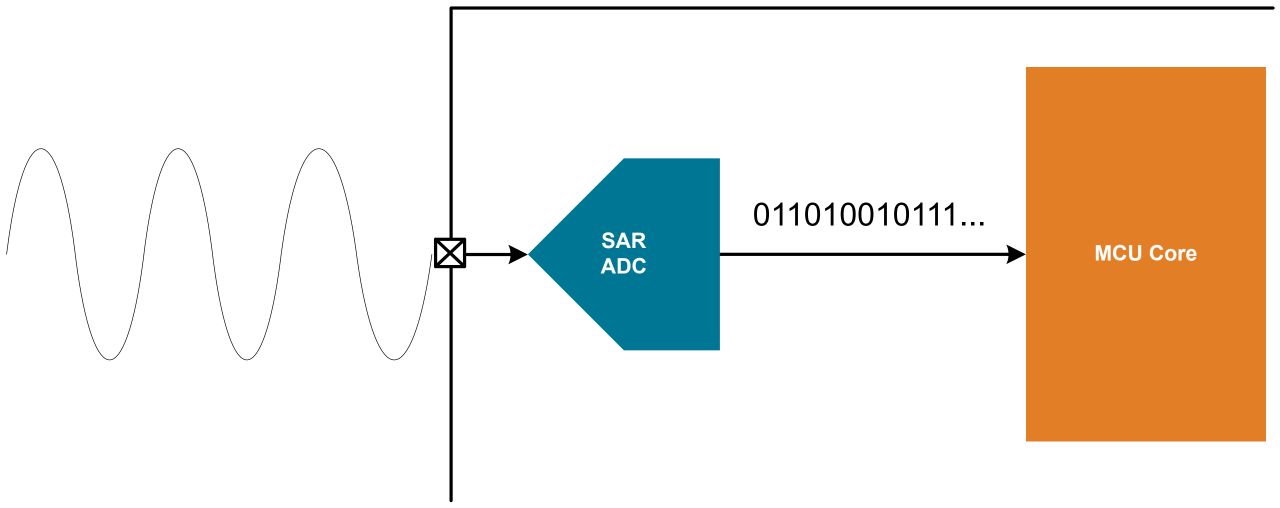
AN919: Using the EFM8LB1 ADC

This application note shows general operation and usage of the EFM8LB1's and EFM8BB3's ADC.

In addition, this document describes the advanced features of the ADC including Window Compare, Autoscan mode, and Low Power mode. This document also includes detailed operation instructions, accompanied by software examples, for each of these features. Software examples can be found within Simplicity Studio by navigating to the Software Examples tile, then to the ADC / Peripheral Driver category for the EFM8LB1 and EFM8BB3 devices.

KEY POINTS

- The EFM8LB1 ADC has up to 14 bits of resolution with up to 24 sampling inputs.
- The available autoscan mode allows for autonomous sample collection.
- The ADC also features low-power modes reduce current consumption.



1. Introduction

The EFM8LB1 and EFM8BB3 ADC is a successive-approximation-register (SAR) ADC with 14-bit (EFM8LB1 only), 12-bit, and 10-bit modes, integrated track-and hold and a programmable window detector. The ADC is fully configurable under software control via several registers. The ADC may be configured to measure different signals using the analog multiplexer including up to twenty external pins and four internal signals. The voltage reference for the ADC is selectable between internal and external reference sources. The ADC has an automatic scan mode that allows for the storage of up to 64 samples in memory without CPU intervention. The ADC can be configured to power down between samples and also run in a low-power mode to reduce energy consumption.

