Freescale Semiconductor
Application Note

Document Number: AN4076
Rev 1, 02/2012

Data Manipulation and Basic Settings of the MMA8451, 2, 3Q

by: Kimberly Tuck
Applications Engineer

1.0 Introduction

It is important to understand how to program the MMA8451, 2, 3Q to extract and manipulate the acceleration data. These devices have been designed to be compatible with a shared memory map. For details on the differences of the embedded features please review our selector guide. The MMA8451Q has the most embedded features which include eight different sample rates, 32 different cutoff frequencies for the high-pass filter, Three dynamic ranges and four oversampling modes. 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 FIFO can collect the regular low-pass filtered data as well as the data from the high-pass filter. The manipulation of the data into different formats is important for algorithm development and for display. This application note accompanies the MMA8451, 2, 3Q Driver Code and will explain the following:

- Changing the operational modes (Standby, Active 2g, Active 4g and Active 8g)
- Changing Oversampling Modes
- Changing the Data Rate
- Changing the High-Pass Filter Cutoff Frequency
- 8-bit data vs. 14/12/10-bit data
- Changing Data Formats (hex to counts to decimal numbers)
- Streaming XYZ data polling vs. Streaming XYZ data with interrupts
- Using the FIFO in the MMA8451Q

1.1 Key Words

Accelerometer, Output Data Rate, Standby Mode, Active Mode, High-Pass Filter Cutoff Frequency, 8-bit Data, 14-bit Data, 12-bit Data, 10-bit Data Hexadecimal Numbers, Decimal Numbers, Data Formats, Streaming Data, Counts, Polling, Interrupts, FIFO Data, Flush, Sensor Toolbox Demo Board, Driver Code, High-Pass Filtered Data, Low-Pass Filtered Data

TABLE OF CONTENTS

1.2 Summary

A. There is a Standby Mode which responds to I^2C communication but doesn't allow for updated data. There are also three dynamic ranges: 2g, 4g and 8g which can be used to observe the change in sensitivity and full acceleration range while the device is active.

B. An example of how to set the data rate is shown. There are eight different data rates ranging from 1.56 Hz to 800 Hz.

C. An example of how to set the High-Pass Filter Cutoff Frequency is given. The high-pass filtered output data is affected by the filter cutoff frequency settings.

D. There are four different oversampling modes that can be set.

E. An example and the format conversions for manipulating 14/12/10/8-bit data converting 2's complement hex data to two different formats, which include formatting to signed integer (counts) and signed decimal fractions in g's for high-pass filtered data or low-pass filtered data.

F. An example of how to set up the device to poll the data or configure an interrupt service routine is shown.

G. An example of how to configure the FIFO to store and flush data.

H. There is a driver available that will run on the MMA8451, 2, 3Q 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 MMA845xQ Consumer 3-axis Accelerometer 3 by 3 by 1 mm

The MMA8451, 2, 3Q has a selectable dynamic range of ±2g, ±4g, ±8g. The device has 8 different output data rates, selectable high-pass filter cutoff frequencies, and in some cases high-pass filtered output data available. The resolution of the data and the embedded features is dependant on the specific device.

Note: The MMA8451, 2, 3Q has a different memory map than that of the MMA8450Q.

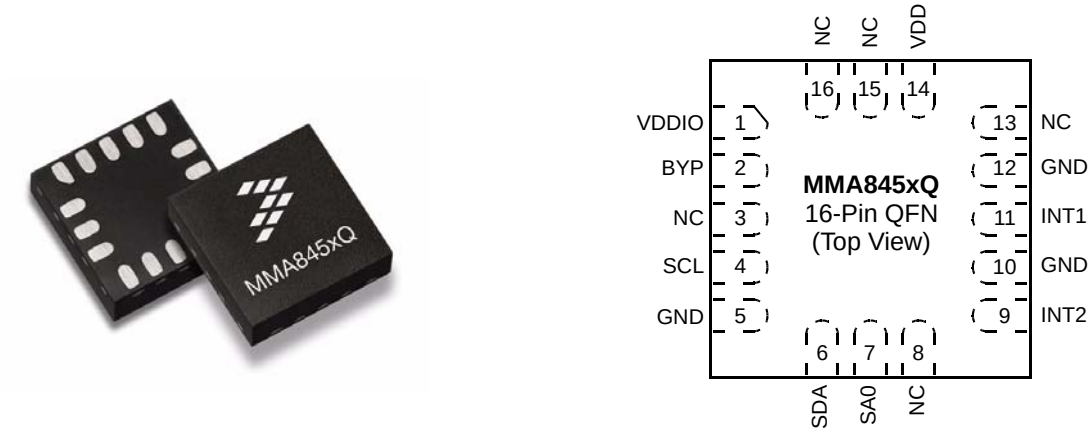

Figure 1. MMA8451, 2, 3Q Consumer 3-axis Accelerometer 3 mm by 3 mm by 1 mm

2.1 Output Data, Sample Rates and Dynamic Ranges of all Three Products

2.1.1 MMA8451Q

1. **14-bit data**
   2g (4096 counts/g = 0.25 mg/LSB) **4g** (2048 counts/g = 0.5 mg/LSB) **8g** (1024 counts/g = 1 mg/LSB)
2. **8-bit data**
   2g (64 counts/g = 15.6 mg/LSB) **4g** (32 counts/g = 31.25 mg/LSB) **8g** (16 counts/g = 62.5 mg/LSB)
3. **Embedded 32 sample FIFO (MMA8451Q)**

