Freescale Semiconductor, Inc.
Application Note

Document Number: AN5101
Rev. 0, 04/2015

The High-Performance Data Acquisition Circuit

By Jan Tomecek

1. Introduction

Currently many applications use external high-performance sigma-delta analog to digital converters (SD ADC) to convert analogue signals to a digital representation. Amongst the main advantages of such high performance SD ADCs are 24-bit dynamic range of measurement, outstanding linearity and high signal-to-noise ratio (SNR). These devices are designed to measure and communicate raw measured quantities with other devices through either an SPI or an I^2C communication bus. They feature one to sixteen analogue input channels with optional programmable gain of the measurement.

Besides external SD ADCs also variety of microcontrollers is integrated with a 24-bit SD ADC. Such microcontrollers are designed to convert analogue signals into digital representation and to perform advanced digital signal processing via a single chip.

The Freescale Kinetis-M microcontroller series can be programmed to acquire analogue signals at various sampling rates, communicate measured values through a variety of communication buses and to emulate the functionalities of many popular 24-bit SD ADCs.

Contents

This application note describes the basic features of the Kinetis-M microcontroller and data acquisition firmware that transforms this highly integrated microcontroller into a four-channel, 24-bit SD ADC data acquisition circuit with SPI communication interface.

The Kinetis-M microcontrollers integrate up to four SD ADCs. These SD ADCs measure analogue signals within the range ±0.5 V with 92 dB SNR. As opposed to stand-alone, dedicated external SD ADCs, a microcontroller based solution allows other signal processing steps to be added into the firmware application. This allows the SNR of the integrated SD ADC within the microcontroller to be further improved by averaging measured values using a software low-pass filter.

The data acquisition firmware for Kinetis-M microcontrollers, described in the following sections, sets the output sample rate to 2.8 ksps and programs the SPI module to take output measurements in an LTC244x data format (see *Section 3.3.2 - LTC244x data format*.) If device pinout and measurement parameters are not so strict then the Kinetis-M microcontroller, programmed with the firmware covered in this document, can be used as an alternative to the LTC2440/5 devices [1]. (Moreover, the high-performance SD ADCs with maximum conversion rates up to 92 ksps, wide selection of communication interfaces including I^2C, SPI, UART, ARM® Cortex®-M0+ core and the tiny LGA 5 x 5 mm^2 package makes the Kinetis-M microcontrollers suitable for emulating many types of 24-bit SD ADCs – see *Figure 1*.

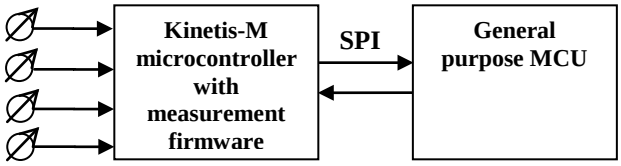


Figure 1. Block schematic of described firmware

2. Block diagram

Figure 2 shows the block diagram of the data acquisition firmware. This firmware consists of three essential parts for the application:

- **Measurement engine**
- **Filtering & Result conversion**
- **SPI communication**

The measurement engine performs analog-to-digital conversion of analogue input signals and basic signal processing. The filtering block averages measured data to further improve signal-to-noise ratio and the result conversion block converts smoothed data to the required output format. Finally, the SPI communication block sends smoothed data to the master device, a general purpose MCU or PC. In addition, the SPI communication block also receives commands from the master device, performs decoding commands and controls accordingly the operation of the measurement engine.

Figure 2. Block diagram of the data acquisition firmware

3. Implementation

This section describes the implementation of the data acquisition firmware. Although filtering and result conversion processes are carried out by one block (*Figure 2*), these processes are discussed in separate subsections due to their complexity. In addition, synchronization between all blocks of the data acquisition firmware is a key feature and is also described in a separate subsection.

3.1 Measurement engine

The measurement engine uses four 24-bit SD ADCs with true differential inputs to measure input analogue signals. There are several possibilities how to configure this peripheral which affects sampling frequency, measuring range, SNR and so on. The data acquisition firmware configures all four channels to operate in interrupt mode. This mode asserts an interrupt event when the conversion is completed. Because all channels are triggered simultaneously, only one interrupt service routine (ISR) and callback function is required to process data from all channels. As all channels are configured to operate in continuous mode, only the initial trigger is needed to initiate simultaneous conversions in the same period without extra startup delays. The programmable gain amplifier (PGA) for amplification of the input analogue signals is turned off. The oversampling ratio (OSR) is set to 2048. The system clock frequency for SD ADC is generated by the PLL module and it is 12.288 MHz. Due to the 6.5 MHz maximal SD ADC clock frequency, the system clock frequency generated by the PLL must be divided by two in the module's clock divider. The output sample rate in Hz is then given:

$$f_s = \frac{afe\_clock}{clock\_div} \cdot OSR\ [Hz] \qquad \text{Eq. 1}$$

And with mentioned values

$$f_s = \frac{12288000}{2} \cdot 2048 = 3000\ Hz \qquad \text{Eq. 2}$$

SD ADC ISR will occur every 333 µs after the actual voltage on the analogue input is sampled. The SNR is typically 92dB. More parameters of the measurement engine are described in the *Section 5 - Parameters* section of this document.

3.2 Filtering

To further improve the SNR measurement, the sampled raw data can be smoothed by a Finite Impulse Response (FIR) filter that is implemented in the software [2]. The FIR filter is a digital filter with a finite impulse response. For the discrete time FIR filter, the output signal y[n] of the discrete FIR filter is defined:

$$y[n] = \sum_{i=0}^{N} a[i] \cdot x[n-i] \qquad \text{Eq. 3}$$

Where x[n] is the input signal, N is the filter order and $a[i]$ is the *i*th value of the filter's impulse response.

