InvenSense Inc.
1745 Technology Drive, San Jose, CA, 95110 U.S.A.
Tel: +1 (408) 501-2200 Fax: +1 (408) 988-7339
Website: www.invensense.com

Document Number: AN-XX-XXXX-XX
Revision: 1.0
Release Date: 02/21/2014

MPU Hardware Offset Registers Application Note

A printed copy of this document is
NOT UNDER REVISION CONTROL
unless it is dated and stamped in red ink as,
"REVISION CONTROLLED COPY."

TABLE OF CONTENTS

1. Revision History

Revision Date	Revision	Description
02/21/2014	1.0	Document created
06/27/2014	1.1	Modified Set 6500 Accel Bias function
07/28/2014	1.2	Modified scale range for the accel bias from +-8G to +-16G

2. Reference

Please check the Register Map, EVB, and Production Specification of the relevant parts.

http://www.invensense.com/mems/gyro/catalog.html

3. Purpose

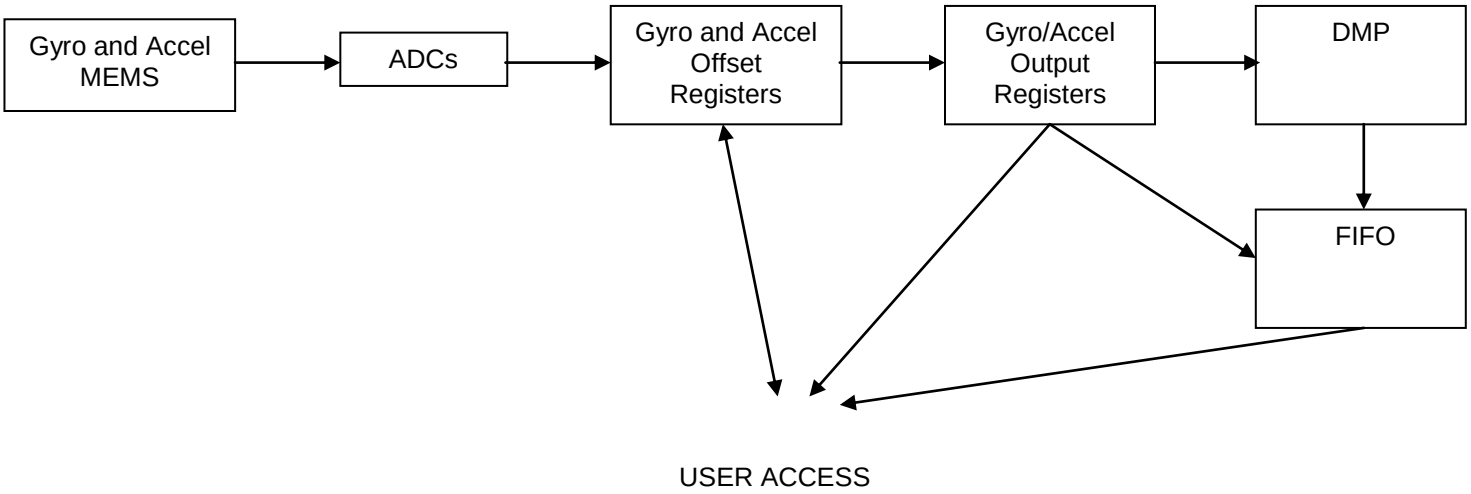
All MEMS sensors are highly recommended to calibrate either at factory or dynamically while in use to get more accurate data. Invensense software and DMP has several algorithms dealing with calibration. However if you are not using these algorithms, offset cancellation can still be achieved using the build in hardware offset registers in the MPU device.

Several Invensense MPU and ICM devices contain offset cancellation registers for the Accelerometer and gyroscope sensors. This document details those registers, a small example on how to get gyro and accel offsets, and how to apply those offsets into these registers.

The MPU devices in which has been tested and confirmed for this documentation are MPU6050, MPU9150, MPU6500, MPU6515, and MPU9250.

4. Data Signal Diagram

How these offset register works is that all data from the MEMS sensors will have these offsets applied before outputting to the data registers for users to read. The offset application is also before all usage concerning FIFO and DMP. Therefore the data in the FIFO, output registers, and used in the DMP will already have these offsets included.



5. Finding the Offset Biases for Accel and Gyro

There are a number of ways to get the biases for the Accel and Gyro, but the general idea is that the device is stationary in a known orientation typically with the MPU part faced up or down.

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.

5.1 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];
        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;
}
```

6. Gyro Offset Registers

6.1 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

6.2 Register Format and Descriptions

The Gyro registers at boot up will have a default value of 0. The value of the bias inputted needs to be in +-1000dps sensitivity range. This means each 1 dps = 32.8 LSB

6.3 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;
}
```

7. Accel Offset Registers

7.1 Register Locations

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

Addr	Name	Description
0x06	XG_OFFS_USRH	Accel X-axis offset cancellation register high byte
0x07	XG_OFFS_USRL	Accel X-axis offset cancellation register low byte
0x08	YG_OFFS_USRH	Accel Y-axis offset cancellation register high byte
0x09	YG_OFFS_USRL	Accel Y-axis offset cancellation register low byte
0x0A	ZG_OFFS_USRH	Accel Z-axis offset cancellation register high byte
0x0B	ZG_OFFS_USRL	Accel Z-axis offset cancellation register low byte

MPU6500/MPU6515/ and MPU9250 are located at the following

Addr	Name	Description
0x77	XG_OFFS_USRH	Accel X-axis offset cancellation register high byte
0x78	XG_OFFS_USRL	Accel X-axis offset cancellation register low byte
0x7A	YG_OFFS_USRH	Accel Y-axis offset cancellation register high byte
0x7B	YG_OFFS_USRL	Accel Y-axis offset cancellation register low byte
0x7D	ZG_OFFS_USRH	Accel Z-axis offset cancellation register high byte
0x0E	ZG_OFFS_USRL	Accel Z-axis offset cancellation register low byte

7.2 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 bootup 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. Format is in +-16G in which 1mg = 2048 LSB
3. Bit 0 on the low byte of each axis is a reserved bit and needs to be preserved.

7.3 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;
}
```

8. Misc

Bias offsets once inputted into the Offset Cancellation Registers will immediate take into effect. However after power off and on the values will need to be reloaded back into these registers. It is recommended that the bias values are saved so that can be easily reapplied after boot up.