2.1.2 MMA8452Q

1. **12-bit data**
   2g (1024 counts/g = 1 mg/LSB) **4g** (512 counts/g = 2 mg /LSB) **8g** (256 counts/g = 3.9 mg/LSB)
2. **8-bit data**
   2g (64 counts/g = 15.6 mg/LSB) **4g** (32 counts/g = 31.25 mg/LSB) **8g** (16 counts/g = 62.5 mg/LSB)

2.1.3 MMA8453Q Note: No HPF Data

1. **10-bit data**
   2g (256 counts/g = 3.9 mg/LSB) **4g** (128 counts/g = 7.8 mg/LSB) **8g** (64 counts/g= 15.6 mg/LSB)
2. **8-bit data**
   2g (64 counts/g = 15.6 mg/LSB) **4g** (32 counts/g = 31.25 mg/LSB) **8g** (16 counts/g = 62.5 mg/LSB)

3.0 Changing Modes of the MMA8451, 2, 3Q

The device can be in either Standby Mode or Active Mode. Most of the settings are changed in Standby Mode but the data does not update and is not enabled until the device is in Active Mode. There are also 3 different dynamic ranges that can be set (2g, 4g, 8g). The dynamic range is changeable only in the Standby Mode. The dynamic range is controlled by setting the FS0 and FS1 bits in register 0x0E. The device changes from Standby to Active Mode via bit 0 in register 0x2A.

Table 1. 0x2A 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	DR2	DR1	DR0	LNOISE	FREAD	**ACTIVE**

Table 2. 0x0E XYZ_DATA_CFG Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	00	0	HPF_Out	0	0	**FS1**	**FS0**

Table 3. Full Scale Selection

FS1	FS0	g Range
0	0	±2g
0	1	±4g
1	0	±8g
1	1	—

3.1 Standby and Active Mode

Most, although not quite all changes to the registers must be done while the accelerometer is in Standby Mode. Current consumption in Standby Mode is typically 1 - 2 µA. To be in Standby Mode the last bit of **CTRL_REG1** must be cleared (**Active** = 0). When Active = 1 the device is in the active mode.

Code Example:

```
void MMA845x_Standby (void)
{
    byte n;
    /*
    **  Read current value of System Control 1 Register.
    **  Put sensor into Standby Mode by clearing the Active bit
    **  Return with previous value of System Control 1 Register.
    */
    n = IIC_RegRead(CTRL_REG1);
    IIC_RegWrite(CTRL_REG1, n & ~ACTIVE_MASK);
}

void MMA845x_Active ()
{
/*
** Set the Active bit in CTRL Reg 1
*/
    IIC_RegWrite(CTRL_REG1, (IIC_RegRead(CTRL_REG1) | ACTIVE_MASK));
}
```

3.2 2g Active Mode

In order to enter 2g Active Mode, the MMA8451, 2, 3Q must first be put into Standby Mode prior to changing the **FS** bits to 00 (as per Table 3).

Code Example:

```
    /*
    **Put the part in Standby Mode
    */
    MMA845x_Standby();
/*
**Write the 2g dynamic range value into register 0x0E
*/
IIC_RegWrite(XYZ_DATA_CFG_REG, (IIC_RegRead(XYZ_DATA_CFG_REG) & ~FS_MASK));
/*
**Put the part back into the Active Mode
*/
MMA845x_Active();
```

3.3 4g Active Mode

In order to enter 4g Active Mode, the MMA8451, 2, 3Q must first be put into Standby Mode prior to changing the **FS** bits 01 (as per Table 3).

Code Example:

```
    /*
    **Put the part in Standby Mode
    */
    MMA845x_Standby();
/*
**Write the 4g dynamic range value into register 0x0E
*/
IIC_RegWrite(XYZ_DATA_CFG_REG, (IIC_RegRead(XYZ_DATA_CFG_REG) & ~FS_MASK));
IIC_RegWrite(XYZ_DATA_CFG_REG, (IIC_RegRead(XYZ_DATA_CFG_REG) | FULL_SCALE_4G));
/*
**Put the part back into the Active Mode
*/
MMA845x_Active();
```

3.4 8g Active Mode

In order to enter 8g Active Mode, the MMA8451, 2, 3Q must first be put into Standby Mode prior to changing the **FS** bits to 10 (as per Table 3).

