AN3922
Rev 2, 03/2010

Data Manipulation and Basic Settings of the MMA8450Q with Driver Code

by: Kimberly Tuck and Derrick Klotz
Applications Engineers

1.0 Introduction

It is important to understand how to program the MMA8450Q to extract and manipulate the acceleration data. The MMA8450Q has many different features which include seven different sample rates, 28 different cut-off frequencies for the high pass filter, 3 different dynamic ranges with different sensitivities for 12-bit output data, 8-bit output data and 8-bit delta data (data that has been filtered through the high pass filter). It also has a 32 sample FIFO for collecting and storing data, which is the most efficient way to access the data for minimizing the I^2C transactions. The manipulation of the data into different formats is important for algorithm development and for display. This application note accompanies the MMA8450Q_Driver Code and will explain the following:

- Changing the different Modes (Shutdown, Standby, 2g, 4g and 8g)
- Changing the Data Rate
- Changing the High Pass Filter Cut off Frequency
- 8-bit data vs. 12-bit data
- Changing Data Formats (hex to counts to decimal numbers)
- Streaming XYZ data polling vs. Streaming XYZ data with interrupts

1.1 Key Words

Shutdown Mode, Standby Mode, Active Mode, High Pass Filter Cut Off Frequency, 8-bit Data, 12-bit Data, Hexadecimal Numbers, Decimal Numbers, Data Formats, Streaming Data, Counts, Polling, Interrupts, FIFO Data, Flush, Sensor Toolbox Demo Board, Driver Code

TABLE OF CONTENTS

1.2 Summary

A. There are 5 different modes: Shutdown Mode, which provides no I^2C communication, Standby Mode which responds to I^2C communication and then 3 active modes; 2g, 4g and 8g.
B. An example of how to set the data rate is shown. There are 6 active mode data rates.
C. An example of how to set the High Pass Filter Cut-off Frequency is given. The delta data is affected by the filter settings.
D. An example and the format conversions for manipulating 12-bit data converting 2's complement hex data to three different formats, which include formatting to signed hex, signed integer (counts) and signed decimal fractions in g's.
E. An example and the format conversions for manipulating 8-bit data converting 2's complement hex data to three different formats, including signed hex, signed integer (counts) and signed decimal fractions in g's. The 8-bit data conversions are valid for the 8-bit XYZ data and the 8-bit High Pass Filter data.
F. An example of how to set up the device to poll the data or configure an interrupt service routine is shown.
G. There is a driver available that will run on the MMA8450Q Sensor Toolbox Demo Board that provides an example in CodeWarrior® for everything discussed in the application note. The driver runs in RealTerm or HyperTerminal™ and can be used to capture and log data in different formats.

2.0 MMA8450Q Consumer 3-axis Accelerometer 3 x 3 x 1 mm

The MMA8450Q has a selectable dynamic range of ±2g, ±4g and ±8g with sensitivities of 1024 counts/g, 512 counts/g and 256 counts/g respectively. The device offers either 8-bit or 12-bit XYZ output data for algorithm development. The chip shot and pinout are shown in Figure 1.

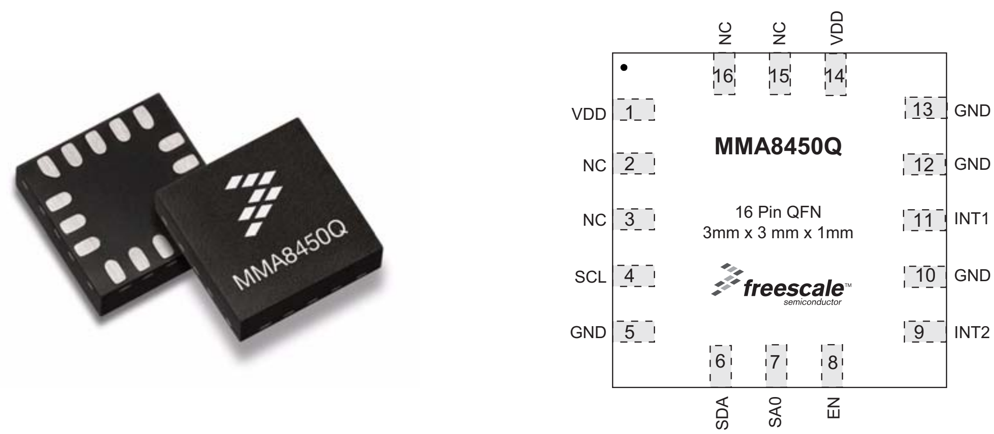

Figure 1. MMA8450Q Consumer 3-axis Accelerometer 3 x 3 x 1 mm

2.1 Key Features of the MMA8450Q

1. Shutdown Mode: Typical < 1 µA, Standby Mode 3 µA
2. Low Power Mode current consumption ranges from 27 µA (1.56 - 50 Hz) to 120 µA (400 Hz)
3. Normal Mode current consumption ranges from 42 µA (1.56 - 50 Hz) to 225 µA (400 Hz)
4. I^2C digital output interface (operates up to 400 kHz Fast Mode)
5. 12-bit and 8-bit data output, 8-bit high pass filtered data output
6. Post Board Mount Offset < ±50 mg typical
7. Self Test X, Y and Z axes

2.2 Two (2) Programmable Interrupt Pins for 8 Interrupt Sources

1. Embedded 4 channels of Motion detection
   a. Freefall or Motion detection: 2 channels
   b. Tap detection: 1 channel
   c. Transient detection: 1 channel
2. Embedded orientation (Portrait/Landscape) detection with hysteresis compensation
3. Embedded automatic ODR change for auto-wake-up and return-to-sleep
4. Embedded 32 sample FIFO
5. Data Ready Interrupt

2.3 Application Notes for the MMA8450Q

The following is a list of Freescale Application Notes written for the MMA8450Q:

- **AN3915**, *Embedded Orientation Detection Using the MMA8450Q*
- **AN3916**, *Offset Calibration of the MMA8450Q*
- **AN3917**, *Motion and Freefall Detection Using the MMA8450Q*
- **AN3918**, *High Pass Filtered Data and Transient Detection Using the MMA8450Q*
- **AN3919**, *MMA8450Q Single/Double and Directional Tap Detection*
- **AN3920**, *Using the 32 Sample First In First Out (FIFO) in the MMA8450Q*
- **AN3921**, *Low Power Modes and Auto-Wake/Sleep Using the MMA8450Q*
- **AN3922**, ***Data Manipulation and Basic Settings of the MMA8450Q***
- **AN3923**, *MMA8450Q Design Checklist and Board Mounting Guidelines*

3.0 Changing Modes of the MMA8450Q

There are five different modes that the MMA8450Q can be in. These modes include Shutdown, Standby, 2g Active, 4g Active and 8g Active. The Standby and Active modes are controlled by the last two bits of the System Control 1 Register (at 0x38), FS1 and FS0, while Shutdown is controlled by the signal level on the EN pin (pin 8).

