Freescale Semiconductor
Application Note

Document Number: AN3948
Rev. 1, 12/2009

i.MX25 Integrated Analog-to-Digital Converter

by *Multimedia and Applications Division*
Freescale Semiconductor, Inc.
Austin, TX

The i.MX25 multimedia applications processor has an embedded analog-to-digital converter (ADC) in addition to the broad external memory and digital connectivity peripherals. The general-purpose ADC on the i.MX25 was designed to encompass the touch-screen controller (TSC). However, TSC functions have been disabled in some versions of i.MX25.

This application note discusses the use of the general-purpose ADC that is enabled on all versions of i.MX25. Key benefits of an on-chip ADC include the following:

- Low power consumption
- High resolution and accuracy
- Reduced BOM costs

Contents

freescale semiconductor

1 i.MX25 ADC Features

The ADC converts an input analog voltage to a digital number proportional to the magnitude of the voltage. The i.MX25 on-chip ADC is flexible; provides accurate, high-speed conversions; has high resolution; and is power-efficient. It is based on the successive-approximation (SAR) ADC architecture.

1.1 High Speed Conversions

The ADC module uses only one root clock generated from the i.MX25 IPG clock, which eliminates the complexity of using a multi-clock domain. The IPG root clock on i.MX25 is typically 66.67 MHz, but in low-power mode, it can be reduced to 33.33 MHz or 16.66 MHz. Additionally, the ADC module root clock can be gated on or off in the clock control module (CCM), or the ADC clock can be gated off within the ADC module in the IPG_CLKEN bit field of the TGCR register. This can further reduce power consumption when in low power modes or when not using the ADC.

The actual clock driving ADC logic is derived from the IPG root clock. The root clock needs to be divided down to provide a 1.6667 MHz clock driving the ADC. This conversion is controlled with the ADC clock divider, which is set in the ADCCLKCFG[4–0] bit field in the TGCR register. Figure 1 shows the clock diagram for generating the clock to the ADC, and Table 1 below provides configuration examples for various IPG clock frequencies.

