AN4156
Application note

Hardware abstraction layer for Android

Introduction

This application note provides guidelines for successfully integrating STMicroelectronics sensors (accelerometer, magnetometer, gyroscope and pressure) into the Android operating system.

The configuration files of the sensor HAL (hardware abstraction layer) are discussed along with the issues and possible solutions for successfully integrating different kinds of sensors.

Finally, the building and installation of this library is also described.

Contents

List of figures

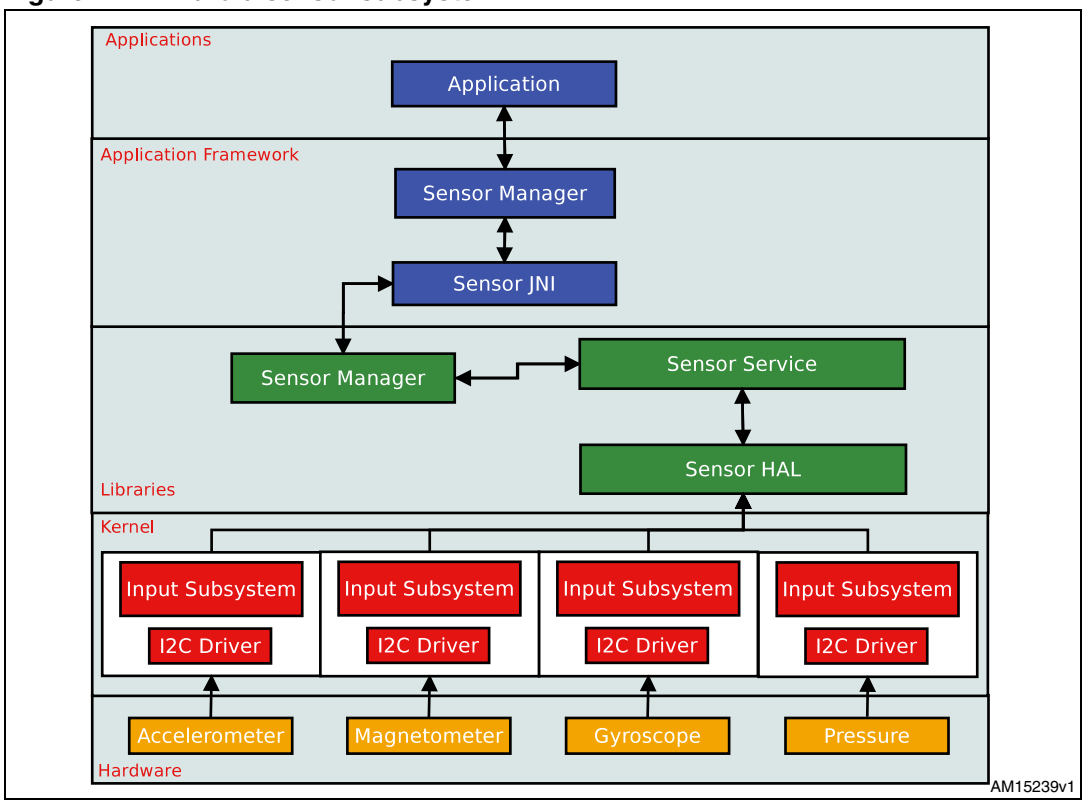
1 Sensor HAL: overview

The Android sensor HAL is the library which provides the links by kernel-space drivers to the Android sensor service and Android sensor manager.

The architecture of the Android sensor framework is shown in *Figure 1*:

- Application framework: applications that make use of the sensors utilize the “Application framework” to get data from the devices. The communication starts in the sensor manager class to then pass to the lower layer through the sensor JNI (Java native interface).
- Sensor libraries: these libraries have the purpose of creating a sophisticated interface for the upper layer. This is done through the sensor manager class, sensor service class, and the sensor HAL, which is the focus of this document.
- Kernel: in this layer we find Linux device drivers created using the input subsystem, a generic Linux framework for all input devices such as the mouse, keyboard, joystick, etc. The data are exported to the user space through the sysfs virtual file system (`/sys/class/input/`). The driver sends/receives data to/from the sensor through the stable Linux subsystem I2C.