Code Example:

```
    /*
    **Put the part in Standby Mode
    */
    MMA845x_Standby();
/*
**Write the 8g dynamic range value into register 0x0E
*/
IIC_RegWrite(XYZ_DATA_CFG_REG, (IIC_RegRead(XYZ_DATA_CFG_REG) & ~FS_MASK));
IIC_RegWrite(XYZ_DATA_CFG_REG, (IIC_RegRead(XYZ_DATA_CFG_REG) | FULL_SCALE_8G));
/*
**Put the part back into the Active Mode
*/
MMA845x_Active();
```

4.0 Setting the Data Rate

The active mode Output Data Rate (ODR) and Sleep Mode Data Rate are programmable via other control bits in the CTRL_REG1 register, seen in Table 4. Unless the sleep mode is enabled the active mode data rate is the data rate that will always be enabled. Table 5 shows how the DR2:DR0 bits affect the ODR. These are the active mode data rates available. The default data rate is DR = 000, 800 Hz.

Table 4. 0x2A 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	**DR2**	**DR1**	**DR0**	LNOISE	FREAD	ACTIVE

Table 5. Output Data Rates

DR2	DR1	DR0	Output Data Rate (ODR)	Time Between Data Samples
0	0	0	800 Hz	1.25 ms
0	0	1	400 Hz	2.5 ms
0	1	0	200 Hz	5 ms
0	1	1	100 Hz	10 ms
1	0	0	50 Hz	20 ms
1	0	1	12.5 Hz	80 ms
1	1	0	6.25 Hz	160 ms
1	1	1	1.563 Hz	640 ms

Code Example:

```
/*
**  Adjust the desired Output Data Rate value as needed.
*/
DataRateValue <<= 3;
/*
**  Put the device into Standby Mode
*/
MMA845x_Standby();
/*
**  Write in the Data Rate value into Ctrl Reg 1
*/
IIC_RegWrite(CTRL_REG1,IIC_RegRead(CTRL_REG1) & ~DR_MASK);
IIC_RegWrite(CTRL_REG1, IIC_RegRead(CTRL_REG1)| DataRateValue);
/*
**  Put the device into the active mode again.
*/
MMA845x_Active();
```

5.0 Setting the Oversampling Mode

There are four different oversampling modes. There is a normal mode, a low noise + power mode, a high-resolution mode and a low-power mode. The difference between these are the amount of averaging of the sampled data, which is done internal to the device. The following chart shows the amount of averaging at each data rate, which is the OSRatio (oversampling ratio). There is a trade-off between the oversampling and the current consumption at each ODR value.

Table 6. Oversampling Modes with Current and OS Ratio per ODR

Mode	Normal		Low Noise Low Power		High Resolution		Low Power	
ODR	Current μA	OS Ratio	Current μA	OS Ratio	Current μA	OS Ratio	Current μA	OS Ratio
1.5625	24	128	8	32	165	1024	6	16
6.25	24	32	8	8	165	256	6	4
12.5	24	16	8	4	165	128	6	2
50	24	4	24	4	165	32	14	2
100	44	4	44	4	165	16	24	2
200	85	4	85	4	165	8	44	2
400	165	4	165	4	165	4	85	2
800	165	2	165	2	165	2	165	2

The following are code examples of how to change to different oversampling modes.

Code Example:

```
/** Oversampling Mode: Normal MODS=00
/*
    **  Put the device into Standby Mode
*/
MMA845x_Standby();
/*
    **  Clear the Mode bits in CTRL_REG2
*/

IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (IIC_RegRead(SlaveAddressIIC, CTRL_REG2) &
~MODS_MASK));

/*
    **  Put the device into Active Mode
*/

MMA845x_Active();



/*
** Oversampling Mode: Low Noise Low Power MODS = 01
*/
/*
    **  Put the device into Standby Mode
*/
MMA845x_Standby();
/*
    **  Clear the Mode bits in CTRL_REG2
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (IIC_RegRead(SlaveAddressIIC, CTRL_REG2) &
~MODS_MASK));
/*
    **  Set the MODS bits to 01 for Low Noise Low Power Mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (IIC_RegRead(SlaveAddressIIC, CTRL_REG2) |
MODS0_MASK));
```

```
/*
    **  Put the device into Active Mode
*/
MMA845x_Active();

/*
** Oversampling Mode: HI RESOLUTION MODS =10
*/
/*
    **  Put the device into Standby Mode
*/
MMA845x_Standby();
/*
    **  Clear the Mode bits in CTRL_REG2
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (IIC_RegRead(SlaveAddressIIC, CTRL_REG2) &
~MODS_MASK));
/*
    **  Set the MODS bits to 10 for Hi Resolution Mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (IIC_RegRead(SlaveAddressIIC, CTRL_REG2) |
MODS1_MASK));
/*
    **  Put the device into Active Mode
*/
MMA845x_Active();


/*
** Oversampling Mode: LOW POWER MODS = 11
*/
/*
**  Put the device into Standby Mode
*/
MMA845x_Standby();
/*
    **  Clear the Mode bits in CTRL_REG2
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (IIC_RegRead(SlaveAddressIIC, CTRL_REG2) &
~MODS_MASK));
/*
    **  Set the MODS bits to 11 for Low Power Mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (IIC_RegRead(SlaveAddressIIC, CTRL_REG2) |
MODS_MASK));
/*
    **  Put the device into Active Mode
*/
MMA845x_Active();
```