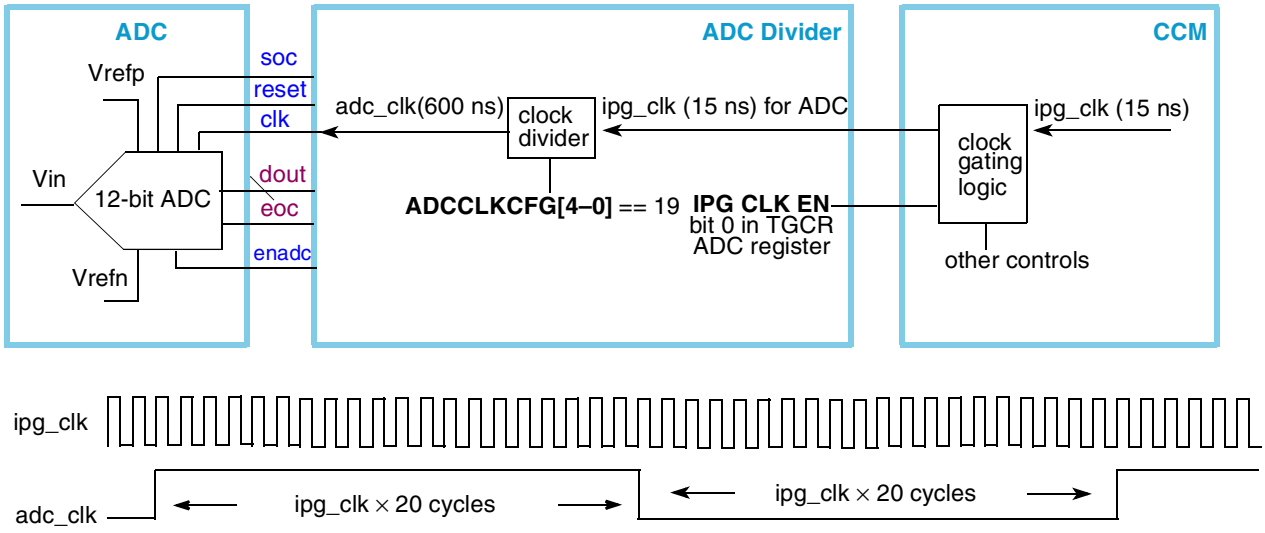

Figure 1. ADC Clock Generation

Table 1. Clock Examples

IPG clock		ADCCLKCFG	ADC Clock[1]	
66.67 MHz	15 ns	19	1.667 MHz	600 ns
33.33 MHz	30 ns	9	1.667 MHz	600 ns
16.67 MHz	60 ns	4	1.667 MHz	600 ns

1 The ADC clock frequency should be less than 1.75 MHz.

The 1.6667 MHz clock (period of 600 ns) to the ADC allows for a maximum implementation of about 119 K samples (12 bits per sample) per second.

1.2 High Accuracy

The ADC converts an input analog voltage to a corresponding digital code, but with any ADC, error is associated with the conversion. The curve that describes this behavior is called the actual transfer function. The curve that describes an ideal (zero error) ADC conversion behavior is called the ideal transfer function. The ideal transfer function of an ADC is simply a straight line from the minimum input voltage (Vrefl or NGND_ADC) to the maximum input voltage (Vrefh or ADC_REF). The ideal transfer function is then divided into steps by the number of codes the ADC is capable of resolving, meaning that the input voltage range is divided into steps with each step having the same width or size.

The ideal code width (ICW), also known as 1 LSB, is:

$$ICW = 1LSB = (Vrefh - Vrefl) / 2^N \qquad \text{Eqn. 1}$$

where N is the width of the ADC. For i.MX25, this is 12 bits.

The ideal transfer function is then:

$$Code = (Vin - Vrefl) / 1LSB \qquad \text{Eqn. 2}$$

Figure 2 shows a graph of the ideal transfer function.

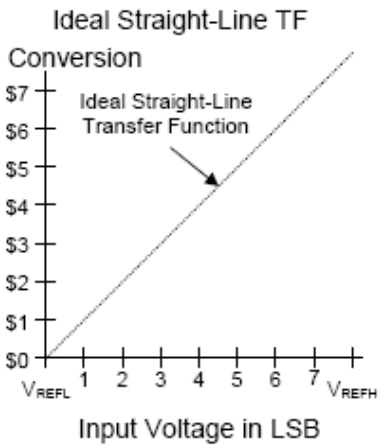

Figure 2. Ideal Transfer Function Graph

This graph assumes the ADC is perfectly linear or that a given change in input voltage creates the same change in conversion code regardless of the input's initial level. However, nothing is ever ideal in the analog world, and all ADCs exhibit some non-linearity. This non-linearity is typically specified by differential non-linearity (DNL) and integral non-linearity (INL).

Differential non-linearity (DNL) is the maximum of the differences in each conversion's current code width (CCW) and the ideal code width (ICW). DNL is the most critical parameter. It specifies an ADC's

performance in high-accuracy applications because it represents the ADC's ability to relate a small change in input voltage to the correct change in code conversion. DNL is defined as:

$$Code\ DNL = CCW - ICW \qquad \textit{Eqn. 3}$$

$$DNL = Max\ (Code\ DNL) \qquad \textit{Eqn. 4}$$

Two other critical parameters used in defining ADC operation are related to DNL: missing codes and monotonicity. An ADC has missing codes if an infinitesimally small change in voltage causes a change in the result of two codes, with the intermediate code never being set. A DNL of –1.0 LSB indicates that the ADC has missing codes. An ADC is monotonic if the conversion result continually increases with increasing voltage (and vice versa). A non-monotonic ADC may give a lower conversion result for a higher input voltage, which can also mean that the same conversion may result from two separate voltage ranges. Some literature suggests that a DNL of greater than 1 LSB may indicate non-monotonicity. For i.MX25 the ADC's DNL is specified as ±0.75 LSB.

