Freescale Semiconductor
Application Note

Document Number: AN4590
Rev. 0, 9/2012

Using DMA to Emulate ADC Flexible Scan Mode on Kinetis K Series

by: **Lukas Vaculik**
Rožnov pod Radhoštem
Czech Republic

Contents

1 Introduction

The Kinetis K series of microcontrollers offers an analog-to-digital controller (ADC) that supports up to two input channels through its built-in Scan mode. The channel number is defined by the ADCx_SCA and ADCx_SCB registers, and the conversion results are available, after conversion, in the ADCx_RA and ADCx_RB registers. The Kinetis K series of microcontrollers also offers a powerful and complex DMA periphery (with up to 16 channels) that can be combined with the ADC to allow the scanning of more than two channels. This application note describes how to combine the ADC and DMA into a powerful and flexible periphery supplying a resultant data stream into SRAM from any ADC input.

2 Implementation detail

2.1 DMA transfer terminology

The following terms are used in this application note's discussion of DMA transfers:

- In a *minor loop*, one transfer group is started for each request. A minor loop can transfer from 1 to 4 GB of basic units. The basic transfer unit is 8 bits.So, for 16-bit data, one minor loop supports two transfers.

- A *major loop* chains multiple minor loops together. A major loop can modify source and destination addresses; it can also support a circular buffer mode after the major loop has finished. The end of a major loop transfer can be handled asynchronously by an interrupt request. A half transfer finish can also be handled by an interrupt. A combination of half-transfer and full-transfer finishes can be implemented using a double-buffer principle.
- *Linking* is a special eDMA feature for chaining more than one DMA channel. Linking channels allows you to start more than one transfer with a single request by defining a sequence of channels to be converted. The request starts the transfer on one DMA channel, and when that channel finishes, the transfer on the next channel starts. Channel linking is defined separately for major and minor loop finishes.

2.2 Flexible Scan mode process

ADC Flexible Scan mode requires two DMA channels for one ADC converter. DMA channel 1, with a higher priority, transfers the resultant ADC data from the ADCx_RA register to a memory buffer in the SRAM. DMA channel 0, with a lower priority, transfers a future ADC channel setting (input multiplexer channel and single end/differential mode) from the constant buffer, which could be placed in SRAM or flash memory. The following steps occur in Flexible Scan mode:

1. The conversion complete flag ADCxSC1A.COCO requests a DMA transfer for channel 1.
2. The channel 1 transfer finishes and the resulting value is transferred to the SRAM buffer.
3. Since channel 1 is linked to channel 0, the channel 1 finish requests a transfer start on channel 0.