3.2.1 Moving average filter

The moving average filter is the simplest form of a discrete FIR filter. Because all values of the impulse response are the same, and equal to 1/*N*, the output signal y[n] of the discrete moving average filter is defined:

$$y[n] = \frac{1}{N}\sum_{i=0}^{N} x[n-i] \qquad \text{Eq. 4}$$

The main goal of the moving average filter is to eliminate from the measured signal most of the frequency components above the measurement bandwidth. Therefore, one of the key factors is filter frequency response. *Figure 3* shows the magnitude responses of the moving average filters of lengths 8, 12, 16 taps and 3.0 kHz sampling frequency. From these magnitude responses, it is clear that characteristics of moving average filters are similar to low-pass filters.

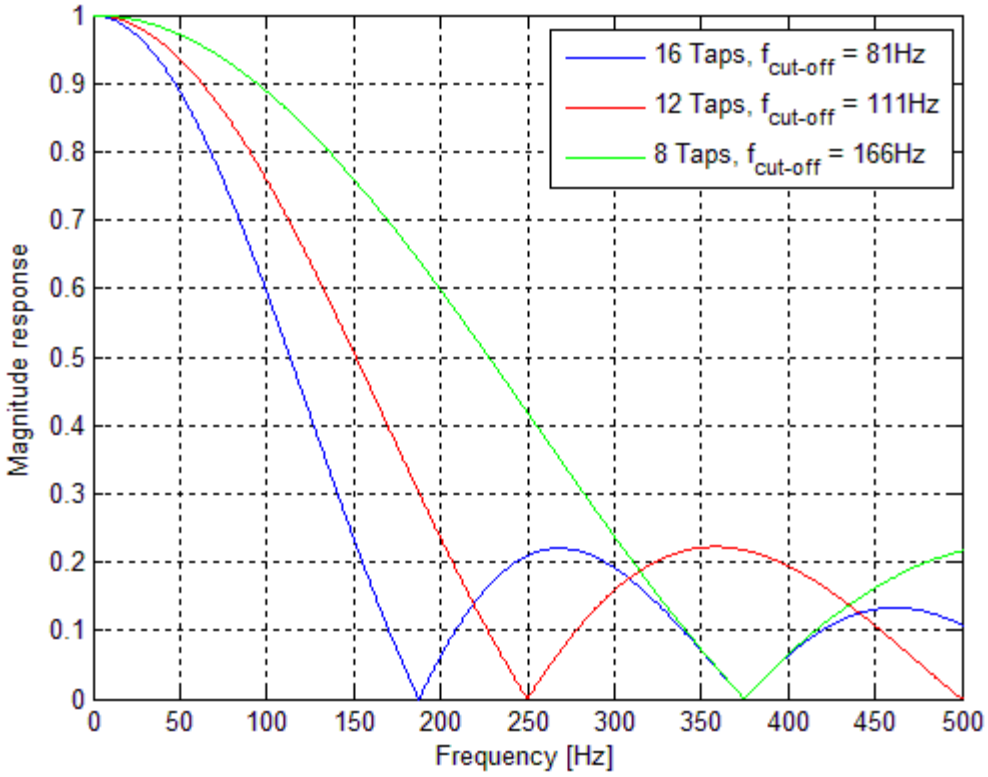


Figure 3. Magnitude frequency response of moving average filters

The magnitude frequency response of the moving average filter is calculated through the following equation:

$$H(\omega) = \frac{sin(\frac{\omega \cdot N}{2})}{N \cdot sin(\frac{\omega}{2})} \qquad \text{Eq. 5}$$

where $\omega = 2 \cdot \pi \cdot f/f_s$, f is input signal frequency and f_s is sampling frequency.

From this equation, the filter cut-off frequency (frequency with signal magnitude reduced by 3 dB) can be approximated for an arbitrary N as:

$$f_{-3dB} \approx \frac{1299}{N} \qquad \text{Eq. 6}$$

For example, when N = 16 (the same value is used in the data acquisition firmware) the cut-off frequency of the respective moving average filter is 81.187 Hz. For the sake of computation efficiency, it is recommended to select N as a power of two because the filter equation can then be computed as a trailing sum of samples with a single arithmetic shift instruction to divide by N.

3.3 Result conversion

This section describes the conversion between Kinetis-M and LTC244x data formats. The Kinetis-M data format relates to the numerical representation of measurements within the result registers of the SD ADC. The LTC244x data format is used by data acquisition firmware to represent measured data which is communicated to a host device through SPI bus.

Each SD ADC channel has its own 32-bit result register in the Kinetis-M microcontrollers – see *Figure 4*.

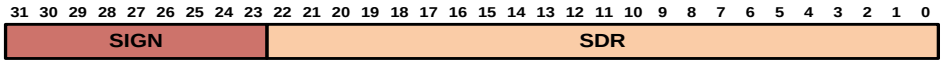


Figure 4. Kinetis-M - AFE result format (right justified)

In this figure, the SIGN part contains sign bits and the SDR part shows the absolute value of the sample data result. This arrangement is used when the right justified format of the result is selected. The range of the result is as follows:

$$-2^{(N-1)} \leq Integer \leq \left[2^{(N-1)} - 1\right] \qquad \text{Eq. 7}$$

The maximal positive value is 2,147,483,647 and the maximum negative value is -2,147,483,648.

3.3.1 Kinetis-M data format

Table 1 shows several examples of the left justified data format of the SD ADC result registers as implemented on the Kinetis-M microcontroller device. Note that bit 31 represents a sign, bit 30 is the most significant bit and bit 9 is the least significant bit of the digitized analogue value. Bits 8 to 0 are always zeroed.

