Using the 32 Sample First In First Out (FIFO) in the MMA8450Q

by: Kimberly Tuck
Applications Engineer

1.0 Introduction

The MMA8450Q has a built in 32 sample first in, first out buffer capable of storing either 12-bit data or 8-bit data. The FIFO is very beneficial for saving overall system power by putting the processor into sleep mode until it needs to process data from the accelerometer. The idea is to configure the MMA8450Q to monitor a desired interrupt, putting the processor in a low power mode until it needs to respond to the accelerometer. This minimizes the system's overall power consumption, increasing the life of the battery. The embedded FIFO is a proven benefit as it limits how often the processor needs to read the data. The FIFO allows the processor to sleep longer while samples are being collected inside the sensor.

Higher sample rate data can be captured in the FIFO and accessed at a reasonable update time without increasing computational throughput by accessing every sample individually.

1.1 Key Words

Accelerometer, Output Data Rate, 32 Sample FIFO, 12-bit Data, 8-bit Data, I^2C Bus, Flushing, Algorithm Development, Circular Buffer Mode, Fill Buffer Mode, Watermark, Overflow, Sensor, Power Savings, Multi-read, Processor, MCU

TABLE OF CONTENTS

1.2 Summary

A. The embedded FIFO is highly beneficial for system power savings and minimizing traffic across the I^2C bus.

B. The FIFO can be flushed at all sample rates. At 400 Hz reading out 12-bit data the FIFO must be flushed more often to ensure no data is missed.

C. The FIFO allows for remarkable power savings of the system by allowing the host processor/MCU to go into a sleep mode while the accelerometer independently stores the data, up to 32 samples.

D. The FIFO can be configured to be in a circular buffer mode or a fill buffer mode, which is application dependent.

E. The FIFO is very useful for further enhancing embedded algorithms to extract more details from what caused the interrupt, with the ability to read previous history.

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.

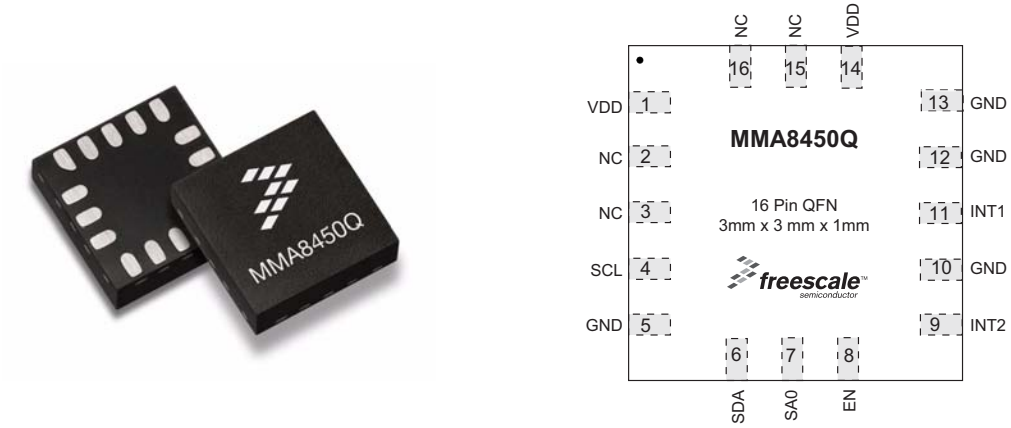

Figure 1. MMA8450Q Consumer 3-axis Accelerometer 3 x 3 x 1 mm

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 Applications Using the FIFO

The FIFO is used typically for the following functions:

- Data logging- flushing once every 32 samples for power savings and to relieve bus contention
- Storing the previous history leading up to an event, for power savings and relives bus contention
- The FIFO can also be programmed to stop on an event trigger if the device was in sleep mode and the sample rate changes when the event occurs

3.1 Power Savings Using the FIFO Data Logging

The FIFO is very beneficial for saving overall system power by putting the processor into sleep mode until it needs to process data from the accelerometer. The idea is to configure the MMA8450Q to monitor a desired interrupt, putting the processor in a low power mode until it needs to respond to the accelerometer. This maximizes the time that the processor spends in a sleep or low power mode and ultimately will minimize the system's overall power consumption, increasing the life of the battery. The FIFO allows the processor to sleep longer while samples are being collected inside the sensor. This also minimizes the traffic across the I^2C bus.

The timing of the data rate and the bus speed should be chosen with care. As an example the accelerometer is put in Low Power Mode sampling at 50 Hz (20 ms) with the FIFO running in Fill Mode and the FIFO interrupt enabled. The interrupt would be used to trigger the processor to wake up, service the interrupt, and flush the 32 samples. New data cannot be stored into the FIFO while it is being flushed. Therefore the processor must wake up, service the interrupt and flush the data within 20 ms before the next sample is available.

