

Agenda

1. Module Overview
2. On-chip Interconnections and Inter-module Dependencies
3. Hardware Configuration
4. Software Configuration
5. Example Use Case
6. ADC Frequently Asked Questions (FAQs)

 External Use | 1 

The agenda for this training includes:

- A module overview
- The on-chip interconnections and inter-module dependencies
- Hardware and software configurations
- An example use case for reference
- And a few frequently asked questions



Let's first begin with an overview of the module.

16-Bit SAR ADC Module Features and Application Benefits

Features

- Differential or single-ended conversions
- Programmable gain amplifier
- Hardware average and hardware automatic compare

Application Benefits

- 16-bit accuracy at low-power operation
- Flexible scan using DMA connection



External Use | 3

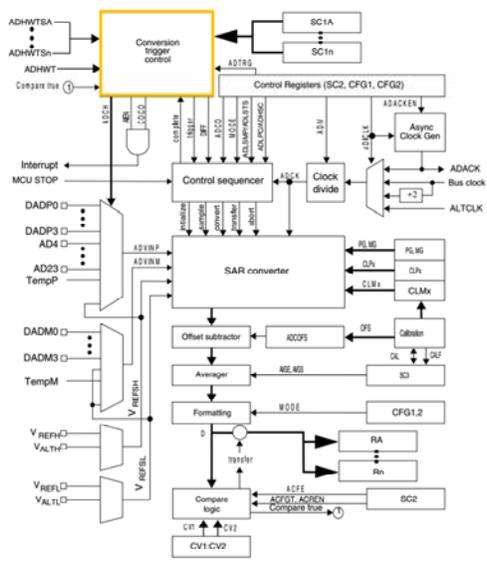


16-Bit SAR ADC Module Features and Application Benefits

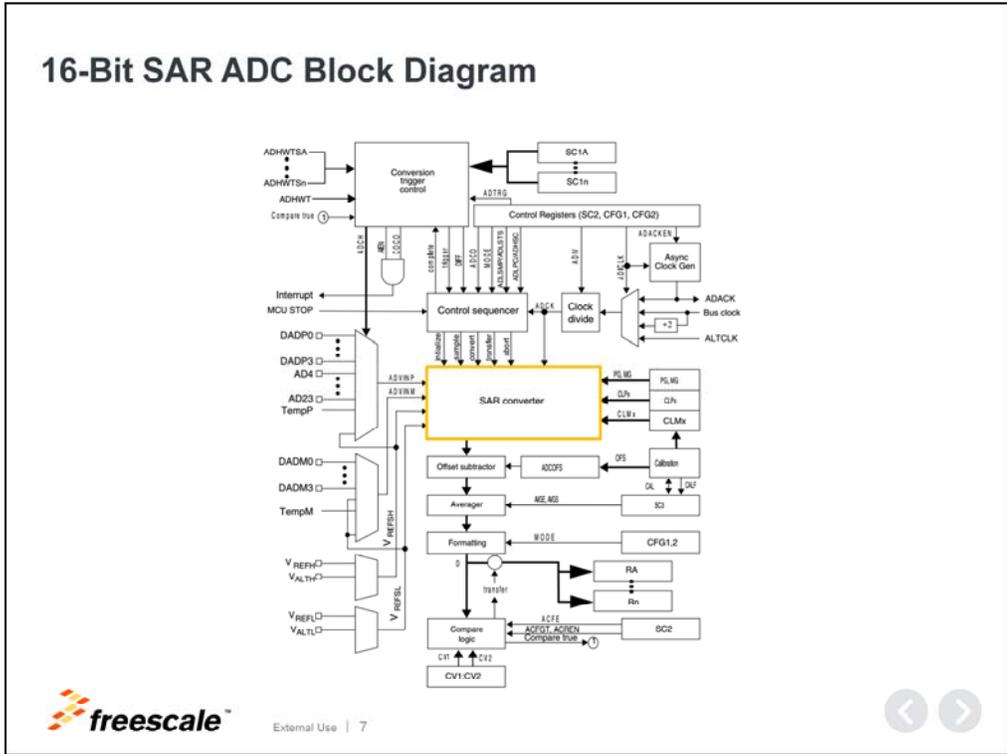
Some of the features of the 16-bit SAR ADC module include: Differential or single-ended conversions. Select input channels have the capability to increase the dynamic range up to 64 times through a programmable gain amplifier. The ADC also comes with a configurable hardware average feature for up to 32 samples, and a hardware automatic compare.

The integration of the ADC allows for some key application benefits. For example, the ADC can sample an input while it is in a low-power mode such as VLPS. Another example is the ability to combine the ADC hardware trigger and the conversion complete DMA channel so that an application can periodically sample an input without CPU intervention.