Table 1. Kinetis-M result format

Measured voltage [V]	31	30	29	28	27	26	25	…	9	8	…	0
>0.5	0	1	1	1	1	1	1		1	0		0
0.5	0	1	1	1	1	1	1		0	0		0
0.25	0	1	0	0	0	0	0		0	0		0
0	0	0	0	0	0	0	0		0	0		0
-0.25	1	1	0	0	0	0	0		0	0		0

Table 1. Kinetis-M result format

-0.5	1	0	0	0	0	0	0		1	0		0
<-0.5	1	0	0	0	0	0	0	…	0	0		0

3.3.2 LTC244x data format

Table 2 shows several examples of the LTC244x data format. This format is slightly different in comparison to the Kinetis-M data format. It also has valid 23 SDR bits where bits 31 and 30 have no value (dummy), 29 bit represents a sign, bit 27 is the most significant bit and bit 5 is the least significant bit of the digitized analogue value. Bits 4 to 0 have no value and their states may vary.

NOTE

Bit 31 is the EOC (End of Conversion) bit in the LTC244x data format. When the data packet is read from the LTC2440/5 device with this bit de-asserted then the data packet contains the result from the last conversion. This bit is read asserted in all other cases.

Table 2. LTC result format

Measured voltage [V]	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**…**	**0**
>0.5	0	0	1	1	0	0	0		0
0.5	0	0	1	0	1	0	0		0
0.25	0	0	1	0	1	0	0		
0	0	0	1	0	0	0	0		0
-0.25	0	0	0	1	1	0	0		
-0.5	0	0	0	1	0	0	0		0
<-0.5	0	0	0	0	1	1	1	…	1

3.3.3 Conversion process

There are four result conversions implemented in the data acquisition firmware application for the Kinetis-M microcontroller device.

- **Positive saturation** – Positive saturation occurs when the measured voltage is higher than 0.5 V. If the left justified result format is being used, the value in the result register is 0x7FFFFF00. According to the LTC244x data format a value of 0x30000000 is needed. The following code carries out the necessary conversion from Kinetis-M data format to LTC244x data format:

Positive saturation conversion code

```
if(filt_res0 == 0x7FFFFF00)      /* Positive saturation                                    */
{
```

```
*((unsigned volatile long*)(adc_buff)) = 0x30000000;
return;
}
```

- **Negative saturation** – If the measured value is less than -0.5 V then negative saturation occurs. With left justified result format the value in the AFE result register is 0x80000000. According to the LTC244x data format a value of 0x0FFFFFFF is needed. The following code carries out the required conversion:

Negative saturation conversion code

```
else if(filt_res0 == 0x80000000) /* Negative saturation                                    */
{
*((unsigned volatile long*)(adc_buff)) = 0xFFFFFFF;
return;
}
```

- **Positive result** – For a positive, including zero, non-saturated conversion result you must carry out several steps. Firstly, the AFE result must be logical shifted to the right by two bits due to the LTC dummy bits on MSBs. Next the converted value is divided by two and the sign bit is set when the result is positive in the LTC244x data format. The remaining differences in sub resolution bits can be ignored. The described shift operation and divide operation can be done in one logical shift to the right by three.

Positive result conversion code

```
raw0 = (filt_res0 & ~(1<<31)); /* Store the filtered and shifted result value with cleared sign bit  */
if(!(filt_res0 & (1<<31)))     /* Positive result - Sign bit (of KM3x result format) is zero         */
{
raw_res0 = raw0>>3;           /* Do a shift and set the sign bit according to the LTC244x format     */
raw_res0 |= 1<<29;
}
```

- **Negative value** – If the measured value is less than zero and bigger than -0.5 V the result has to be converted by carrying out the following steps. Firstly, the result must be logical shifted to the right by two bits, due to the dummy bits of the LTC, and then the sign bit has to be cleared. Secondly, the offset value 0xFFFFFFF must be added to the value obtained through the previous step. The following code carries out the described conversion:

Negative result conversion code

```
else                          /*  Negative result - Sign bit (of KM3x result format) is one       */
{
raw0>>=3;                     /* Do a shift and convert value to the LTC244x format               */
raw_res0 = raw0 + (0xFFFFFFF);
}
```

3.4 SPI communication

This section describes communication between the Kinetis-M based data acquisition circuit and a host device. The host device will most-commonly be a general purpose microcontroller.

3.4.1 Communication interface

After the measured value is filtered, converted to the required format and stored in the memory the final step is to send the value to a host microcontroller device. For this purpose an SPI communication interface is used. The SPI communication interface uses five pins to transmit and receive data:

- **SPI MISO** – Master in, slave out data line
- **SPI MOSI** – Master out, slave in data line
- **SPI SPSCK** – SPI clock signal
- **SPI SS** – SPI slave select line
- **BUSY** – Slave busy bit line

3.4.2 Communication format

Although the SPI module of the Kinetis-M microcontroller is designed to operate in 8-bit and 16-bit modes, when the FIFO buffer is enabled the module can transfer up to 48-bit data without interruption via a clock signal from the master side. In the described firmware, 32-bit words are transferred, so the Kinetis-M device acts like a 32-bit SPI slave peripheral in this case.

NOTE

Only the SPI1 module supports the FIFO buffer.

Moreover, there is a possibility to keep the /SS signal in a permanent low state. However if the device is to work in this mode, the /SS signal cannot be used for transfer triggering. In this case communication has to be triggered by the SPSCK signal. The Kinetis-M SPI module allows triggering by SPSCK signal only if the CPHA mode is enabled (C1[CPHA] bit must be set). The CPHA mode causes the clock signal shift as shown in *Figure 5*. Simultaneously, if the C1[CPOL] bit is de-asserted a first rising edge on SPSCK line triggers the transfer. A falling edge then becomes an active edge.