The FIFO overflow is asserted every 32 samples. The user has the option of flushing either the 12-bit data or the 8-bit data. For the 12-bit data each sample consists of three 12-bit values, each stored as 2 bytes. Therefore, when full, the FIFO will contain 192 bytes. For the 8-bit data each sample consists of three 8-bit values, each stored as 1 byte. Therefore in this case when full the FIFO will contain 96 bytes. An I^2C burst access has about 3 extra bytes of "overhead", for a total of 195 bytes in the 12-bit data flush and 99 bytes in total for the 8-bit data flush. Also the Start, Stop and Repeat Start I^2C transactions take a minimum of 0.6 μs. A 1 μs value will be added for each of these. Assuming that each I^2C byte requires a 10-bit transfer window (8 for data, 1 for the acknowledge and 1 for bus idle), the time required to perform an I^2C burst read of N samples can be calculated as follows:

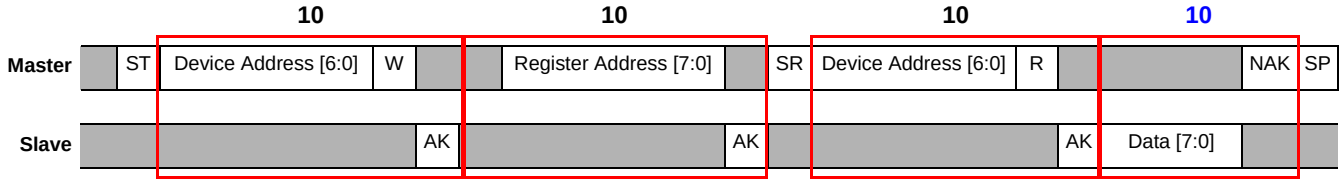

Figure 2. I^2C Single Byte Read Transaction

12- bit Data Flush Calculations

FIFORead(N) = ((((N · 3 · 2) + 3) · 10) / I^2C bit rate)

FIFORead(32) = 1950 / I^2C bit rate

For an I^2C bit rate of 400 kHz, FIFORead(32) + 3 μs = 4.878 ms.

For an I^2C bit rate of 400 kHz, FIFORead(16) + 3 μs = 2.478 ms.

8-bit Data Flush Calculations

FIFORead(N) = ((((N · 3) + 3) · 10) / I^2C bit rate)

FIFORead(32) = 990 / I^2C bit rate

For an I^2C bit rate of 400 kHz, FIFORead(32) + 3 μs = 2.478 ms.

Note that bursting out 32 samples of 12-bit XYZ data consecutively takes 1950 bits to perform the transaction. By bursting XYZ 12-bit data each time new data is ready requires 2880 bits, as this requires 32 iterations. The start, stop and repeat start transactions calculated at 1 μs each start to add up over 32 iterations.

$$\text{DataReadyRead(32)} = (((3 \cdot 2) + 3) \cdot 10) \cdot 32 = 2880 \text{ bits}/I^2C \text{ bit rate}$$

$$\text{For an } I^2C \text{ bit rate of 400 kHz, DataReadyRead(32)} = 7.2 \text{ ms} + 3\ \mu s \cdot 32 = 7.296 \text{ ms}$$

It is seen that using the FIFO to pull out all 32 samples at one time saves on the overhead. This allows the application processor to do other things or to remain in a low power mode for longer.

Example conditions are given for a processor with the wake timing and current consumption values in Table 1. In Wake Mode the example processor uses a total of 12 mA, while in sleep mode it only uses 0.5 mA. It take 3 ms to wake the processor from sleep and read the FIFO status. As shown above, a 12-bit data flush from the accelerometer FIFO takes close to 5 ms while an 8-bit flush of the FIFO takes 2.5 ms. With these example conditions, the average current consumption and the percentage of saved current consumption can be calculated.

Note: current consumption and wake times on different processors/MCUs will vary but this same methodology applies. The next several sections will show the analysis of flushing the FIFO at different sample rates using the assumed conditions from Table 1.

Table 1. Example Conditions

Wake-Up Time	12-bit Flush	8-bit Flush	Sleep Mode	Wake Mode
3 ms	5 ms	2.5 ms	0.5 mA	12 mA

3.1.1 Flushing the FIFO at 100 Hz ODR and Below

At 100 Hz (or less) output data rate the processor can wake up and flush the FIFO without missing any samples. The following is a timing diagram typical of how the FIFO and processor would be configured for sample rates 100 Hz or less. The FIFO collects data until the overflow flag interrupt is asserted. Then the processor wakes up and flushes all the data out of the FIFO before the next sample is ready. From sample 1 to sample 32 the processor is in sleep mode. **Note:** The processor will also be asleep during the later part of the interval before the first sample is ready. The details of the sleep to wake timing are captured below in Figure 3 and Table 2. The total current is calculated assuming 0.5 mA in sleep mode and 12 mA in active mode.

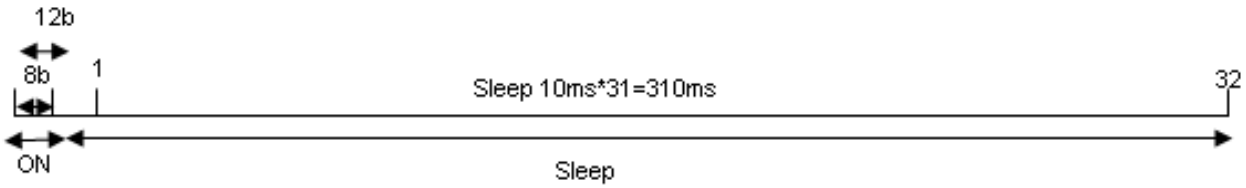

Figure 3. Timing of the FIFO at 100 Hz ODR Showing Sleep and Wake Timing

Table 2. Wake to Sleep Timing at 100 Hz ODR

Data	ODR	Total Time	Sleep Time	Wake Time	Watermark	Wake/Total	Sleep/Total	Total Current	Current Savings
12-bit	100	320 ms	312 ms	8 ms	32	8/320	312/320	0.7875 mA	93.4%
8-bit	100	320 ms	314.5 ms	5.5 ms	32	5.5/320	314.5/320	0.6978 mA	94.2%

3.1.2 Flushing the FIFO at 200 Hz ODR

When the data rate is set to 200 Hz the processor can be triggered by the watermark set at 31 samples, giving 5 ms to turn on, which is more than enough time. Then when the overflow flag asserts the FIFO is flushed, which takes almost 5 ms for flushing the 12-bit data and 2.5 ms for flushing the 8-bit data. The FIFO can be flushed at 200 Hz ODR without missing any samples by waking up from the Watermark interrupt set at sample 31 as shown in Figure 4. If the 12-bit data flush takes longer than the 5 ms then the first sample of the next data set will be missed. The results of the sleep to wake timing and current drain are captured in Table 3.