Figure 3 shows how DNL affects the transfer function graph.

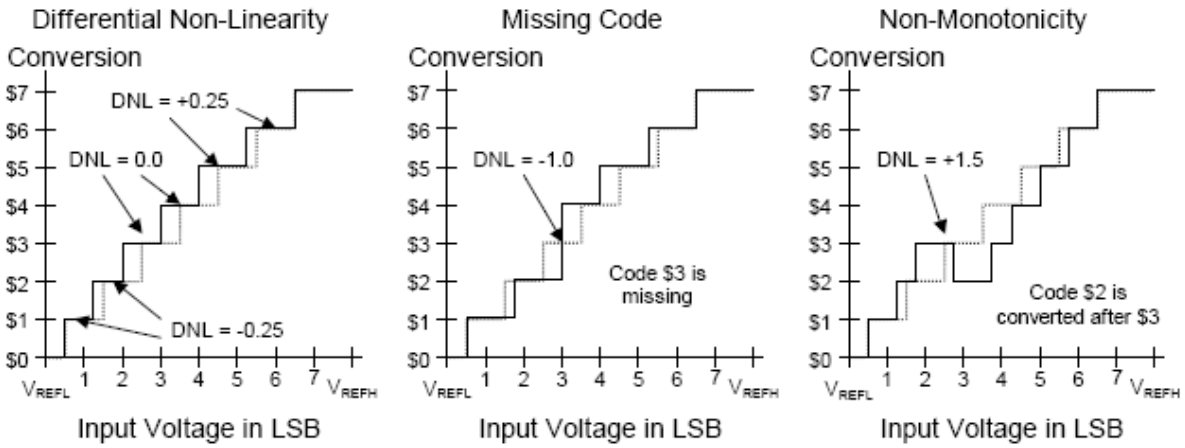

Figure 3. Differential Non-Linearity, Missing Codes, and Non-Monotonicity Graphs

Integral non-linearity is defined as the sum from the first to the current conversion (integral) of the non-linearity at each code (Code DNL). For example, if the sum of the DNL up to a particular point is 1 LSB, the total code width to that point is 1 LSB greater than the sum of the ideal code width. Therefore, the current point converts one code lower than the ideal conversion.

INL represents the curvature in the actual transfer function relative to the ideal transfer function, or the difference between the current and the ideal transition voltages. For i.MX25, INL is specified as ±2 LSB. In general INL is defined as:

$$Code\ INL = V\ (current\ transition) - V\ (ideal\ transition) \qquad \textit{Eqn. 5}$$

$$INL = Max\ (Code\ INL) \qquad \textit{Eqn. 6}$$

Figure 4 shows a representation of how INL affects the transfer function graph.

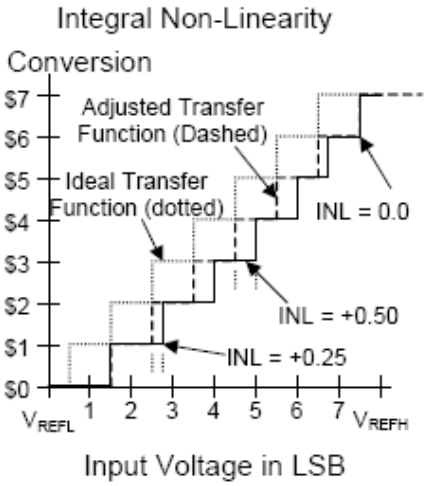

Figure 4. Integral Non-Linearity

1.3 High Resolution

The embedded ADC on i.MX25 is a 12-bit ADC providing 12 bits of resolution. This gives up to 4096 different possible conversions. The resolution voltage per conversion step is ADC_REF divided by 4096. In a typical use case, when using an external reference, this is 3.3 V ÷ 4096, which is about 800 µV per conversion step, a typical ICW.