NOTE

The CPHA mode must also be set on the SPI master for communication with the Kinetis-M device.

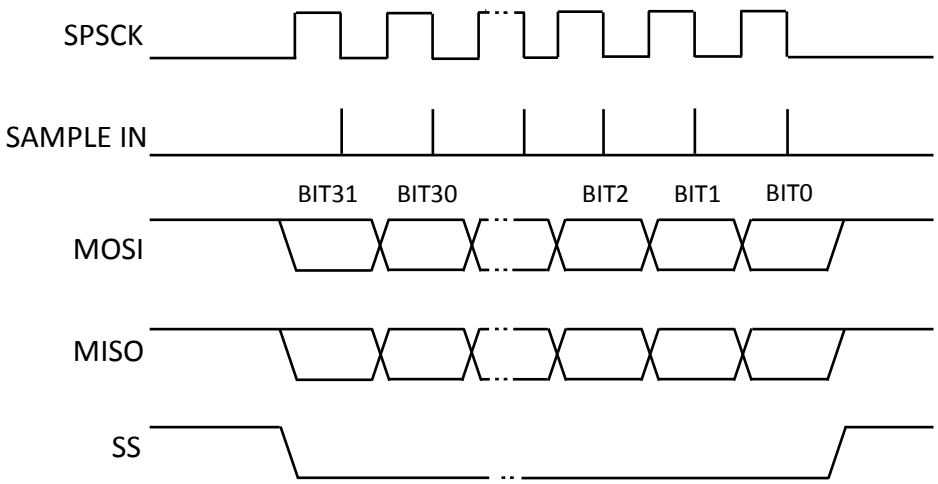


Figure 5. General SPI clock format when the CPHA bit is asserted

The next configuration of the SPI module is the following. The SPI FIFO is configured to *receive nearly full interrupt* mode. That means that interrupt occurs when the receive FIFO buffer reaches 32 bits and then the SPI ISR callback function is called. This is the same situation when the measured result was transferred to the master (from transmit FIFO buffer) and a new command from SPI master was received (into receive FIFO buffer), simultaneously.

In the SPI ISR callback function several steps are done. The busy bit is asserted as a first step and then the received command is stored from receive FIFO buffer to a memory. Next, the system timer (SysTick) is enabled and triggered (this step is described in more depth in the **Busy bit function** subsection). Finally, the SPI module is disabled causing all communication from the SPI master to be ignored. This step is signaled by the asserted busy bit.

The following SPI ISR callback function, represented by the *spi1_isr_optim* function in the source code, performs command decoding according to the steps described above.

Code for result storing and asserting the busy bit

```
GPIO_Set(FGPIOG, PIN0);              /* Assert the busy bit                                          */
if(SPI1_S & SPI_S_RNFULLF_MASK){     /* If the SPI RX FIFO buffer is near to full..                  */
  SPI_GetWordFromFifo(SPI1,(uint16*)spi_buff,2); /* ..store the received bytes to the spi_buff array  */
}
SYST_Enable();                       /* Enable and trigger system timer                              */
SPI1_C1 &= ~SPI_C1_SPE_MASK;         /* Disable SPI module                                           */
```

Figure 6 shows the communication format for transferring a command from an SPI master to an SPI slave device. Together with transferring a new command to the slave using the MOSI communication line, the last measured data are read by the master device from the MISO communication line.

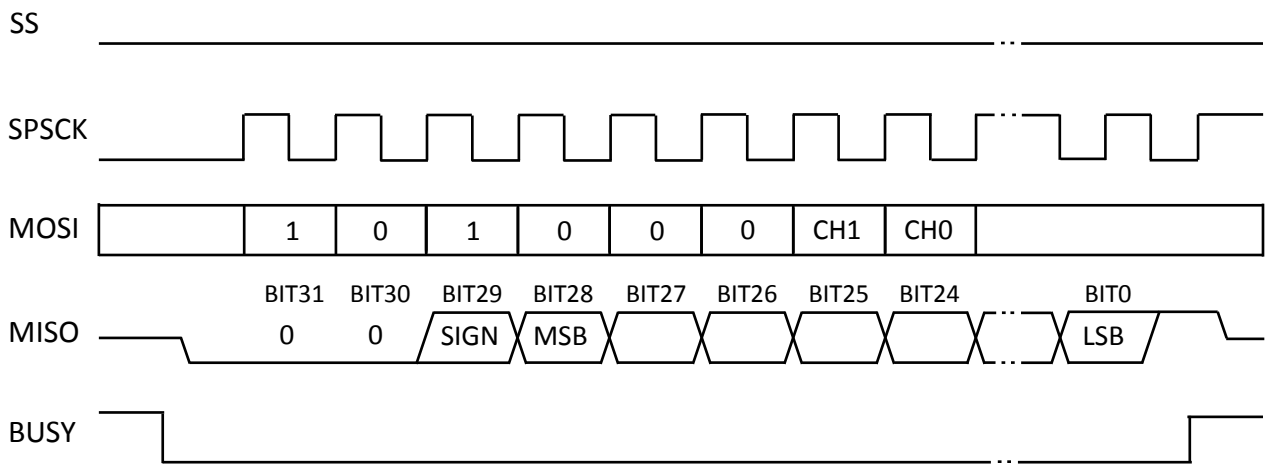


Figure 6. SPI MOSI command format

The bits 31 to 26 are fixed in the SPI master command. The measurement engine channel whose result will be read on the next communication transfer is selected based on decoding bits 25 (CH1) and 24 (CH0), see *Table 3*.