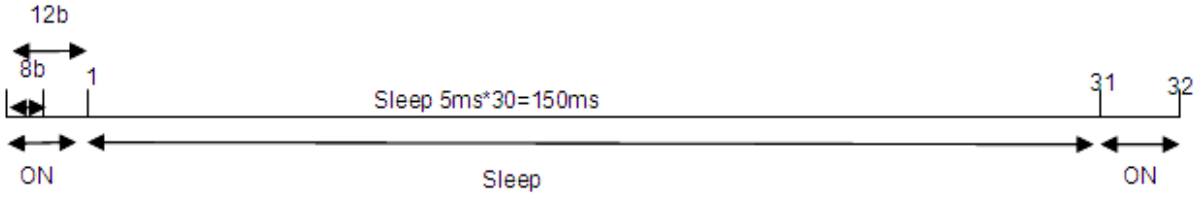

Figure 4. Timing of the FIFO at 200 Hz ODR Showing Sleep and Wake Timing

Table 3. Wake to Sleep Timing at 200 Hz ODR

Data	ODR	Total Time	Sleep Time	Wake Time	Watermark	Wake/Total	Sleep/Total	Total Current	Current Savings
12-bit	200	160 ms	150 ms	10 ms	31	10/160	150/160	1.218 mA	89.8%
8-bit	200	160 ms	152.5 ms	7.5 ms	31	7.5/160	152.5/160	1.039 mA	91.3%

3.1.3 Flushing the FIFO at 400 Hz ODR

When sampling at 400 Hz, there is a new sample every 2.5 ms, which does not allow a lot of time to wake and flush without missing samples. At 400 Hz the best way to configure the FIFO to avoid losing data is to set the Watermark for 30 samples. This is the trigger to interrupt the processor to wake up. Then, when the overflow flag is asserted, a 16 sample (12-bit data) flush occurs, which takes 2.475 ms. Next, the processor will go immediately to sleep and continue cycling through this pattern, waking up at the watermark then flushing the last 16 samples when the overflow flag asserts. When flushing 8-bit samples the FIFO should have enough time to flush the entire buffer. Figure 5 shows the timing for flushing 12-bit data at 400 Hz ODR. Figure 6 shows the timing for the 8-bit data flush at 400 Hz.

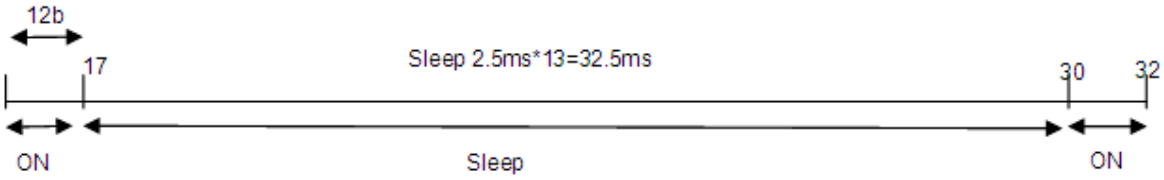

Figure 5. Timing of the FIFO at 400 Hz ODR Flushing 12-bit data Showing Sleep and Wake Timing

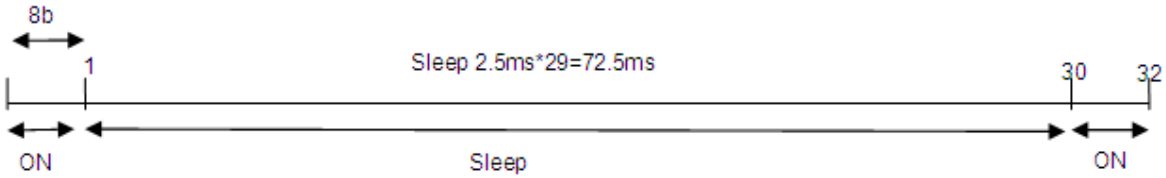

Figure 6. Timing of the FIFO at 400 HZ ODR Flushing 8-bit data Showing Sleep and Wake Timing

Table 4 presents all the calculations at 400 Hz flushing 12-bit data and 8-bit data without missing samples.

Table 4. Wake to Sleep Timing at 400 Hz ODR

Data	ODR	Total Time	Sleep Time	Wake Time	Watermark	Wake/Total	Sleep/Total	Total Current	Current Savings
12-bit	400	40 ms	32.5 ms	7.5 ms	14	7.5/40	32.5/40	2.656 mA	77.9%
8-bit	400	80 ms	72.5 ms	7.5 ms	30	7.5/80	72.5/80	1.578 mA	86.9%

Table 5 summarizes the wake and sleep timing for all sample rates of the MMA8450Q. The total current consumed per cycle and the current savings as a percentage are calculated based on the amount of time the processor is in wake vs. sleep.