6.0 Setting the High-Pass Filter Cutoff Frequency

The HP_FILTER_CUTOFF register (at 0x0F) sets the high-pass cutoff frequency, Fc, for the data. The output of this filter is provided in the output data registers (0x01 to 0x06). Note that the high-pass filtered output data is available for the MMA8451Q and the MMA8452Q only. The MMA8453Q has the internal high-pass filter for the embedded functions but does not have access to the output data. The available cutoff frequencies change depending upon the set Output Data Rate.

Table 7. 0x0F 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	Pulse_HPF_Byp	Pulse_LPF_EN	0	0	**SEL1**	**SEL0**

Table 8 presents the different cutoff frequencies for the high-pass filter based on the different set data rates. Note that the cutoff frequencies change based on output data rate and also the oversampling mode.

Table 8. HP_FILTER_CUTOFF Setting Options

Oversampling Mode = Normal									
SEL1	SEL0	800 Hz	400 Hz	200 Hz	100 Hz	50 Hz	12.5 Hz	6.25 Hz	1.56 Hz
0	0	16 Hz	16 Hz	8 Hz	4 Hz	2 Hz	2 Hz	2 Hz	2 Hz
0	1	8 Hz	8 Hz	4 Hz	2 Hz	1 Hz	1 Hz	1 Hz	1 Hz
1	0	4 Hz	4 Hz	2 Hz	1 Hz	0.5 Hz	0.5 Hz	0.5 Hz	0.5 Hz
1	1	2 Hz	2 Hz	1 Hz	0.5 Hz	0.25 Hz	0.25 Hz	0.25 Hz	0.25 Hz
Oversampling Mode = Low Noise Low Power									
0	0	16 Hz	16 Hz	8 Hz	4 Hz	2 Hz	0.5 Hz	0.5 Hz	0.5 Hz
0	1	8 Hz	8 Hz	4 Hz	2 Hz	1 Hz	0.25 Hz	0.25 Hz	0.25 Hz
1	0	4 Hz	4 Hz	2 Hz	1 Hz	0.5 Hz	0.125 Hz	0.125 Hz	0.125 Hz
1	1	2 Hz	2 Hz	1 Hz	0.5 Hz	0.25 Hz	0.063 Hz	0.063 Hz	0.063 Hz
Oversampling Mode = High Resolution									
0	0	16 Hz	16 Hz	16 Hz	16 Hz	16 Hz	16 Hz	16 Hz	16 Hz
0	1	8 Hz	8 Hz	8 Hz	8 Hz	8 Hz	8 Hz	8 Hz	8 Hz
1	0	4 Hz	4Hz	4 Hz	4 Hz	4 Hz	4 Hz	4 Hz	4 Hz
1	1	2 Hz	2Hz	2 Hz	2 Hz	2 Hz	2 Hz	2 Hz	2 Hz
Oversampling Mode = Low Power									
0	0	16 Hz	8 Hz	4 Hz	2 Hz	1 Hz	0.25 Hz	0.25 Hz	0.25 Hz
0	1	8 Hz	4 Hz	2 Hz	1 Hz	0.5 Hz	0.125 Hz	0.125 Hz	0.125 Hz
1	0	4 Hz	2 Hz	1 Hz	0.5 Hz	0.25 Hz	0.063 Hz	0.063 Hz	0.063 Hz
1	1	2 Hz	1 Hz	0.5 Hz	0.25 Hz	0.125 Hz	0.031 Hz	0.031 Hz	0.031 Hz

To set the cutoff frequency, a value from 0x00 to 0x03 must be chosen for the **SEL** bits, as per Table 8. In order to make this change, the sensor must be in Standby Mode prior to writing to the HP_FILTER_CUTOFF register.

Consider an example where the device is operating in 2g Active Mode with a 400 Hz ODR and the default Fc of 16 Hz. The following code demonstrates how to change Fc to 4 Hz without making any other operational modifications. This is done by changing the SEL bits to 10, (as per Table 8).

Code Example:

```
/*
**  Select the desired cutoff frequency.
*/
CutOffValue=2;
/*
** Put the device in Standby Mode
*/
MMA845x_Standby();
/*
** Write in the cutoff value
*/
IIC_RegWrite(HP_FILTER_CUTOFF_REG,(IIC_RegRead(HP_FILTER_CUTOFF_REG)& ~SEL_MASK);
```

```
IIC_RegWrite(HP_FILTER_CUTOFF_REG,(IIC_RegRead(HP_FILTER_CUTOFF_REG)| CutOffValue);
/*
 ** Put the device back in the active mode
 */
MMA845x_Active();
```

6.1 High-Pass Filtered Data or Low-Pass Filtered Data

Registers 0x01 through 0x06 are used to read the X, Y, Z data. The device can be configured to produce high-pass filtered data or low-pass filtered data by setting or clearing the HPF_Out bit in the XYZ_Data_Cfg Register 0x0E. The following code example shows how to set the HPF_Out bit.