Table 3. Bits for channel selection in SPI master's command

CH1	CH0	Selected channel
0	0	Channel 0
0	1	Channel 1
1	0	Channel 2
1	1	Channel 3

3.4.3 Busy bit function

A function of the busy bit is to signal if a new measurement result is ready to transmit or not. If the busy bit is asserted the result is not yet ready. After a falling edge on this line occurs, the SPI master can start a new data transfer.

The busy bit does not only ensure time needed for filtering and format conversion of the measured result but it also ensures synchronization between conversions of the measurement engine and SPI communication. The timing details of the busy bit are as follows. The measurement engine sampling frequency is 3 kHz. In addition, time needed to smooth, format and store smoothed result is no more than 20µs. This means that the busy bit has to be asserted for a length of time according to the following equation:

$$t_{busy} \geq t_{sampling} + t_{processing}$$
$$t_{busy} \geq 333.\overline{3} \cdot 10^{-6} + 20 \cdot 10^{-6} \quad \text{Eq. 8}$$

The SysTick timer acts as timer for the busy bit. The SysTick timer is configured to generate an interrupt request whenever its counter reaches zero. The timer's counter is preloaded by value according to Eq. 9:

$$value = t_{busy} \cdot f_{core} + 1$$
$$value = 353.\overline{3} \cdot 10^{-6} \cdot 48 \cdot 10^{6} + 1 = 16801$$

Eq. 9

Initially, the timer is stopped until the software trigger occurs in SPI ISR callback function. At the same time the busy bit is asserted in this callback function. After the t_{busy} time the SysTick ISR callback function is called. There are several steps executed by the SysTick ISR callback function. At first the SPI IP module is enabled. Then the processed result from the desired channel is pushed to the SPI transmit FIFO buffer to wait for the next SPI communication. Next the timer is stopped and the timer's counter is cleared. At the end of the callback function the busy bit is de-asserted. The implementation is shown in the following source code.

Code for busy bit de-assertion and result store to the SPI FIFO transmit buffer

```
static void syst_callback(void)
  {
  SPI1_C1 |= SPI_C1_SPE_MASK;           /* SPI module enabling                                                    */
  if((spi_buff[0] & 0xFF00) == 0xA000){  /* If the SPI master device is requiring result from AFE's channel 0,
the converted result of this channel is stored into the SPI transmit FIFO buffer                                 */
  SPI_PutWord(SPI1,adc_buff[1]);
  SPI_PutWord(SPI1,adc_buff[0]);
   }
   if((spi_buff[0] & 0xFF00) == 0xA100){  /* If the SPI master device is requiring result from AFE's channel 1,
the converted result of this channel is stored into the SPI transmit FIFO buffer                                 */
   SPI_PutWord(SPI1,adc_buff[3]);
   SPI_PutWord(SPI1,adc_buff[2]);
   }
   if((spi_buff[0] & 0xFF00) == 0xA200){  /* If the SPI master device is requiring result from AFE's channel 2,
the converted result of this channel is stored into the SPI transmit FIFO buffer                                 */
       SPI_PutWord(SPI1,adc_buff[5]);

       SPI_PutWord(SPI1,adc_buff[4]);
   }
   if((spi_buff[0] & 0xFF00) == 0xA300){  /* If the SPI master device is requiring result from AFE's channel 3,
the converted result of this channel is stored into the SPI transmit FIFO buffer                                 */
       SPI_PutWord(SPI1,adc_buff[7]);
       SPI_PutWord(SPI1,adc_buff[6]);
   }
   SYST_Disable();                      /* Hold the SysTick timer                                                 */
   SYST_ClrCntrVal();                   /* Clear the SysTick timer's counter                                      */
   GPIO_Clr(FGPIOG, PIN0);              /* Clear the busy bit                                                     */
   }
```

3.4.3.1 Busy bit at the beginning of application

As was mentioned in *Section 3.2 - Filtering*, for averaging the measured values, the last N (relates to the FIR filter order) results from the previous measurements have to be stored in the memory. This requirement is not satisfied at the launch of the firmware. Therefore, at the launch of the firmware application the SPI communication module is disabled and the busy bit is asserted until N=16 values are collected and stored in the memory. This initial delay is controlled by SysTick timer which is configured according to the following formula:

$$t_{busy_{first}} \geq 16 \cdot t_{sampling} + t_{processing}$$
$$t_{busy_{first}} \geq 16 \cdot 333.\overline{3} \cdot 10^{-6} + 20 \cdot 10^{-6}$$

Eq. 10

The first value read from the Kinetis-M device is a dummy word of value 0x80000000.

3.5 Application structure and timing

The application contains three main parts whose operations have to be synchronized.

- **Measurement & result processing** – The first part is included in the AFE ISR callback function (*afe_ch1_callback*). This part carries out result conversion, result filtering and result storing. As was mentioned previously in *Section 3.1 - Measurement engine*, the AFE ISR callback is called with a frequency of 3 kHz and the callback duration is less than 20 us.
- **Communication** – The second part is handling of the SPI communication with the host device. Because the Kinetis-M device acts like an SPI slave in the described firmware, the communication must be synchronized with measurements. When the host device begins communication, the SPI ISR callback function (*spi1_isr_optim*) is called which signals that the measured and formatted results have been automatically sent to the host device from the SPI transmit FIFO buffer. Simultaneously, a command from the host device containing information about the next measurement has been stored in the SPI receive FIFO buffer. At the end of this callback function the busy bit is asserted and the SPI module is disabled.
- **Busy bit** – The main functionality of the third part is the busy bit de-assertion. This functionality is processed by the SysTick timer ISR callback function (*syst_callback*). When the busy bit is de-asserted, the SPI module is enabled and the result from the measurement & result processing part is stored in the SPI transmit FIFO buffer. Afterwards, the application firmware is ready to transmit the measured data and to receive the next measurement command.