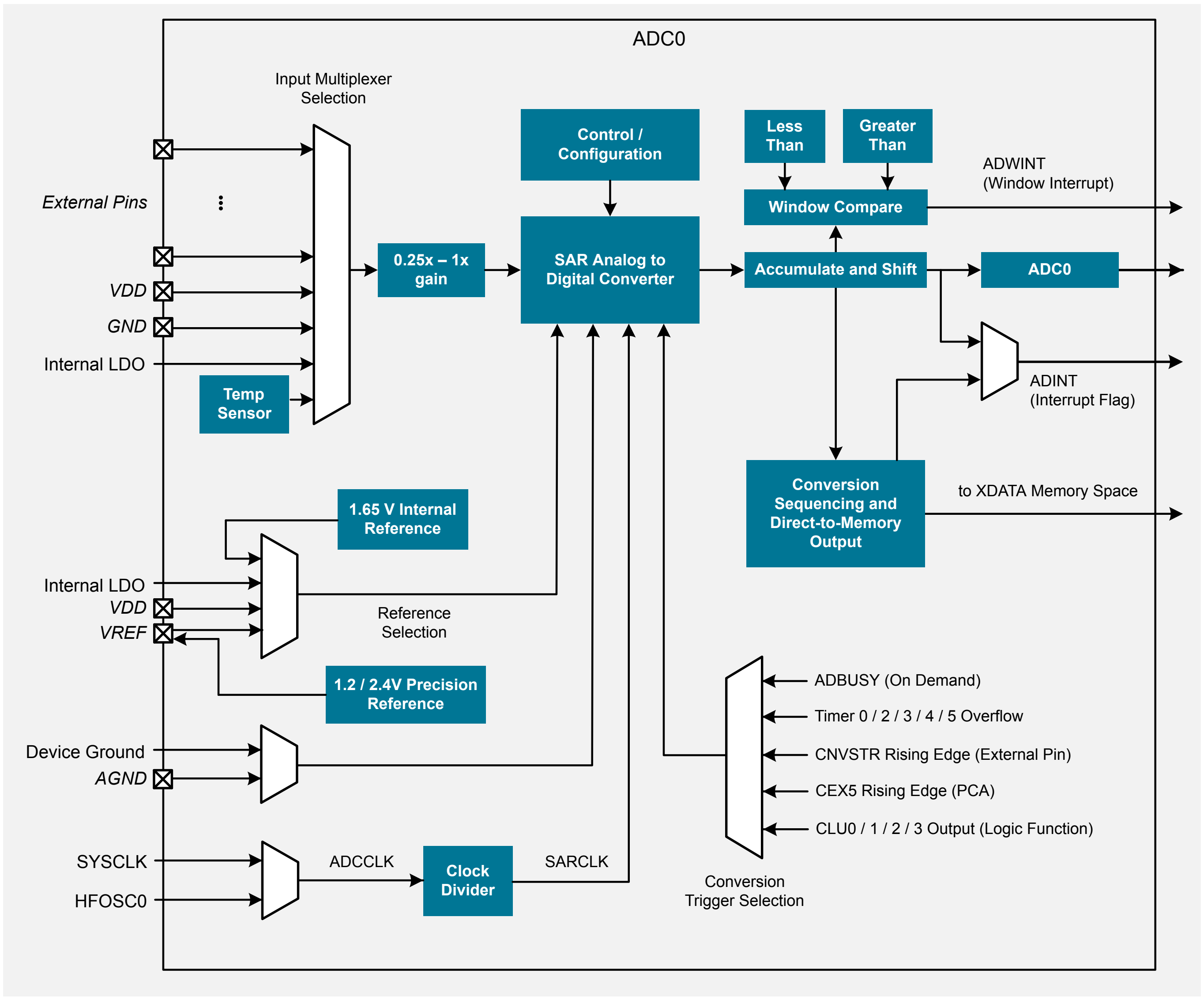

ADC0
Input Multiplexer Selection
External Pins
VDD
GND
Internal LDO
Temp Sensor
0.25x – 1x gain
Control / Configuration
SAR Analog to Digital Converter
Less Than
Greater Than
Window Compare
ADWINT (Window Interrupt)
Accumulate and Shift
ADC0
ADINT (Interrupt Flag)
Conversion Sequencing and Direct-to-Memory Output
to XDATA Memory Space
1.65 V Internal Reference
Internal LDO
VDD
VREF
Reference Selection
1.2 / 2.4V Precision Reference
Device Ground
AGND
SYSCLK
HFOSC0
ADCCLK
Clock Divider
SARCLK
ADBUSY (On Demand)
Timer 0 / 2 / 3 / 4 / 5 Overflow
CNVSTR Rising Edge (External Pin)
CEX5 Rising Edge (PCA)
CLU0 / 1 / 2 / 3 Output (Logic Function)
Conversion Trigger Selection

Figure 1.1. Analog to Digital Converter Block Diagram

2. General Operation

2.1 Functional Description

2.1.1 Input Selection

The ADC has an analog multiplexer which allows selection of external pins, the on-chip temperature sensor, the internal regulated supply, the VDD supply, or GND. ADC input channels are selected using the ADC0MX register.
Note: Any port pins selected as ADC inputs should be configured as analog inputs in their associated port configuration register, and configured to be skipped by the crossbar.

2.1.2 Gain Setting

The ADC has gain settings of 1x, 0.75x, 0.5x and 0.25x. In 1x mode, the full scale reading of the ADC is determined directly by VREF. In the other modes, the full-scale reading of the ADC occurs when the input voltage is equal to VREF divided by the selected gain. For example, in 0.5x mode, the full scale input voltage is VREF / 0.5 = VREF x 2. The lower gain settings can be useful to obtain a higher input voltage range when using a small VREF voltage, or to measure input voltages that are between VREF and the supply voltage. Gain settings for the ADC are controlled by the ADGN field in register ADC0CN0. Note that even with the lower gain settings, voltages above the supply rail cannot be measured directly by the ADC.

2.1.3 Voltage Reference Options

The voltage reference multiplexer is configurable to use a number of different internal and external reference sources. The ground reference mux allows the ground reference for ADC0 to be selected between the ground pin (GND) or a port pin dedicated to analog ground (AGND). The voltage and ground reference options are configured using the REF0CN register. The REFSL field selects between the different reference options, while GNDSL configures the ground connection.

2.1.3.1 Internal Voltage Reference

The high-speed internal reference is self-contained and stabilized. It is not routed to an external pin and requires no external decoupling. When selected, the internal reference will be automatically enabled/disabled on an as-needed basis by the ADC. The reference is nominally 1.65 V. The electrical specification tables in the datasheet have more information about the accuracy of this reference source.

2.1.3.2 Supply or LDO Voltage Reference

For applications with a non-varying power supply voltage, using the power supply as the voltage reference can provide the ADC with added dynamic range at the cost of reduced power supply noise rejection. Additionally, the internal LDO supply to the core may be used as a reference. Neither of these reference sources are routed to the VREF pin, and do not require additional external decoupling.

2.1.3.3 External Voltage Reference

An external reference may be applied to the VREF pin. Bypass capacitors should be added as recommended by the manufacturer of the external voltage reference. If the manufacturer does not provide recommendations, a 4.7 μF in parallel with a 0.1 μF capacitor is recommended.
Note: The VREF pin is a multi-function GPIO pin. When using an external voltage reference, VREF should be configured as an analog input and skipped by the crossbar.

2.1.3.4 Precision Voltage Reference

The precision voltage reference source is an on-chip block which requires external bypass (see the VREF chapter for details). The precision reference is routed to the VREF pin. To use the precision reference with the ADC, it should be enabled and settled, and the ADC's REFSL field should be set to the VREF pin setting.