Code Example:

```
/*
** Put the device in Standby Mode
*/
MMA845x_Standby();
/*
** Set the HPF_OUT Bit to enable the HPF Data Out
*/
IIC_RegWrite(XYZ_DATA_CFG_REG, (IIC_RegRead(XYZ_DATA_CFG_REG) | HPF_OUT_MASK));
MMA845x_Active();
```

7.0 14-bit, 12-bit or 10-bit Data Streaming and Data Conversions

The MMA8451Q has 14-bit XYZ data. The MMA8452Q has 12-bit XYZ data and the MMA8453 has 10-bit data. This section is an overview of how to manipulate the data to continuously burst out 14-bit data in different data formats from the MCU. The examples will be shown for the 14-bit data but the reader can understand what changes would be made for the 12-bit data or the 10-bit data. The driver code has all the functions for all data formats available.

The event flag can be monitored by reading the STATUS register (0x00). This can be done by using either a polling or interrupt technique, which is discussed later in Section 9.0 of this document. It is not absolutely necessary to read the STATUS register to clear it. Reading the data clears the STATUS register.

Table 9. 0x00 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 14/12/10-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_00_REG);

if (RegisterFlag.ZYXDR_BIT == 1)
{
    /*
    **  Read 14/12/10-bit XYZ results using a 6 byte IIC access.
    */
    IIC_RegReadN(OUT_X_MSB_REG, 6, &value[0]);
    /*
    **  Copy and save each result as a 16-bit left-justified value.
    */
    x_value.Byte.hi = value[0];
    x_value.Byte.lo = value[1];
    y_value.Byte.hi = value[2];
    y_value.Byte.lo = value[3];
    z_value.Byte.hi = value[4];
    z_value.Byte.lo = value[5];
}
```

The corresponding 16-bit results in left-justified format (2's complement numbers) are provided as an example of the register and variable formats for the X-axis result below:

Table 10. 0x00 x_value.Byte.hi: X_MSB Register MMA8451Q (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD13	XD12	XD11	XD10	XD9	XD8	XD7	XD6

Table 11. 0x01 x_value.Byte.lo: X_LSB Register MMA8451Q (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD5	XD4	XD3	XD2	XD1	XD0	0	0

Table 12. 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
XD13	XD12	XD11	XD10	XD9	XD8	XD7	XD6	XD5	XD4	XD3	XD2	XD1	XD0	0	0

7.1 Converting 14-bit 2's Complement Hex Number to 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: 0xABCC = -1349
0x5443 = +1349

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 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(). 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_s14dec_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 >>2) / 1000);
   r = (data.Word >>2) % 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);
}
```

7.2 Converting 14-bit 2's Complement Hex Number to Signed Decimal Fraction in g's

Converting to a signed value into g's requires performing the same operations as shown above 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 13, represents the sign of the result (either positive or negative).

- In 2g Active Mode 1g = 4096 counts. Therefore Bit 12 is the only bit that will contribute to an integer value of either 0, 1. 2^{12} = 4096. Bits 11 through 0 will be fractional values.
- In 4g Active Mode 1g = 2048 counts. Therefore Bits 12 and 11 will contribute to an integer value of 0, 1, 2, or 3. Bits 10 through 0 will be fractional values.
- In 8g Active Mode 1g = 1024 counts. Therefore Bits 12, 11 and 10 will contribute to an integer value of 0, 1, 2, 3, 4, 5, 6, and 7. Bits 9 through 0 will be fractional values.

Table 13. Full Scale Value with Corresponding Integer Bits and Fraction Bits

Full Scale Value	Counts/g	Sign Bit	Integer Bits	Fraction Bits
2g	4096	13	12 (2^{12} = 4096)	0 through 11
4g	2048	13	12 (2^{12} = 4096), 11 (2^{11} = 2048)	0 through 10
8g	1024	13	12 (2^{12} = 4096), 11 (2^{11} = 2048), 10 (2^{10} = 1024)	0 through 9

7.2.1 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 word format bits 13 and 14, as can be seen in Table 14. The row labeled as "MMA8451Q 14b" shows where the 14-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") from the word format.

Table 14. 2g Active Mode 14-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
MMA8451Q 14b	13	12	11	10	9	8	7	6	5	4	3	2	1	0	x	x
Integer/Fraction	±	**I**	F	F	F	F	F	F	F	F	F	F	F	F	x	x
MSB/LSB	M	**M**	M	M	M	M	M	M	L	L	L	L	L	L	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 15.