The 12-bit output logic coding of the conversion data is provided in unsigned binary. This means that at the bottom of the scale when the analog input is set to NGND_ADC, the 12-bit output conversion is 0x000. Similarly, when the analog input voltage is set to VREF at the top of the scale, the 12-bit output conversion is 0xFFF. This is commonly referred to as full scale.

1.4 Low-Power Consumption

The integrated ADC is a power-efficient 12-bit ADC, which is powered by two supplies. The i.MX25 core voltage QVDD powers the ADC's digital control logic, while the NVCC_ADC provides power to the analog portion of the ADC. During conversions the current draw on NVCC_ADC is about 2.6 mA and about 500 µA current draw on the QVDD supply. When not in use, the ADC can be powered down with less than 10 µA of current consumption on QVDD and under 1 µA on the NVCC_ADC supply.

Additionally, the ADC has three power modes:

- Always-off mode (the ADC is powered down continuously—default setting)
- Power saving mode (ADC is powered down when ADC is not converting)
- Always-on mode (ADC is powered on continuously)

The active ADC power mode is controlled with the POWERMODE[1–0] bit field in the TGCR register.

1.5 Analog Inputs

i.MX25 can provide up to eight general-purpose ADC inputs. There are five TSC inputs and three general-purpose auxiliary inputs. However, all of the ADC inputs, including the five touch screen inputs, can be used to convert analog input signals even when the TSC function is not used or disabled. This allows up to eight general-purpose inputs.

The ADC has a wide input range. Zero scale is the voltage equal to NGND_ADC. Full scale is equal to the voltage on REF (REF is recommended to be set at NVCC_ADC). All of the ADC inputs can have inputs anywhere between the negative reference and the positive reference voltages. The internal reference is fixed at 2.5 V, but an external reference can range from 2.5 V up to NVCC_ADC for flexibility.

2 External Signal Description

Table 2 shows the ADC i.MX25 external signals.

Table 2. ADC i.MX25 External Signals

PAD	Function	I/O
NVCC_ADC	ADC main power supply	I
NGND_ADC	ADC ground	I/O
REF[1]	External reference voltage	I
XP	ADC Analog Input	I
XN	ADC Analog Input	I
YP	ADC Analog Input	I
YN	ADC Analog Input	I
WIPER	ADC Analog Input	I
INAUX0	ADC Analog Input	I
INAUX1	ADC Analog Input	I
INAUX2	ADC Analog Input	I

1 REF should be tied to NVCC_ADC or left floating if an external reference is not used. However, for optimal ADC operations, it is recommended to use an external low-noise reference.

3 Functional Description and Programming the ADC

Some additional details on the ADC architecture and design can be found in the TSC/ADC chapter of the *i.MX25 Reference Manual*.

For the i.MX25, the ADC start-up sequence and ADC conversion operations are transparent to the software programmer and are automatically implemented by the ADC hardware. The ADC startup sequence includes a dummy conversion cycle at the end of the start-up sequence to let the voltages of the internal analog nodes settle to the right levels. The results from that conversion should be disregarded.

3.1 Memory Map Definitions for Programming Example

The memory map used in the programming example described in this entire section is as follows:

Example 1. ADC Memory Map Used in the Programming Example

```
// ADC registers used
#define ADC_BASE          0x50030000
#define ADC_TGCR          (ADC_BASE + 0x000)// TSC General cfg reg
#define ADC_TGSR          (ADC_BASE + 0x004)// TSC General Status reg
#define ADC_TICR          (ADC_BASE + 0x008)// TSC IDLE cfg reg
#define ADC_GCQFIFO       (ADC_BASE + 0x800)// General Convert Queue FIFO
#define ADC_GCQCR         (ADC_BASE + 0x804)// General Queue Control reg
#define ADC_GCQSR         (ADC_BASE + 0x808)// General Queue Status reg
#define ADC_GCQMR         (ADC_BASE + 0x80C)// General Queue Mask reg
#define ADC_GCQITEM_7_0 (ADC_BASE + 0x820)// General Queue ITEM 7-0
#define ADC_GCQITEM_15_8(ADC_BASE + 0x824)// General Queue ITEM 15-8
#define ADC_GCC0          (ADC_BASE + 0x840)// General Convert Config 0
#define ADC_GCC1          (ADC_BASE + 0x844)// General Convert Config 1
#define ADC_GCC2          (ADC_BASE + 0x848)// General Convert Config 2
#define ADC_GCC3          (ADC_BASE + 0x84C)// General Convert Config 3
#define ADC_GCC4          (ADC_BASE + 0x850)// General Convert Config 4
#define ADC_GCC5          (ADC_BASE + 0x854)// General Convert Config 5
#define ADC_GCC6          (ADC_BASE + 0x858)// General Convert Config 6
#define ADC_GCC7          (ADC_BASE + 0x85C)// General Convert Config 7

// CCM registers used in example controlling clock gating
#define CCM_CGR2          (0x53F80014) // Clock Gating Register 2
```

3.2 Starting-Up/Enabling the ADC

Internal reference should only be enabled if it is going to be used. For better performance, a low-noise external reference is recommended. The recommended initial start-up sequence to enable the ADC is as follows:

Example 2. Enabling the ADC

```
*(unsigned int*)(CCM_CGR2) &= ~(1<<13);  // disable adc ipgclk in CCM
*(unsigned int*)(ADC_TGCR) &= ~(1<<0);   // disable adc ipgclken bit
*(unsigned int*)(CCM_CCTL) |=  (1<<15);  // set ipgclk ctl by adc enable bit
*(unsigned int*)(CCM_CGR2) |=  (1<<13);  // enable adc ipgclk in CCM
*(unsigned int*)(ADC_TGCR) |=  (1<<0);   // enable adc ipgclken bit
*(unsigned int*)(ADC_TGCR) |=  (1<<1);   // self-reset adc
while(*(unsigned int*)(ADC_TGCR) & (1<<1));  // wait until self-reset is done
*(unsigned int*)(ADC_TGCR) |=  (1<<8);   // adc in power saving mode
*(unsigned int*)(ADC_TGCR) |=  (1<<10);  // enable internal ref
```

3.3 Setting-Up the Conversion Queues

To make the conversion process more transparent to the software programmer, two individual conversion queues are implemented in the ADC control logic hardware.

- Touch Screen-Convert-Queue(TCQ) for 4-wire/5-wire touch screen measurement purpose
- General ADC-Convert-Queue(GCQ) for general measurement purpose: “temperature,” “pressure,” “voltage,” and so on

When the programmer triggers a conversion to start, the ADC converts every entry in the queue. Each queue contains sixteen possible measurement entries. Both the TSQ and GCQ can be used for general purpose conversions, even in versions of i.MX25 where TSC is not enabled.

When a conversion is triggered to start an ADC acquisition, the ADC takes all measurements pointed to by the conversion queue. A total of sixteen conversions can be made with a single trigger on each conversion queue. The ADC starts the conversion pointed to by the first entry in the conversion queue. It then goes through each entry one by one and stores the sampled result into the corresponding FIFO until it reaches the end of the conversion queue, which is determined by the LAST_ITEM_ID bit-field defined in the queue control register.

Each item in the conversion queue can point to any of the convert configuration registers. This allows the convert configuration register to define the actual measurement and the measurement to be called anywhere and as many times as needed in each conversion queue. Programmers can define a conversion using one of the convert configuration registers and use it anywhere in any of the queues.

Figure 5 shows how the ADC queues are structured.