Table 5. Power Savings Using FIFO at Different Data Rates

ODR	Time between Samples	Sleep/Total Ratio 12-bit	Current Consumption 12-bit Data Flush mA	Current Savings 12-bit Data (%)	Sleep/Total Ratio 8-bit	Current Consumption 8-bit Data Flush mA	Current Savings 8-bit Data (%)
1.56 Hz	641 ms	99.96%	0.505	95.8%	99.97%	0.503	95.8%
12.5 Hz	80 ms	99.69%	0.536	95.5%	99.79%	0.524	95.6%
50 Hz	20 ms	98.75%	0.644	94.6%	99.14%	0.599	95.0%
100 Hz	10 ms	97.50%	0.788	93.4%	98.28%	0.698	94.2%
200 Hz	**5 ms**	**93.75%**	**1.219**	**89.8%**	**95.31%**	**1.039**	**91.3%**
400 Hz	**2.5 ms**	**81.25%**	**2.656**	**77.9%**	**90.63%**	**1.578**	**86.9%**

From Table 5, these values can be related to the amount of time that a typical lithium ion battery for a cell phone would last. This gives a representation of power savings related to battery life time. The percentage saved for current consumption is for the application processor only. Table 6 incorporates the current consumption of the processor and the accelerometer in full power mode to give the average total current consumption. An example lithium-ion cell phone battery stores 1200 mA hours. Based on this information a comparison is made. This shows the total current consumption (processor + accelerometer) at all sample rates when the processor is continuously polling data and therefore always in the wake state.

Table 6. Example Li-Ion Battery Life Calculations without the FIFO to Data Log Data

ODR LP-Low Power N-Normal	Processor Current Consumption	MMA8450Q Current Consumption	Total Consumption	AA LI-ion Battery Life 1200 mAh (1200 mAh/Total mA) Time (h)	Time (Days)
1.56 Hz (LP)	12	0.027	12.027	99.77	4.16
1.56 Hz (N)	12	0.042	12.042	99.65	4.16
12.5 Hz (LP)	12	0.027	12.027	99.77	4.16
12.5 Hz (N)	12	0.042	12.042	99.65	4.16
50 Hz (LP)	12	0.027	12.027	99.77	4.16
50 Hz (N)	12	0.042	12.042	99.65	4.16
100 Hz (LP)	12	0.042	12.042	99.65	4.15
100 Hz (N)	12	0.072	12.072	99.40	4.14
200 Hz (LP)	12	0.072	12.072	99.40	4.14
200 Hz (N)	12	0.132	12.132	98.91	4.12
400 Hz (LP)	12	0.120	12.120	99.01	4.13
400 Hz (N)	12	0.225	12.225	98.16	4.09

When the processor is continuously running, the accelerometer current consumption has a small effect on the battery life because the processor uses much more current than the accelerometer. The ability to use the accelerometer to put the processor in a sleep mode can have a significant impact on the battery life (Table 7). The current consumption of the processor is based on the 12-bit data that was explained from Table 5. Note in Table 7 the far column on the right displays the battery life improvement by using the FIFO to data log the data. This shows that at the highest sampling rate in Normal Mode the battery life improves 4.2x what it would by polling the data with the processor continually running. At the lowest sample rate in Low Power Mode, the savings is 22.6x longer battery life.

Table 7. Example Li-Ion Battery Life Calculations Using the FIFO to Data Log 12-bit Data

ODR LP-Low Power N-Normal	Processor Current Consumption 12-bit Data Flush mA	MMA8450Q Current Consumption 12-bit Data Flush mA	Total Consumption 12-bit Data Flush mA	Li-Ion Battery 1200 mAh (1200 mAh/Total mA) Time (h)	Time (Days)	Battery Life Improvement (x Longer)
1.56 Hz (LP)	0.505	0.027	0.532	2255.64	93.98	22.6
1.56 Hz (N)	0.505	0.042	0.547	2193.78	91.41	22.0
12.5 Hz (LP)	0.536	0.027	0.563	2131.44	88.81	21.4
12.5 Hz (N)	0.536	0.042	0.578	2076.12	86.51	20.8
50 Hz (LP)	0.644	0.027	0.671	1788.38	74.52	17.9
50 Hz (N)	0.644	0.042	0.686	1749.27	72.89	17.6
100 Hz (LP)	0.788	0.042	0.830	1445.78	60.24	14.5

Table 7. Example Li-Ion Battery Life Calculations Using the FIFO to Data Log 12-bit Data

100 Hz (N)	0.788	0.072	0.860	1395.35	58.14	14.0
200 Hz (LP)	1.219	0.072	1.291	929.51	38.73	9.4
200 Hz (N)	1.219	0.132	1.351	888.23	37.01	9.0
400 Hz (LP)	**2.656**	**0.120**	**2.776**	**432.28**	**18.01**	**4.4**
400 Hz (N)	**2.656**	**0.225**	**2.881**	**416.52**	**17.36**	**4.2**

Note: A similar analysis can be done for the 8-bit data using the FIFO.

3.2 Power Savings Using the FIFO to Collect the History Leading up to an Event Trigger

Another use for the FIFO is the ability to analyze the data that occurred right up to the point of an interrupt triggering event. After the interrupt flag of the event is set, the FIFO (configured in Circular Mode) can be flushed to extract the previous 32 samples of data leading up to the event. If it is desirable for the FIFO to hold the data in the FIFO after the interrupt, then this can only be done when there is a transition from Wake to Sleep Mode only. Otherwise the FIFO must be flushed after the event to store the data in the processor for further analysis. This technique is discussed in AN3919 for analyzing directional tap. The single tap is configured and the FIFO is configured for Circular Buffer Mode to run at 400 Hz. When the tap interrupt flag is set, the FIFO is read within 15 ms of the interrupt to collect the full signature of the tap to analyze the data leading up to the event, and the data during the event. This technique can be particularly important when tracking events over a long period of time. The MCU or processor can remain asleep until the event has triggered and it can add up to substantial power savings.