The *timing diagram for the application* is shown in appendix I.

4. Test Results

The KM34Z TWR board has been used during testing. The board configuration is shown in *Figure 7*.

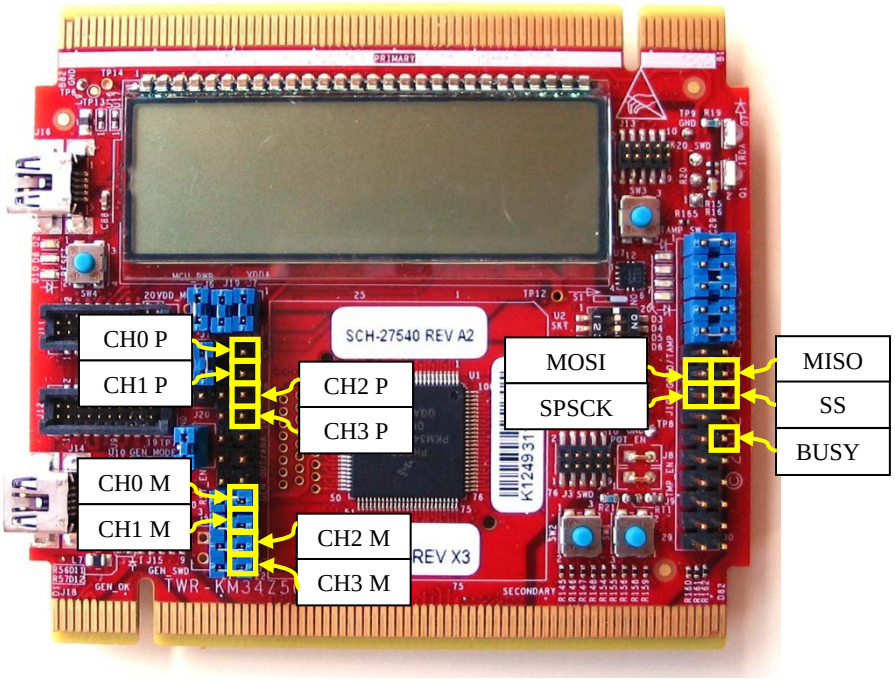


Figure 7. KM34 TWR board – pin connection

NOTE

Please ignore the jumpers on the CH0 M, CH1 M CH2 M and CH3 M pins.

4.1 Communication with the Kinetis-M device

For communication with the Kinetis-M device an arbitrary host device supporting SPI interface can be used. Although the length of data packet of the Kinetis-M device with described firmware is 32 bits, data transfer may also be split into 2 x 16 bits or 4 x 8 bits packets. This is useful when the SPI peripheral on the host side does not support 32-bit mode. The following figures show SPI signals for various tested communication modes.

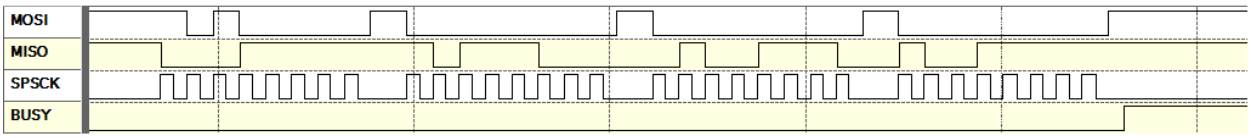


Figure 8. SPI communication when host SPI is in 8-bit mode

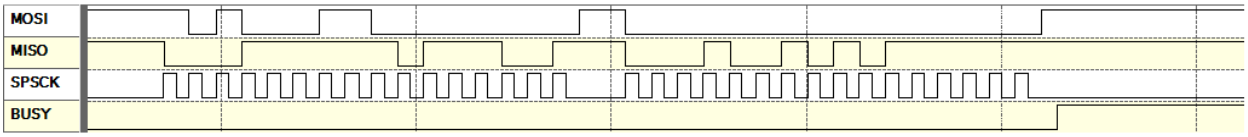


Figure 9. SPI communication when host SPI is in 16-bit mode

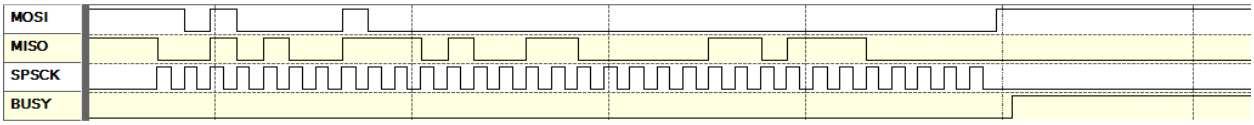


Figure 10. SPI communication when host SPI is in 32-bit mode

Due to software implementation there is a time delay between the last falling edge of the SPSCK signal and the rising edge on the busy line. This time is about 700 ns when the 48 MHz core clock is selected.

4.2 Visualization of the received data

Figure 11 shows a graphical representation of measurements received from the Kinetis-M device that has been programmed with firmware emulating popular LTC2440/5 measurement devices.

