Pawel Gburzynski

MPU9250 accelerometer/gyro/compass

(preliminary) driver

August 4, 2021

Introduction

MPU9250 is an accelerometer/gyroscope/compass combo manufactured by InvenSense Inc. Documentation available from the manufacturer (data sheet + specification) covers most of the technical intricacies (in a moderately confusing sort of way). That

documentation is not absolutely required to understand how to use our driver and what you can expect from the sensor (under the present driver).

The sensor optionally offers high-level functionality known as DMP (Digital Motion Processing). That functionality is not implemented in the present driver. To implement it requires some reverse engineering, but it can be done, should a need arise. It is a high-power, high-cpu-load function relying on special code loaded into the chip which then requires special processing in software. The manufacturer has provided source code which probably can be adapted for PicOS with some effort.

Another feature that is excluded from the present driver, but can be easily added in the future (this one truly easily) is FIFO access, i.e., high-speed data acquisition, e.g., for high-bandwidth, real-time analysis of motion, gesture recognition, dead reckoning, and so on. The feature is mostly excluded, because it requires a different interface than our standard **`read_sensor`** operation intended for low-bandwidth data extraction, a few bytes at a time.

In the meantime, four years after writing the above paragraph, I am updating this document to cover an extension added to the driver to accommodate (reasonably) high-speed continuous data collection from the sensor. I have decided against using FIFO for this purpose because 1) the speed gain would be probably insignificant (the values still have to be read over the I2C interface), 2) I couldn't find in the documentation how to use interrupts to properly recognize the FIFO status (other than overflow), such that the FIFO could act as a smooth two-ended buffer. But it is relatively easy to use interrupts (sensor events) for synchronous and timed readouts of fresh data at rates of order 256 readings per second.

Summary of the sensor's functionality

The sensor has two basic configurable modes of operation:

1. low power (LP) motion detection mode (triggering motion events)
2. measurement mode (on demand data acquisition)

In the second mode, the sensor can be optionally set to generate events at regular intervals when new data can be read. The maximum readout rate (at which the registers containing the sensor indications are internally updated) is 1KHz, i.e., 1024 times per second. The events can be generated at this frequency or at any lower frequency obtained by skipping a prescribed number of clock ticks from 0 to 255 (corresponding to rates from 4 to 1024 events per second). This internal clock can assist in collecting timed series of readings. It isn’t significantly more helpful than a system clock, but the strobes generated by the sensor clock fall on the boundaries of the formal availability of new values.

Only the accelerometer is available in the LP motion detection mode. Besides receiving events on exceeded acceleration thresholds, the application can also read the acceleration values at any time.

In the second mode, the set of sensors is configurable and can comprise any combination of these options:

- acceleration
- gyro
- compass
- temperature

The last option is the free (die) temperature readout that can be obtained from the sensor. For example, when acceleration and temperature have been selected from the above options, the sensor *reading* (as returned by the standard **`read_sensor`** operation) will consists of four numbers: the three-element acceleration vector + the temperature reading. With the complete set of selections, a single sensor reading consists of 10 numbers (3 vectors and 1 scalar)

In the LP motion detection mode, a standard sensor event is triggered when the acceleration (along any of the three axes) exceeds a declared threshold.

For acceleration, the measurement range, aka FSR [Full Scale Range] comes in four options: ±2g, ±4g, ±8g, ±16g. For the gyroscope, the options are: ±250, ±500, ±1000, and ±2000 degrees per second. For the compass (magnetometer) the (immutable) FSR is ±4800μT (microtesla). The resolution is 15 bits in all cases, i.e., each value is returned as a 16-bit signed integer.

Two more configuration parameters for the driver are:

- low pass filter setting (for damping rapid changes): 8 levels
- wakeup rate in the LP motion-detection mode: 12 levels

The second parameter only applies to the accelerometer operating in the LP motion detection mode. The LPF bandwidth selection applies to acceleration and gyro. For the compass, each reading stands for itself and is taken exactly at the time of issue of the **`read_sensor`** function.

Driver interface

The sensor must be explicitly turned on, which operation determines its mode and parameters. This is accomplished by invoking:

```
mpu9250_on (lword options, byte threshold);
```

where **`options`** is a collection of fields composed by or'ring a few constants defined in mpu9250.h. In particular, the LPF setting is determined by one of these constants:

Constant name	Accel (Hz)	Gyro (Hz)	Temperature (Hz)
`MPU9250_LPF_256`	460	250	4000
`MPU9250_LPF_188`	184	184	188
`MPU9250_LPF_98`	92	92	98

MPU9250_LPF_42	41	41	42
MPU9250_LPF_20	20	20	20
MPU9250_LPF_10	10	10	10
MPU9250_LPF_5	5	5	5
MPU9250_LPF_2100	460	3600	4000

The wakeup rate in LP mode can be any of these:

Constant name	Rate (Hz)
MPU9250_LPA_02	0,24
MPU9250_LPA_05	0,49
MPU9250_LPA_1	0,98
MPU9250_LPA_2	1,95
MPU9250_LPA_4	3,91
MPU9250_LPA_8	7,81
MPU9250_LPA_16	15,63
MPU9250_LPA_32	31,25
MPU9250_LPA_64	62,5
MPU9250_LPA_128	125
MPU9250_LPA_256	250
MPU9250_LPA_512	500