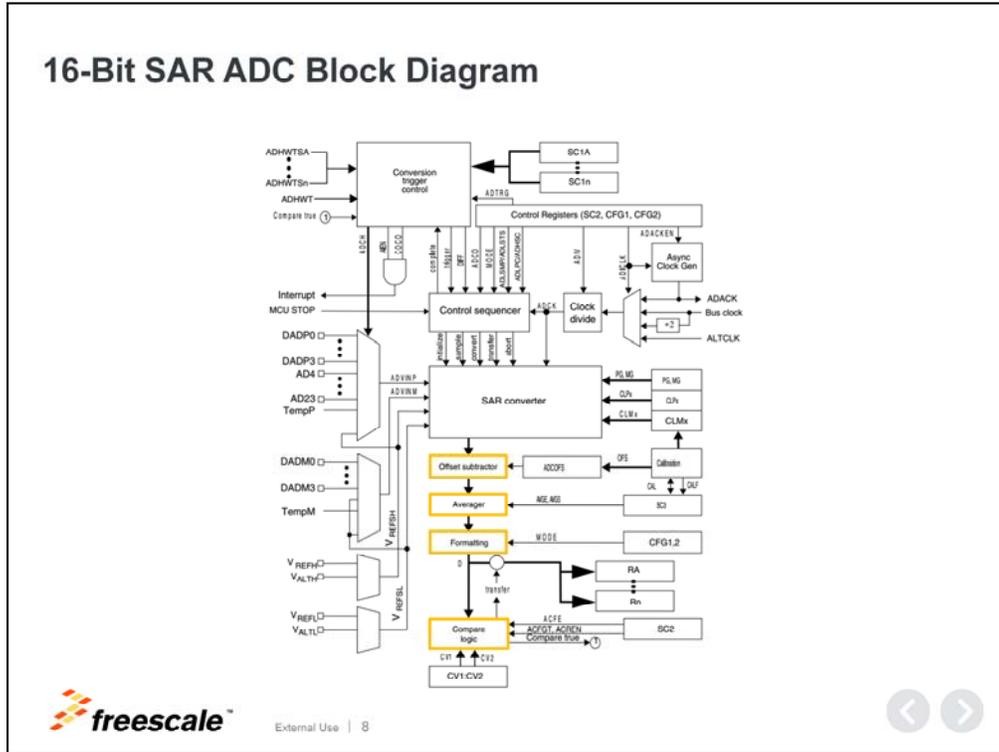
16-Bit SAR ADC Block Diagram



The conversion trigger control handles starting ADC conversion either through software or hardware trigger.



The SAR converter is the heart of the ADC. In this block diagram, the ADC analog input is converted into a digital value.



After the SAR converter provides a result, the digital value passes through an offset subtractor, averager, formatting and compare logic; the ADC conversion complete flag, or COCO, is now set.

ADC Channels

Single ended: 16-bit, 12-bit, 10-bit, and 8-bit modes

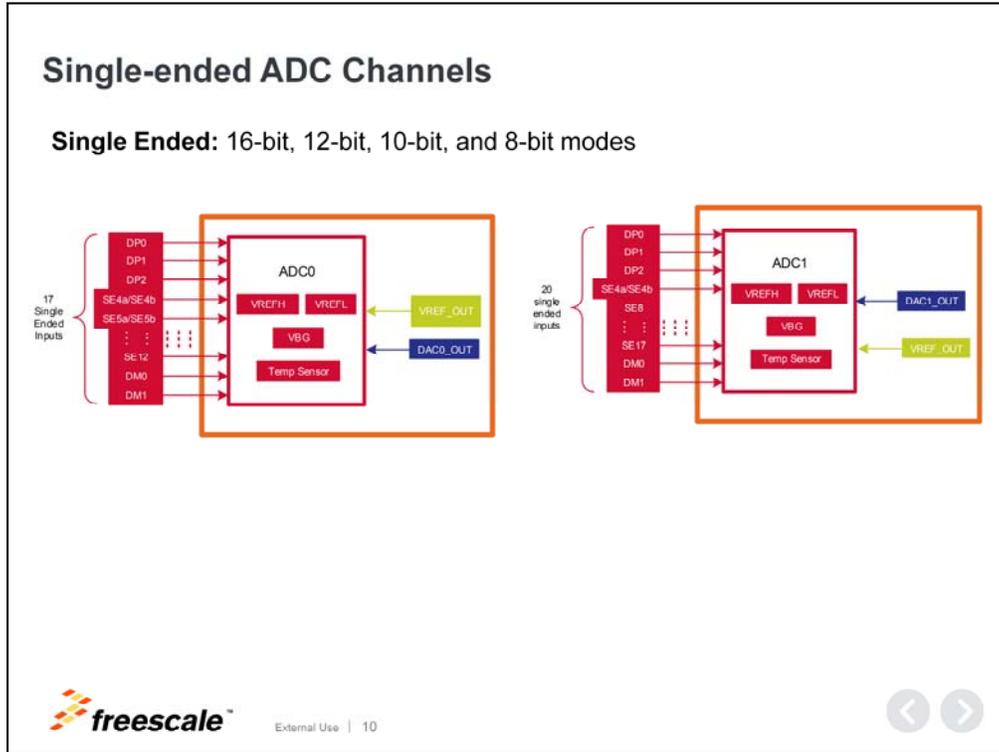
Differential: 16-bit, 13-bit, 11-bit and 9-bit modes

 External Use | 9 

ADC Channels

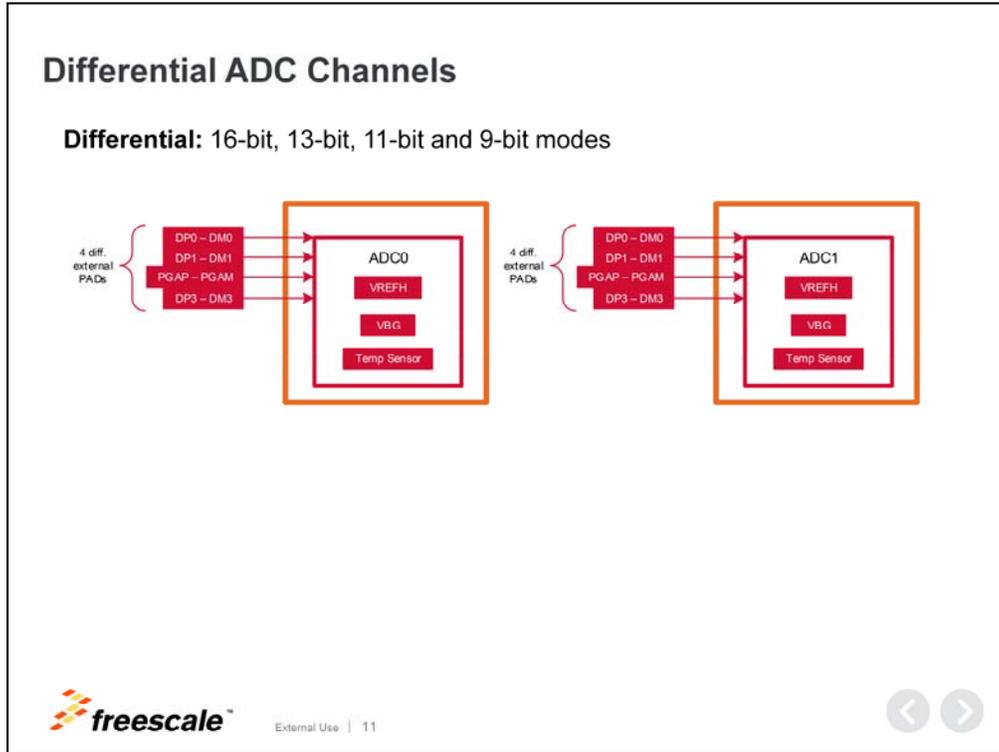
The ADCs support both single-ended and differential channels. To understand which channels are single-ended or differential, refer to your Kinetis MCU Reference Manual.