2.1.3.5 Ground Reference

To prevent ground noise generated by switching digital logic from affecting sensitive analog measurements, a separate analog ground reference option is available. When enabled, the ground reference for the ADC during both the tracking/sampling and the conversion periods is taken from the AGND pin. Any external sensors sampled by the ADC should be referenced to the AGND pin. If an external voltage reference is used, the AGND pin should be connected to the ground of the external reference and its associated decoupling capacitor. The separate analog ground reference option is enabled by setting GNDSL to 1. Note that when sampling the internal temperature sensor, the internal chip ground is always used for the sampling operation, regardless of the setting of the GNDSL bit. Similarly, whenever the internal high-speed reference is selected, the internal chip ground is always used during the conversion period, regardless of the setting of the GNDSL bit.
Note: The AGND pin is a multi-function GPIO pin. When using AGND as the ground reference to the ADC, AGND should be configured as an analog input and skipped by the crossbar.

2.1.4 Clocking

The ADC clock (ADCCLK) can be selected from one of two sources using the ADCLKSEL field in ADC0CF0. The default selection is the system clock (SYSCLK). For applications requiring faster conversions but using a slower system clock, the HFOSC0 oscillator may be selected as the ADC clock source. ADCCLK is used to clock registers and other logic in the ADC.

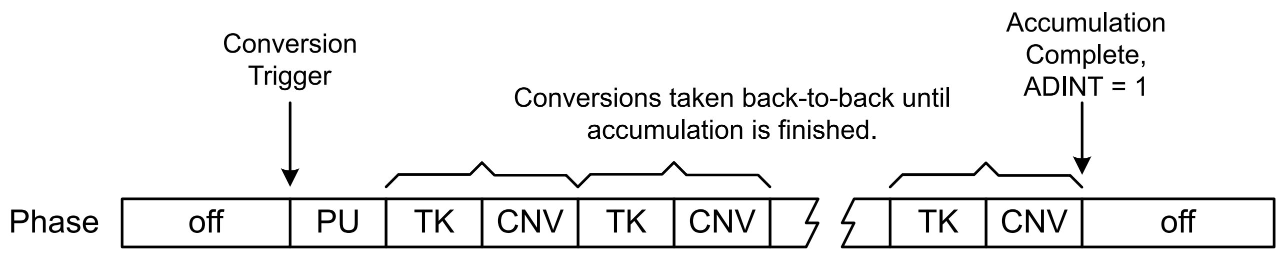
The conversion process is driven by the SAR clock (SARCLK). SARCLK is a divided version of the ADCCLK. The ADSC field in ADC0CF0 determines the divide ratio for SARCLK. In most applications, SARCLK should be adjusted to operate as fast as possible, without exceeding the maximum SAR clock frequency of 18 MHz.

2.1.5 Timing

Each ADC conversion may consist of multiple phases: power-up, tracking, and conversion. The power-up phase allows time for the ADC and internal reference circuitry to power on before sampling the input and performing a conversion. The power-up phase is optional, and used only when the ADC is configured to power off after the conversion is complete (IPOEN = 1). When IPOEN = 1, the ADC will power up, accumulate the requested number of conversions, and then power back off. The power-up phase is only present before the first conversion.

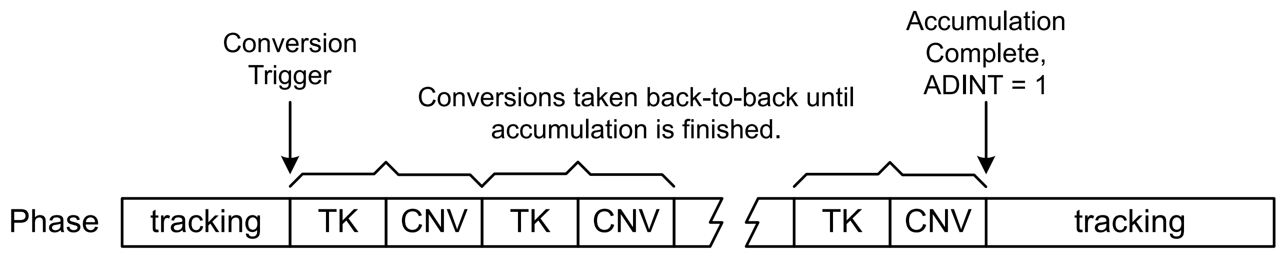
The tracking phase is the time period when the ADC multiplexer is connected to the selected input and sampled. Tracking can be defined to occur whenever a conversion is not in progress, or the ADC may be configured to track the input for a specific time prior to each conversion. When accumulating multiple conversions, it is important that the ADTK field be programmed for sufficient tracking between each conversion.

At the end of the tracking phase, the sample/hold circuit disconnects the input from the selected channel, and the sampled voltage is then converted to a digital value during the conversion phase.


Conversion Trigger
Conversions taken back-to-back until accumulation is finished.
Accumulation Complete, ADINT = 1
Phase
off
PU
TK
CNV
TK
CNV
TK
CNV
off

off = ADC shut down.
PU = Power-Up Phase. Timing Defined by ADPWR field.
TK = Tracking Phase. Timing Defined by ADTK field.
CNV = Conversion Phase. Timing depends on resolution and SARCLK.

Figure 2.1. ADC Timing With IPOEN = 1


Conversion Trigger
Conversions taken back-to-back until accumulation is finished.
Accumulation Complete, ADINT = 1
Phase
tracking
TK
CNV
TK
CNV
TK
CNV
tracking

tracking = Converter tracking selected input any time conversion is not in progress.
TK = Tracking Phase. Timing Defined by ADTK field.
CNV = Conversion Phase. Timing depends on resolution and SARCLK.

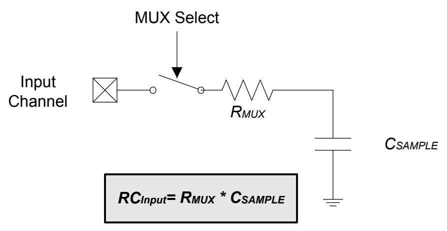
Figure 2.2. ADC Timing With IPOEN = 0

2.1.5.1 Input Tracking

Each ADC conversion must be preceded by a minimum tracking time to allow the voltage on the sampling capacitor to settle, and for the converted result to be accurate.