Figure 1. Android sensor subsystem



2 Sensor HAL: files

In this section we examine the sensor HAL library files and its architecture model. There are two configuration files in the library: `configuration.h` and `sensors.h`.

The `configuration.h` file is used to set some parameters such as the names of the sensors, full scale defaults, names of the sysfs files, enable or disable debug information, etc.

The `sensors.h` file is used to set the convert value of the data, the axis mapping through rotation matrices and the event type of the drivers. The UML graph of the HAL library structure is shown in *Figure 4*.

The library is written in `C++` language using the object-oriented approach, for each sensor there is a custom class file (`AccelSensor.cpp`, `MagnSensor.cpp`, `GyroSensor.cpp` and `PressTempSensor.cpp`) which extends the common base class (`SensorBase.cpp`).

2.1 Configuration.h

As mentioned above, the configuration file is used to describe some parameters of the sensors. In this short paragraph we describe all the parameters and what they mean. An example of a configuration file for the LSM303DLHC accelerometer is shown in *Figure 2*.

The first macro visible in this file, ANDROID_JELLY_BEAN, is used as the define if the library is built for Android Jelly Bean version (4.1).

The parameters of the other sensors are the same as those used in this accelerometer example.

```
..
...
/* ACCELEROMETER SENSOR \*/
#define SENSOR_ACC_LABEL                "LSM303DLHC 3-axis Accelerometer"
#define SENSOR_ACC_INCLUDE_FILE_NAME    "lsm303dlhc.h"
#define SENSOR_DATANAME_ACCELEROMETER   LSM303DLHC_ACC_DEV_NAME
#define ACCEL_DELAY_FILE_NAME           "pollrate_ms"
#define ACCEL_ENABLE_FILE_NAME          "enable_device"
#define ACCEL_RANGE_FILE_NAME           "range"
#define ACCEL_MAX_RANGE                 16*GRAVITY_EARTH
#define ACCEL_MAX_ODR                   400
#define ACCEL_POWER_CONSUMPTION         0.033f
#define ACCEL_DEFAULT_FULLSCALE         4
...
..
```

2.1.1 SENSOR_ACC_LABEL

This macro is used to define the sensor name visible in the Android application.

2.1.2 SENSOR_ACC_INCLUDE_FILE_NAME

This macro is used to define the name of the header file of the device driver, located in the "include" subfolder.

2.1.3 SENSOR_ACC_DATANAME_ACCELEROMETER

This macro is necessary to search the device in the input subsystem. This name is visible in the header file of the driver (in this case it is set in the LSM303DLHC_ACC_DEV_NAME macro) and used in the implementation file of the driver through the input device structure called input_dev->name.

2.1.4 SENSOR_DELAY_FILENAME, SENSOR_ENABLE_FILENAME, SENSOR_RANGE_FILENAME

ACCEL_DELAY_FILE_NAME, ACCEL_ENABLE_FILE_NAME and ACCEL_RANGE_FILE_NAME are necessary to define the names of the files needed to read/write sensor data (these files are located in the sysfs virtual file system [/sys/class/i2c-adapter/i2c-n/n-00xx/], where n is the I2C bus number and xx the sensor I2C address).

2.1.5 ACCEL_MAX_RANGE, ACCEL_MAX_RANGE

These macros are used to inform Android about the operating range for the sensors. When Android requires the value for the maximum range of measurements: this value is set through the ACCEL_MAX_RANGE macro and expressed in m/s^2 (in this example, 16 is the full scale of the accelerometer) for the maximum ODR (output data rate) defined in the ACCEL_MAX_ODR macro and expressed in Hz (400 Hz is the maximum ODR for the accelerometer).

2.1.6 ACCEL_POWER_CONSUMPTION

Power consumption of the sensor expressed in mA.

2.1.7 ACCEL_DEFAULT_FULLSCALE

The ACCEL_DEFAULT_FULLSCALE macro is used to define the default full scale used when Android requires the use of this sensor (in this example, 4g).