A feature of this ADC, related to channels, is that the SAR resolution can be software configurable, so even when you have a 16-bit SAR, you can obtain 12, 10 or 8-bit resolutions.



Single-ended ADC Channels

For single-ended channels the possible resolutions can be 16, 12, 10, or 8-bit mode. Single-ended channels have internal inputs like DAC outputs, VREF, or VBG – also known as bandgap. The bandgap is an internal 1.2 volts that can be used to measure battery levels. Single-ended inputs can also come from external pins.

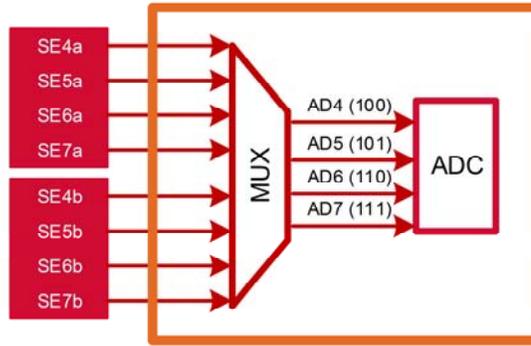


Differential ADC Channels

Differential channels also have internal or external inputs. Differential channel resolutions may be configured to 16, 13, 11, or 9-bit mode. Resolutions smaller than 16-bit add one resolution bit from single-ended to use for the sign. Differential channels have rail to rail inputs. For more information on levels and performance, please refer to your Kinetis MCU datasheet.

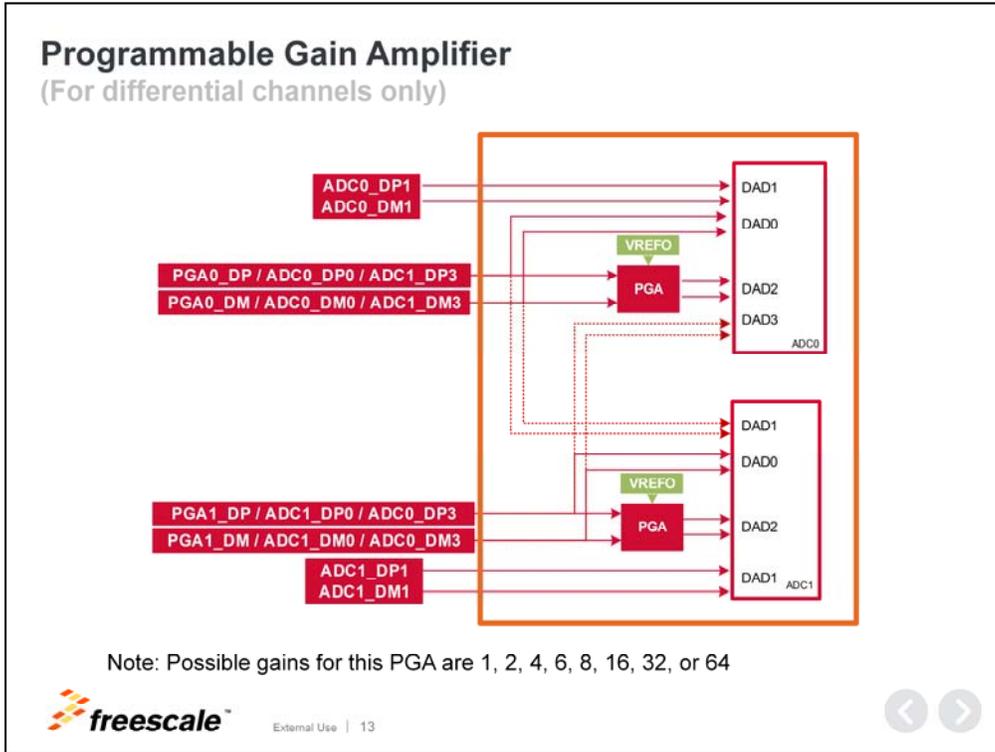
Single-ended Multiplexer

- Allow you to select between alternate set of channels
- This type of configuration increases the total number of channels



Single-ended Multiplexer

For single-ended inputs, the ADC has software that multiplexes from a series of single-ended channels labeled with an A or B suffix. This feature allows you to increase the number of ADC channels at the pinout and also creates a redundancy for systems that may require it.



Programmable Gain Amplifier (For differential channels only)

Some differential channels have a programmable gain amplifier, also known as a PGA.

The built-in PGA is designed to increase the dynamic range by amplifying low-amplitude signals before they are fed to the 16-bit ADC. Possible gains for this PGA are: 1, 2, 4, 6, 8, 16, 32, or 64.

The PGA only works with differential input signals between 0 -1.2 V. This is because the internal 1.2 VREF is the only possible ADC voltage referenced when using PGA.

Hardware (HW) Average

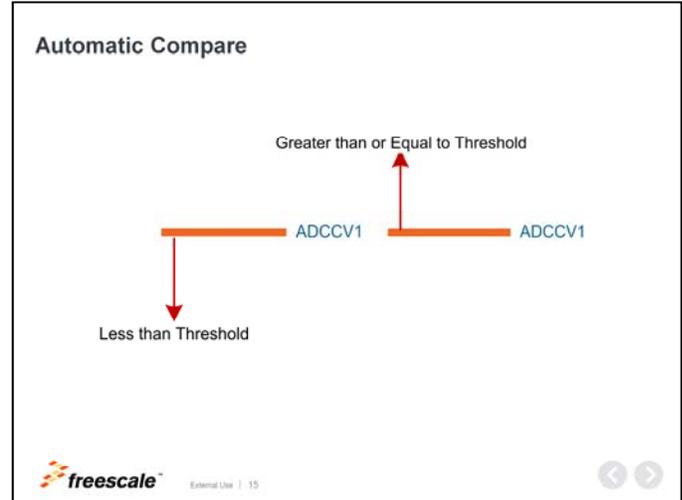
- When using HW average, keep in mind:
 - ✓ The more HW averaging, the slower the sampling rate
 - ✓ Make sure not to under-sample dynamic inputs
 - ✓ Dynamic performance depends on choosing the right amount of averaging. In general, the more HW averaging the higher the ADC accuracy.
- When the ADC is used to sample a dynamic input signal, it may be more suitable to employ hardware or digital filtering to reduce the impact of noise.