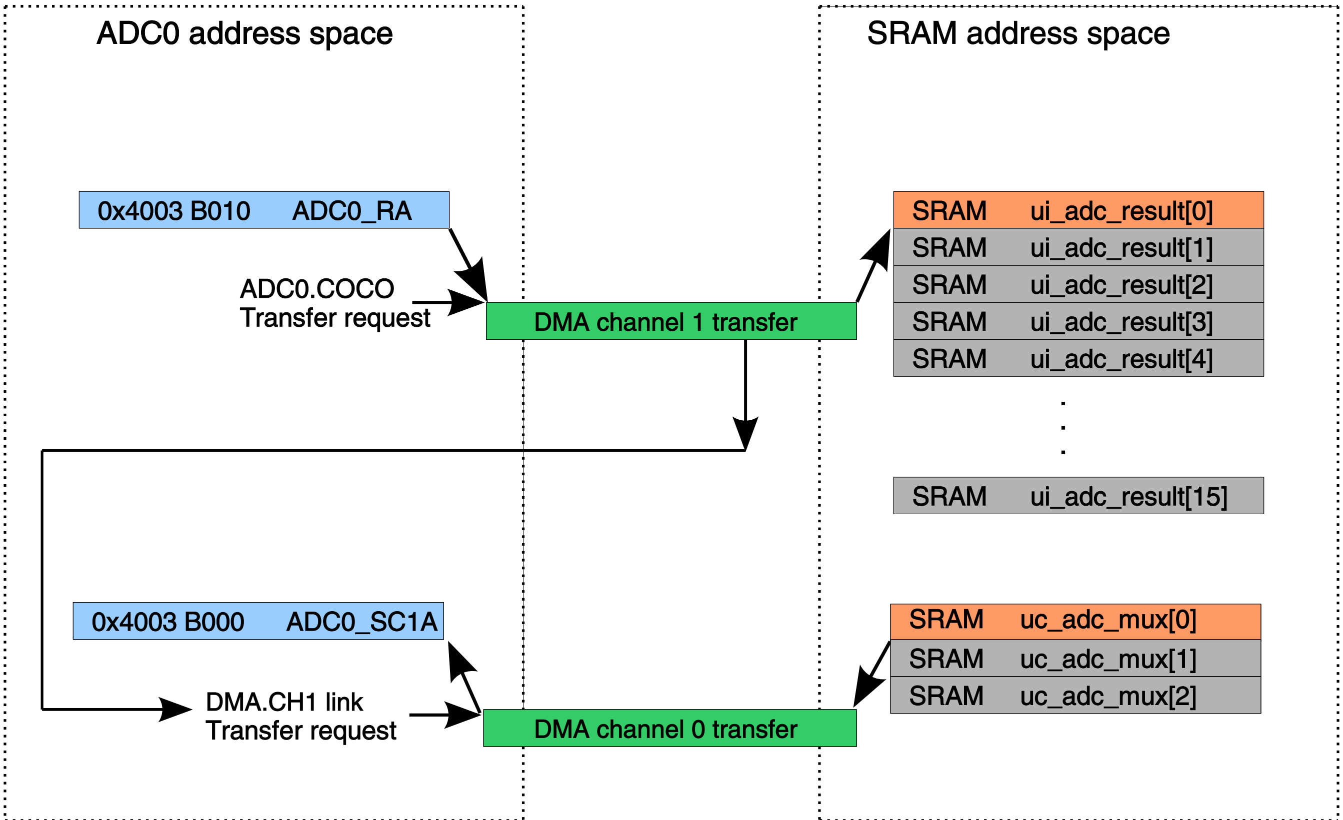

Figure 1. Data flow between periphery and SRAM

2.3 DMA special settings

2.3.1 Channel priority

DMA channel priority settings are important because in ADC software trigger mode, conversion is started by a write to ADCn_SC1A; thus, the resultant data must be read first and then followed by a write of the next channel setting, because writing to the ADC starts the next conversion. DMA channel 1 is used to transfer ADC result data to the SRAM buffer. DMA channel 0 is used to change the ADC input mux.

2.3.2 Major and minor loop linking

Single scan mode (only one measurement from some ADC channels is required) should use only minor loop linking. This is defined in the DMA_TCDx_BITER_ELINKNO and DMA_TCDx_CITER_ELINKNO registers.

Continuous scan mode (circular measurement from some ADC channels is required) needs to use both major and minor loop linking. This is defined in the DMA_TCDx_BITER_ELINKNO, DMA_TCDx_CITER_ELINKNO, and DMA_TCD1_CSR registers. It is necessary for continuous scan mode to use both major and minor loop linking because there is no generation of a minor loop finish request after a major loop finish.

3 Example

This example uses the Freescale TRW-K60N512 development board and IAR Embedded Workbench® version 6.30.1.3142 as its test environment.

3.1 Example process flow

The example code that accompanies this application note demonstrates a continuous scan conversion from three ADC channels.

- Each channel is measured four times, so the SRAM result buffer size is $3 \times 4 = 12$ (the real buffer size is 16, to demonstrate that only 12 data field parts are written).
- The ADC works in hardware trigger mode, with the PDB timer serving as the trigger source.
- Scanning is executed in continuous mode; thus, after a major loop has finished, the result buffer pointer address_0 is reloaded and the conversion begins again from the start buffer address.