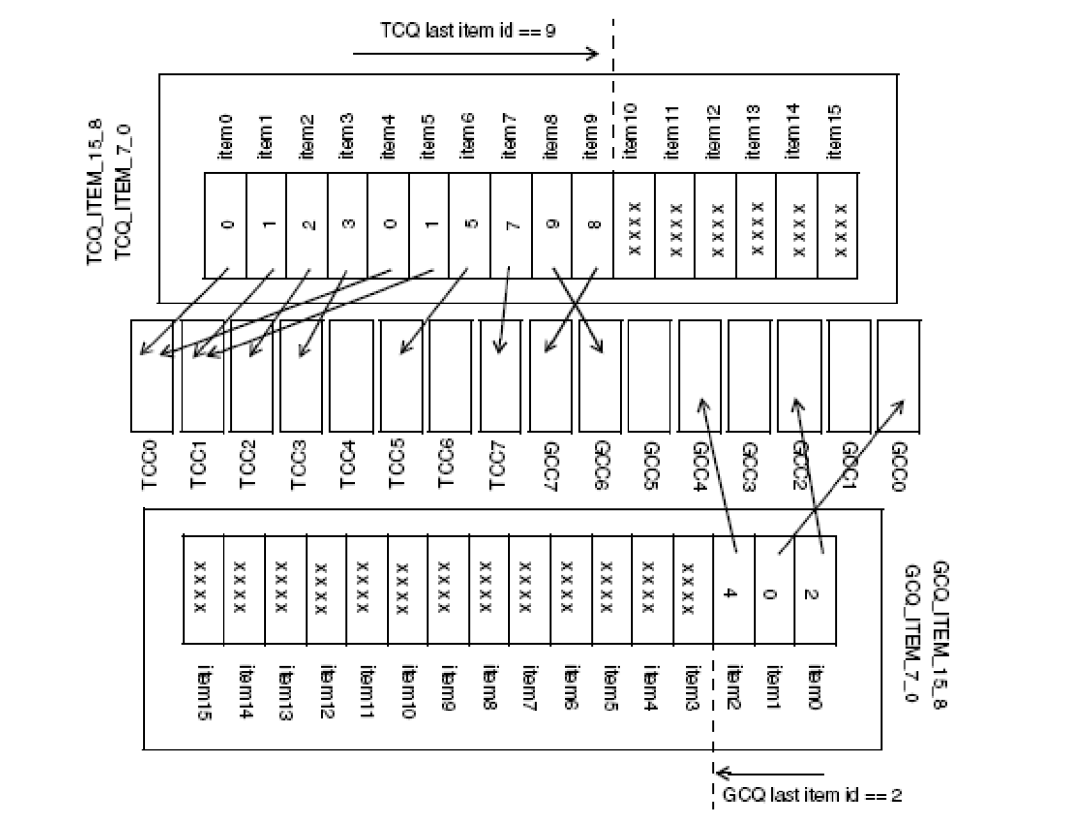

Figure 5. Convert Configuration and Queue Structure

Example 3 shows how the general conversion queue can be configured to make three different measurements each time that an ADC acquisition is triggered. Keep in mind that the ADC_GCQITEM registers can be programmed to take the measurements of the conversion configuration registers in any order.

Example 3. Setting-Up the General Conversion Queue

```
// ADC registers used
*(unsigned int*)(ADC_GCQCR) &= 0xFFFF0000;  // set GCQCR values to 0
*(unsigned int*)(ADC_GCQCR) |= (2<<8);  // FF watermark at 3, trig interrupt
*(unsigned int*)(ADC_GCQCR) |= (2<<4);  // last item =2, three items in queue
*(unsigned int*)(ADC_GCQCR) |= (1<<1);  // queue will get trig with FQS bit
*(unsigned int*)(ADC_GCQMR) &= ~(1<<15);// enable FDRY IRQ, in mask reg
*(unsigned int*) ADC_GCQITEM_7_0 = 0x210;  // Item[0]=GCC0,[1]=GCC1,[2]=GCC2
```

Additionally, this example enabled an interrupt to occur after there are three entries in the FIFO, essentially triggering after all entries in the conversion queue have completed because there are only three entries in the queue of this example. Also, setting the FQS bit triggers the conversion to start.