Hardware (HW) Average

The hardware average feature performs an average of multiple conversions. The number of conversions is determined by two bits in the ADC registers, which can select 4, 8, 16, or 32 conversions to be averaged. After the input is sampled and converted, the result is placed in an accumulator from which an average is calculated once the selected number of conversions have been completed.

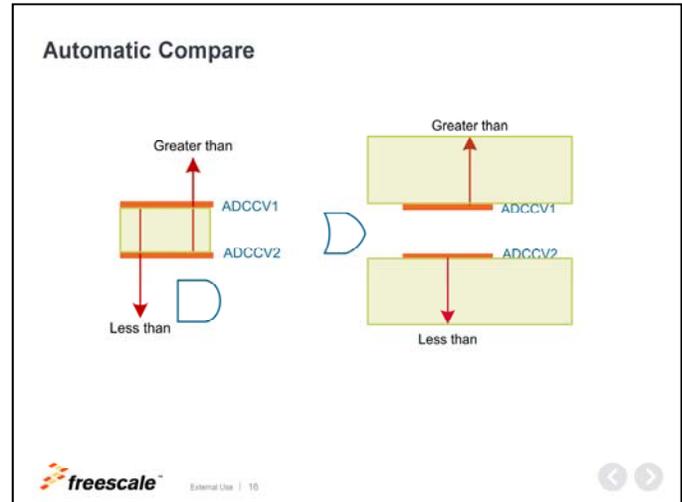
When hardware average is selected, the completion of a single conversion will not set the ADC conversion complete flag. When all selected conversions are completed, the average conversion result is transferred into the data result registers, and then ADC conversion complete is set.

When the ADC has a dynamic input, the number of samples to average needs to be chosen carefully. Averaging multiple samples impacts the sampling rate, therefore, the ADC configuration can be a source of aliasing.



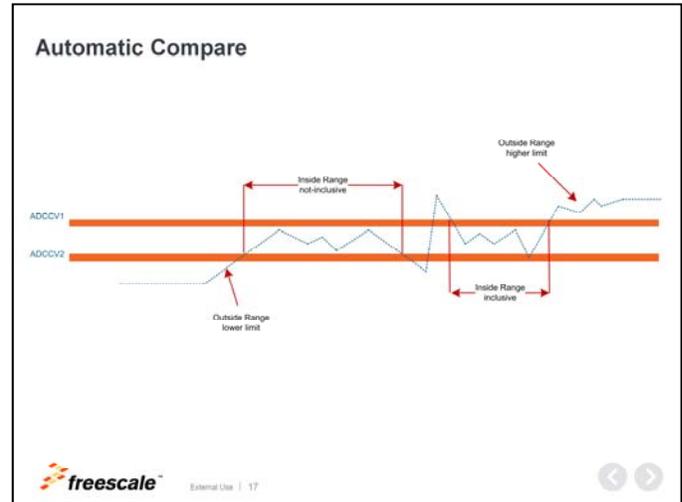
Automatic Compare

The compare function can be configured to check whether the result is, less than or greater-than-or-equal-to a single compare value.



Automatic Compare

The compare function can be configured to check whether the result is within or outside a range determined by two compare values.



Automatic Compare

ADCCV1 and ADCCV2 are software configurable like the automatic compare condition. This flexibility allows you to test several conditions on an input waveform without the need to analyze raw data.

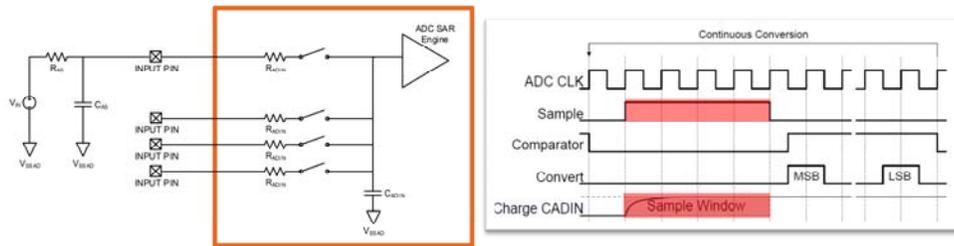
Sample Time

Sample time is the time that the internal switch is closed and sample capacitor (C_{ADIN}) is charged.

Software configurable

Short sample time – 4 ADC clocks

Long sample time – 6, 10, 16, 24 ADC clocks



Sample Time

The sample time is a time window for charging a sample capacitor. When sampling is complete, the converter is isolated from the input channel and the SAR algorithm converts the voltage at the sample capacitor.

Sample time can be either short or long. Since the ADC performance is linked to the charge of the sample capacitor, longer sample times are better.

Conversion Speed

Conversion time can be calculated using the following equation

$$\text{Conversion Time} = \text{SFCA} + \text{AverageNum} \times (\text{BCT} + \text{LSTA} + \text{HSCA})$$

Component	Meaning
SFCA	Single or first continuous adder
AverageNum	Average number factor, could be 1,4,8,16, or 32
BCT	Base conversion time
LSTA	Long sample time adder
HSCA	High-speed conversion time adder

Example: Conversion speed for a short sample time with high-speed single conversion

For 8-bit mode (18 MHz)

$$\begin{aligned} \text{Conversion Time} &= (5 \text{ ADCK} + 5 \text{ BUSCLK}) + 1 \times (17 \text{ ADCK} + 0 + 0) \\ &= 1.489 \mu\text{s} \end{aligned}$$

For 16-bit mode (12 MHz)

$$\begin{aligned} \text{Conversion Time} &= (5 \text{ ADCK} + 5 \text{ BUSCLK}) + 1 \times (25 \text{ ADCK} + 0 + 0) \\ &= 2.767 \mu\text{s} \end{aligned}$$



External Use | 19



Conversion Speed

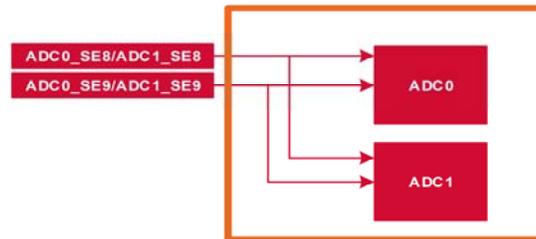
The conversion time can be calculated using the equation listed here, where the conversion time is equal to the single or first continuous adder + (plus) the average number factor X (multiplied) by the summation of the base conversion time which is related to the resolution, the long sample time adder which is related by the sample time, and the high-speed conversion time adder.