3.3 FIFO Behavior During Wake to Sleep Transitions

The following table describes the different behaviors of the FIFO under the wake/sleep conditions.

Table 8. Behavior of FIFO under Wake/ Sleep Conditions

FIFO INT ENABLED	Wake from Sleep Enabled	Result (Assuming that the FIFO is set up to accept samples in either Fill or Circular Mode)
NO	NO	• FIFO will fall asleep when the sleep timer times out and no other interrupt wakes the system. • There is an AUTOMATIC flush and the FIFO starts refilling at the Sleep ODR from 0. • If another functional block causes the device to wake, the FIFO will FLUSH itself again and start filling at Wake ODR.
YES	NO	• With the interrupt enabled, the FIFO can be read and flushed (clearing the interrupt) to keep the device from falling asleep. This is dependant on the sleep time out value and how fast the FIFO is clearing the interrupt. • If the system does fall asleep, *(and no interrupts occur during the time-out period)*, the FIFO AUTOMATICALLY flushes and starts refilling at the Sleep ODR from 0. • When the device wakes up again by an interrupt, the FIFO AUTMOATICALLY flushes and starts from 0 and stores at the Wake ODR.
NO	YES	• FIFO will fall asleep if no wake events occur within the time out period. • Last data remains here in FIFO until it is flushed. • Once the FIFO is flushed, it will start collecting the new data at the current ODR.
YES	YES	• With interrupt enabled, the FIFO can be read and flushed (clearing the interrupt) to keep the device from falling asleep. • If the system does fall asleep, *(and no interrupts occur during the tim out period)*, then the FIFO will stop collecting any data. • The last data will be held in FIFO. • Once the FIFO is flushed, it will start collecting the new data at the current ODR.

When the FIFO is configured and the auto-wake/sleep is configured with the FIFO Wake from Sleep bit set (Reg 0x3A bit 7) the data in the FIFO is held until the FIFO is flushed. If a new sample arrives before the FIFO is flushed a FIFO Gate Error occurs indicating at least one sample has been missed. The FIFO Gate Error bit is set by the system in register 0x14, bit 6. This bit will clear when the FIFO is flushed. This can be useful to see the data up to the point before the device changes ODR.

Table 9. 0x14 SYSMOD: System Mode Register (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PERR	FGERR	0	0	0	0	SYSMOD1	SYSMOD0

4.0 Embedded Settings of the FIFO

The following section discusses the different registers involved in configuring the FIFO.

1. Register 0x16 bit 7 – Set **FDE** bit
2. Registers 0x11(0x01), 0x12(0x05), 8-bit and12-bit FIFO data
3. Register 0x13 FIFO Set-Up Register
4. Register 0x10 FIFO Status Register
5. Register 0x3B Interrupt Enable Register bit 6
6. Register 0x3C Interrupt Configuration Register bit 6, Route INT1 or INT2

4.1 Register 0x16: XYZ_DATA_CFG Sensor Data Configuration Register

The **XYZ_DATA_CFG** register configures the 3-axis acceleration data and event flag generator based on ODR. First the **FDE** (Bit 7) must be set (1). This points the sample data to the FIFO buffer. The Sensor Data Configuration Register is shown in Table 10.

Table 10. 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

4.2 Register 0x11-0x12: F_8DATA and F_12DATA FIFO Data

F_8DATA is Register 0x11 is shown in Table 11 and provides access to the previous (up to) 32 samples of X, Y, and Z axis acceleration data, at 8-bit resolution. Use **F_12DATA** (Table 12) to access the same FIFO data at 12-bit resolution. The advantage of **F_8DATA** access is much faster download of the sample data, since it is represented by only 3 bytes per sample (**OUT_X_MSB**, **OUT_Y_MSB**, and **OUT_Z_MSB**). The host application should initially perform a single byte read of the FIFO status byte (address 0x10) to determine the status the FIFO and if it is determined that the FIFO contains data sample(s), the FIFO contents can also be read from register address location 0x01 (8-bit data) or 0x05 (12-bit data). When the **FDE** bit is set to logic 1, the **F_12DATA** FIFO root data pointer shares the same address location as the **OUT_X_MSB** register (0x05); therefore all 12-bit accesses of the FIFO buffer data must use the I^2C register address 0x05. All reads to the register address 0x02, 0x03, 0x06, 0x07, 0x08, 0x09, and 0x0A return a value of 0x00. The **F_8DATA** FIFO root data pointer shares the same address location as the **OUT_X_MSB** register (0x01);

Table 11. 0x11 F_8DATA: 8-bit FIFO Data Register Points to Register 0x01 (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 12. Register 0x12 F_12DATA: 12-bit FIFO Data Register Points to Register 0x05 (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

4.3 Register 0x13: F_SETUP FIFO Setup Register

The setup register shown in Table 13, is used to configure the options for the FIFO. The FIFO can operate in three (3) states which are defined by the Mode Bits. The watermark bits are configured to set the number of samples of data to trigger the watermark event flag. The maximum number of samples is 32.