Table 1. 0x38 CTRL_REG1 Register (Read/Write

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ASLP_RATE1	ASLP_RATE0	0	DR2	DR1	DR0	**FS1**	**FS0**

Table 2. Full Scale Selection

FS1	FS0	Mode	g Range
0	0	Standby	—
0	1	Active	±2g
1	0	Active	±4g
1	1	Active	±8g

3.1 Shutdown Mode

The MMA8450Q is in Shutdown Mode when a digital logic low level is applied to the EN pin (pin 8). In this mode there are no I^2C transactions available and the current consumption of the sensor is < 1 μA. This is the lowest current consumption state. In the case of the Sensor Toolbox Demo Board, the EN pin is controlled by the Port C1 GPIO pin of the microcontroller (MCU).

Code Example:

```
/*
** Put the MMA8450Q into Shutdown Mode.
*/
PTCD_PTCD1 = 0;
```

3.2 Standby Mode

Most, although not quite all changes to the registers must be done while the MMA8450Q is in Standby Mode. Current consumption in Standby Mode is typically 3 μA. To be in Standby Mode the last two bits of CTRL_REG1 must be cleared (FS = 00).

Code Example:

```
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Hold the values for sleep and active Data Rates & clear FS bits.
*/
CTRL_REG1Data &= (ASLP_RATE_MASK + DR_MASK);
/*
** Write the value back into CTRL_REG1 with FS = 00 = Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data);
```

3.3 2g Active Mode

In order to enter 2g Active Mode, the MMA8450Q must first be put into Standby Mode prior to changing the **FS** bits to 01 (as per Table 2). In 2g Active Mode the 12-bit data sensitivity is 1024 counts per g and the 8-bit data sensitivity is 64 counts per g. The current consumption in the active mode depends upon the selected data rate and Low Power Mode.

Code Example:

```
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Write the value back into CTRL_REG1 with FS = 01 = 2g Active Mode.
*/
IIC_RegWrite(CTRL_REG1, (CTRL_REG1Data & (ASLP_RATE_MASK+DR_MASK) | FS0_MASK));
```

3.4 4g Active Mode

In order to enter 4g Active Mode, the MMA8450Q must first be put into Standby Mode prior to changing the **FS** bits 10 (as per Table 2). In 4g Active Mode the 12-bit data sensitivity is 512 counts per g, and the 8-bit data sensitivity is 32 counts per g.

Code Example:

```
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Write the value back into CTRL_REG1 with FS = 10 = 4g Active Mode.
*/
IIC_RegWrite(CTRL_REG1, (CTRL_REG1Data & (ASLP_RATE_MASK+DR_MASK) | FS1_MASK));
```

3.5 8g Active Mode

In order to enter 8g Active Mode, the MMA8450Q must first be put into Standby Mode prior to changing the **FS** bits to 11 (as per Table 2). In 8g Active Mode the 12-bit data sensitivity is 256 counts per g, and the 8-bit data sensitivity is 16 counts per g.

Code Example:

```
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Write the value back into CTRL_REG1 with FS = 11 = 8g Active Mode.
*/
IIC_RegWrite(CTRL_REG1, (CTRL_REG1Data & (ASLP_RATE_MASK+DR_MASK) | FS0_MASK+FS1_MASK));
```

4.0 Setting the Data Rate

The active mode Output Data Rate (ODR) and Sleep Mode Rate are programmable via other control bits in the CTRL_REG1 register, seen in Table 3. Unless the sleep mode is enabled the active mode data rate is the data rate that will always be enabled. Table 4 shows how the DR2:DR0 bits affect the ODR. For more details on how to use the sleep mode, refer to AN3921 “Low Power Modes and Auto Wake Sleep Using the MMA8450Q”. The default data rate is DR = 000, 400 Hz.

Table 3. 0x38 CTRL_REG1 Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ASLP_RATE1	ASLP_RATE0	0	DR2	DR1	DR0	FS1	FS0

The following table is a list of all the available active mode data rates.

Table 4. Output Data Rates

DR2	DR1	DR0	Output Data Rate (ODR)	Time Between Data Samples
0	0	0	400 Hz	2.5 ms
0	0	1	200 Hz	5 ms
0	1	0	100 Hz	10 ms
0	1	1	50 Hz	20 ms
1	0	0	12.5 Hz	80 ms
1	0	1	1.563 Hz	640 ms

Code Example:

```
/*
** Adjust the desired Output Data Rate value as needed.
*/
DataRateValue <<= 2;
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Write the value back into CTRL_REG1 with the desired Output Data Rate.
*/
IIC_RegWrite(CTRL_REG1, (CTRL_REG1Data & (ASLP_RATE_MASK+FS_MASK) | DataRateValue));
```

5.0 Setting the High Pass Filter Cut-off Frequency

The HP_FILTER_CUTOFF register (at 0x17) sets the high pass cut off frequency, Fc, for the data. The output of this filter is provided in the **OUT_X_DELTA**, **OUT_Y_DELTA**, and **OUT_Z_DELTA** registers (at 0x0C, 0x0D and 0x0E, respectively). These are the 8-bit high pass filtered data output for X, Y and Z. Note that the available cut-off frequencies change depending upon the set Output Data Rate.

Table 5. 0x17 HP_FILTER_CUTOFF: High Pass Filter Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	**SEL1**	**SEL0**

Table 6 presents the different cut-off frequencies for the high pass filter based on the different set data rates.

Table 6. HP_FILTER_CUTOFF Setting Options

SEL1	SEL0	Fc @ ODR = 400 Hz	Fc (Hz) @ ODR = 200 Hz	Fc (Hz) @ ODR = 100 Hz	Fc (Hz) @ ODR = 50 Hz	Fc (Hz) @ ODR = 12.5 Hz	Fc (Hz) @ ODR = 1.563 Hz
0	0	4	2	1	0.5	0.125	0.01
0	1	2	1	0.5	0.25	0.063	0.007
1	0	1	0.5	0.25	0.125	0.031	0.004
1	1	0.5	0.25	0.125	0.062	0.016	0.002

To set the cut off frequency, a value from 0x00 to 0x03 must be chosen for the **SEL** bits, as per Table 6. In order to make this change, the sensor must be in Standby Mode prior to writing to the HP_FILTER_CUTOFF register, after which the previous Active mode would normally be reselected.

Consider an example where the MMA8450Q is operating in 2g Active Mode with a 400 Hz ODR and the default Fc of 4 Hz. The following code demonstrates how to change Fc to 1 Hz without making any other operational modifications. This is done by changing the SEL bits to 10 (as per Table 6).

Code Example:

```
/*
** Select the desired cut-off frequency.
*/
CutOffValue=2;
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Write to the HP_FILTER_CUTOFF register to set the cut-off frequency.
*/
IIC_RegWrite(HP_FILTER_CUTOFF_REG, CutOffValue);
/*
** Write to CTRL_REG1 restoring the previous operating mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data);
```

6.0 12-bit Data Streaming and Data Conversions

The MMA8450Q can provide 12-bit XYZ data. This section is an overview of how to manipulate the data to continuously burst out 12-bit data in different data formats from the MCU.