2.2 Sensor.h

This configuration file is used to change the event type generated by the driver through the input subsystem, to define rotation matrix as regards the ENU Android coordinate system and, finally, to define the factor of the data conversion. In most cases the event type does not change, however, this parameter is set through the EVENT_TYPE_****_* macro, according to the driver: see input set_abs_params() function in the input driver. The most important parameters of this file are the rotation matrices: they are necessary to rotate the single coordinate system of each sensor to only one "general" coordinate system of the board/smartphone. In fact, the Android O.S. expects sensor data in only one coordinate system called ENU (East, North, Up), visible in *Figure 2*.

Figure 2. ENU coordinate system

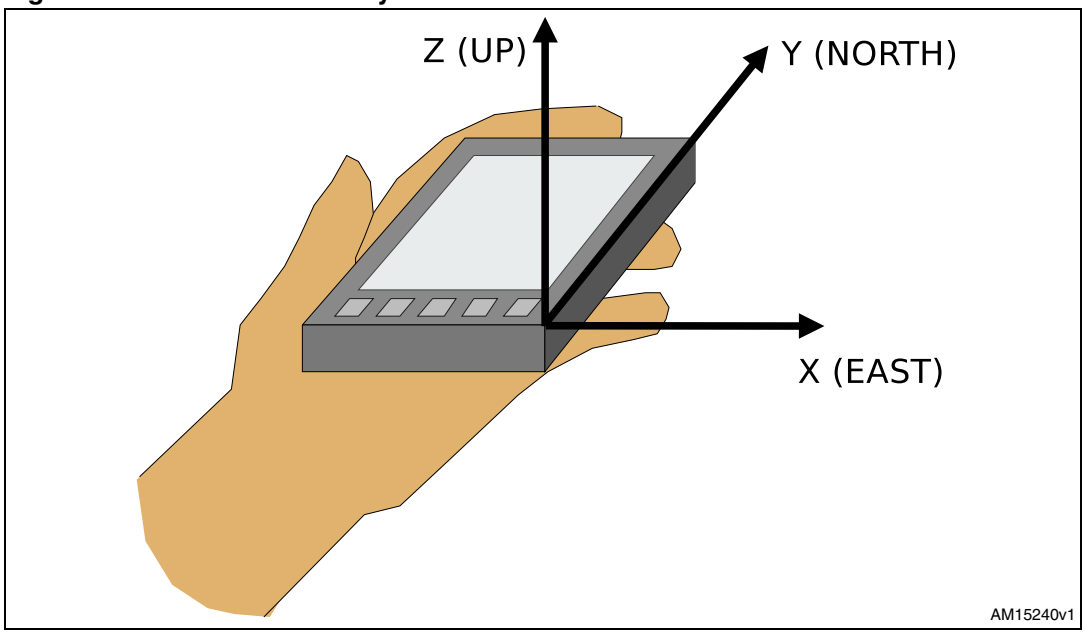
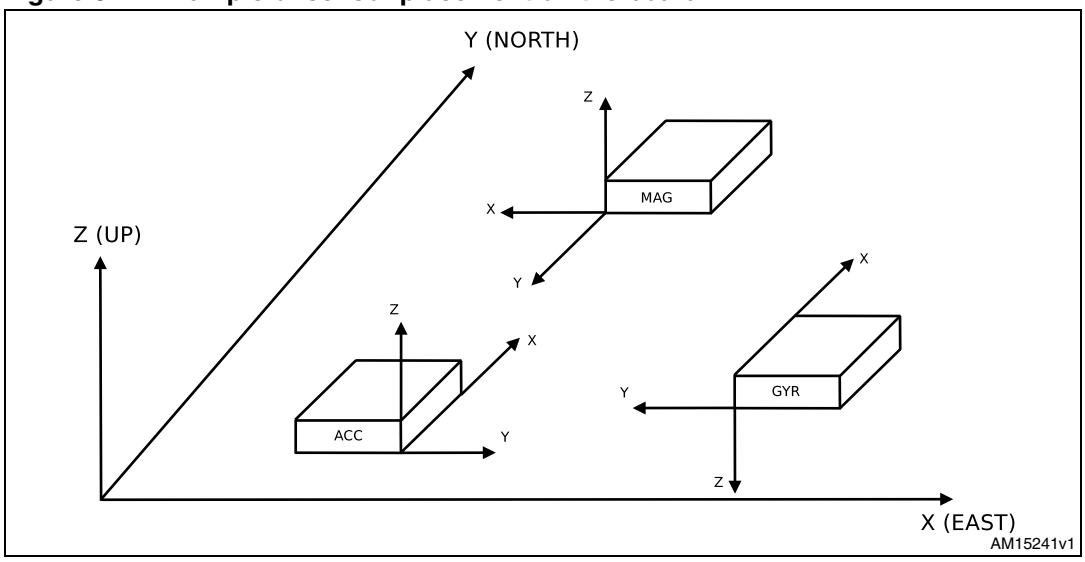