Table 13. Register 0x13 F_SETUP: FIFO Setup Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
F_MODE1	F_MODE0	F_WMRK5	F_WMRK4	F_WMRK3	F_WMRK2	F_WMRK1	F_WMRK0

4.3.1 Changing Modes of the FIFO

First note that the Watermark bits 0 through 5 can only be written in standby mode. The FIFO “Disable” bits 6 and 7 can be activated in Active or Standby Mode. Note the **F_Mode** cannot be switched between the two operational modes (01 and 10) in Active Mode. The **F_Mode** bits must first be set to “Disable” then they can be set to one of the two operational modes. This allows the operational modes to be switched from Fill to Circular or vise versa without going to Standby.

To change Modes while in Active try the following sequence:

A. Set the Watermark 1 to 32 counts
B. Set **F_Mode** bit 6 and 7 to “Fill” 01
C. Set Active Mode 2g Mode
D. Set **F_Mode** bit 6 and 7 to “Disable” 00
E. Set **F_Mode** bit 6 and 7 to “Circular” 10

A FIFO sample count exceeding the watermark event does not stop the FIFO from accepting new data. The FIFO update rate is dictated by the selected system ODR. In active mode the ODR is set by the **DR** bits in the **CTRL_REG1** register (0x38) and when auto-sleep is active the ODR is set by the **ASLP_RATE** field in the **CTRL_REG1** register (0x38).

When a byte is read from the FIFO buffer the oldest sample data in the FIFO buffer is returned and also deleted from the front of the FIFO buffer, while the FIFO sample count is decremented by one. It is assumed that the host application shall use the I2C multi-read transaction to dump the FIFO.

4.4 Register 0x10: F_STATUS FIFO Status Register

The FIFO Status Register shown in Table 14, is used to retrieve information about the FIFO. This register has a flag for the overflow and watermark. It also has a counter that can be checked to review the number of samples stored in the buffer.

Table 14. Register 0x10 F_STATUS: FIFO STATUS Register (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
F_OVF	F_WMRK_FLAG	F_CNT5	F_CNT4	F_CNT3	F_CNT2	F_CNT1	F_CNT0

The **F_OVF** and **F_WMRK_FLAG** flags remain asserted while the event source is still active, but the user can clear the FIFO interrupt bit flag in the interrupt source register **(INT_SOURCE Reg 0x15**) by reading the **F_STATUS** register (0x10).

The **F_OVF** bit flag will assert when the FIFO has overflowed and the **F_WMRK_FLAG** bit flag will assert when the **F_CNT** value is greater than then **F_WMRK** value. These interrupts remain asserted until the **F_STATUS** is read. Both the Watermark flag and the Overflow flag cause a System Interrupt for the FIFO in Register 0x15 when the FIFO interrupt is enabled.

5.0 Configuring the FIFO to an Interrupt Pin

In order to set up the system to route to a hardware interrupt pin, the System Interrupt (bit 6 in Reg 0x3B) must be enabled. The MMA8450Q allows for eight (8) separate types of interrupts. One (1) of these is reserved for the FIFO.

Step 1: Enable the Interrupt in Register 0x3B shown in Table 15.

Table 15. 0x3B CTRL_REG4 Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INT_EN _ASLP	INT_EN _FIFO	INT_EN _TRANS	INT_EN _LNDPRT	INT_EN _PULSE	INT_EN _FF_MT_1	INT_EN _FF_MT_2	INT_EN_DRDY

The **INT_EN_FIFO** interrupt enable bit allows the FIFO function to route its event detection flag to the interrupt controller of the system. The interrupt controller routes the enabled function to either the INT1 or INT2 pin.

To enable the FIFO, set bit 6 in Register 0x3B as follows:

Code Example: IIC_RegWrite (0x3B, 0x40);

Step 2: Route the interrupt to INT1 or to INT2. This is done in register 0x3C shown in Table 16.

Table 16. 0x3C CTRL_REG5 Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INT_CFG_ASLP	INT_CFG_FIFO	INT_CFG_TRANS	INT_CFG_LNDPRT	INT_CFG_PULSE	INT_CFG_FF_MT_1	INT_CFG_FF_MT_2	INT_CFG_DRDY

Note: To route the FIFO to INT1 set bit 6 in register 0x3C. Clear bit 6 to set the FIFO to INT2.

Code Example: IIC_RegWrite (0x3C,0x40); //Set to INT1

5.1 Reading the System Interrupt Status Source Register

In the interrupt source register, the status of the various embedded features can be determined. This is shown in Table 17.

The bits that are set (logic ‘1’) indicate which function has asserted an interrupt; conversely, the bits that are cleared (logic ‘0’) indicate which function has not asserted or has de-asserted an interrupt. The interrupts are rising edge sensitive. The bits are set by a low to high transition and are cleared by reading the appropriate interrupt source register.

Table 17. 0x15 INT_SOURCE: System Interrupt Status Register (Read Only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRC_ASLP	SRC_FIFO	SRC_TRANS	SRC_LNDPRT	SRC_PULSE	SRC_FF_MT_1	SRC_FF_MT_2	SRC_DRDY

6.0 Example Code Using the FIFO

The following are three examples for configuring the FIFO. Table 18 shows all the registers of importance for using the FIFO.

