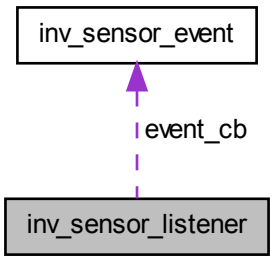
- SensorTypes.h

8.36 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

8.36.1 Detailed Description

Sensor event listener definition.

Examples:

ExampleDeviceIcm20648EMD.c, and ExampleDeviceIcm20948EMD.c.

The documentation for this struct was generated from the following file:

- SensorTypes.h

8.37 sensor_type_icm20648 Struct Reference

ICM20648 driver states definition.

`#include <Icm20648.h>`

8.37.1 Detailed Description

ICM20648 driver states definition.

The documentation for this struct was generated from the following file:

- Icm20648.h

8.38 sensor_type_icm20948 Struct Reference

ICM20948 driver states definition.

`#include <Icm20948.h>`

8.38.1 Detailed Description

ICM20948 driver states definition.

The documentation for this struct was generated from the following file:

- Icm20948.h

Chapter 9

Example Documentation

9.1 ExampleDeviceIcm20648EMD.c

This example shows how to use the Device API for ICM20648 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/DeviceIcm20648.h"

/*
 * High resolution sleep implementation for Icm20648.
 * Used at initilization stage. ~100us is sufficient.
 */
void inv_icm20648_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 Icm20648.
 */
uint64_t inv_icm20648_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 icm20648 device object
 */
static inv_device_icm20648_t device_icm20648;

static uint8_t dmp3_image[] = {
        0
// #include "path/to/Icm30630Dmp3Image.h"
};

/*
 * 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 ICM20648 Device
         * Pass to the driver:
         * - reference to serial interface object,
         * - reference to listener that will catch sensor events,
         */
        inv_device_icm20648_init2(&device_icm20648, &serif_instance, &
      sensor_listener, dmp3_image, sizeof(dmp3_image));

        /* Initialize akm9911 compass */
        inv_device_icm20648_init_aux_compass(&device_icm20648,
      INV_ICM20648_COMPASS_ID_AK09911, 0x0C);

        /*
         * Simply get generic device handle from Icm20648 Device
         */
        device = inv_device_icm20648_get_base(&device_icm20648);

        /*
         * Just get the whoami
         */
        rc += inv_device_whoami(device, &whoami);
        /* ... do something with whoami */
```

```
        /*
         * Configure and initialize the Icm20648 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);

        /* ... */

        /*
         * 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 ExampleDeviceIcm20948EMD.c

This example shows how to use the Device API for ICM20948 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/DeviceIcm20948.h"
```

```
/*
 * High resolution sleep implementation for Icm20948.
 * Used at initilization stage. ~100us is sufficient.
 */
void inv_icm20948_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 Icm20948.
 */
uint64_t inv_icm20948_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 icm20948 device object
 */
static inv_device_icm20948_t device_icm20948;

static uint8_t dmp3_image[] = {
        0
// #include "path/to/Icm30630Dmp3Image.h"
};

/*
 * 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 ICM20948 Device
         * Pass to the driver:
         * - reference to serial interface object,
         * - reference to listener that will catch sensor events,
         */
        inv_device_icm20948_init2(&device_icm20948, &serif_instance, &
      sensor_listener, dmp3_image, sizeof(dmp3_image));

        /*
         * Simply get generic device handle from Icm20948 Device
         */
        device = inv_device_icm20948_get_base(&device_icm20948);

        /*
         * Just get the whoami
         */
        rc += inv_device_whoami(device, &whoami);
        /* ... do something with whoami */

        /*
         * Configure and initialize the Icm20948 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);

        /* ... */

        /*
         * 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 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