Table 15. 2g Active Mode 14-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
MMA8451Q 14b	11	10	9	8	7	6	5	4	3	2	1	0	x	x	x	x
Integer/Fraction	F	F	F	F	F	F	F	F	F	F	F	F	x	x	x	x
Fraction Bits	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	x	x	x	x
MSB/LSB	M	M	M	M	M	M	L	L	L	L	L	L	0	0	0	0

This leaves six MSB bits and six LSB bits after shifting left by two. Therefore there are 12-bits for the fraction portion in 2g Active Mode. The 2g Active Mode has the highest number of bits for the fraction portion with 12-bits because it has the highest sensitivity. In Table 16 the decimal value is rounded to the fourth decimal place because the final fraction number will have four significant digits. This should be sufficient.

The values shown in Table 16 are translated here into C macros for use in the code example at the end of this section:

Table 16. 2g Active Mode Fraction Values

	2g Mode	Calculation	Rounded to 4th Decimal Place	Integer Number
2^{-1}	$2^{11} = 2048$	2048/4096 = 0.5	0.5000	5000
2^{-2}	$2^{10} = 1024$	1024/4096 = 0.25	0.2500	2500
2^{-3}	$2^{9} = 512$	512/4096 = 0.125	0.1250	1250
2^{-4}	$2^{8} = 256$	256/4096 = 0.0625	0.0625	625
2^{-5}	$2^{7} = 128$	128/4096 = 0.03125	0.0313	313
2^{-6}	$2^{6} = 64$	64/4096 = 0.015625	0.0156	156
2^{-7}	$2^{5} = 32$	32/4096 = 0.0078125	0.0078	78
2^{-8}	$2^{4} = 16$	16/4096 = 0.00390625	0.0039	39
2^{-9}	$2^{3} = 8$	8/4096 = 0.001953125	0.0020	20
2^{-10}	$2^{2} = 4$	4/4096 = 0.0009765652	0.0010	10
2^{-11}	$2^{1} = 2$	2/4096 = 0.000488281	0.0005	5
2^{-12}	$2^{0} = 1$	1/4096 = 0.00024414	0.0002	2

```
#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
#define FRAC_2d11            5
#define FRAC_2d12            2
```

For each of the 12 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 (625 + 156 + 39 = 820) which corresponds to 0.0820. The highest fractional value occurs when all fraction bits are set (5000 + 2500 + 1250 + 625 + 313 + 156 + 78 + 39 + 20 + 10 + 5 + 2 = 9998) which corresponds to 0.9998. In 2g Active Mode the resolution is 0.244 mg (1/4096). Calculating out the fraction to four significant digits gives a resolution of 0.2 mg for 2g Active Mode.

7.2.2 4g Active Mode

In 4g Active Mode there are 2 integer bits and 11 fraction bits as shown in Table 17.

Table 17. 4g Active Mode 14-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
MMA8451Q 14b	13	12	11	10	9	8	7	6	5	4	3	2	1	0	x	x
Integer/Fraction	±	I	I	F	F	F	F	F	F	F	F	F	F	F	x	x
MSB/LSB	M	**M**	**M**	M	M	M	M	M	L	L	L	L	L	L	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 18. Table 18 shows the 11 bits of the fraction with the corresponding decimal values for each bit identified in Table 19.

Table 18. 4g Active Mode 14-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
MMA8451Q 14b	10	9	8	7	6	5	4	3	2	1	0	x	x	x	x	x
Integer/Fraction	F	F	F	F	F	F	F	F	F	F	F	x	x	x	x	x
Fraction Bits	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	x	x	x	x	x
MSB/LSB	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0

Table 19. 4g Active Mode Fraction Values

	4g Mode	**Calculation**	**Rounded to 4th Decimal Place**	**Integer Number**
2^{-1}	$2^{10} = 1024$	1024/2048 = 0.5	0.5000	5000
2^{-2}	$2^{9} = 512$	512/2048 = 0.25	0.2500	2500
2^{-3}	$2^{8} = 256$	256/2048 = 0.125	0.1250	1250
2^{-4}	$2^{7} = 128$	128/2048 = 0.0625	0.0625	625
2^{-5}	$2^{6} = 64$	64/2048 = 0.03125	0.0313	313
2^{-6}	$2^{5} = 32$	32/2048 = 0.015625	0.0156	156
2^{-7}	$2^{4} = 16$	16/2048 = 0.0078125	0.0078	78
2^{-8}	$2^{3} = 8$	8/2048 = 0.00390625	0.0039	39
2^{-9}	$2^{2} = 4$	4/2048 = 0.00195312	0.0020	20
2^{-10}	$2^{1} = 2$	2/2048 = 0.00097656	0.0010	10
2^{-11}	$2^{0} = 1$	1/2048 = 0.00048828	0.0005	5

For each of the 11 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 (1250 + 313 + 78 = 1641) which corresponds to 0.1641. The highest fractional value occurs when all fraction bits are set (5000 + 2500 + 1250 + 625 + 313 + 156 + 78 + 39 + 20 + 10+5 = 9996) which corresponds to 0.9996. The resolution in 4g Active Mode is 0.488 mg. Calculating the fractional value to four significant digits results in a resolution of 0.5 mg.