For example, on an 8-bit resolution mode with an 18MHz ADC clock frequency, using a short sample time and no hardware average, the conversion time will be 1.489 microseconds. For a 16-bit 12MHz ADC clock, it is 2.767 microseconds.

Interleaved Channels

Two ADCs cover the same external pin, providing:

- Higher speed rate
- Better efficiency
- More flexibility
- Frequent calibration without stopping measurements



Interleaved Channels

The majority of Kinetis K series devices have more than one instance. For multiple ADC instances, Kinetis channel pinouts are laid out in a way to support interleaving.

Interleaving allows two ADCs to cover the same external pin to increase speed of sampling or frequency of calibration. This allows for higher speed rates, better efficiency, more flexibility and enables frequent calibration without stopping measurements.

Low-power Mode Effect on 16-bit SAR ADC

ADC16-bit SAR:

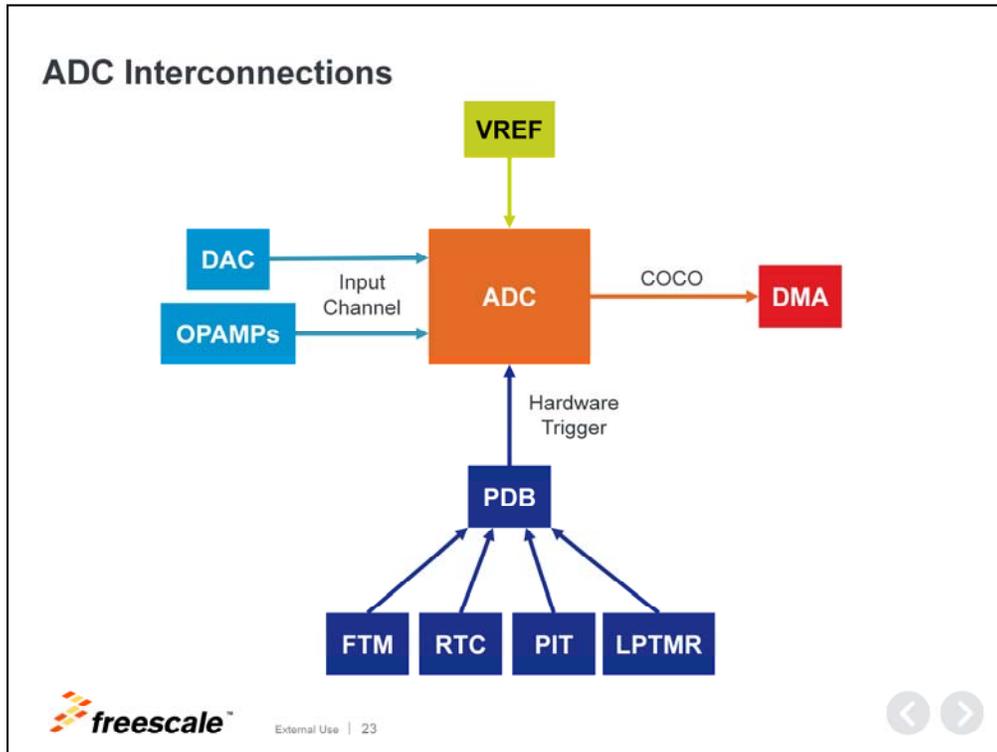
- Is fully functional in VLPR and VLPW low-power modes
 - Must use its own internal clock (ADACK) in the STOP and VLPS modes
- Retains state in LLS mode
- The ADC turns off in VLLSx low-power mode

Low-power Mode Effect on 16-bit SAR ADC

- The ADC is fully functional in VLPR and VLPW low-power modes. Is also is fully functional but must use its own internal clock (ADACK) in the STOP and VLPS modes.
- The ADC also retains state in LLS mode.
- Please note that the ADC turns off in VLLSx low-power mode.

2. On-chip Interconnections and Inter-module Dependencies

Next, let's turn to on-chip interconnection and inter-module dependencies.



ADC Interconnections

ADC internal connections include:

- DAC and OPAMPs, which have internal connections to input channels
- Internal 1.2V voltage reference peripheral like the ADC VREF
- Programmable delay block (or PDB) to hardware trigger conversion on the ADC
- The PDB can be triggered by other timers like the FTM, RTC, PIT, or the LPTMR. If the application needs to sync ADC conversions with these modules, the PDB can be bypassed or hardware triggered by one of the above mentioned times.
- The ADC can enable it's DMA internal channel to transfer ADC results to the DMA after the ADC conversion complete flag is set



Now, onto Hardware Configuration.

Design Practices for Hardware

Power Supplies

- Separate regulators for digital and analog supplies

Grounds

- No inductive coupling between VSS and VSSA

Trace Routing

- Keep digital traces **away** from analog supply, ADC reference, and ADC inputs

ADC Input

- Any filtering meets spec time constant constraints
- For 16-bit performance, you should **strongly** consider buffering the inputs



External Use | 25



Design Practices for Hardware

These are some recommendations to improve ADC performance:

- For power supplies, separate regulators for digital and analog supplies as this will ensure cleaner a voltage reference for the SAR algorithm. Use large “tank” caps at the regulator output for stability, decoupling caps of various values changes clock to the MCU.
- For grounds, do not use inductive coupling between VSS and VSSA. Use isolated digital and analog planes resistively coupled by a 0Ω resistor at a “startpoint” directly underneath the VSSA pin.
- For trace routing, keep digital traces away from the analog supply, ADC reference and ADC inputs. This means vertically as well as horizontally, and will likely mean more layers for multi-layered boards. Keep analog power and traces as short as possible.
- For ADC inputs, make sure any filtering on the input meets the time constant constraints outlined in the spec. Keep source resistance as low as possible. For 16-bit performance, customers should strongly consider buffering the inputs.



Let's move onto Software Configuration.