The XYZ_DATA_CFG register (at 0x16) has control bits used to enable the internal event flag upon the detection of new data, which would occur at the selected Output Data Rate. At least one of the data ready enable bits (ZDEFE, YDEFE and XDEFE shown in Table 7) must be set to activate the event flag upon the detection of an updated sample.

Table 7. 0x16 XYZ_DATA_CFG: Sensor Data Configuration Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FDE	0	0	0	0	**ZDEFE**	YDEFE	XDEFE

The following line of code will enable the activation of the event flag upon a new Z-axis sample by writing to the XYZ_DATA_CFG register with ZDEFE = 1:

IIC_RegWrite(XYZ_DATA_CFG_REG, ZDEFE_MASK);

Once configured, the event flag can be monitored by reading the STATUS register (at 0x04). This can be done by using either a polling or interrupt technique, which is discussed later in Section 8.0 of this document. Regardless of the technique used, the STATUS register needs to be read and the appropriate flag monitored.

Table 8. 0x00, 0x04, 0x0B STATUS: Data Status Registers (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ZYXOW	ZOW	YOW	XOW	**ZYXDR**	ZDR	YDR	XDR

The ZYXDR flag is set whenever there is new data available in any axis. The following code example monitors this flag and, upon the detection of new data, reads the 12-bit XYZ data into an array (`value[]`) in RAM with a single, multi-byte I^2C access. These values are then copied into 16-bit variables prior to further processing.

Code Example:

```
/*
** Poll the ZYXDR status bit and wait for it to set.
*/
RegisterFlag.Byte = IIC_RegRead(STATUS_04_REG);

if (RegisterFlag.ZYXDR_BIT == 1)
{
    /*
    ** Read 12-bit XYZ results using a 6 byte IIC access.
    */
    IIC_RegReadN(OUT_X_LSB_REG, 6, &value[0]);
    /*
    ** Copy and save each result as a 16-bit left-justified value.
    */
    x_value.Byte.lo = value[0] << 4;
    x_value.Byte.hi = value[1];
    y_value.Byte.lo = value[2] << 4;
    y_value.Byte.hi = value[3];
    z_value.Byte.lo = value[4] << 4;
    z_value.Byte.hi = value[5];
}
```

Note in the previous code that each lower byte result is shifted to the left four places thereby configuring the corresponding 16-bit results in left-justified format (2's complement numbers). An example of the register and variable formats for the X-axis result is provided here:

Table 9. 0x05 OUT_X_MSB: X_MSB Register (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD11	XD10	XD9	XD8	XD7	XD6	XD5	XD4

Table 10. 0x06 OUT_X_LSB: X_LSB Register (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	XD3	XD2	XD1	XD0

Table 11. x_value 16-bit 2's Complement Result

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD11	XD10	XD9	XD8	XD7	XD6	XD5	XD4	XD3	XD2	XD1	XD0	0	0	0	0

6.1 Converting a 12-bit 2's Complement Hex Number to a Signed Hex Number

Converting to a signed hexadecimal number means that the value is converted from 2's complement data to a hexadecimal number with either a leading + or – sign.

The sign of the result is easy to determine by simply checking if the high byte of the value is greater than 0x7F. If so, then the value is a negative number and needs to be transformed by performing a 2's complement conversion. This involves executing a 1's complement (i.e., switch all 1's to 0's and all 0's to 1's) and followed by adding 1 to the result. The following code outputs the data in this format:

Code Example:

```
void SCI_s12int_Out (tword data)
{
    byte c;
    /*
    ** Determine sign and output
    */
    if (data.Byte.hi > 0x7F)
    {
        SCI_CharOut ('-');
        data.Word = ~data.Word + 1;
    }
     else
    {
    SCI_CharOut ('+');
    }
    SCISendString ("0x");
    /*
    ** Calculate and output result
    */
    c = (data.Byte.hi >>4);
    SCI_NibbOut(c);
    c = (data.Byte.hi & 0x0F);
    SCI_NibbOut(c);
    c = (data.Byte.lo >>4);
    SCI_NibbOut(c);
}
```

Here is the routine used above which will output a single nibble character:

```
void SCI_NibbOut(byte data)
{
    byte c;
    c = data + 0x30;
    if (c > 0x39)
    c += 0x07;
    SCI_CharOut(c);
}
```

6.2 Converting a 12-bit 2's Complement Hex Number to a Signed Integer (Counts)

Converting to a signed value into counts implies that the 2's complement hex number is converted to an integer number with a + or – sign.

Example: 0xABC0 = -1348
0x5440 = +1348

This conversion is similar to the one shown previously with the added step of converting a 12-bit binary value into a decimal result that could contain up to 4 digits (i.e., 0x7FF = +2047). The code below performs this conversion. It also adds the additional output formatting step of replacing each leading zero digit with a space character, which is done by passing 0xF0 to SCI_NibbOut() (shown in the previous code example). Upon close examination it is seen that this routine will add 0x30 to 0xF0, resulting in a value of 0x120 which gets truncated to 0x20 – the ASCII space character.

Code Example:

```
void SCI_s12dec_Out (tword data)
{
    byte a, b, c, d;
    word r;
    /*
    ** Determine sign and output
    */
    if (data.Byte.hi > 0x7F)
    {
        SCI_CharOut ('-');
        data.Word = ~data.Word + 1;
    }
    else
    {
        SCI_CharOut ('+');
    }
    /*
    ** Calculate decimal equivalence:
    **    a = thousands
    **    b = hundreds
    **    c = tens
    **    d = ones
    */
    a = (byte)((data.Word >>4) / 1000);
    r = (data.Word >>4) % 1000;
    b = (byte)(r / 100);
    r %= 100;
    c = (byte)(r / 10);
    d = (byte)(r % 10);
    /*
    ** Format adjustment for leading zeros
    */
    if (a == 0)
    {
        a = 0xF0;
        if (b == 0)
        {
            b = 0xF0;
            if (c == 0)
            {
            c = 0xF0;
            }
        }
    }
    /*
    ** Output result
    */
    SCI_NibbOut (a);
    SCI_NibbOut (b);
    SCI_NibbOut (c);
    SCI_NibbOut (d);
}
```

6.3 Converting a 12-bit 2's Complement Hex Number to a Signed Decimal Fraction in g's

Converting to a signed value into g's requires performing the same operations as shown previoulsy with the added step of resolving the integer and fractional portions of the value. The scale of the accelerometer's Active Mode (i.e., either 2g, 4g or 8g) determines the location of the inferred radix point separating these segments and thereby the overall sensitivity of the result. In all cases, the most significant bit, Bit 11, represents the sign of the result (either positive or negative).