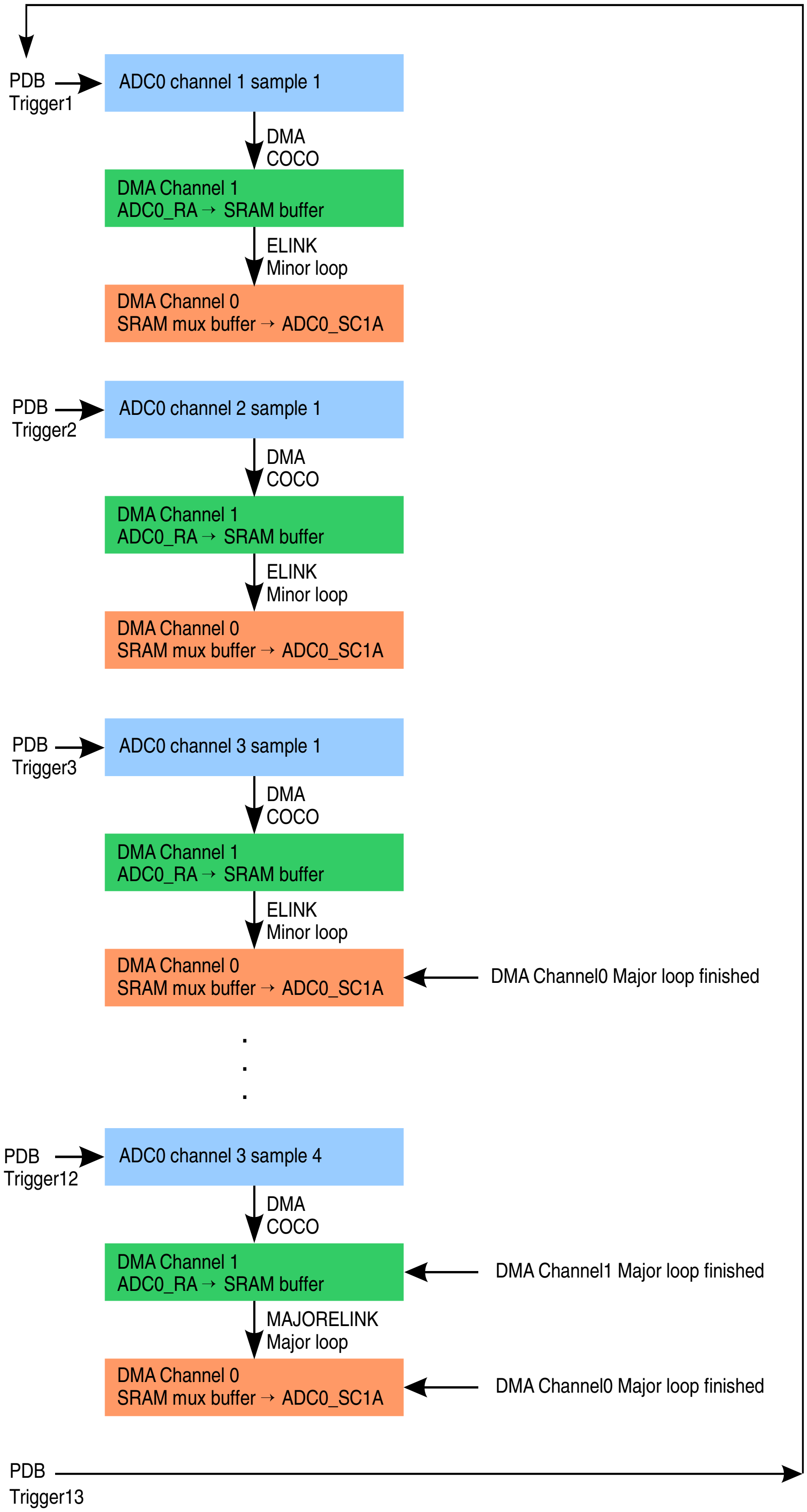

Figure 2. Example process flow

3.2 Functions in the main file

SIM_Init	Initialize clocks and oscillator
FLL_Init	FLL periphery initialization
VREF_Init	Voltage reference initialization
PDBCH0TRG0_Init	PDB channel 0 initialization for hardware trigger ADC0
PDB_Init	Common PDB initialization
ADC_ExecCalib	Internal calibration procedure for ADC0
ADC_Init	ADC0 initialization
DMACH0_CH1_init	DMA channel 0 and channel 1 initialization