Kinetic SDK (KSDK) v1.2 ADC16 Driver

```

#define ADC0_INST 0

adc16_converter_config_t  adcUserConfig;
adc16_hw_average_config_t  adcHwAverageConfig;
adc16_chn_config_t        adcChnConfig;
uint16_t                  adcSampleValue;

// ADC Initialization
adcUserConfig.clkSrc = kAdc16ClkSrcOfBusClk;
adcUserConfig.clkDividerMode = kAdc16ClkDividerOf4;
adcUserConfig.continuousConvEnable = false;
adcUserConfig.dmaEnable = true;
adcUserConfig.highSpeedEnable = true;
adcUserConfig.hwTriggerEnable = false;
adcUserConfig.longSampleTimeEnable = false;
adcUserConfig.refVoltSrc = kAdc16RefVoltSrcOfVref;
adcUserConfig.resolution = kAdc16ResolutionBitOfSingleEndAs16;
ADC16_DRV_Init(ADC0_INST, &adcUserConfig);

// Enable hardware average
adcHwAverageConfig.hwAverageEnable = true;
adcHwAverageConfig.hwAverageCountMode = kAdc16HwAverageCountOf32;
ADC16_DRV_ConfigHwAverage(ADC0_INST, &adcHwAverageConfig);

// Configure ADC channel
adcChnConfig.chnIdx = kAdc16Chn0;
adcChnConfig.diffConvEnable = false;
adcChnConfig.convCompletedIntEnable = false;
ADC16_DRV_ConfigConvChn(ADC0_INST, 0U, &adcChnConfig);

// Convert ADC Channel
ADC16_DRV_WaitConvDone(ADC0_INST, 0U);
adcSampleValue = ADC16_DRV_GetConvValueRAM(ADC0_INST, 0U);

```


External Use | 27

Kinetic SDK (KSDK) v1.2 ADC16 Driver

To initialize the ADC16 driver, prepare configuration structure `adcUserConfig` and populate it with the desired configuration. After the configuration structure is available, initialize the ADC module by calling the API of [ADC16_DRV_Init\(\)](#) function.

If the application needs to enable hardware average, prepare the hardware average configuration structure and initialize the ADC registers by calling the API of [ADC16_DRV_ConfigHwAverage\(\)](#) function.

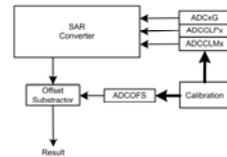
After the ADC registers are configured, the ADC channel configuration structure needs to be filled with the input channel, a binary selection of single-ended or differential channel, and then whether the ADC conversion complete flag will trigger an interrupt. The [ADC16_DRV_ConfigConvChn\(\)](#) function transfers the channel configuration structure to the ADC registers.

There are three types of use cases for the Kinetic SDK ADC driver: One-time-trigger, interrupt and blocking

This software example uses one-time-trigger.

Calibration

- Calibration needs to be done at least once for the generation of offset and gain compensation values
- These values are automatically subtracted (offset) and multiplied (gain) during the conversion sequence to compensate for internal errors
- If the ADC uses more than one configuration, calibrate the one that needs the highest accuracy



Calibration

There are three main sub blocks important for understanding how the Kinetis SAR module works. There is a capacitive DAC, a comparator, and the SAR engine that controls the module. Of these blocks, the DAC is the most susceptible to variation that causes linearity problems in the SAR.

The ADC contains a self-calibration function that balances the DAC capacitor bank. Such calibration is required to achieve the specified accuracy. Calibration must run after any reset. The calibration function sets the offset, minus-side and plus-side calibration values. These values are automatically subtracted and multiplied during the conversion sequence to compensate for internal errors.

If the ADC is used in more than 1 configuration, then calibrate the configuration that needs the highest accuracy.

Calibration

```

adc16_converter_config_t  adcUserConfig;
adc16_hw_average_config_t  adcHwAverageConfig;
adc16_calibration_param_t  adcUserCalibration;

// ADC Initialization
adcUserConfig.lowPowerEnable = false;
adcUserConfig.asyncClkEnable = false;
adcUserConfig.clkSrc = kAdc16ClkSrcOfBusClk;
adcUserConfig.clkDividerMode = kAdc16ClkDividerOf8;
adcUserConfig.resolution = kAdc16ResolutionBitOf12or13;
adcUserConfig.highSpeedEnable = false;
adcUserConfig.longSampleTimeEnable = true;
adcUserConfig.longSampleCycleMode = kAdc16LongSampleCycleOf24;
adcUserConfig.hwTriggerEnable = false;
adcUserConfig.refVoltSrc = kAdc16RefVoltSrcOfVref;
adcUserConfig.continuousConvEnable = false;
adcUserConfig.dmaEnable = false;
ADC16_DRV_Init(ADC0_INST, &adcUserConfig);

// Enable Hardware Average
adcHwAverageConfig.hwAverageEnable = true;
adcHwAverageConfig.hwAverageCountMode = kAdc16HwAverageCountOf32;
ADC16_DRV_ConfigHwAverage(ADC0_INST, &adcHwAverageConfig);

// Run calibration
ADC16_DRV_GetAutoCalibrationParam(ADC0_INST, &adcUserCalibration);
ADC16_DRV_SetCalibrationParam(ADC0_INST, &adcUserCalibration);

```

ADC Offset Correction Register (ADC0_OFS)
ADC Plus-Side Gain Register (ADC0_PG)
ADC Minus-Side Gain Register (ADC0_MG)
ADC Plus-Side General Calibration Value Register (ADC0_CLP0)
ADC Plus-Side General Calibration Value Register (ADC0_CLP5)
ADC Plus-Side General Calibration Value Register (ADC0_CLP4)
ADC Plus-Side General Calibration Value Register (ADC0_CLP3)
ADC Plus-Side General Calibration Value Register (ADC0_CLP2)
ADC Plus-Side General Calibration Value Register (ADC0_CLP1)
ADC Plus-Side General Calibration Value Register (ADC0_CLP0)
ADC Minus-Side General Calibration Value Register (ADC0_CLM0)
ADC Minus-Side General Calibration Value Register (ADC0_CLM5)
ADC Minus-Side General Calibration Value Register (ADC0_CLM4)
ADC Minus-Side General Calibration Value Register (ADC0_CLM3)
ADC Minus-Side General Calibration Value Register (ADC0_CLM2)
ADC Minus-Side General Calibration Value Register (ADC0_CLM1)
ADC Minus-Side General Calibration Value Register (ADC0_CLM0)