- In 2g Active Mode 1g = 1024 counts. Therefore Bit 10 is the only bit that will contribute to an integer value of either 0, 1. 2^10 = 1024.
- In 4g Active Mode 1g = 512 counts. Therefore Bits 10 and 9 will contribute to an integer value of 0, 1, 2, or 3.
- In 8g Active Mode 1g = 256 counts. Therefore Bits 10, 9 and 8 will contribute to an integer value of 0,1, 2, 3, 4, 5, 6, and 7.

This is summarized in Table 12.

Table 12. Full Scale Value with Corresponding Integer Bits and Fraction Bits

Full Scale Value	Counts/g	Sign Bit	Integer Bits	Fraction Bits
2g	1024	11	10 (2^{10} = 1024)	0 through 9
4g	512	11	10 (2^{10} = 1024), 9 (2^{9} = 512)	0 through 8
8g	256	11	10 ($2^{1\,0}$= 1024), 9 (2^{9} = 512), 8 (2^{8} = 256)	0 through 7

6.3.1 2g Active Mode

Adjusting the data into 16-bit left-justified format, the implied radix point of a result when operating in 2g Active Mode is between bits 13 and 14, as can be seen in Table 13. The row labeled as "MMA8450Q 12-bit" shows where the 12-bits of the result are placed in this format, with the row labeled as "MSB/LSB" indicating which result register was the source of the data, either the most-significant byte ("M") or the least-significant byte ("L"). The row labeled as "Integer/Fraction" shows that Bit 15 is the sign bit ("±") while the single integer bit is located at Bit 14 ("I").

Table 13. 2g Active Mode 12-bit Data Conversion to Decimal Fraction Number

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	11	10	9	8	7	6	5	4	3	2	1	0	x	x	x	x
Integer/Fraction	±	I	F	F	F	F	F	F	F	F	F	F	x	x	x	x
MSB/LSB	M	M	M	M	M	M	M	M	L	L	L	L	0	0	0	0

Once the sign and integer of the result have been determined, the result is logically shifted to the left by two binary locations, leaving only the fraction portion of the result, as can be seen in Table 14.

Table 14. 2g Active Mode 12-bit Data in Word Format After Left Shift to Eliminate Integer and Sign Bits

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	9	8	7	6	5	4	3	2	1	0	x	x	x	x	x	x
Integer/Fraction	F	F	F	F	F	F	F	F	F	F	x	x	x	x	x	x
Fraction Bits	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	x	x	x	x	x	x
MSB/LSB	M	M	M	M	M	M	L	L	L	L	0	0	0	0	0	0

This leaves 6 MSB bits and 4 LSB bits after shifting left by two. Therefore there are 10 bits for the fraction portion in 2g Active Mode. The 2g Active Mode has the highest number of bits for the fraction portion with 10 bits because it has the highest sensitivity. In Table 15 the decimal value is rounded to the fourth decimal place because the final fraction number will have four significant digits.

Table 15. 2g Active Mode Fraction Values

	2g Mode	Calculation	Rounded to 4th Decimal Place	Integer Number
2^{-1}	2^{9} = 512	512/1024 = 0.5	0.5000	5000
2^{-2}	2^{8} = 256	256/1024 = 0.25	0.2500	2500
2^{-3}	2^{7} = 128	128/1024 = 0.125	0.1250	1250
2^{-4}	2^{6} = 64	64/1024 = 0.0625	0.0625	625
2^{-5}	2^{5} = 32	32/1024 = 0.03125	0.0313	313
2^{-6}	2^{4} = 16	16/1024 = 0.015625	0.0156	156
2^{-7}	2 = 8	8/1024 = 0.0078125	0.0078	78
2^{-8}	2^{2} = 4	4/1024 = 0.00390625	0.0039	39
2^{-9}	2^{1} = 2	2/1024 = 0.001953125	0.0020	20
2^{-10}	2^{0} = 1	1/1024 = 0.0009765652	0.0010	10

The values shown in Table 15 are translated here into C macros for use in the code example at the end of this section:

```
#define FRAC_2d1        5000
#define FRAC_2d2        2500
#define FRAC_2d3        1250
#define FRAC_2d4         625
#define FRAC_2d5         313
#define FRAC_2d6         156
#define FRAC_2d7          78
#define FRAC_2d8          39
#define FRAC_2d9          20
#define FRAC_2d10         10
```

For each of the ten fraction bits, if the value of the bit is set then the corresponding decimal value will be added to the total. As an example, if bits 8, 6 and 4 are set then the total will be (2500 + 625 + 156 = 3281) which corresponds to 0.3281. The highest fractional value occurs when all fraction bits are set (5000 + 2500 + 1250 + 625 + 313 + 156 + 78 + 39 + 20 + 10 = 9991) which corresponds to 0.9991. In 2g Active Mode the resolution is 0.976 mg. Calculating out the fraction to 4 significant digits gives a resolution of 0.9 mg for 2g Active Mode, which should be good enough.

6.3.2 4g Active Mode

In 4g Active Mode there are two integer bits and nine fraction bits as shown in Table 16.

Table 16. 4g Active Mode 12-bit Data Conversion to Decimal Fraction Number

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	11	10	9	8	7	6	5	4	3	2	1	0	x	x	x	x
Integer/Fraction	±	I	I	F	F	F	F	F	F	F	F	F	x	x	x	x
MSB/LSB	M	M	M	M	M	M	M	M	L	L	L	L	0	0	0	0

In this case, logically shifting the sample to the left by three binary locations leaves the fractional portion of the result, shown in Table 17. Table 17 shows the nine bits of the fraction with the corresponding decimal values for each bit identified in Table 18.

Table 17. 4g Active Mode 12-bit in Word Format After Left Shift to Eliminate Integer and Sign Bits

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	8	7	6	5	4	3	2	1	0	x	x	x	x	x	x	x
Integer/Fraction	F	F	F	F	F	F	F	F	F	x	x	x	x	x	x	x
Fraction Bits	-1	-2	-3	-4	-5	-6	-7	-8	-9	x	x	x	x	x	x	x
MSB/LSB	M	M	M	M	M	L	L	L	L	0	0	0	0	0	0	0

Table 18. 4g Active Mode Fraction Values

	4g Mode	Calculation	Rounded to 4th Decimal Place	Integer Number
2^{-1}	$2^8 = 256$	256/512 = 0.5	0.5000	5000
2^{-2}	$2^7 = 128$	128/512 = 0.25	0.2500	2500
2^{-3}	$2^6 = 64$	64/512 = 0.125	0.1250	1250
2^{-4}	$2^5 = 32$	32/512 = 0.0625	0.0625	625
2^{-5}	$2^4 = 16$	16/512 = 0.03125	0.0313	313
2^{-6}	$2^3 = 8$	8/512 = 0.015625	0.0156	156
2^{-7}	$2^2 = 4$	4/512 = 0.0078125	0.0078	78
2^{-8}	$2^1 = 2$	2/512 = 0.00390625	0.0039	39
2^{-9}	$2^0 = 1$	1/512 = 0.001953125	0.0020	20

For each of the nine fraction bits, if the value of the bit is set then the corresponding decimal value will be added to the total. As an example, if bits 8, 6 and 4 are set then the total will be (5000 + 1250 + 313 = 6563) which corresponds to 0.6563. The highest fractional value occurs when all fraction bits are set (5000 + 2500 + 1250 + 625 + 313 + 156 + 78 + 39 + 20 = 9981) which corresponds to 0.9981. The resolution in 4g Active Mode is 1.95 mg. Calculating the fractional value to 4 significant digits results in the resolution being 1.9 mg, which should be good enough.