Settling Time Requirements

The absolute minimum tracking time is given in the electrical specifications tables, and will vary based on whether the ADC is in low power mode. It may be necessary to track for longer than the minimum tracking time specification, depending on the application. For example, if the ADC input is presented with a large series impedance, it will take longer for the sampling cap to settle on the final value during the tracking phase. The exact amount of tracking time required is a function of all series impedance (including the internal mux impedance and any external impedance sources), the sampling capacitance, and the desired accuracy.



Note: The value of C_{SAMPLE} depends on the PGA gain. See the electrical specifications for details.

Figure 2.3. ADC Equivalent Input Circuit

The required ADC0 settling time for a given settling accuracy (SA) may be approximated as follows:

$$t = \ln\left(\frac{2^n}{SA}\right) \times R_{TOTAL} \times C_{SAMPLE}$$

Where: SA is the settling accuracy, given as a fraction of an LSB (for example, 0.25 to settle within 1/4 LSB)

t is the required settling time in seconds

R_{TOTAL} is the sum of the ADC mux resistance and any external source resistance.

C_{SAMPLE} is the size of the ADC sampling capacitor.

n is the ADC resolution in bits.

When measuring any internal source, R_{TOTAL} reduces to R_{MUX}. See the electrical specification tables in the datasheet for ADC minimum settling time requirements as well as the mux impedance and sampling capacitor values.

Configuring the Tracking Time

The ADTK field configures the amount of time which will be allocated for input tracking by the ADC conversion logic.

When IPOEN is set to 1, firmware must always configure the ADTK field to allow adequate tracking and settling of the selected input. The tracking time will be applied after the power-up phase is complete, and before the conversion begins.

When IPOEN is cleared to 0, the ADC-timed tracking phase will still be applied before every conversion. If ADRPT is configured to accumulate multiple conversions, firmware must configure the ADTK bits to ensure that adequate tracking is given to every conversion. However, the ADC will continue to track the input whenever it is not actively performing a conversion. ADTK may be set to zero, provided that ADRPT is configured for single conversions, and adequate tracking time is allowed for in-between every conversion.

2.1.5.2 Power-Up Timing

The ADC requires up to 1.2 us to power up and settle all internal circuitry. When IPOEN is set to 1, the ADC will power down between conversions to save energy. Firmware must configure the ADPWR field to allow adequate time for the ADC and internal reference circuitry to power up before each conversion.

When IPOEN is cleared to 0, the ADPWR time is not applied. This is primarily useful when operating the ADC in faster data acquisition systems. When firmware enables the ADC from a powered-down state, it must take the required power time into account before initiating a conversion. Once the ADC is powered on in this mode, it will remain powered up and the power-up time is not needed between subsequent conversions.

2.1.5.3 Conversion Resolution and Timing

The conversion resolution is adjusted using the ADBITS field in ADC0CN1, and selectable between 14-bit (EFM8LB1 only), 12-bit , and 10-bit modes. The total amount of time required for a conversion is equal to:

Total Conversion Time = [RPT × (ADTK + NUMBITS + 1) × T(SARCLK)] + (T(ADCCLK) × 4)

where RPT is the number of conversions represented by the ADRPT field and ADCCLK is the clock selected for the ADC. Up to one SYSCLK of synchronization time is also required when triggering from the external CNVSTR pin source.

2.1.6 Initiating Conversions

Conversions may be initiated in many ways, depending on the programmed state of the ADCM bitfield. The following options are available as conversion trigger sources:

1. Software-triggered—Writing a 1 to the ADBUSY bit initiates conversions.
2. Hardware-triggered—An automatic internal event such as a timer overflow initiates conversions.
3. External pin-triggered—A rising edge on the CNVSTR input signal initiates conversions.
   Note: The CNVSTR pin is a multi-function GPIO pin. When the CNVSTR input is used as the ADC conversion source, the associated port pin should be skipped in the crossbar settings.

Basic converter operation is straightforward. The selected conversion trigger will begin the conversion cycle. Writing a 1 to ADBUSY provides software control of ADC0 whereby conversions are performed "on-demand". All other trigger sources occur autonomous to code execution. Each conversion cycle may consist of one or more conversions, as determined by the ADRPT setting. Individual conversions from the ADC will be accumulated until the requested number of conversions has been accumulated. When the converter is finished accumulating conversions, the ADINT flag will be posted and firmware may read the output results from the ADC data registers (ADC0H:ADC0L). Note that the first conversion in an accumulation sequence is triggered from the selected trigger source, while all subsequent conversions in an accumulation sequence will be self-triggered upon completing the previous conversion.

During any conversion, the ADBUSY bit is set to logic 1 and reset to logic 0 when the conversion is complete. However, the ADBUSY bit should not be used to poll for ADC conversion completion. It will read back 0 whenever the converter is not in the conversion phase, and results may not yet be available in the ADC data registers. The ADC interrupt flag (ADINT) should be polled instead, when writing polled-mode firmware.

2.2 Operation

2.2.1 Polled Mode

In polled mode, each conversion of the ADC is manually triggered. The software then polls to see if the conversion has been completed. Once a conversion has been completed, the result is retrieved and a new conversion is started. This example shows the steps necessary to acquire single samples using the ADC in polled mode.

Initialization sequence:

1. Configure the input mux settings: Write ADC0MX to the desired channel. If the input is a port pin, configure the pin as an analog input through the PnMDIN register.
2. Configure the ADC's clock: select the ADC clock source with ADCLKSEL and configure the SARCLK by setting ADSC, making sure to comply with the ADC's absolute maximum SARCLK specification.
3. Configure the ADC's voltage reference: select the reference by setting REFSL. Set ADGN to configure the gain.
4. Write to ADTK to configure the tracking time to meet the ADC's and input tracking time requirements.
5. Enable the ADC. Write 1 to ADEN.
6. Configure the ADC for manual conversions. Write 0 to ADCM.
7. Clear ADINT.
8. Begin an ADC conversions. Write 1 to ADBUSY.
9. Poll ADINT until it equals 1.
10. Read the result from ADC0H and ADC0L.
11. If more conversions are required, repeat from #8.