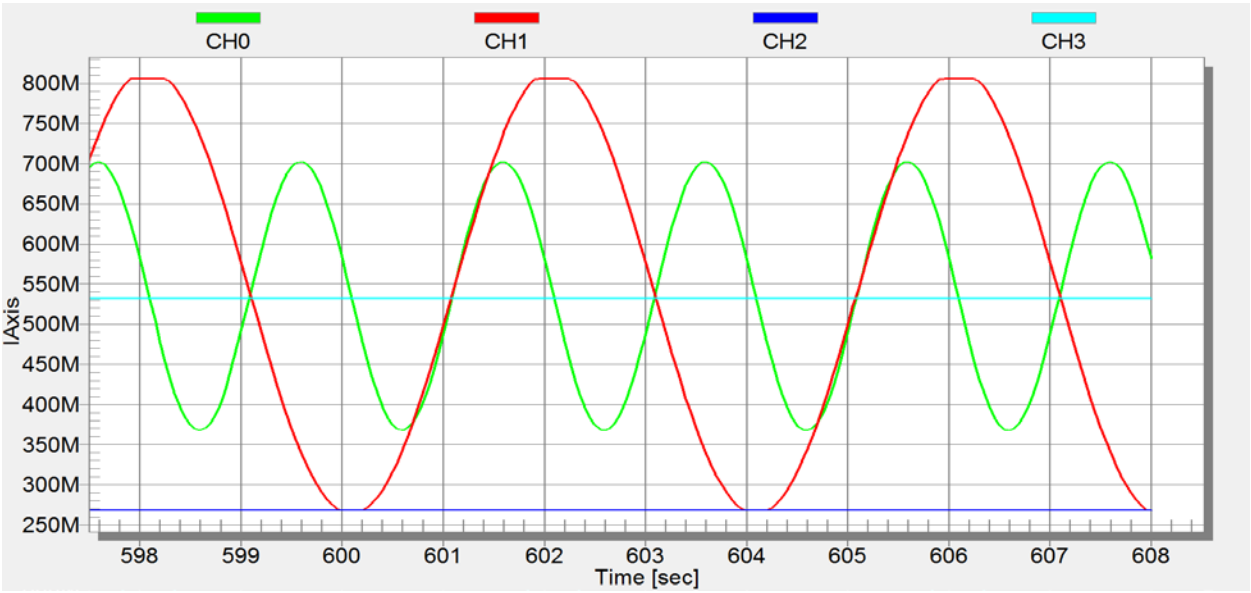


Figure 11. Received data on SPI master side shown by FreeMASTER tool

Another KM34Z50 TWR board was used as a host side during this test. The SPI peripheral was configured in 16-bit master mode without a FIFO buffer. You must set the CPHA mode of SPI communication on both boards. On this host side the following code for data acquisition was used.

Part of the code for the SPI master device for data receiving from the Kinetis-M device

```
if(msgFlag == 0)
      {
       spi_buff[0] = SPI_TxRxWord(SPI1, spi_data0[0]);
       spi_buff[1] = SPI_TxRxWord(SPI1, spi_data0[1]);
       result_chn3 = (spi_buff[0]<<16)|spi_buff[1];
      }
else if(msgFlag == 1)
      {
       spi_buff[0] = SPI_TxRxWord(SPI1, spi_data1[0]);
       spi_buff[1] = SPI_TxRxWord(SPI1, spi_data1[1]);
       result_chn0 = (spi_buff[0]<<16)|spi_buff[1];
```

```
      }
else if(msgFlag == 2){
       spi_buff[0] = SPI_TxRxWord(SPI1, spi_data2[0]);
       spi_buff[1] = SPI_TxRxWord(SPI1, spi_data2[1]);
       result_chn1 = (spi_buff[0]<<16)|spi_buff[1];
      }
else if(msgFlag == 3){
       spi_buff[0] = SPI_TxRxWord(SPI1, spi_data3[0]);
       spi_buff[1] = SPI_TxRxWord(SPI1, spi_data3[1]);
       result_chn2 = (spi_buff[0]<<16)|spi_buff[1];
      }
else if(msgFlag == 4){
        SPI_PutWordToFifo(SPI1,spi_data0,2);
      }
```

5. Parameters

Table 4 lists the most common parameters of the Kinetis-M device with the described application firmware.

Table 4. Parameters of Kinetis-M device with described firmware

Parameter	Value	Unit
OSR	2048	-
ENOB	15.3	bits
Output data rate	2.8	kHz
SNR	92	dB
SINAD	78	dB
CMMR	70	dB
PSRR	60	dB
Input range	+/-0.5	mV

The parameters (OSR, ENOB and so on) in this table are minimum values as outlined in the Kinetis-M device datasheet [6]. These parameter values are further improved by using the moving average filter described in *Section 3.2 - Filtering*. For example, when the filter has 16-tap the ENOB value will increase by approximately 2 bits.

6. Summary

This application note describes data acquisition firmware for Kinetis-M microcontrollers. With this firmware the Kinetis-M device can emulate a four-channel sigma-delta analogue-to-digital converter. The Kinetis-M device communicates with other devices via a SPI bus. Therefore, an arbitrary device supporting this bus could be used as a host device. This communication interface uses a 32-bit data format and specific commands for measurement channel selection, which is very similar to an LTC244x ADC interface. The output data format is also similar to an LTC244x ADC interface. The Kinetis-M

device with this firmware acts like a slave data acquisition module with 32-bit SPI communication. This functionality is possible due to the SPI FIFO mode supported by the device's SPI1 module.

The Kinetis-M device measures analogue input signals via four SD ADCs. In the described application, these SD ADCs are clocked by a very accurate low jitter PLL module. To achieve the best SNR an OSR value of 2048 is selected. In this case the sampling frequency of the measurement engine is 3 kHz for each channel. The maximal output data rate of the device is 2.8 ksps.