6.3.3 8g Active Mode

In 8g Active Mode there are three integer bits, leaving eight bits for the fraction as per Table 19.

Table 19. 8g Active Mode 12-bit Data Conversion to Decimal Fraction Number

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	11	10	9	8	7	6	5	4	3	2	1	0	x	x	x	x
Integer/Fraction	±	I	I	I	F	F	F	F	F	F	F	F	x	x	x	x
MSB/LSB	M	M	M	M	M	M	M	M	L	L	L	L	0	0	0	0

The fractional portion of the result can be extracted by logically shifting the sample to the left by four binary locations. Once again, this result is shown in Table 20; with Table 21 providing the corresponding decimal values.

Table 20. 8g Active Mode 12-bit in Word Format After Left Shift to Eliminate Integer and Sign Bits

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	7	6	5	4	3	2	1	0	x	x	x	x	x	x	x	x
Integer/Fraction	F	F	F	F	F	F	F	F	x	x	x	x	x	x	x	x
Fraction Bits	-1	-2	-3	-4	-5	-6	-7	-8	x	x	x	x	x	x	x	x
MSB/LSB	M	M	M	M	L	L	L	L	0	0	0	0	0	0	0	0

Table 21. 8g Active Mode Fraction Values

	8g Mode	Calculation (256 counts/g)	Rounded to 4th Decimal Place	Integer Number
2^{-1}	$2^7 = 128$	128/256 = 0.5	0.5000	5000
2^{-2}	$2^6 = 64$	64/256 = 0.25	0.2500	2500
2^{-3}	$2^5 = 32$	32/256 = 0.125	0.1250	1250
2^{-4}	$2^4 = 16$	16/256 = 0.0625	0.0625	625
2^{-5}	$2^3 = 8$	8/256 = 0.03125	0.0313	313
2^{-6}	$2^2 = 4$	4/256 = 0.015625	0.0156	156
2^{-7}	$2^1 = 2$	2/256 = 0.0078125	0.0078	78
2^{-8}	$2^0 = 1$	1/256 = 0.00390625	0.0039	39

For each of the eight fraction bits, if the value of the bit is set then the corresponding decimal value will be added to the total. As an example, if bit 7, 6 and 4 are set then the total will be (5000+2500+625=8125) which corresponds to 0.8125. The highest fractional value occurs when all fraction bits are set (5000+2500+1250+625+313+156+78+39 =9961) which corresponds to 0.9961. The resolution in 8g Active Mode is 3.90mg. Calculating the fractional value to 4 significant digits results in the resolution being 3.9mg.

Here is the code example which performs the conversion of a 12-bit signed 2-s complement value into a signed decimal fraction displayed in g's.

Code Example:

```
void SCI_s12frac_Out (tword data)
{
    BIT_FIELD value;
    word result;
    byte a, b, c, d;
    word r;
    /*
    ** Determine sign and output
    */
    if (data.Byte.hi > 0x7F)
    {
        SCI_CharOut ('-');

        data.Word &= 0xFFF0;
        data.Word = ~data.Word + 1;
    }
    else
    {
        SCI_CharOut ('+');
    }
    /*
    ** Determine integer value and output
    */
    if (full_scale == FULL_SCALE_2G)
    {
        SCI_NibbOut((data.Byte.hi & 0x40) >>6);
        data.Word = data.Word <<2;
    }
    else if (full_scale == FULL_SCALE_4G)
    {
        SCI_NibbOut((data.Byte.hi & 0x60) >>5);
        data.Word = data.Word <<3;
    }
    else
    {
    SCI_NibbOut((data.Byte.hi & 0x70) >>4);
    data.Word = data.Word <<4;
}
SCI_CharOut ('.');
/*
** Determine fractional value
*/
result = 0;
value.Byte = data.Byte.hi;
if (value.Bit._7 == 1)
    result += FRAC_2d1;
if (value.Bit._6 == 1)
    result += FRAC_2d2;
if (value.Bit._5 == 1)
    result += FRAC_2d3;
if (value.Bit._4 == 1)
    result += FRAC_2d4;
```

```
data.Word = data.Word <<4;
value.Byte = data.Byte.hi;

if (value.Bit._7 == 1)
    result += FRAC_2d5;
if (value.Bit._6 == 1)
    result += FRAC_2d6;
if (value.Bit._5 == 1)
    result += FRAC_2d7;
if (value.Bit._4 == 1)
    result += FRAC_2d8;

if (full_scale != FULL_SCALE_8G)
{
    if (value.Bit._3 == 1)
        result += FRAC_2d9;
    if (full_scale == FULL_SCALE_2G)
        if (value.Bit._2 == 1)
            result += FRAC_2d10;
}
/*
** Convert fractional value to 4 decimal places
*/
r = result % 1000;
a = (byte)(result / 1000);
b = (byte)(r / 100);
r %= 100;
c = (byte)(r / 10);
d = (byte)(r % 10);
/*
** Output fractional value
*/
SCI_NibbOut (a);
SCI_NibbOut (b);
SCI_NibbOut (c);
SCI_NibbOut (d);
SCI_CharOut ('g');
}
```

7.0 8-bit XYZ Data or Delta Data Streaming and Conversions

The MMA8450Q can provide 8-bit XYZ data or delta data. Reading these values requires the same initialization set up as was shown in the previous section for 12-bit data. One or several of the X, Y or Z data ready enable bits must be enabled for the sensor to detect an updated sample. These control bits are located in the XYZ_DATA_CFG register (at 0x16), see Table 7. In this example the Z Data Ready is enabled with the following code:

```
IIC_RegWrite(XYZ_DATA_CFG_REG, ZDEFE_MASK);
```

As was shown when using 12-bit data, the ZYXDR flag in the STATUS register must be monitored. However, there are three separate locations available for this register, making it easier and faster to perform multi-byte I^2C data read accesses. Reading the STATUS register located at 0x00 (STATUS_00_REG), should be done when fetching 8-bit data. Reading this same register at location 0x04 (STATUS_04_REG) is intended for 12-bit data, as was shown in the previous section of this document. When 8-bit delta data is desired, the third location at 0x0B (STATUS_0B_REG) is preferred.