7.2.3 8g Active Mode

In 8g Active Mode there are three integer bits, leaving ten bits for the fraction as per Table 20.

Table 20. 8g Active Mode 14-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
MMA8451Q 14b	13	12	11	10	9	8	7	6	5	4	3	2	1	0	x	x
Integer/Fraction	±	I	I	I	F	F	F	F	F	F	F	F	F	F	x	x
MSB/LSB	M	M	M	M	M	M	M	M	L	L	L	L	L	L	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 21 with Table 22 providing the corresponding decimal values.

Table 21. 8g Active Mode 14-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
MMA8451Q 14b	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	L	L	L	L	L	L	0	0	0	0	0	0

Table 22. 8g Active Mode Fraction Values

	8g Mode	Calculation 256 counts/g	Rounded to 4th Decimal Place	Integer Number
2^{-1}	2^9 = 512	512/1024 = 0.5	0.5000	5000
2^{-2}	2^8 = 256	258/1024 = 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 = 0.01563	0.0156	156
2^{-7}	2^3 = 8	8/1024 = 0.007812	0.0078	78
2^{-8}	2^2 = 4	4/1024 = 0.003906	0.0039	39
2^{-9}	2^1 =2	2/1024 = 0.001953	0.0020	20
2^{-10}	2^0 =1	1/1024 = 0.000976	0.0010	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 bit 9, 6 and 4 are set then the total will be (5000 + 625 + 156 = 5781) which corresponds to 0.5781. 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. The resolution in 8g Active Mode is 0.977 mg. Calculating the fractional value to four significant digits results in 1 mg resolution.

Below is the code example which performs the conversion of a 14-bit signed 2-s complement value into a signed decimal fraction displayed in g's. This routine can be used to also convert 12-bit, 10-bit or 8-bit data. The extra unused bits will remain zeros.

Code Example:

```
void SCI_s14frac_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 &= 0xFFFC;
      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 mantissa 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 (value.Bit._3 ==1)
       result += FRAC_2d9;
    if (value.Bit._2 ==1)
       result += FRAC_2d10;
    //
    if (full_scale != FULL_SCALE_8G)
    {
       if (value.Bit._1 == 1) {
          result += FRAC_2d11;
       }
       if (full_scale == FULL_SCALE_2G)
       {
          if (value.Bit._0 == 1)
             result += FRAC_2d12;
    }
}
```

8.0 8-bit XYZ Data Streaming and Conversions

The MMA8451, 2, 3Q can provide 8-bit XYZ high-pass filtered data or low-pass filtered data. The F_READ bit in Register 0x2A must be set read out 8-bit data.

As was shown with the 14-bit data, the ZYXDR flag in the STATUS register should be monitored.

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_MSB_REG, 3, &value[0]);
    /*
    **  Copy and save each result as a 16-bit left-justified value.
    */
    x_value.Byte.hi = value[0];
    x_value.Byte.lo = 0;
    y_value.Byte.hi = value[1];
    y_value.Byte.lo = 0;
    z_value.Byte.hi = value[2];
    z_value.Byte.lo = 0;
}
```

The 8-bit values can be converted to the various formats described previously for the 14-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 14-bit X-axis sample, the result of this procedure is shown below.

Table 23. 0x01 OUT_X_MSB_REG: X_MSB Register (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD13	XD12	XD11	XD10	XD9	XD8	XD7	XD6

Table 24. 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
XD13	XD12	XD11	XD10	XD9	XD8	XD7	XD6	0	0	0	0	0	0	0	0

Refer to Section 7.0 for further details regarding applicable data conversion formats. Specific 8-bit versions of the previous routines are provided in the following sections. Note for the MMA8451Q the MSB values are XD13 to XD6. For the MMA8452Q the MSB values are labeled XD11 through XD4. For the MMA8453Q, the MSB values are labeled XD9 through XD2.

8.1 Converting 8-bit 2's Complement Hex Number to 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 three 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 7.1.

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);
}
```

8.2 Converting 8-bit 2's Complement Hex Number to 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 14-bit data in Section 7.2. Therefore, only the details regarding the use of 2g Active Mode are described here, in Section 8.2.1.

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.

Table 25. 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

8.2.1 2g Active Mode

The subroutine provided in Section 2.0 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 26.

Table 26. 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
MMA8451, 2, 3Q 12b	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 27.

Table 27. 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
MMA8451, 2, 3Q 12b	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 28. These values are rounded to the fourth decimal place because the final fraction number will have four significant digits.

Table 28. 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 four significant digits gives a resolution of 15.6 mg for 2g Active Mode, which is more than enough resolution.

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.

9.0 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.

9.1 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 MMA8451, 2, 3Q, required to continuously poll 14-bit, 12-bit or 10-bit XYZ data.