Software Example: ADC Lib Polled

This example demonstrates using the ADC to sample a voltage in polled mode. This is a simple method to use the ADC that does not require interrupts.

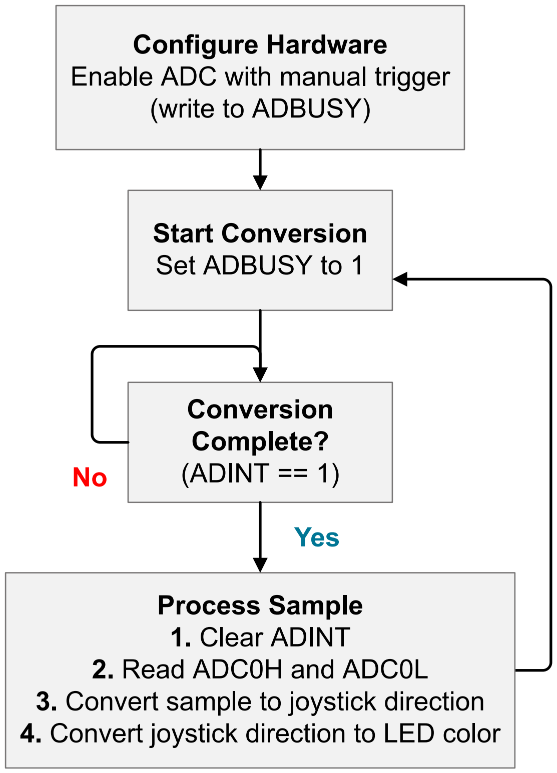


Figure 2.4. Polled Mode Example

In this example, the ADC is configured to sample the STK's joystick on pin P1.7, requiring manual triggers to start a conversion. In the main loop, an ADC conversion is manually initiated. The loop then polls the conversion complete interrupt flag until the conversion is complete. The conversion complete flag is then cleared, and the joystick's voltage is used to change the color of the LED on the STK. The loop then begins again, starting another conversion.

2.2.2 Periodic Interrupt

Interrupt Mode

A more common mode of operation is interrupt mode. In this mode, the ADC triggers an interrupt when a conversion is completed, allowing the CPU to perform other tasks or idle to conserve power while the conversion is underway. This example shows the steps necessary to acquire samples in interrupt mode.

Initialization sequence:

1. Configure the input mux settings: Write ADC0MX to the desired channel. If the input is a port pin, configure the pin as an analog input through the PnMDIN register.
2. Enable the ADC. Write 1 to ADEN.
3. Clear ADINT.
4. Begin an ADC conversions: Either start the conversion trigger source, or if the trigger source is already running, switch the ADC to use it by writing to ADCM.
5. Perform other tasks or, if desired, place the CPU into idle mode to conserve power.

Interrupt Routine:

1. Clear ADINT.
2. Read the result from ADC0H and ADC0L.

Software Example: ADC Lib Interrupt

This example demonstrates using the ADC to sample a voltage using conversion complete interrupts.

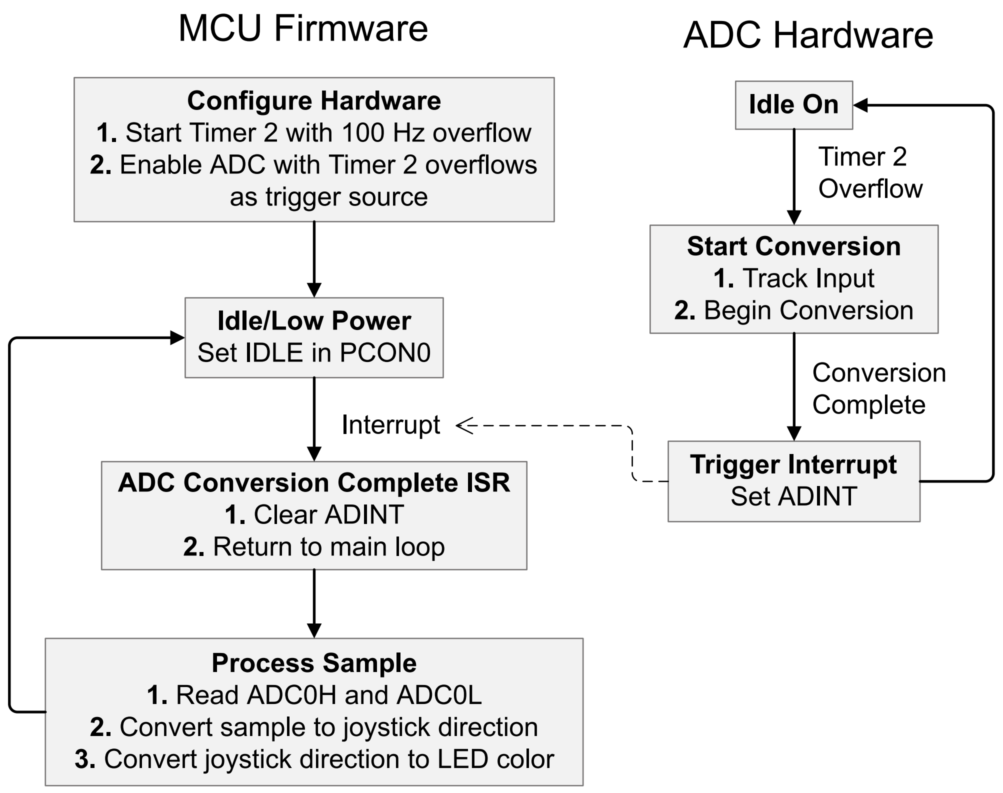


Figure 2.5. Interrupt Example

Timer 2 is configured to overflow at approximately 100Hz. The ADC is configured to sample the STK's joystick (P1.7), and to start a conversion on each Timer 2 overflow. When a conversion is completed, a conversion complete interrupt is triggered which signals to the main loop to process the latest sample. The joystick voltage is then used to change the color of the STK's LED.