To optimize the SNR, measured data is additionally filtered with a 16-tap software moving average filter. The measured results are filtered after every conversion. Additionally, the result from the channel, which will be sent on the next transfer, is converted to the LTC244x data output format. These filtering and converting operations take about 20µs @ 48 MHz core clock. Because the Kinetis-M device is a slave device in the described application the host device may start data transfer independently and therefore operations have to be synchronized. Due to this the functionality of busy bit is integrated in the firmware. When the busy bit is asserted the result is not yet ready and all communication from the side of the master is ignored. For the best data throughput, the host device should check the busy line and start the communication when the busy bit is de-asserted.

As shown in this application note, the data acquisition circuit based on the Kinetis-M microcontroller series can be successfully used to emulate many popular, high-performance 24-bit SD ADCs. Characteristics of emulated ADCs can be further improved by additional signal processing. For example, the firmware described in this application note transforms the Kinetis-M microcontroller into a full-featured data acquisition circuit that communicates with other devices using SPI communication and data format compliant with the LTC244x ADCs.

7. References

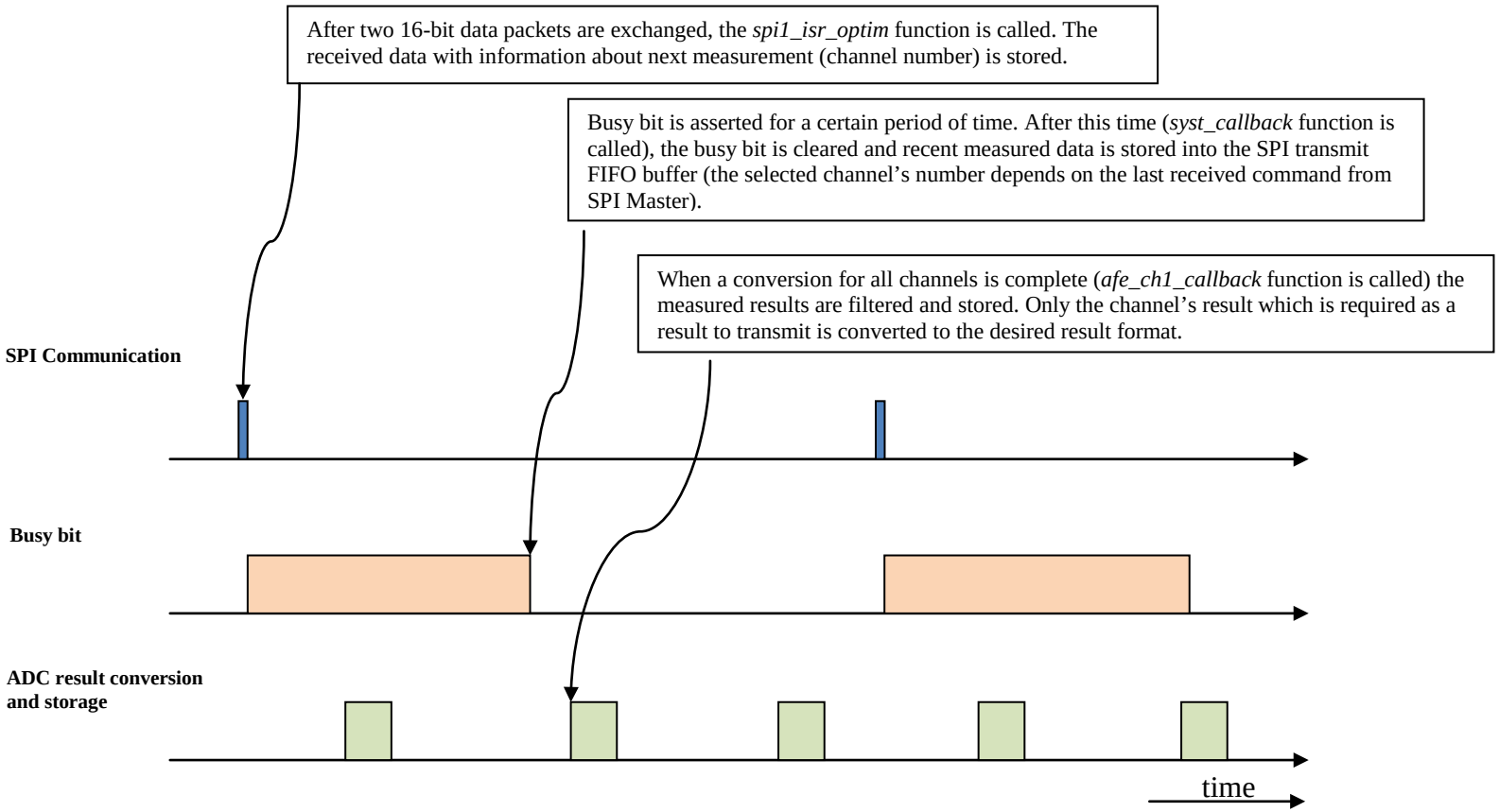
The following documents are useful when using the Kinetis-M High Speed Data Acquisition Firmware.

1. *LTC2440 Datasheet*, cds.linear.com/docs/en/datasheet/2440fd.pdf
2. *Handbook for Digital Signal Processing*, Sanjit K. Mitra, James F. Kaiser (John Wiley & Sons, 1993, USA)
3. *Circular buffer*, en.wikipedia.org/wiki/Circular_buffer
4. *Frequency response of FIR filters*, eas.uccs.edu/wickert/ece2610/lecture_notes/ece2610_chap6.pdf

The following documents can be found on freescale.com. Additional documents not listed here can be found on the Kinetis-M Series product page.

5. *Kinetis-M Reference Manual*
6. *Kinetis-M Datasheet*

Appendix I. Timing Diagram



8. Revision history

Table 5. Revision History

Revision number	Date	Substantive changes
0	04/2015	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Document Number: AN5101
Rev 0
04/2015