External Use | 29

⏪
⏩

Calibration

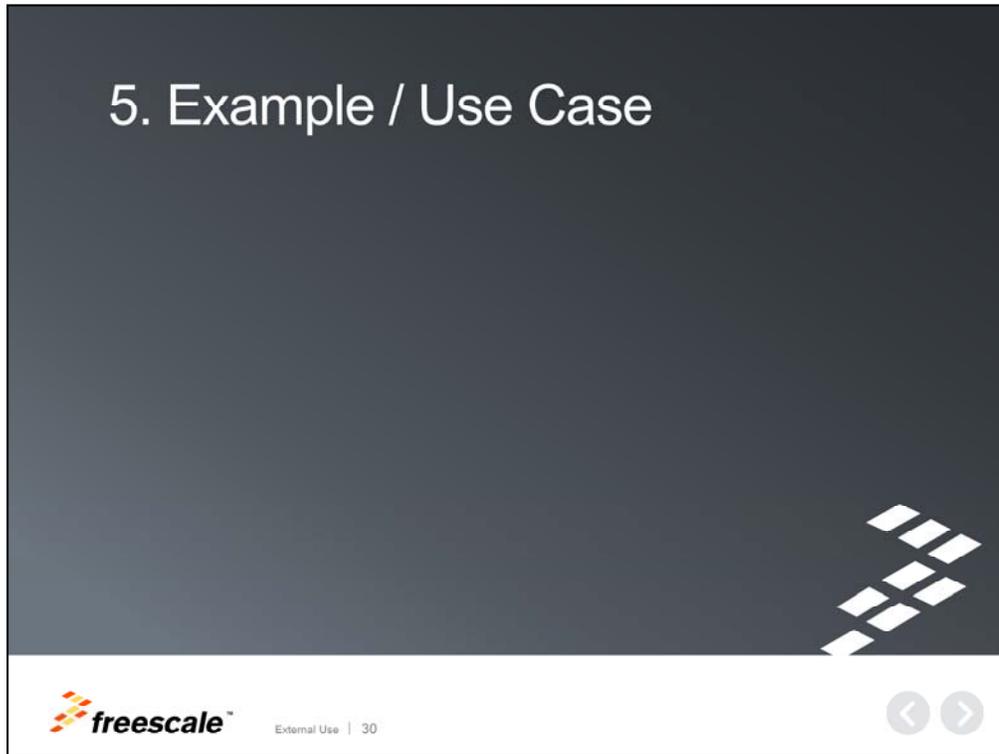
Prior to calibration, configure ADCs for optimum calibration results by

- Setting the hardware averaging to the maximum average of 32
- Set the ADC clock frequency less than or equal to 4MHz, and
- Set the longest sample time available by an extra 24 ADC clocks

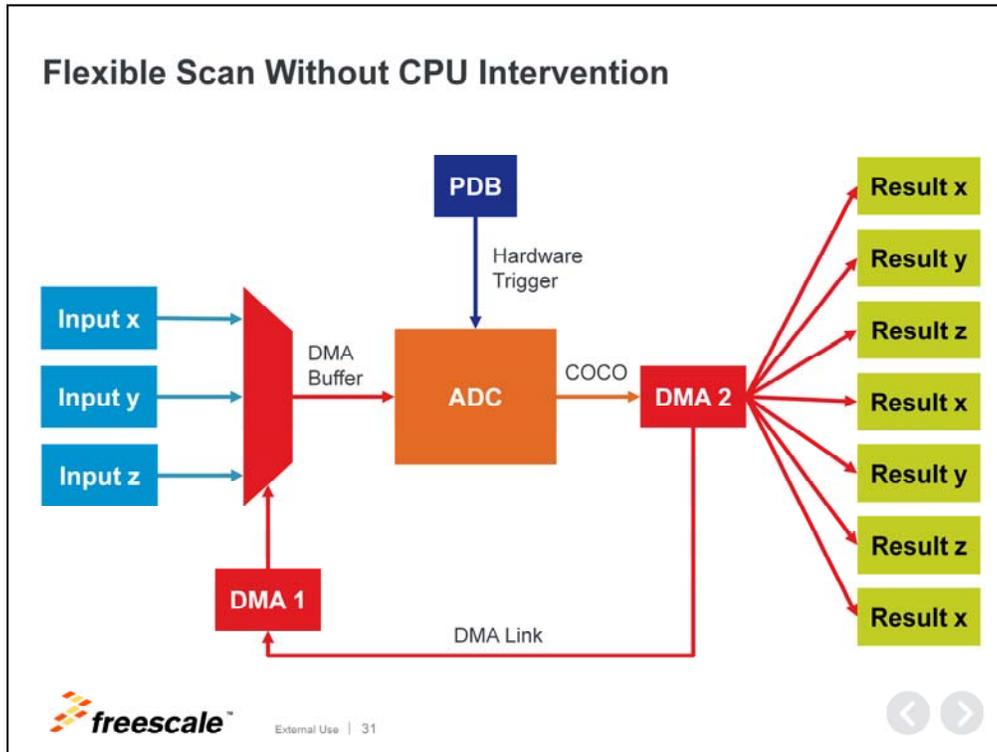
After the ADC is initialized, the software needs to call the

[ADC16_DRV_GetAutoCalibrationParam\(\)](#) and [ADC16_DRV_SetAutoCalibrationParam\(\)](#) functions.

Store calibration registers into non-volatile memory and re-write them into their corresponding registers whenever the ADC register values are cleared.