After each iteration of the main loop, the device is put into idle mode to conserve power until the next interrupt fires.

2.2.3 Low Power

The ADC has two main features for reducing power consumption, Low Power mode and Idle Powered-off.

Low Power mode can be enabled to decrease current consumption at the cost of additional minimum tracking time. Please see the datasheet for this minimum tracking time specification.

To use the Low Power mode:

1. Set ADLPM in ADC0CF1 to 1.
2. Adjust for the additional tracking time required. This can be accomplished by either setting ADTK, or, if Idle Powered-off is disabled, through an additional delay between conversions.

The second feature, Idle Powered-off, will power down the ADC between conversions. This adds an additional 1.2 us power-up delay before a conversion can be started. Additionally, if this feature is enabled, the ADC must be configured to meet tracking requirements by setting ADTK, since the ADC will not be tracking the source between conversions.

To use Idled Powered-off:

1. Set IPOEN in ADC0CN0 to 1.
2. Configure the power-up delay to be greater than or equal to 1.2 us by setting ADPWR in ADC0CF2.

   Given the undivided ADC input clock F_{adcclk} (either SYSCLK or HFOSC0), the formula for determining this is given by: ADPWR = $(((F_{adcclk} \times 1.2 \text{ us}) - 2)/4 - 1)$ rounded up. For example, with an ADC clock of 12 MHz, the result of this formula would be 2.1 rounded up to 3. This would result in a power-up delay of 1.5 us. The result must be rounded up since, using this example, an ADPWR setting of 2 would result in a power-up delay of 1.167 us, which would not satisfy the ADC's minimum power-up time requirement.
3. Configure the tracking delay by setting ADTK to satisfy the minimum settling time required for the sampling capacitor voltage to settle. The required tracking delay must be set through ADTK since the ADC will be powered down, and thus not tracking, between conversions.

Software Example: ADC Lib Interrupt Low Power

This example demonstrates using the ADC to sample a voltage using conversion complete interrupts with the ADC's low power features enabled, saving a substantial amount of current compared to the periodic interrupt example.

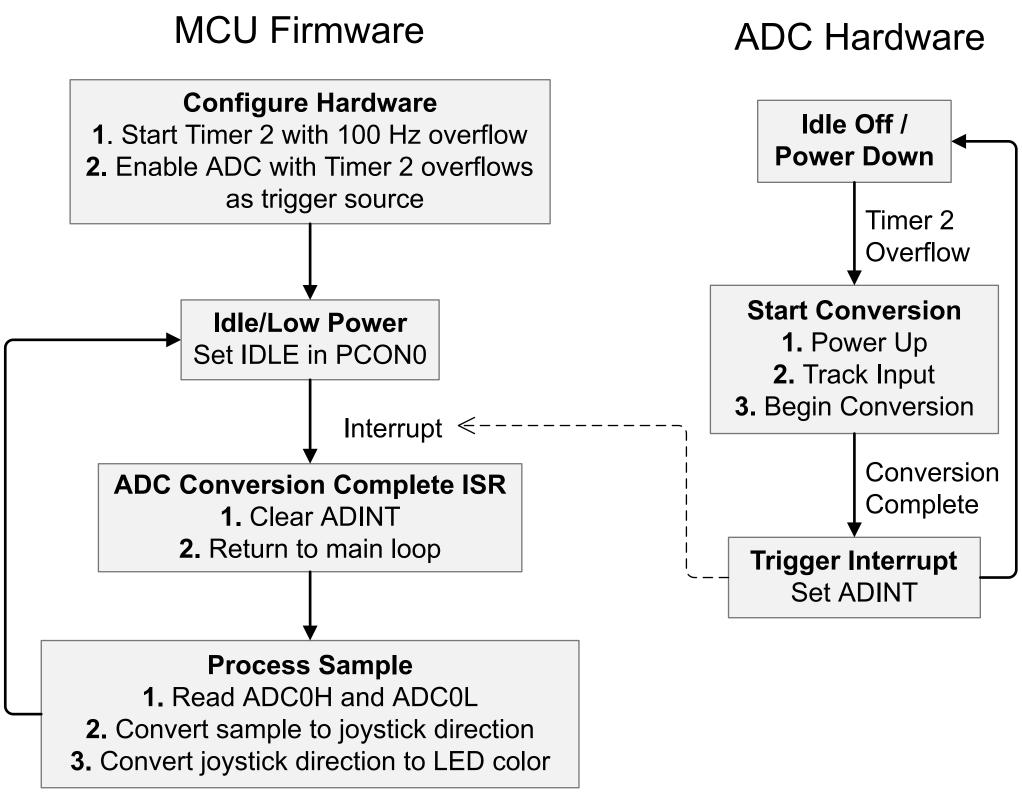


Figure 2.6. Low Power Example

This example is identical to the Periodic Interrupt example, except that the ADC's Low Power Mode is enabled, and the ADC is configured to power down between conversions. These changes result in approximately 1 mA of current savings with a negligible decrease in ADC throughput.

3. Autoscan Mode

In addition to basic conversions, the ADC includes a flexible autoscan mode, which offloads much of the firmware tasks required to collect information from the ADC. Autoscan allows multiple output words from the ADC to be collected on up to four contiguous ADC channels without firmware intervention. ADC outputs are written to a firmware-designated area of XDATA space in the order they are received. The firmware specifies the number of desired output words (up to 64) before a scan begins. When active, the scanner will collect the requested number of output words from the ADC. At the end of a scan sequence, the autoscan hardware stores the current state of select register fields, generates an interrupt, and optionally continues with a new scan.