3.4 Defining Each Conversion

There can be up to sixteen different measurement configurations in the ADC by using all eight general conversion configuration registers and all eight TSC conversion configuration registers. This example only uses three of the general conversion configuration registers, which are pointed to by the GCQ_ITEM_7_0 register that was set up in the previous section. The conversion configuration registers give the ability to configure parameters like high and low references used for measurement, settling time, input used, and so on. More detailed information about the registers can be found in the TSC/ADC chapter of the *i.MX25 Reference Manual*.

Example 4 shows how to configure three measurements, all on the AUX0 input line, with different parameters making each of the measurements unique.

Example 4. Setting-Up Conversion Configuration Registers

```
// set up GCC0
*(unsigned int*)(ADC_GCC0) = 0x0; // reset to 0
*(unsigned int*)(ADC_GCC0) |= (0x7F<<24); // set settling time (3x8)+1 cycle
*(unsigned int*)(ADC_GCC0) |= (1<<8); // positive ref set to External Ref
*(unsigned int*)(ADC_GCC0) |= (5<<4);// input channel select INAUX_0
*(unsigned int*)(ADC_GCC0) |= (3<<2);// neg reference set to NGND_ADC

// set up GCC1
*(unsigned int*)(ADC_GCC1) = 0x0; // reset to 0
*(unsigned int*)(ADC_GCC1) |= (0xFF<<24); // set settling time at max
*(unsigned int*)(ADC_GCC1) |= (1<<8);// positive ref set to External Ref
*(unsigned int*)(ADC_GCC1) |= (5<<4); // input channel select INAUX_0
*(unsigned int*)(ADC_GCC1) |= (3<<2);// neg reference set to NGND_ADC

// set up GCC2
*(unsigned int*)(ADC_GCC2) = 0x0;// reset to 0
*(unsigned int*)(ADC_GCC2) |= (0xFF<<24);  // set settling time at max
*(unsigned int*)(ADC_GCC2) |= (1<<8);    //
*(unsigned int*)(ADC_GCC2) |= (1<<7); // use Internal Reference
*(unsigned int*)(ADC_GCC2) |= (5<<4);// input channel select INAUX_0
*(unsigned int*)(ADC_GCC2) |= (3<<2);// neg reference set to NGND_ADC
```

3.5 Reading Acquisition Data

The acquisition data from each conversion queue can be stored into their corresponding FIFOs. There are two internal independent FIFOs, each with sixteen 16-bit entries used for storing the TCQ and GCQ conversion queue results. These FIFOs help pass sampled data from the ADC to the CPU data bus. Both FIFOs have the same structure. Each FIFO contains sixteen entries and two pointers (a read and write pointer). Each entry is 16-bits for storing the 12-bit conversion result and a 4-bit item ID used to distinguish which item in the queue the conversion result corresponds to. Figure 6 shows the 16-bit FIFO entry format.

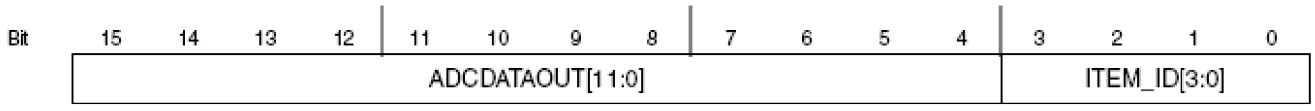

Figure 6. 16-bit FIFO Entry Format

The FIFO pointers are automatically maintained by the built-in ADC hardware. The write pointer always points to the next entry to be written. On a FIFO write operation, if the FIFO is not full, the FIFO entry location pointed to by the write pointer is written and the pointer is incremented to the next location. Similarly the read pointer always points to the current FIFO entry to be read. On a FIFO read operation, if the FIFO is not empty, the FIFO word location pointed to by the read pointer is read and the pointer is incremented to the next location.

The FIFO pointers are automatically updated when the ADC acquires new sampled data (updates the write pointer) or when the user reads the contents of the queue FIFO registers (read pointer updated). After reading the queue FIFO register for either the TSC queue or the general-purpose queue (TCQFIFO, GCQFIFO), the data read is the data that was currently pointed to by the read pointer, and the read pointer is updated to point to next location. Figure 7 shows the FIFO structure.