Table 18. Registers of Importance for the FIFO

Reg	Name	Definition	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
10	F_STATUS	FIFO Status R	F_OVF	F_WMRK_FLAG	F_CNT5	F_CNT4	F_CNT3	F_CNT2	F_CNT1	F_CNT0
11	F_8DATA	8-bit FIFO Data R	XD11	XD10	XD9	XD8	XD7	XD6	XD5	XD4
12	F_12DATA	12-bit FIFO Data R	0	0	0	0	XD3	XD2	XD1	XD0
13	F_SETUP	FIFO Setup R/W	F_MODE1	F_MODE0	F_WMRK5	F_WMRK4	F_WMRK3	F_WMRK2	F_WMRK1	F_WMRK0
14	SYSMOD	System Mode R	PERR	FGERR	0	0	0	0	SYSMOD1	SYSMOD0
15	INT_SOURCE	Interrupt Status R	SRC_ASLP	SRC_FIFO	SRC_TRANS	SRC_LNDPRT	SRC_PULSE	SRC_FF_MT_1	SRC_FF_MT_2	SRC_DRDY
3A	CTRL_REG3	Control Reg3 R/W (Wake Interrupts from Sleep)	FIFO_GATE	WAKE_TRANS	WAKE_LNDPRT	WAKE_PULSE	WAKE_FF_MT_1	WAKE_FF_MT_2	IPOL	PP_OD
3B	CTRL_REG4	Control Reg4 R/W (Interrupt Enable Map)	INT_EN _ASLP	INT_EN _FIFO	INT_EN _TRANS	INT_EN _LNDPRT	INT_EN _PULSE	INT_EN _FF_MT_1	INT_EN _FF_MT_2	INT_EN_DRDY
3C	CTRL_REG5	Control reg5 R/W (Interrupt Configuration)	INT_CFG_ASLP	INT_CFG_FIFO	INT_CFG_TRANS	INT_CFG_LNDPRT	INT_CFG_PULSE	INT_CFG_FF_MT_1	INT_CFG_FF_MT_2	INT_CFG_DRDY

6.1 Power Minimization Example: Data logger Collecting 12-bit Data 100 Hz

Step 1: Go to Standby Mode to configure the device

IIC_RegWrite(0x38, 0x08); //Ctrl Reg1: 100 Hz mode, standby

Step 2: Register 0x16 Set FDE bit =1 to enable the FIFO

IIC_RegWrite(0x16, 0x84); //Set FDE = 1, and leave Event flag on Z-axis

Step 3: Set the FIFO to Fill Buffer Mode in Register 0x13 F_Setup

IIC_RegWrite(0x13, 0x80); //FIFO Set to Fill Mode

Step 4: Set the FIFO Interrupt Pin to Int1 to flush the data every 32 samples when the overflow flag asserts

IIC_RegWrite(0x3B, 0x40); //Enable the interrupt Pin for the FIFO

IIC_RegWrite(0x3C, 0x40); //Set the interrupt to route to INT1

Step 5: Put device in 2g Active Mode

IIC_RegWrite(0x38, 0x09); //2g Active Mode 100 Hz

Step 6: Write an Interrupt Service Routine to Clear the interrupt and Flush the data from the FIFO

The interrupt service routine must be able to wake the device and flush the data within 10 ms at this sample rate. It will take the 12-bit data 5 ms to flush 32 samples at 400 kHz as shown in calculations above.

```
Interrupt void isr_KBI (void)
{
      WAKE_MCU;
      //clear the interrupt flag
      CLEAR_KBI_INTERRUPT;

      //Determine the source of the interrupt by reading the system interrupt
      Int_SourceSystem = IIC_RegRead(0x15);
      // Set up Case statement here to service all of the possible interrupts
      if ((Int_SourceSystem&0x40)==0x40)
      {
            // 1. Read the Status Register to Clear the Overflow Flag Interrupt
```

```
                IIC_RegRead(0x10);

                // 2. This would be used to burst out 12 bit data
                ComObj.ReadDataBurst(CurDeviceAddress, 0X05, ref databuf12, 192);
                //This would be used to burst out 8 bit data
                ComObj.ReadDataBurst(CurDeviceAddress, 0X01, ref databuf8, 96);
            }
        }
```

6.2 Event Detection Waiting for a Tap Event to Flush the Data for Further Analysis 400 Hz ODR, 8g Mode

Step 1: Put device in Standby Mode

IIC_RegWrite(0x38, 0x00); //Ctrl Reg1: 400 Hz mode, standby

Step 2: Register 0x16 Set FDE bit = 1 to enable the FIFO

IIC_RegWrite(0x16, 0x84); //Set FDE = 1, and leave Event flag on Z-axis

Step 3: Set the FIFO to Circular Buffer Mode in Register 0x13 F_Setup

IIC_RegWrite(0x13, 0x40); //FIFO Set to Circular Buffer Mode

Step 4: Enable X and Y and Z Single Pulse

IIC_RegWrite(0x2F, 0x15)

Step 5: Set Threshold 1.55g on X and 2.58g on Z

Note: Every step is 0.258g

- 1.55g/ 0.258g = 6 counts
- 2.58g/0.258g = 10 counts

IIC_RegWrite(0x31, 0x06); //Set X Threshold to 1.55g

IIC_RegWrite(0x32, 0x06); //Set Y Threshold to 1.55g

IIC_RegWrite(0x33, 0x0A); //Set Z Threshold to 2.58g

Step 6: Set Time Limit for Tap Detection to 50ms

Note: 400 Hz ODR, Time step is 0.625 ms per step

- 50 ms/0.625 ms = 80 counts

IIC_RegWrite(0x34,0x50); //50 ms

Step 7: Set Latency Timer to 300 ms

- **Note**: 400 Hz ODR, Time step is 1.25 ms per step
- 300ms/1.25 ms = 240 counts