Figure 3 shows the accelerometer, magnetometer and gyroscope placed on a PCB. The orientation of the device on the PCB can be done in the driver code through the `axis_map_x`, `axis_map_y` and `axis_map_z` variables or with rotation matrices.

Figure 3. Example of sensor placement on the board



The transformation matrices are defined as:

$$\begin{bmatrix} x_b & y_b & z_b \end{bmatrix} = \begin{bmatrix} x_s & y_s & z_s \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} = \begin{bmatrix} x_s \cdot x_1 + x_s \cdot y_1 + z_s \cdot z_1 \\ x_s \cdot x_2 + x_s \cdot y_2 + z_s \cdot z_2 \\ x_s \cdot x_3 + x_s \cdot y_3 + z_s \cdot z_3 \end{bmatrix}^T$$

where x_b, y_b, z_b are board coordinates and x_s, y_s, z_s are sensor coordinates.

The single coordinate system for each sensor is visible in the sensor datasheet.

In this case, for example, the rotation matrices are defined as:

- Accelerometer:

$$\begin{bmatrix} x_b & y_b & z_b \end{bmatrix} = \begin{bmatrix} x_a & y_a & z_a \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix}^T$$

- Magnetometer:

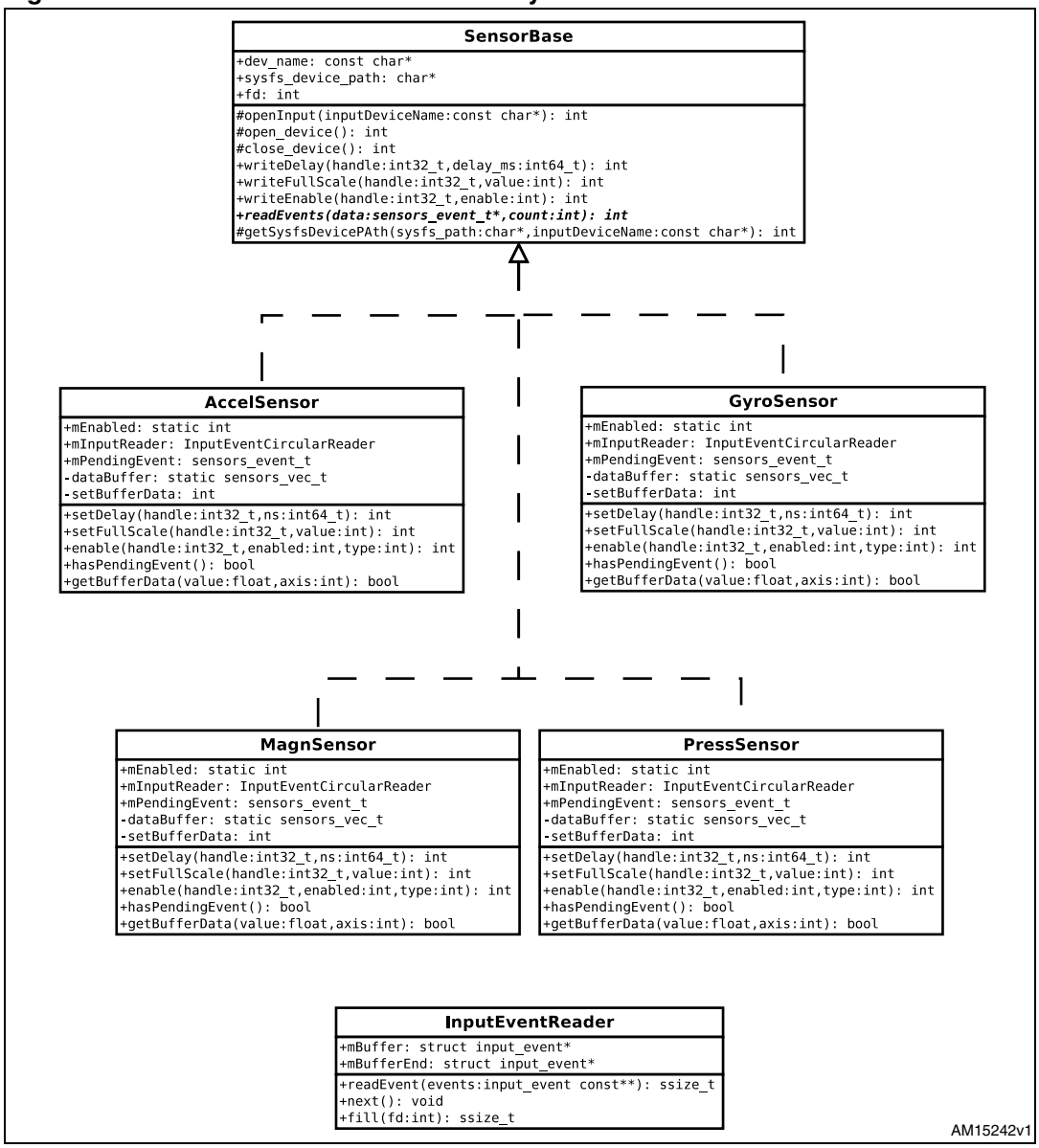
$$\begin{bmatrix} x_b & y_b & z_b \end{bmatrix} = \begin{bmatrix} x_m & y_m & z_m \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix}^T$$

- Gyroscope:

$$\begin{bmatrix} x_b & y_b & z_b \end{bmatrix} = \begin{bmatrix} x_g & y_g & z_g \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix}^T$$

In the last section of the configuration file the user must define what the conversion factors are to convert driver data values into the Android default measurement units. Android expects accelerometer data in m/s^2, magnetometer data in uT and the gyroscope data in rad/sec.