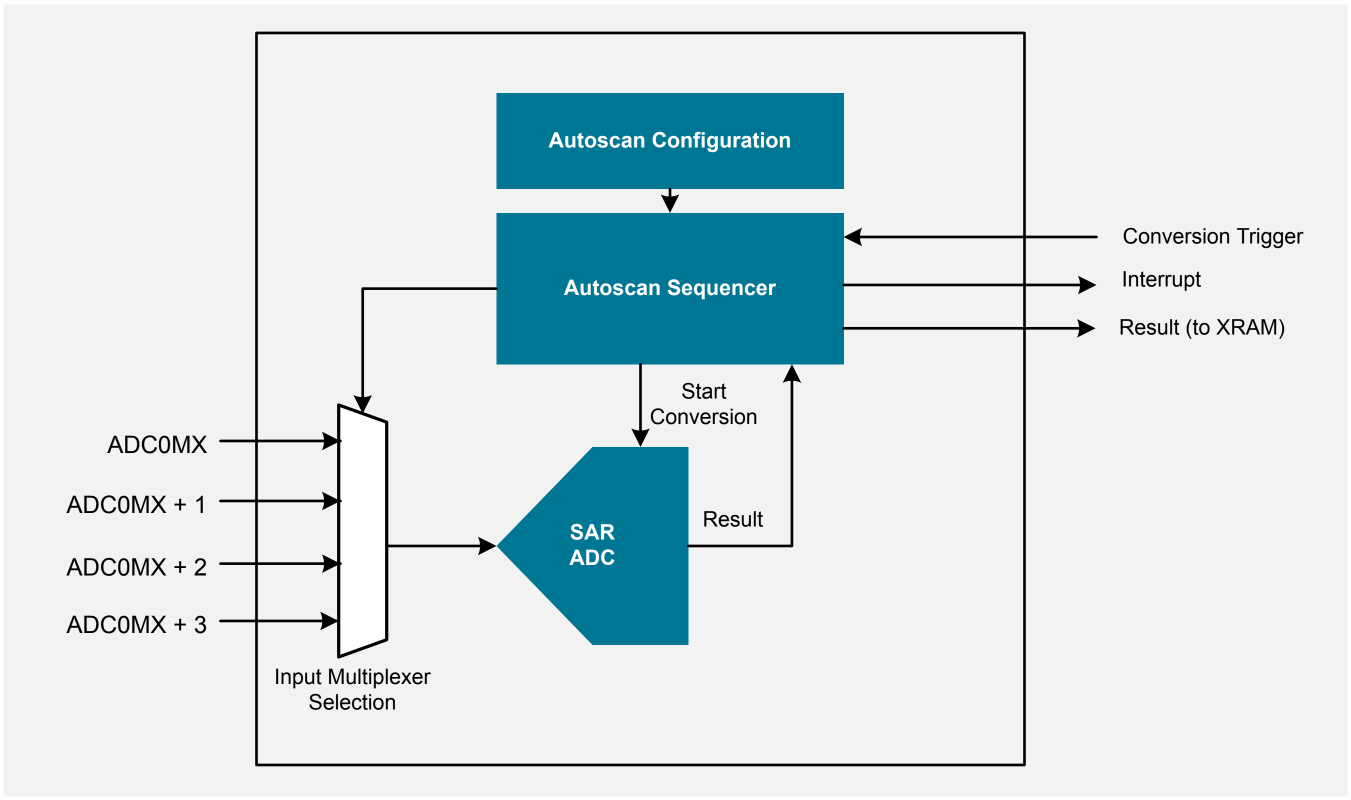


Figure 3.1. Autoscan Block Diagram

3.1 Functional Description

Trigger Configuration

In autoscan mode, the ADC may be triggered by any of the trigger source options selected by ADCM. The STEN bit in ADC0ASCF controls whether multiple triggers or a single trigger is required to complete the scan operation. When STEN is cleared to 0 (MULTIPLE_TRIGGERS), each (accumulated) conversion in the scan requires a new conversion trigger event. For example, if Timer 3 is the selected ADC trigger source and the autoscan hardware is configured to accumulate 20 sets of 4 conversions, Timer 3 would need to overflow 20 times to generate a trigger event for each conversion. When STEN is set to 1 (SINGLE_TRIGGER), an entire scan will be performed using a single trigger. In the preceding example, the first conversion would be triggered from a Timer 3 overflow event, and then the rest of the conversions would be automatically triggered by the scan hardware as each conversion completes.
Note: The converter must not be in the process of a normal conversion when entering autoscan mode. For this reason, firmware should ensure that the desired trigger source will not trigger the ADC before ASEN is set to 1. The simplest way to do this is to leave ADCM configured for software triggers until after ASEN is set to 1, and then select the desired trigger source.

Channel Configuration

The scanner hardware is capable of collecting data from up to four contiguous ADC channels in sequence. The ADC0MX register defines the first channel to be converted, and the NASCH field in ADC0ASCF defines the number of channels (1, 2, 3, or 4) to be converted. Channels are converted in circular fashion, one at a time. For example, if ADC0MX is configured to 0x02, NASCH is configured to convert three channels, and nine conversions are requested, the autoscan hardware will collect a conversion from ADC0MX = 0x02, then ADC0MX = 0x03, then ADC0MX = 0x04, then repeat at ADC0MX = 0x02, and so on until nine conversions are collected (three conversions on each of the three channels).

The ADRPT setting is valid in autoscan mode, and each accumulated sample counts as one conversion output from the autoscan hardware. If ADRPT is configured to accumulate 4 conversions and the scanner is configured to collect 9 samples, a total of 9 x 4, or 36 conversions will be performed. When scanning through multiple channels, the ADC will accumulate the requested number of conversions on each channel before proceeding to the next channel.

