Using Hardware Offset Registers for Calibration

PURPOSE AND SCOPE

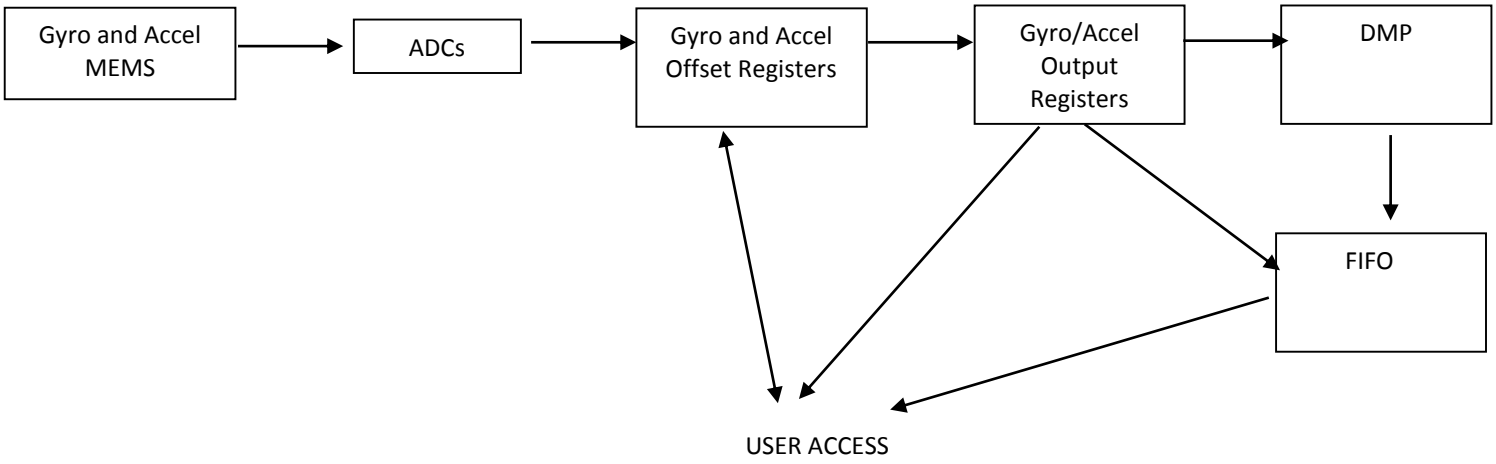
All MEMS sensors are highly recommended to calibrate either at factory or dynamically while device is in-use to get more accurate sensor data. InvenSense software and DMP has several algorithms dealing with calibration. However, if you are not using these software algorithms, offset cancellation can still be achieved using the built-in Hardware Offset Registers in InvenSense motion devices.

Several InvenSense MPU and ICM devices contain offset cancellation registers for the accelerometer and gyroscope sensors. While these registers are listed in the Register Maps, customers typical have several questions on how to use these registers. This document will

- Details the Hardware Offset Registers,
- Provides a small example on how to get gyro and accel offsets
- How to apply those offsets into the Hardware Offset Registers.

DATA SIGNAL DIAGRAM

All data sampled from the MEMS sensors will have the offsets in the Offset Hardware Registers applied before outputting to the sensor data registers for users. The Offsets application is also before all usage concerning FIFO and DMP. Therefore, the data in the FIFO, sensor data output registers, and used in the DMP will already have these offsets included.



FINDING THE OFFSET BIASES FOR ACCEL AND GYRO

There are a number of ways to get the biases for the Accel and Gyro, the general idea is that the device is stationary in a known orientation typically with the InvenSense part faced up or down. This is so that we can isolate the gravity on the Z axis and remove it from the accel samples. It is possible to determine the biases in other orientations however for simplicity it is best to isolate the gravity vector to a particular axis. In InvenSense default code and in our example we expect the gravity on the Accel Z axis.

Because the device is stationary the expected gyro output of each axis is 0, 0, 0 and the accel output is 0, 0, +1G. With this information if we take samples of each axis we will be able to determine the average offset from the ideal values and those values will be the Offset Biases. In the example below from our Motion Driver, we read each sensor axis 200 samples then average it to get the biases.