Figure 4. UML of hardware abstraction layer



3 How to build and install the Android sensor HAL

The build system in Android is very easy thanks to the Android build environment and to the Android makefiles. After having successfully downloaded and compiled the Android sources, the user can compile and add/substitute the sensor HAL library.

To do this, copy the sensor HAL library folder in the android sources path, usually located in:

```
[Root Android Sources]/vendor/[vendor name]/[boardname]/
```

Before the build operation of the library, the user must initialize the Android environment:

```
[Root Android Sources]$ source build/envsetup.sh
```

```
[Root Android Sources]$ lunch [target board]
```

It is now possible to build the library; just launch the "mm" command in the HAL folder. The result of this process is a dynamic library located in:

```
[Root Android Sources]/out/target/product/[board
name]/system/lib/hw/sensors.[board name].so
```

The user can add this library or substitute the existing library using `adb push`, but they must remount the Android system partition to push it:

```
[Root Android Sources]$ adb shell]
```

```
[Android shell]$ mount
```

```
...
```

```
/dev/block/platform/omap/omap_hsmmc.0/by-name/system /system ext4
ro,relatime,barrier=1,data=ordered 0 0
```

```
...
```

```
[Android shell]$ mount -o remount,rw
/dev/block/platform/omap/omap_hsmmc.0/by-name/system
```

```
[Root Android Sources]/out/target/product/[board
name]/system/lib/hw/adb push sensors.[board name].so
/system/lib/hw/
```

4 Revision history

Table 1. Document revision history

Date	Revision	Changes
24-Sep-2012	1	Initial release.