The first code example shown here demonstrates how to read the 8-bit sample data by polling the STATUS register located at 0x00. Note that the samples are saved in 16-bit left-justified format in order to be able to effectively reuse the data conversion subroutines previously described.

Code Example:

```
/*
** Poll the ZYXDR status bit and wait for it to set.
*/
RegisterFlag.Byte = IIC_RegRead(STATUS_00_REG);

if (RegisterFlag.ZYXDR_BIT == 1)
{
    /*
    ** Read 8-bit XYZ results using a 3 byte IIC access.
    */
    IIC_RegReadN(OUT_X_MSB8_REG, 3, &value[0]);
    /*
    ** Copy and save each result as a 16-bit left-justified value.
    */
    x_value.Byte.lo = 0;
    x_value.Byte.hi = value[0];
    y_value.Byte.lo = 0;
    y_value.Byte.hi = value[1];
    z_value.Byte.lo = 0;
    z_value.Byte.hi = value[2];
}
```

The second code example applies to reading the 8-bit delta data. In this case the STATUS register is polled using address 0x0B. The 16-bit left-justified format is used here as well. In fact, the routine shown here reads the 8-bit sample data as well, which is a step required in order to clear the ZYXDR flag.

Code Example:

```
/*
** Poll the ZYXDR status bit and wait for it to set.
*/
RegisterFlag.Byte = IIC_RegRead(STATUS_0B_REG);

if (RegisterFlag.ZYXDR_BIT == 1)
{
    /*
    ** Read 8-bit XYZ results using a 3 byte IIC access to clear the ZYXDR bit.
    */
    IIC_RegReadN(OUT_X_MSB8_REG, 3, &value[0]);
    /*
    ** Read 8-bit XYZ delta data using a 3 byte IIC access.
    */
    IIC_RegReadN(OUT_X_DELTA_REG, 3, &value[3]);
    /*
```

```
    ** Copy and save each result as a 16-bit left-justified value.
    */
    x_value.Byte.lo = 0;
    x_value.Byte.hi = value[3];
    y_value.Byte.lo = 0;
    y_value.Byte.hi = value[4];
    z_value.Byte.lo = 0;
    z_value.Byte.hi = value[5];
}
```

The 8-bit values can be converted to the various formats described previously for the 12-bit samples using the same conversion subroutines, provided that the data is formatted appropriately. This is easily done by simply copying the sample into the upper 8 bits and “zero-filling” the lower byte, as shown in the code examples above. As a means of comparison with the 12-bit X-axis sample shown in Table 9, Table 10 and Table 11, the result of this procedure is shown below.

Table 22. 0x01 OUT_X_MSB8_REG: X_MSB Register (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD11	XD10	XD9	XD8	XD7	XD6	XD5	XD4

Table 23. x_value 16-bit 2’s Complement Result

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD11	XD10	XD9	XD8	XD7	XD6	XD5	XD4	0	0	0	0	0	0	0	0

Refer to Section 6.0 for further details regarding applicable data conversion formats. Specific 8-bit versions of the previous routines are provided in the following sections.

7.1 Converting 8-bit 2’s Complement Hex Number to a Signed Hex Number

As was shown in Section 6.1, converting to a signed hexadecimal number means that the value is converted from 2’s complement data to a hexadecimal number with either a leading + or – sign.

The sign of the result is easy to determine by simply checking if it is greater than 0x7F. If so, then the value is a negative number and needs to be transformed by performing a 2’s complement conversion. This involves executing a 1’s complement (i.e., switch all 1’s to 0’s and all 0’s to 1’s) and followed by adding 1 to the result. The following code outputs the data in this format:

Code Example:

```
void SCI_s8int_Out (byte data)
{
    byte c;
    /*
    ** Determine sign and output
    */
    if (data > 0x7F)
    {
        SCI_CharOut ('-');
        data = ~data + 1;
    }
    else
    {
        SCI_CharOut ('+');
    }
    SCISendString ("0x");
    /*
    ** Calculate and output result
    */
    c = (data >>4);
    SCI_NibbOut(c);
    c = (data & 0x0F);
    SCI_NibbOut(c);
}
```

Note that subroutine SCI_NibbOut() is described in Section 6.1.

7.2 Converting an 8-bit 2's Complement Hex Number to a Signed Integer Number

Converting to a signed value into counts implies that the 2's complement hex number is converted to an integer number with a + or – sign.

Example: 0xAB = -84
0x54 = +84

This conversion is similar to the one shown previously with the added step of converting an 8-bit binary value into a decimal result that could contain up to 3 digits (i.e., 0x7F = +127). The code below performs this conversion. It also adds the additional output formatting step of replacing each leading zero digit with a space character, as described in Section 6.2.

Code Example:

```
void SCI_s8dec_Out (byte data)
{
    byte a, b, c;
    word r;
    /*
    ** Determine sign and output
    */
    if (data > 0x7F)
    {
        SCI_CharOut ('-');
        data = ~data + 1;
    }
    else
    {
        SCI_CharOut ('+');
    }
    /*
    ** Calculate decimal equivalence:
    **    a = hundreds
    **    b = tens
    **    c = ones
    */
    a = (byte)(data / 100);
    r = (data) % 100;
    b = (byte)(r / 10);
    c = (byte)(r % 10);
    /*
    ** Format adjustment for leading zeros
    */
    if (a == 0)
    {
        a = 0xF0;
        if (b == 0)
        {
            b = 0xF0;
        }
    }
    /*
    ** Output result
    */
    SCI_NibbOut (a);
    SCI_NibbOut (b);
    SCI_NibbOut (c);
}
```

7.3 Converting 8-bit 2's Complement Hex Number to a Signed Decimal Fraction in g's

The mechanics of converting 8-bit data to a signed value into g's is done in the same manner as that shown for 12-bit data in Section 6.3. Therefore, only the details regarding the use of 2g Active Mode are described here in Section 7.3.1.

The scale of the accelerometer's Active Mode (i.e., either 2g, 4g or 8g) determines the location of the inferred radix point separating these segments and thereby the overall sensitivity of the result. In all cases the most significant bit, Bit 7, represents the sign of the result (either positive or negative).

- In 2g Active Mode 1g = 64 counts. Therefore Bit 6 is the only bit that will contribute to an integer value of either 0, 1. 2^6 = 64.
- In 4g Active Mode 1g = 32 counts. Therefore Bits 6 and 5 will contribute to an integer value of 0, 1, 2, or 3.
- In 8g Active Mode 1g = 16 counts. Therefore Bits 6, 5 and 4 will contribute to an integer value of 0,1, 2, 3, 4, 5, 6, and 7.

This is summarized in Table 24.