InvenSense Inc.
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: AN-xxxx
Revision: x.x
Release Date: MM/DD/YYYY

Using HW Offset Registers for Calibration

EXAMPLE CODE FROM MOTION DRIVER 5.1.2

```
static int get_biases(long *gyro, long *accel)
{
    unsigned char data[MAX_PACKET_LENGTH];
    unsigned char packet_count, ii;
    unsigned short fifo_count;

    data[0] = 0x01;
    data[1] = 0;
    if (i2c_write(st.hw->addr, st.reg->pwr_mgmt_1, 2, data))
        return -1;
    delay_ms(200);
    data[0] = 0;
    if (i2c_write(st.hw->addr, st.reg->int_enable, 1, data))
        return -1;
    if (i2c_write(st.hw->addr, st.reg->fifo_en, 1, data))
        return -1;
    if (i2c_write(st.hw->addr, st.reg->pwr_mgmt_1, 1, data))
        return -1;
    if (i2c_write(st.hw->addr, st.reg->i2c_mst, 1, data))
        return -1;
    if (i2c_write(st.hw->addr, st.reg->user_ctrl, 1, data))
        return -1;
    data[0] = BIT_FIFO_RST | BIT_DMP_RST;
    if (i2c_write(st.hw->addr, st.reg->user_ctrl, 1, data))
        return -1;
    delay_ms(15);
    data[0] = st.test->reg_lpf;
    if (i2c_write(st.hw->addr, st.reg->lpf, 1, data))
        return -1;
    data[0] = st.test->reg_rate_div;
    if (i2c_write(st.hw->addr, st.reg->rate_div, 1, data))
        return -1;
    data[0] = st.test->reg_gyro_fsr;
    if (i2c_write(st.hw->addr, st.reg->gyro_cfg, 1, data))
        return -1;

    if (hw_test)
        data[0] = st.test->reg_accel_fsr | 0xE0;
    else
        data[0] = test.reg_accel_fsr;
    if (i2c_write(st.hw->addr, st.reg->accel_cfg, 1, data))
        return -1;
    if (hw_test)
        delay_ms(200);

    /* Fill FIFO for test.wait_ms milliseconds. */
    data[0] = BIT_FIFO_EN;
    if (i2c_write(st.hw->addr, st.reg->user_ctrl, 1, data))
        return -1;

    data[0] = INV_XYZ_GYRO | INV_XYZ_ACCEL;
    if (i2c_write(st.hw->addr, st.reg->fifo_en, 1, data))
        return -1;
    delay_ms(test.wait_ms);
    data[0] = 0;
    if (i2c_write(st.hw->addr, st.reg->fifo_en, 1, data))
        return -1;

    if (i2c_read(st.hw->addr, st.reg->fifo_count_h, 2, data))
        return -1;

    fifo_count = (data[0] << 8) | data[1];
    packet_count = fifo_count / MAX_PACKET_LENGTH;
    gyro[0] = gyro[1] = gyro[2] = 0;
    accel[0] = accel[1] = accel[2] = 0;

    for (ii = 0; ii < packet_count; ii++) {
        short accel_cur[3], gyro_cur[3];
        if (i2c_read(st.hw->addr, st.reg->fifo_r_w, MAX_PACKET_LENGTH, data))
            return -1;
        accel_cur[0] = ((short)data[0] << 8) | data[1];
        accel_cur[1] = ((short)data[2] << 8) | data[3];
        accel_cur[2] = ((short)data[4] << 8) | data[5];
```


InvenSense Inc.
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: AN-100-00
Revision: 1.4
Release Date: 9/21/2016

```
        accel[0] += (long)accel_cur[0];
        accel[1] += (long)accel_cur[1];
        accel[2] += (long)accel_cur[2];
        gyro_cur[0] = (((short)data[6] << 8) | data[7]);
        gyro_cur[1] = (((short)data[8] << 8) | data[9]);
        gyro_cur[2] = (((short)data[10] << 8) | data[11]);
        gyro[0] += (long)gyro_cur[0];
        gyro[1] += (long)gyro_cur[1];
        gyro[2] += (long)gyro_cur[2];
    }
    gyro[0] = (long)(((long long)gyro[0]) / test.gyro_sens / packet_count);
    gyro[1] = (long)(((long long)gyro[1]) / test.gyro_sens / packet_count);
    gyro[2] = (long)(((long long)gyro[2]) / test.gyro_sens / packet_count);
    accel[0] = (long)(((long long)accel[0]) / test.accel_sens /
        packet_count);
    accel[1] = (long)(((long long)accel[1]) / test.accel_sens /
        packet_count);
    accel[2] = (long)(((long long)accel[2]) / test.accel_sens /
        packet_count);
    /* remove gravity from the accel Z */
    if (accel[2] > 0L)
        accel[2] -= test.accel_sens;
    else
        accel[2] += test.accel_sens;

    return 0;
}
```

GYRO OFFSETS REGISTERS

Please refer to the motion components register maps for the precise location and format. To current date, the location of the Offset Registers could be different but the format should be the same. For this example we will be using the MPU parts to illustrate how to apply the HW offset register settings for calibration.