IIC_RegWrite(0x35,0xF0); //300 ms

Step 8: Set the Tap Interrupt Pin to INT1 to Alert to MCU to wake up and flush the FIFO

IIC_RegWrite(0x3B, 0x08); //Enable the interrupt Pin Tap

IIC_RegWrite(0x3C, 0x08); //Set the interrupt to route to INT1

Step 9: Put Device in 8g Active Mode

IIC_RegWrite(0x38, 0x03); //8g Active Mode

Step 10: Write an Interrupt Service Routine to Clear the interrupt and Flush the data from the FIFO

The interrupt service routine will wake the MCU, and then wait 10-12 ms to allow a few more samples to come into the buffer, and then it will flush the FIFO. Sometimes the rebound acceleration data can be very insightful to understanding an event. Therefore it is a good idea to capture the data right before the event, during the event and maybe a few samples after the event. All of this can be accomplished with the FIFO.

```
Interrupt void isr_KBI (void)
{
      WAKE_MCU;
      //clear the interrupt flag
      CLEAR_KBI_INTERRUPT;
      //Determine the source of the interrupt by first reading the system interrupt
      register
      Int_SourceSystem = IIC_RegRead(0x15);
      // Set up Case statement here to service all of the possible interrupts
      if ((Int_SourceSystem&0x08)==0x08)
      {
            // 1. Read the Status Register to Clear the Tap Status Flag
            TapStatus = IIC_RegRead(0x30);
            TIME_DELAY(10); //10ms delay to allow 4-5 more samples
            // 2. If 8 bit data only required
             ComObj.ReadDataBurst(CurDeviceAddress, 0X01, ref databuf8, 96);
            // Or if 12 bit data required
            ComObj.ReadDataBurst(CurDeviceAddress, 0X05, ref databuf12, 192);
      }
}
```

6.3 Auto-Wake Sleep Trigger Using the FIFO to Hold the Data that Saved Before the ODR Changed

Step 1: Put the device in Standby Mode, ASLEEP = 50 Hz, Wake DR = 400 Hz

IIC_RegWrite (0x38, 0x00);

Step 2: Enable Auto Sleep: To enable the Auto-Wake/Sleep set bit 1 in Register 39, the SLPE bit.

- **Note:** Register 0x39 CTRL_REG2

CTRLReg2Data = IIC_RegRead(0x39); //Store value in the Register

CTRLReg2Data| = 0x02; //Set the Sleep Enable Bit

IIC_RegWrite(0x39, CTRLReg2Data); Write the updated value in CTRL Register 2.

Step 3: Set the Sleep Timer for 20 seconds to time out

Note: Time step at 400 Hz ODR = 320 ms steps

20s/ 0.32s = 62.5, rounded to 63 counts

IIC_RegWrite(0x37, 0x3F);

Step 4: Set the FIFO Gate and the Motion Block 1 in the Wake From Sleep Register

WakeRegData = IIC_RegRead(0x3A); //Store Register contents

WakeRegData| = 0x88; Set the Wake from Motion and the FIFO Gate

IIC_RegWrite(0x3A, WakeRegData);

Step 5: Configure the FIFO, enabling FDE=1 from Register 0x16

IIC_RegWrite(0x16, 0x84); //Set FDE = 1, and leave Event flag on Z axis

Step 6: Put the FIFO in Circular Buffer Mode

IIC_RegWrite(0x13, 0x40); //FIFO Set to Circular Buffer Mode

Step 7: Set up the Motion Freefall Block 1 (INT2), and Auto Sleep (INT1)

IIC_RegWrite(0x3B, 0x84); // Enable Motion Block, Auto Sleep

IIC_RegWrite(0x3C, 0x80); //Set INT1 to AutoSleep, Set INT2 to Motion

Step 8: Enable X, Y, Z with “OR” Condition for Motion Detection

IIC_RegWrite(0x23, 0x6A);

Step 9: Set the Threshold to 2g in 2g Mode

Note: In 2g mode each count is 15.75 mg

- 2g/0.01575g

IIC_RegWrite(0x25, 0x7F);

Step 10: Set the debounce counter to eliminate false readings at 50 Hz (Sleep ODR)

IIC_RegWrite(0x26, 0x04);

Step 11: Put the device in 2g Active Mode, Wake sample rate is 400 Hz

IIC_RegWrite(0x38, 0x01);

Step 12: Set up the Interrupt Service Routine

```
Interrupt void isr_KBI (void)
{
      WAKE_MCU;
      //clear the interrupt flag
      CLEAR_KBI_INTERRUPT;
      //Determine the source of the interrupt by first reading the system interrupt
      register
      Int_SourceSystem = IIC_RegRead(0x15);
      // Set up Case statement here to service all of the possible interrupts
      if ((Int_SourceSystem&0x80)==0x80)
      {
            //Perform an Action since AutoSleep Flag has been set
            //Read the System Mode to clear the system interrupt
            Int_SysMod = IIC_RegRead(0x14);
            if (Int_SysMod==0x02)
            {
                  // sleep mode
                  //Flush Data to MCU and store
                  ComObj.ReadDataBurst(CurDeviceAddress, 0X01, ref databuf8, 96);

            }
            else if (Int_SysMod==0x01)
            {
                  //Wake Mode
                  //Flush Data to MCU and store
                  ComObj.ReadDataBurst(CurDeviceAddress, 0X01, ref databuf8, 96);

            }
            else
            {
                  //Error
            }
      }
}
```

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

AN3920
Rev. 0
02/2010