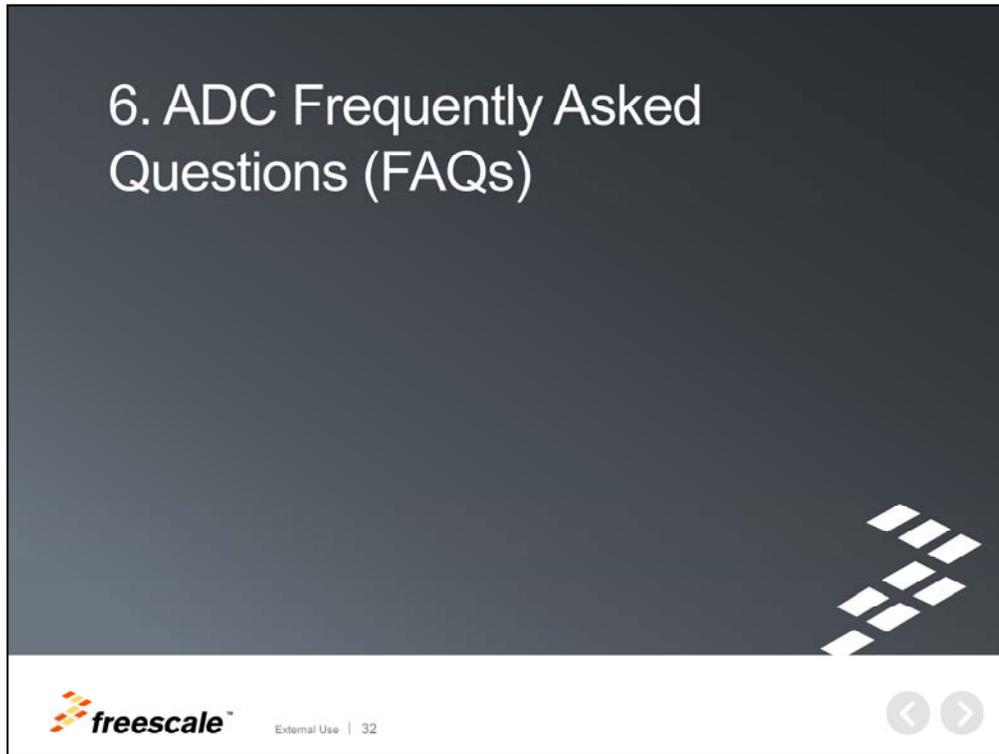
Now, let's go over an example use case.



Flexible Scan Without CPU Intervention

This is a possible use case to emulate ADC flexible scan mode by combining the ADC, 2 DMA channels and a PDB.

In this configuration The MCU uses less resources by using one ADC. The first DMA emulates a multiplexer so the application can convert several inputs. The ADC is PDB hardware triggered. If the ADC DMA is enabled we can use the second DMA to transfer ADC results to a result buffer. Using the DMA link feature we can link DMA2 to be the trigger for DMA1 and use it to change the source input at the precise moment.



Finally, review some frequently asked questions.

ADC FAQs

1. What is interleaving?

This feature allows two ADCs to cover the same external pin to increase speed of sampling or frequency of calibration.

2. Does ADC work in all low-power MCU modes?

Yes, ADC will work down to VLPS as long as it is configured with a valid clock source. Make sure your choice of clock source is active in the desired low-power mode.

3. Can an external trigger be applied?

Yes, an external trigger can be applied, for example triggering by the PDB which is in turn triggered by an external trigger. So not only can you trigger externally, but you can also trigger a sequence of up to four conversions.



External Use | 33



ADC FAQs

1. What is interleaving?

This feature allows two ADCs to cover the same external pin to increase speed of sampling or frequency of calibration.

2. Does ADC work in all low-power MCU modes?

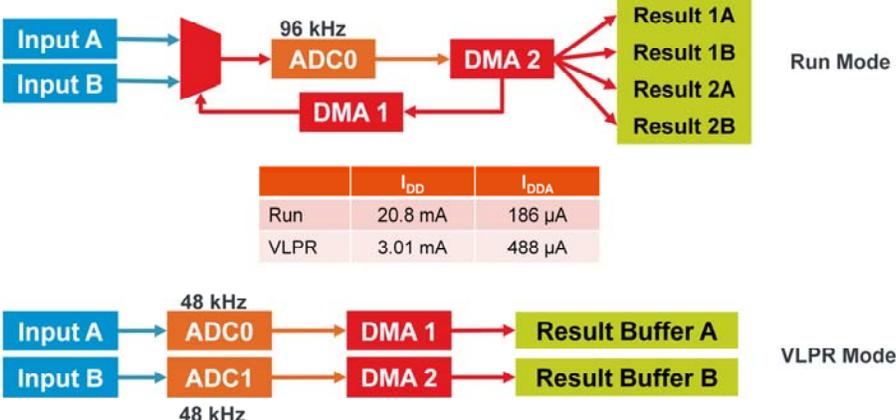
Yes, ADC will work in all low-power MCU modes as long as it is configured with a valid clock source. Make sure your choice of clock source is active in the desired low-power mode.

3. Can an external trigger be applied?

Yes, an external trigger can be applied, for example triggering by the PDB which is in turn triggered by an external trigger. So not only can you trigger externally, but you can also trigger a sequence of up to four conversions.

ADC FAQs – Continued

4. How do I get the best current consumption?
 To obtain the best current consumption, the application needs to sample 2 48kHz inputs without CPU load



	I_{DD}	I_{DDA}
Run	20.8 mA	186 μ A
VLPR	3.01 mA	488 μ A

freescalse™ External Use | 34

4. How do I get the best current consumption?

To obtain the best current consumption, the application needs to sample 2 48kHz inputs without CPU load

To address this request the application can use the same setup from the example use case section and use one ADC instance at 96kHz interleaving 2 samples with the MCU at run mode.

A second approach uses two ADC instances sampling one input at 48kHz with conversion complete DMAs filling the result buffers and the MCU configured at VLPR.

This table here shows that although the analog current is more than doubled by the fact that you have two ADCs turned ON, the overall power gets reduced by the ability to take samples while the MCU is in a low-power state like VLPR.

References

- **App notes:**

- [AN4568](#) Understanding the 16-bit ADC PGA in Kinetis K Series

- [AN4560](#) Using DMA to Emulate ADC Flexible Scan Mode

- [AN4373](#) Cookbook for SAR ADC Measurements

- **Website:** Freescale.com/Kinetis

- **Community:** community.freescale.com/community/Kinetis

- [DOC-103932](#) Understanding ADC specifications

- [DOC-104395](#) Using DMA to Emulate ADC Flexible Scan Mode with KSDK

- [DOC-102013](#) 16-bit SAR ADC Calibration



External Use | 35



References

This concludes our presentation on 16bit SAR ADC.

For additional ADC references, please visit the application notes listed here.

Also, we invite you to visit us on the web at Freescale.com/Kinetis and check out our community page.



www.Freescale.com

© 2015 Freescale Semiconductor, Inc. | *External Use*