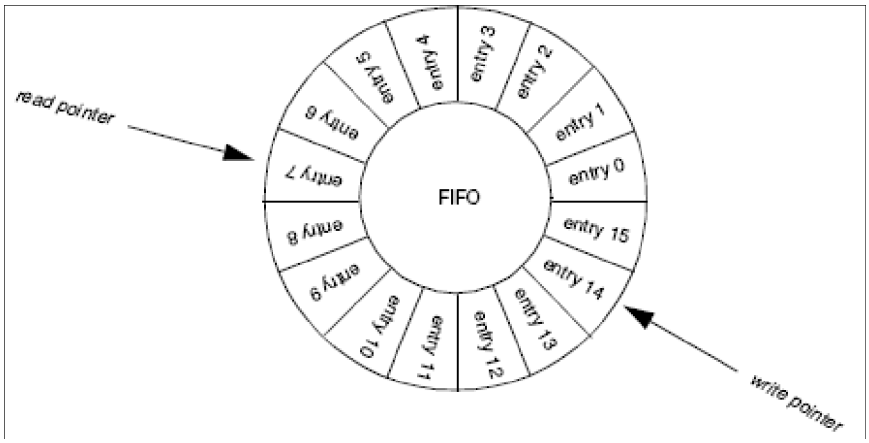

Figure 7. FIFOs Structure

Example 5 shows how acquisition data can be read out of the FIFOs. Keep in mind in the previous example the queue control register was set up so that setting the Force Queue Start (FQS) bit starts the conversion queue and triggers an interrupt once the FIFO has three samples in the FIFO. Example 5 shows how the ADC Interrupt Service Routine (ISR) can be set up to read that acquisition data.

Example 5. ADC ISR to Read Acquisition Data

```
unsigned int adc_data_buf[1000];  //global array to store data
unsigned int adc_buf_index;             // global so can be updated anywhere

void ADC_ISR(void){
*(unsigned int*)(AVIC_INTDISNUM) = 46;          // disable adc interrupts
if((*(volatile unsigned int *) ADC_GCQSR & (1<<15))  != 0) //FDRY interrupt
{
        if(adc_buf_index<1000) {
                // pop first item in the queue
                adc_data_buf[adc_buf_index++]=reg32_read(ADC_GCQFIFO);
                // pop second item in the queue
                adc_data_buf[adc_buf_index++]=reg32_read(ADC_GCQFIFO);
                // pop third item in the queue
                adc_data_buf[adc_buf_index++]=reg32_read(ADC_GCQFIFO);
        }
        else{ while(1); } // stop test after 1000 acquisitions
*(unsigned int*)(ADC_GCQSR) |= (1<<15); // clear FDRY interrupt
```

```
        *(unsigned int*)(ADC_GCQSR) |= (1<<2); // clear EOQ interrupt
        }
        *(unsigned int*)(AVIC_INTENNUM) = 46;   // re-enable adc interrupts
        }
```

In this example test, we set up a global array to store 1000 samples. This is only an example to show how to read the acquired ADC data; data can be managed in whatever manner is necessary.

4 References

The following documents provide further information. Freescale documentation is available at www.freescale.com and from the sources listed on the back page. The documentation numbers are included in parentheses for ease of ordering.

- Freescale application note, “ADC Definitions and Specifications” (AN2438)
- The TSC/ADC chapter in *i.MX25 Multimedia Applications Processor Reference Manual* (IMX25RM)

5 Revision History

Table 3 provides a revision history for this application note.

Table 3. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	09/2009	Initial release.
1	12/2009	Corrected Equation 1 and Equation 2. Removed empty BGA column from Table 2. Added ARM logo and copyright info.

THIS PAGE INTENTIONALLY LEFT BLANK

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
P.O. Box 5405
Denver, Colorado 80217
1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN3948
Rev. 1
12/2009