3.3 DMA channel initialization

Channel 1 transfers ADC0 result data from ADC0_RA to SRAM buffer.

```
//*****************************************************************
//**** DMA transfer request source – ADC0 conversion complete
//*****************************************************************
DMAMUX_CHCFG1           = DMAMUX _CHCFG_ENBL|DMAMUX_CHCFG_SOURCE(28);
//*****************************************************************
//**** Source address, ADC0_RA
//*****************************************************************
DMA_TCD1_SADDR          = (uint32) &ADC0_RA;
//*****************************************************************
//**** Source address increment; data is still read for the same address, no increment needed
//*****************************************************************
DMA_TDC1_SOFF           = 0x00;

//*****************************************************************
//**** Source address reload after major loop finishes, no reload needed
//*****************************************************************
DMA_TDC1_SLAST          = 0x00;
//*****************************************************************
//**** Destination address, SRAM buffer [0]
//*****************************************************************
DMA_TDC1_DADDR          = (uint32)&ui_adc_result[0];
//*****************************************************************
//**** Destination address increment in bytes, increment for next buffer address
//**** 16 bit => 2 bytes
//*****************************************************************
DMA_TDC1_DOFF           = 0x02;
//*****************************************************************
//**** Destination address reload after major loop finishes,
//**** must be subtracted from last pointer value, sample number is 12 each and 2 bytes long,
//**** 2 × 12 = 24 and must be subtract -24
//*****************************************************************
DMA_TDC1_DLASTSGA        = -24;
//*****************************************************************
//**** Number of bytes for minor loop (one data transfer), ADC0 result is 16 bits long, so
//**** 2-byte transfer
//*****************************************************************
DMA_TDC1_NBYTES_MLO      = 0x02;
//*****************************************************************
//**** Channel linking and major loop setting, linking after minor loop is enabled to
//**** channel 0 (0x0000), major loop transfers number 12 (0x0C)
```

```
//*************************************************************
DMA_TDC1_BITER_ELONKNO  =  (DMA_BITER_ELINKNO_ELINK_MASK|0x0000|0x0C);

//*************************************************************
//**** Channel linking and major loop setting reload value after major loop finishes,
//**** linking after minor loop is enabled, major loop transfers number 12 (0x0C).
//*************************************************************
DMA_TDC1_CITER_ELONKNO  =  (DMA_CITER_ELINKNO_ELINK_MASK|0x0C);
//*************************************************************
//**** Source and destination data width specification, both source and destination is 16-bit
//*************************************************************
DMA_TDC1_ATTR            =  DMA_ATTR_SSIZE(1)| DMA_ATTR_SDIZE(1);
//*************************************************************
//**** Common channel setting, linking after major loop enable to channel 0,
//**** IRQ request is generated after major loop complete
//*************************************************************
DMA_TDC1_CSR             = (DMA_CSR_MAJORLINKCH(0)|DMA_CSR_MAJORLINCH_MASK|
                            DMA_CSR_INTMAJOR_MASK);


Channel 0 transfers next ADC0 input setting from constant buffer to ADC0_SC1A.

//*************************************************************
//**** DMA transfer request source – always requestor
//*************************************************************
DMAMUX_CHCFG0            = DMAMUX _CHCFG_ENBL|DMAMUX_CHCFG_SOURCE(36) ;
//*************************************************************
//**** Source address, constant buffer in SRAM
//*************************************************************
DMA_TCD1_SADDR           = (uint32) &uc_adc_mux[0];
//*************************************************************
//**** Source address increment, data is 8-bit, 1 byte
//*************************************************************
DMA_TDC1_SOFF            = 0x01;
//*************************************************************
//**** Source address reload after major loop finish, must be subtracted from last
//**** pointer value, sampling channel number is 3 each and 1 byte long, 1 × 3 = 3
//**** and must be subtract -3
//*************************************************************
DMA_TDC1_SLAST           = -3;
//*************************************************************
//**** Destination address, ADC0 control register
//*************************************************************
DMA_TDC1_DADDR           = (uint32)&ADC0_SC1A;
//*************************************************************
//**** Destination address increment in bytes, no increment needed
//*************************************************************
DMA_TDC1_DOFF            = 0x00;

//*************************************************************
//**** Destination address reload after major loop finish, no address reload needed
//*************************************************************
DMA_TDC1_DLASTSGA        = 0x00;

//*************************************************************
//**** Number of bytes for minor loop (one data transfer), ADC0 input setting value is
//**** 8 bits long, so 1-byte transfer
//*************************************************************
DMA_TDC1_NBYTES_MLO      = 0x01;

//*************************************************************
//**** Channel linking and major loop setting, no linking after minor loop,
//**** major loop transfers number 0x03
//*************************************************************
DMA_TDC1_BITER_ELONKNO  = (DMA_BITER_ELINKNO_ELINK_MASK|0x0000|0x0C);

//*************************************************************
//**** Channel linking and major loop setting reload value after major loop finish,
//**** no linking after minor loop, major loop transfers number 0x03
//*************************************************************
```

```
DMA_TDC1_CITER_ELONKNO  = (DMA_CITER_ELINKNO_ELINK_MASK|0x0C);

//***************************************************************
//**** Source and destination data width specification, both source and destination are 8-bit
//***************************************************************
DMA_TDC1_ATTR           = DMA_ATTR_SSIZE(1)| DMA_ATTR_SDIZE(1);

//***************************************************************
//**** Common channel setting, no linking after major loop, no IRQ request enable
//***************************************************************
DMA_TDC1_CSR            = 0x00;
```

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor
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

Document Number: AN4590
Rev. 0, 9/2012