The configuration of sensors is described by these constants:

MPU9250_SEN_ACCEL
MPU9250_SEN_GYRO
MPU9250_SEN_COMPASS
MPU9250_SEN_TEMP

The ranges (FSR) for the accelerometer and the gyro are:

MPU9250_ACCEL_RANGE_2 (this is the default [0])
MPU9250_ACCEL_RANGE_4
MPU9250_ACCEL_RANGE_8
MPU9250_ACCEL_RANGE_16

MPU9250_GYRO_RANGE_250 (this is the default [0])
MPU9250_GYRO_RANGE_500
MPU9250_GYRO_RANGE_1000
MPU9250_GYRO_RANGE_2000

These constants select the interpretation of sensor events (only one can be chosen at a time):

```
MPU9250_LP_MOTION_DETECT
MPU9250_LP_SYNC_READ
```

The first one selects the motion detection mode where the events generated by the sensor will be interpreted as motion events. The second constant selects events generated by the sensor's internal clock for strobing data readings. With that option, the most significant byte of **options** provides the divider of the basic 1KHz rate.

The **threshold** argument of **mpu9250_on** provides the threshold value for motion detection and is only interpreted when **MPU9250_LP_MOTION_DETECT** is on. The value is in 4 mg increments with the range from 0 to 1020 mg. Note that its interpretation does not depend on the FSR setting.

For example:

```
mpu9250_on (MPU9250_LP_MOTION_DETECT       |
            MPU9250_SEN_ACCEL              |
            MPU9250_LPF_188                |
            MPU9250_LPA_4, 32);
```

sets up the device for LP motion detection with the filter bandwidth of 184 Hz (value recommended for motion detection), with the wakeup frequency of 3.91 Hz, and the threshold of 32 x 4 = 128 mg. Note that the FSR of the accelerometer is only applicable to readouts of acceleration values, not to the comparison against the threshold. Any sensor selections beyond the accelerometer made together with **MPU9250_LP_MOTION_DETECT** would be ignored. In fact, accelerometer selection is superfluous as the motion detection flag makes sure that the accelerometer is selected as the only component.

For another example:

```
mpu9250_on (MPU9250_LP_SYNC_READ           |
            MPU9250_SEN_ACCEL              |
            MPU9250_SEN_GYRO               |
            MPU9250_LPF_5                  |
            15 << 24, 0);
```

selects the accelerometer and the gyro with the LPF of 5 Hz. The sensor will be triggering events at the rate of 1024 / (15 + 1) Hz, i.e., 128 times per second.

The sensor is stopped by invoking:

```
mpu9250_off ();
```


On the systems where the sensor is powered from a µC pin (e.g., CC1350 SENSORTAG), this will power the sensor down completely. Otherwise, the sensor is put into the sleep mode.

Reading values

The sensor value can be read in the standard way:

```
read_sensor (word st, address val);
```

The state argument is ignored (the sensor is always ready to return data). The second argument should point to an array of words whose size depends on the configuration of selected sensors. For example, for the accelerometer alone (e.g., in the LP motion detection mode), the value amounts to a 3-element vector (three words are needed to accommodate it). For the accelerometer + the gyro, the size is two vectors, i.e., 6 words. The temperature readout is the only scalar in the set taking just one word.

The ordering of values in the **`val`** array is: accelerometer, gyro, compass, temperature (the ones not selected are skipped). The vector components are signed 2's complement numbers with the granularity of $1/2^{15}$ of the full-scale range. I couldn't find the temperature conversion formula anywhere in the data sheet. It only says:

Temp_degC = ((TEMP_OUT – RoomTemp_Offset) / Temp_Sensitivity) + 21°C

without providing the constants.

Receiving events

This is accomplished by the standard PicOS operation:

```
wait_sensor (SENSOR_MPU9250, TARGET_STATE);
```

Normally, the function immediately blocks (forces release), so it should be the last statement in the current state. The issuing FSM is suspended and will be resumed in **`TARGET_STATE`** when the event occurs. Needless to say, the FSM can issue other wait requests before calling **`wait_sensor`**.

The event causing **`wait_sensor`** to unblock does not go away until the program executes **`read_sensor`** to retrieve the sensor data. This means that a subsequent **`wait_sensor`**, executed before **`read_sensor`**, will fire immediately.

Here is a sample PicOS code for responding to events triggered by the sensors internal clock:

```
fsm sensor_data_collector {
      word values [6];
      state WAIT_FOR_DATA:
            wait_sensor (SENSOR_MPU9250, DATA_READY);
      state DATA_READY:
```

```
            read_sensor (WNONE, SENSOR_MPU9250, values);
            pass_data_to_app (values);
            sameas WAIT_FOR_DATA;
}
```

The code assumes that the sensor returns 6 words, e.g., corresponding to the second example (see above) of turning the sensor on.

Note that (regardless of whether **MPU9250_LP_SYNC_READ** has been selected or not) it is always legal to issue **read_sensor**, and the function will return the set of recent readings prescribed by the sensor's configuration.