Code Example:

```
/*
** Go to the Standby Mode
*/
MMA845xQ_Standby();
/*
** Clear the F_Read bit to ensure both MSB's and LSB's are indexed
*/
IIC_RegWrite(CTRL_REG1,(IIC_RegRead(CTRL_REG1)& ~FREAD_MASK);
/*
** Go back to Active Mode
*/
MMA845xQ_Active();
/*
** 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_00_REG);

   if (RegisterFlag.ZYXDR_BIT == 1)
   {
      /*
      **  Read 14/12/10-bit XYZ results using a 6 byte IIC access.
      */
      IIC_RegReadN(OUT_X_MSB_REG, 6, &value[0]);
      /*
      **  Copy and save each result as a 16-bit left-justified value.
      */
      x_value.Byte.hi = value[0];
      x_value.Byte.lo = value[1];
      y_value.Byte.hi = value[2];
      y_value.Byte.lo = value[3];
      z_value.Byte.hi = value[4];
      z_value.Byte.lo = value[5];
      /*
      **  Go process the XYZ data.
      */
      GoProcessXYZ(&value[0]);
   }
   /*
   **  Perform other necessary operations.
   */
   etc();
}
```

9.2 Interrupt Routine to Access Data

Streaming data via hardware interrupts is more efficient than polling as the MCU only interfaces with the accelerometer 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 this will be indicated by the overwrite register flags. The following are the register settings to configure the device to generate an interrupt upon each new data ready. The MCU's Interrupt Service Routine (ISR) shown below responds by reading the 14-bit, 12-bit or 10-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 data can also be performed in a similar fashion.

Code Example:

```
/*
** Go to the Standby Mode
*/
MMA845xQ_Standby();
/*
** Clear the F_Read bit to ensure both MSB's and LSB's are indexed
*/
IIC_RegWrite(CTRL_REG1,(IIC_RegRead(CTRL_REG1)& ~FREAD_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.
*/
IIC_RegWrite(CTRL_REG4, INT_EN_DRDY_MASK);
IIC_RegWrite(CTRL_REG5, INT_CFG_DRDY_MASK);
/*
** Go back to Active Mode
*/
MMA845xQ_Active();
/*
**  etc.
*/

/*******************************************************\
* MMA8451,2Q Interrupt Service Routine
\*******************************************************/
   interrupt void isr_MMA8451Q (void)
   {
      /*
      **  Clear the MCU's interrupt flag
      */
      CLEAR_MMA8451Q_INTERRUPT;
      /*
      **  Go read the Interrupt Source Register
      */
      RegisterFlag.Byte = IIC_RegRead(INT_SOURCE_REG);
      if (RegisterFlag.SRC_DRDY_BIT == 1)
   {
      /*
      **  Read 14 12 or 10-bit XYZ results using a 6 byte IIC access.
      */
      IIC_RegReadN(OUT_X_MSB_REG, 6, &value[0]);
      /*
      **  Copy and save each result as a 16-bit left-justified value.
      */
      x_value.Byte.hi = value[0];
      x_value.Byte.lo = value[1];
      y_value.Byte.hi = value[2];
      y_value.Byte.lo = value[3];
```

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

10.0 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 AN4073. The FIFO can be configured in circular buffer mode, discarding oldest data when overflowed and will flush the data every time the watermark is reached. This will set the FIFO interrupt.

Configure the FIFO:

- Circular Buffer Mode 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

Example Code:

```
   /*
   **  Go to Standby Mode
   */
   MMA845xQ_Standby();
   /*
   ** Set F_Mode to Circular, Set the Watermark Value into the F_SETUP register
   */
   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);
   /*
   **  Go to Active Mode
   */
   MMA845xQ_Active();

/*********************************************************\
* MMA8451Q Interrupt Service Routine for the FIFO
\*********************************************************/
interrupt void isr_MMA8451Q (void)
{
   /*
   **  Clear the MCU's interrupt flag
   */
   CLEAR_MMA8451Q_INTERRUPT;
   /*
   **  Go read the Interrupt Source Register
   */
   RegisterFlag.Byte = IIC_RegRead(INT_SOURCE_REG);
```

```
    if (RegisterFlag.SRC_FIFO_BIT == 1)
    {
       /*
       **  Read 14-bit XYZ results using a multi-read IIC access.
       */
       IIC_RegReadN(OUT_X_MSB_REG, WATERMARK_VAL*6, &value[0]);
       /*
       **  Copy and save each result as a 16-bit left-justified value.
       */
       x_value.Byte.hi = value[0];
       x_value.Byte.lo = value[1];
       y_value.Byte.hi = value[2];
       y_value.Byte.lo = value[3];
       z_value.Byte.hi = value[4];
       z_value.Byte.lo = value[5];
       /*
       **  Indicate that new data exists to be processed.
       */
       NEW_DATA = TRUE;
    }
}
```

Related Documentation

The MMA845xQ device features and operations are described in a variety of reference manuals, user guides, and application notes. To find the most-current versions of these documents:

1. Go to the Freescale homepage at:

   http://www.freescale.com/

2. In the Keyword search box at the top of the page, enter the device number MMA845xQ.
3. In the Refine Your Result pane on the left, click on the Documentation link.

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:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN4076
Rev. 1
02/2012