Table 24. Full Scale Value with Corresponding Integer Bits and Fraction Bits

Full Scale Value	Counts/g	Sign Bit	Integer Bits	Fraction Bits
2g	64	7	6 ($2^6 = 64$)	0 through 5
4g	32	7	6 ($2^6 = 64$), 5 ($2^5 = 32$)	0 through 4
8g	16	7	6 ($2^6 = 64$), 5 ($2^5 = 32$), 4 ($2^4 = 16$)	0 through 3

7.3.1 2g Active Mode

The subroutine provided in Section 6.3 can be used to convert 8-bit data, provided that the data has been adjusted into the 16-bit left-justified format, as shown in Table 25.

Table 25. 2g Active Mode 8-bit Data Conversion to Decimal Fraction Number

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	7	6	5	4	3	2	1	0	x	x	x	x	x	x	x	x
Integer/Fraction	±	I	F	F	F	F	F	F	x	x	x	x	x	x	x	x
MSB/LSB	M	M	M	M	M	M	M	M	0	0	0	0	0	0	0	0

Performing a logical shift to the left by two binary locations will provide the result's fractional portion, as can be seen in Table 26.

Table 26. 2g Active Mode 8-bit data in Word Format After Left Shift to Eliminate Integer and Sign Bits

Word Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMA8450Q 12-bit	5	4	3	2	1	0	x	x	x	x	x	x	x	x	x	x
Integer/Fraction	F	F	F	F	F	F	x	x	x	x	x	x	x	x	x	x
Fraction Bits	-1	-2	-3	-4	-5	-6	x	x	x	x	x	x	x	x	x	x
MSB/LSB	M	M	M	M	M	M	0	0	0	0	0	0	0	0	0	0

The decimal values of the six bits of the fractional portion are shown in Table 27. These values are rounded to the fourth decimal place because the final fraction number will have four significant digits.

Table 27. 2g Active Mode Fraction Values

	2g Mode	Calculation (64 counts/g)	Rounded to 4th Decimal Place	Integer Number
2^{-1}	$2^5 = 32$	32/64 = 0.5	0.5000	5000
2^{-2}	$2^4 = 16$	16/64 = 0.25	0.2500	2500
2^{-3}	$2^3 = 8$	8/64 = 0.125	0.1250	1250
2^{-4}	$2^2 = 4$	4/64 = 0.0625	0.0625	625
2^{-5}	$2^1 = 2$	2/64 = 0.03125	0.0313	313
2^{-6}	$2^0 = 1$	1/64 = 0.015625	0.0156	156

For each of the six fraction bits, if the value of the bit is set then the corresponding decimal value will be added to the total. The highest fractional value occurs when all fraction bits are set (5000 + 2500 + 1250 + 625 + 313 + 156 = 9844) which corresponds to 0.9844. In 2g Active Mode the resolution is 15.625 mg. Calculating out the fraction to 4 significant digits gives a resolution of 15.6 mg for 2g Active Mode, which should be good enough.

Note that following the same methodology shown here for 2g Active Mode, the same calculations and conversions can be performed for the 4g and 8g Active Modes.

8.0 Polling Data vs. Interrupts

The data can be polled continuously or it can be set up to a hardware interrupt or exception to the MCU each time new data is ready. Depending on the circumstances, one might be more desirable than the other although polling typically is less efficient.

8.1 Polling Data

Polling requires less configuration of the device and is very simple to implement. However, the MCU must poll the sensor at a rate that is faster then the Output Data Rate. Otherwise, if the polling is too slow, the data samples can be missed. The MCU can detect this condition by checking the overwrite flags in the STATUS register (i.e., ZYXOW, ZOW, YOW, and XOW). The code examples provided so far in this document have primarily described the polling technique. As a summary, here is a more complete example of the basic code, specific to the operation of the MMA8450Q, required to continuously poll 12-bit XYZ data using the 2g Active Mode:

Code Example:

```
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Write the value back into CTRL_REG1 with FS = 01 = 2g Active Mode.
*/
IIC_RegWrite(CTRL_REG1, (CTRL_REG1Data & (ASLP_RATE_MASK+DR_MASK) | FS0_MASK));
    /*
    ** Using a basic control loop, continuously poll the sensor.
    */
for (;;)
{
    /*
    ** Poll the ZYXDR status bit and wait for it to set.
    */
    RegisterFlag.Byte = IIC_RegRead(STATUS_04_REG);

    if (RegisterFlag.ZYXDR_BIT == 1)
    {
        /*
        ** Read 12-bit XYZ results using a 6 byte IIC access.
        */
        IIC_RegReadN(OUT_X_LSB_REG, 6, &value[0]);
        /*
        ** Copy and save each result as a 16-bit left-justified value.
        */
        x_value.Byte.lo = value[0] << 4;
        x_value.Byte.hi = value[1];
        y_value.Byte.lo = value[2] << 4;
        y_value.Byte.hi = value[3];
        z_value.Byte.lo = value[4] << 4;
        z_value.Byte.hi = value[5];
        /*
        ** Go process the XYZ data.
        */
        GoProcessXYZ(&value[0]);
    }
    /*
    ** Perform other necessary operations.
    */
    etc();
}
```

8.2 Interrupt Routine to Access Data

Streaming data via hardware interrupts is more efficient than polling as the MCU only interfaces with the MMA8450Q when it has new data. The data is read only when new data is available. If the data is not read every time, there is a new sample and will be indicated by the overwrite register flags. The following are the register settings to configure the MMA8450Q to generate an interrupt upon each new Z-axis sample (which also corresponds to new X-axis and Y-axis samples as well). The MCU's Interrupt Service Routine (ISR) shown below responds by reading the 12-bit XYZ data and setting a software flag indicating the arrival of new data. It is considered to be good practice to keep ISRs as fast as possible, so the actual processing of this data is not done here. Note the similarities to the polling method. Accessing 8-bit sample or delta data can also be performed in a similar fashion.

Code Example:

```
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Configure the INT pins for Open Drain and Active Low
*/
IIC_RegWrite(CTRL_REG3, PP_OD_MASK);
/*
** Enable the Data Ready Interrupt and route it to INT1.
** Enable the Z-axis Data Event Flag.
*/
IIC_RegWrite(CTRL_REG4, INT_EN_DRDY_MASK);
IIC_RegWrite(CTRL_REG5, INT_CFG_DRDY_MASK);
IIC_RegWrite(XYZ_DATA_CFG_REG, ZDEFE_MASK);
/*
** Write to CTRL_REG1 restoring the previous operating mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data);
/*
** etc.
*/



/*********************************************************\
* MMA8450Q Interrupt Service Routine
\*********************************************************/
interrupt void isr_MMA8450Q (void)
{
    /*
    ** Clear the MCU's interrupt flag
    */
    CLEAR_MMA8450Q_INTERRUPT;
    /*
    ** Go read the Interrupt Source Register
    */
    RegisterFlag.Byte = IIC_RegRead(INT_SOURCE_REG);
    if (RegisterFlag.SRC_DRDY_BIT == 1)
    {
        /*
        ** Read 12-bit XYZ results using a 6 byte IIC access.
        */
        IIC_RegReadN(OUT_X_LSB_REG, 6, &value[0]);
        /*
        ** Copy and save each result as a 16-bit left-justified value.
        */
        x_value.Byte.lo = value[0] << 4;
```

```
            x_value.Byte.hi = value[1];
            y_value.Byte.lo = value[2] << 4;
            y_value.Byte.hi = value[3];
            z_value.Byte.lo = value[4] << 4;
            z_value.Byte.hi = value[5];
            /*
            ** Indicate that new data exists to be processed.
            */
            NEW_DATA = TRUE;
        }
    }
```

9.0 Using the 32 Sample FIFO

The most efficient way to access data, particularly for data logging is to use the internal 32 sample FIFO buffer. This minimizes the number of I^2C transactions. For more information on how to configure the FIFO please refer to AN3920. The FIFO can be configured in circular buffer mode, discarding oldest data when overflowed and will flush 12 bit data every time the watermark is reached. This will set the FIFO interrupt.