REGISTER LOCATION

For MPU6050/MPU6500/MPU6515 and MPU9150/MPU9250, the gyro offset registers are located at

Addr	Name	Description
0x13	XG_OFFS_USRH	Gyro X-axis offset cancellation register high byte
0x14	XG_OFFS_USRL	Gyro X-axis offset cancellation register low byte
0x15	YG_OFFS_USRH	Gyro Y-axis offset cancellation register high byte
0x16	YG_OFFS_USRL	Gyro Y-axis offset cancellation register low byte
0x17	ZG_OFFS_USRH	Gyro Z-axis offset cancellation register high byte
0x18	ZG_OFFS_USRL	Gyro Z-axis offset cancellation register low byte

REGISTER FORMAT AND DESCRIPTIONS

The Gyro registers at boot up will have a default value of 0. Here are the steps to apply the bias into these registers.

- Take the bias you calculated from each axis in degrees per second (dps)
- Convert the DPS to new BIAS = DPS/.031. You can round BIAS to nearest whole number.
- Take the BIAS and insert into the Offset Registers

Please note that the Offset Registers expects a signed number so negative numbers should be in two's compliment format.

EXAMPLE CODE FROM MOTION DRIVER 5.1.2

```
/**
 *  @brief      Push biases to the gyro bias 6500/6050 registers.
 *  This function expects biases relative to the current sensor output, and
 *  these biases will be added to the factory-supplied values. Bias inputs are LSB
 *  in +-1000dps format.
 *  @param[in]  gyro_bias  New biases.
 *  @return     0 if successful.
 */
int mpu_set_gyro_bias_reg(long *gyro_bias)
{
```

```
    unsigned char data[6] = {0, 0, 0, 0, 0, 0};
    int i=0;
    for(i=0;i<3;i++) {
        gyro_bias[i]= (-gyro_bias[i]);
    }
    data[0] = (gyro_bias[0] >> 8) & 0xff;
    data[1] = (gyro_bias[0]) & 0xff;
    data[2] = (gyro_bias[1] >> 8) & 0xff;
    data[3] = (gyro_bias[1]) & 0xff;
    data[4] = (gyro_bias[2] >> 8) & 0xff;
    data[5] = (gyro_bias[2]) & 0xff;
    if (i2c_write(st.hw->addr, 0x13, 2, &data[0]))
        return -1;
    if (i2c_write(st.hw->addr, 0x15, 2, &data[2]))
        return -1;
    if (i2c_write(st.hw->addr, 0x17, 2, &data[4]))
        return -1;
    return 0;
}
```

ACCEL OFFSET REGISTERS

Again please refer to the components register maps for precious format and location. Again our examples will use the MPU devices however the format for MPU and ICM devices are the same as of the date of this document.

REGISTER LOCATIONS

For MPU6050/MPU9150 the registers are located at the following address

Addr	Name	Description
0x06	XA_OFFS_USRH	Accel X-axis offset cancellation register high byte
0x07	XA_OFFS_USRL	Accel X-axis offset cancellation register low byte
0x08	YA_OFFS_USRH	Accel Y-axis offset cancellation register high byte
0x09	YA_OFFS_USRL	Accel Y-axis offset cancellation register low byte
0x0A	ZA_OFFS_USRH	Accel Z-axis offset cancellation register high byte
0x0B	ZA_OFFS_USRL	Accel Z-axis offset cancellation register low byte

MPU6500/MPU6515/ and MPU9250 are located at the following

Addr	Name	Description
0x77	XA_OFFS_USRH	Accel X-axis offset cancellation register high byte
0x78	XA_OFFS_USRL	Accel X-axis offset cancellation register low byte
0x7A	YA_OFFS_USRH	Accel Y-axis offset cancellation register high byte
0x7B	YA_OFFS_USRL	Accel Y-axis offset cancellation register low byte
0x7D	ZA_OFFS_USRH	Accel Z-axis offset cancellation register high byte
0x0E	ZA_OFFS_USRL	Accel Z-axis offset cancellation register low byte

REGISTER FORMAT AND DESCRIPTIONS

Unlike the Gyro, the accel offset registers are not as straight forward to use.

1. Initial values contain the OTP values of the Accel factory trim. Therefore, at boot up there will be a non-zero value in these registers. Users will need to first read the register and apply the biases to that value.
2. Convert the biases of each Accel axis to a scale range of +-16G in which 1G = 2048. So for example if your bias for Accel X is .08G (or 80mg) then BIAS = .08 * 2048 = 163.84. You can round VALUE to nearest whole number in this case it would be 164.
3. Apply the biases by subtracting the bias values from the OTP values.
4. Bit 0 on the low byte of each axis is a reserved bit and needs to be preserved.
5. Write the new value into the Offset Register.

InvenSense Inc.
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: AN-100-00
Revision: 1.4
Release Date: 9/21/2016

Using HW Offset Registers for Calibration

EXAMPLE CODE FROM MOTION DRIVER 5.1.2

```
/**
 *  @brief      Read biases to the accel bias 6500 registers.
 *  This function reads from the MPU6500 accel offset cancellations registers.
 *  The format are G in +-8G format. The register is initialized with OTP
 *  factory trim values.
 *  @param[in]  accel_bias  returned structure with the accel bias
 *  @return     0 if successful.
 */
int mpu_read_6500_accel_bias(long *accel_bias) {
    unsigned char data[6];
    if (i2c_read(st.hw->addr, 0x77, 2, &data[0]))
        return -1;
    if (i2c_read(st.hw->addr, 0x7A, 2, &data[2]))
        return -1;
    if (i2c_read(st.hw->addr, 0x7D, 2, &data[4]))
        return -1;
    accel_bias[0] = ((long)data[0]<<8) | data[1];
    accel_bias[1] = ((long)data[2]<<8) | data[3];
    accel_bias[2] = ((long)data[4]<<8) | data[5];
    return 0;
}
/**
 *  @brief      Push biases to the accel bias 6500 registers.
 *  This function expects biases relative to the current sensor output, and
 *  these biases will be added to the factory-supplied values. Bias inputs are LSB
 *  in +-8G format.
 *  @param[in]  accel_bias  New biases.
 *  @return     0 if successful.
 */
int mpu_set_accel_bias_6500_reg(const long *accel_bias) {
    unsigned char data[6] = {0, 0, 0, 0, 0, 0};
    long accel_reg_bias[3] = {0, 0, 0};

    if(mpu_read_6500_accel_bias(accel_reg_bias))
        return -1;

    // Preserve bit 0 of factory value (for temperature compensation)
    accel_reg_bias[0] -= (accel_bias[0] & ~1);
    accel_reg_bias[1] -= (accel_bias[1] & ~1);
    accel_reg_bias[2] -= (accel_bias[2] & ~1);

    data[0] = (accel_reg_bias[0] >> 8) & 0xff;
    data[1] = (accel_reg_bias[0]) & 0xff;
    data[2] = (accel_reg_bias[1] >> 8) & 0xff;
    data[3] = (accel_reg_bias[1]) & 0xff;
    data[4] = (accel_reg_bias[2] >> 8) & 0xff;
    data[5] = (accel_reg_bias[2]) & 0xff;

    if (i2c_write(st.hw->addr, 0x77, 2, &data[0]))
        return -1;
    if (i2c_write(st.hw->addr, 0x7A, 2, &data[2]))
        return -1;
    if (i2c_write(st.hw->addr, 0x7D, 2, &data[4]))
        return -1;

    return 0;
}
```

SAVING THE BIASES FOR POWER ON/OFF

Bias offsets once inputted into the Offset Cancellation Registers will immediate take into effect. However after power off and on, the offset registers will be cleared and the gyro offset registers will return to ‘0’ values and the accel Offset Registers will return to the default factory values. The sensor values will then be raw data. In order to get calibrated data again, the biases will need to be reloaded back into these registers after power on.

Because it is not feasible to recalculate the biases every time a power on, it is recommended that the bias values are saved in some non-volatile memory so that it can be recalled and applied to the Offset Registers when needed. This way the bias calculations will only need to happen once…most likely at the factory line.

InvenSense Inc.
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: AN-100-00
Revision: 1.4
Release Date: 9/21/2016

REVISION HISTORY

REVISION DATE	REVISION	DESCRIPTION
10/12/2016	x.x	Updated for Release

Compliance Declaration Disclaimer

InvenSense believes the environmental and other compliance information given in this document to be correct but cannot guarantee accuracy or completeness. Conformity documents substantiating the specifications and component characteristics are on file. InvenSense subcontracts manufacturing, and the information contained herein is based on data received from vendors and suppliers, which has not been validated by InvenSense.

This information furnished by InvenSense is believed to be accurate and reliable. However, no responsibility is assumed by InvenSense for its use, or for any infringements of patents or other rights of third parties that may result from its use. Specifications are subject to change without notice. InvenSense reserves the right to make changes to this product, including its circuits and software, in order to improve its design and/or performance, without prior notice. InvenSense makes no warranties, neither expressed nor implied, regarding the information and specifications contained in this document. InvenSense assumes no responsibility for any claims or damages arising from information contained in this document, or from the use of products and services detailed therein. This includes, but is not limited to, claims or damages based on the infringement of patents, copyrights, mask work and/or other intellectual property rights.

Certain intellectual property owned by InvenSense and described in this document is patent protected. No license is granted by implication or otherwise under any patent or patent rights of InvenSense. This publication supersedes and replaces all information previously supplied. Trademarks that are registered trademarks are the property of their respective companies. InvenSense sensors should not be used or sold in the development, storage, production or utilization of any conventional or mass-destructive weapons or for any other weapons or life threatening applications, as well as in any other life critical applications such as medical equipment, transportation, aerospace and nuclear instruments, undersea equipment, power plant equipment, disaster prevention and crime prevention equipment.

InvenSense

©2016 InvenSense, Inc. All rights reserved.