Configure the FIFO:

- Circular Buffer Mode in F_Setup Reg, F_MODE = 01
- Set the Watermark
- Set the FIFO Interrupt
- Route the FIFO Interrupt to INT1 or INT2
- Set the Interrupt Pins for Open Drain Active Low
- Set FDE bit
- Point the Data to the X_LSB bit

Code Example:

```
/*
** Read the contents of the CTRL_REG1 register.
*/
CTRL_REG1Data = IIC_RegRead(CTRL_REG1);
/*
** Write to CTRL_REG1 with FS = 00 to go into Standby Mode.
*/
IIC_RegWrite(CTRL_REG1, CTRL_REG1Data & ~FS_MASK);
/*
** Write to F_SETUP_REG and configure
**   - the FIFO for circular buffer operation
**   - the Watermark at the desired level
*/
IIC_RegWrite(F_SETUP_REG, F_MODE0_MASK + WATERMARK_VAL);
/*
** Enable the FIFO Interrupt and Set it to INT2
*/
IIC_RegWrite(CTRL_REG4, INT_EN_FIFO_MASK);
IIC_RegWrite(CTRL_REG5, ~INT_CFG_FIFO_MASK);
/*
** Configure the INT pins for Open Drain and Active Low
*/
IIC_RegWrite(CTRL_REG3, PP_OD_MASK);
/*
** Set FDE Bit*/
IIC_RegWrite(XYZ_DATA_CFG, FDE_MASK);
```

```
/*********************************************************\
* MMA8450Q Interrupt Service Routine for the FIFO
\*********************************************************/
interrupt void isr_MMA8450Q (void)
{
    /*
    ** Clear the MCU's interrupt flag
    */
    CLEAR_MMA8450Q_INTERRUPT;
    /*
    ** Go read the Interrupt Source Register
    */
    RegisterFlag.Byte = IIC_RegRead(INT_SOURCE_REG);
    if (RegisterFlag.SRC_FIFO_BIT == 1)
    {
        /*
        ** Read 12-bit XYZ results using a multi-read IIC access.
        */
        IIC_RegReadN(OUT_X_LSB_REG, WATERMARK_VAL*6, &value[0]);
        /*
        ** Copy and save each result as a 16-bit left-justified value.
        */
        x_value.Byte.lo = value[0] << 4;
        x_value.Byte.hi = value[1];
        y_value.Byte.lo = value[2] << 4;
        y_value.Byte.hi = value[3];
        z_value.Byte.lo = value[4] << 4;
        z_value.Byte.hi = value[5];
        /*
        ** Indicate that new data exists to be processed.
        */
        NEW_DATA = TRUE;
    }
}
```

10.0 MMA8450Q Driver Code

This application note is accompanied by a driver that has been written in CodeWarrior for a program that will run on the LFSTBEBMMA8450Q demo board system. This code must be loaded into the board via the BDM header on the interface board. An image of the demo board system is shown in Figure 2.

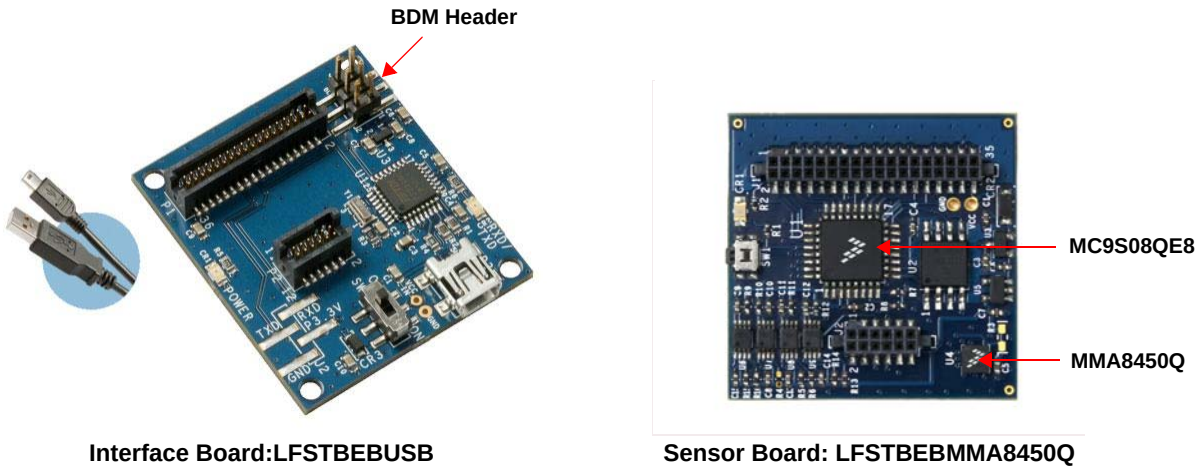

Figure 2. MMA8450Q Demo Board System

Once the code is loaded into the microcontroller via the BDM start a RealTerm, or HyperTerminal session. Find the port connected to the demo board. Set the Baud Rate to 115, 200. Once connected, type “?” in the terminal window. This will list out all the commands. The following are all the commands available:

Mn	: Mode 0 = Shutdown; 1 = Standby; 2 = 2g; 4 = 4g; 8 = 8g
RR xx	: Register xx Read
RW xx = nn	: Register xx Write value nn
RO n	: ODR Hz 0 = 400; 1 = 200; 2 = 100; 3 = 50; 4 = 12.5; 5 = 1.563
RH n	: High Pass Filter 0 - 3
RF	: Report ODR speed, HP Filter frequency and Mode
C	: Read 12-bit XYZ data as signed counts
G	: Read 12-bit XYZ data as signed g's
Sx	: Stream 12-bit XYZ data
Dx	: Stream 8-bit XYZ delta
Ix n	: Stream 12-bit XYZ via INT
Fx ww	: Stream 12-bit XYZ via FIFO
	: x : C = signed counts; G = signed g's
	: n : 1 = INT1; 2 = INT2
	: ww : Watermark = 1 to 31

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN3922
Rev. 2
03/2010