Output Data Configuration

Data from the autoscanner is written directly into XDATA space, starting at an address defined by the 16-bit ADC0ASA register (the combination of the two 8-bit registers ADC0ASAH and ADC0ASAL). ADC0ASA[11:1] correspond directly to bits 11:1 of the XRAM starting address. This means that the starting address must occur on an even-numbered address location. The ENDIAN bit in ADC0ASAL defines the endian-ness of the output data.

Each output word from the ADC will require two bytes of XDATA space. For a single scan consisting of 10 conversions, 20 XDATA bytes are required to hold the output.
Note: The toolchain used for firmware development will not be automatically aware of the location for the scanner output. When using the autoscan function, it is very important for the firmware developer to reserve the area intended for scanner output, to avoid contention with other variables.

Autoscan Operation

When ADC configuration is complete, firmware may place the ADC in scan mode by setting the ASEN bit in ADC0ASCF to 1. Note that the scan does not immediately begin when the ASEN bit is set. ASEN places the ADC into autoscan mode, waiting for the first trigger to occur. When ASEN is set, hardware will copy the contents of the ADC0ASAH, ADC0ASAL, AD0ASCNT and ADC0MX registers, as well as the NASCH field in ADC0ASCF into local registers for the scanner to use. This allows firmware to immediately set up the parameters for the following scan.

If only one scan is desired, firmware can immediately clear ASEN back to 0. Just as setting ASEN does not immediately begin a scan, clearing ASEN does not immediately take the converter out of autoscan mode. Autoscan mode will only be halted if ASEN is 0 at the completion of a scan operation. To terminate a scan in progress, firmware must disable the ADC completely with the ADEN bit.

When the ADC first enters autoscan mode, it waits for the selected conversion trigger to occur. In the case of software-triggered operation, firmware can begin the scan by setting the ADBUSY bit to 1. For timer-triggered conversions, firmware should enable the selected timer.

The scan will proceed according to the configuration options until all of the operations specified by AD0ASCNT have been completed. At the end of a scan operation, the scanner will set the AD0INT bit to 1, and check the status of ASEN. If ASEN is 0, autoscan mode is terminated, and the converter will return to normal mode. If ASEN is 1 however, a new scan is immediately begun, scan settings are loaded into the scanner's local registers, and the ADC waits for the next trigger to occur.

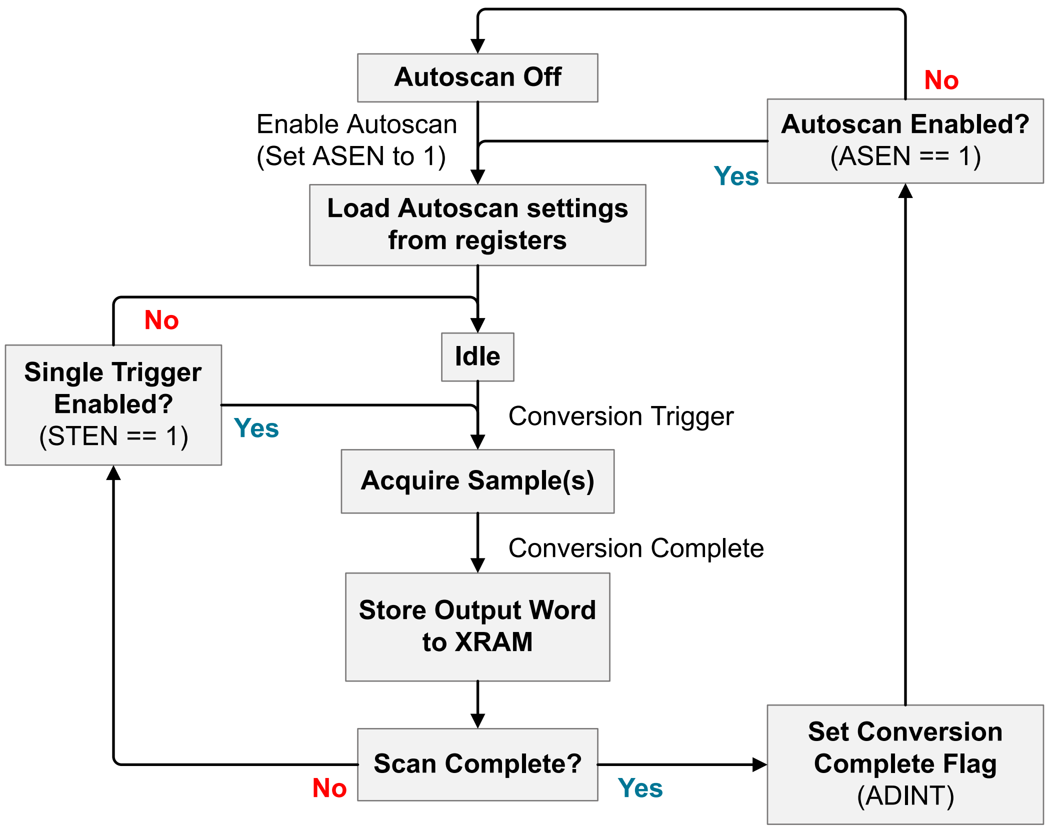


Figure 3.2. Autoscan Flow Diagram

3.2 Operation

Performing a Single Scan

The simplest manner to use the autoscan feature is to initiate a single scan, then return the ADC to normal mode once the scan is complete. This can be accomplished by following these steps:

1. Configure the autoscan in the ADC0ASAH, ADC0ASAL, AD0ASCNT and ADC0MX registers.
2. Write ASEN to 1. The autoscan settings are now loaded and the scan is enabled.
3. Write ASEN to 0. Autoscan mode will be disabled after the scan completes.
4. Begin ADC Conversions.
5. Wait for the scan to be completed as indicated by the ADINT bit in the ADC0CN0 register. If Autoscan Single Trigger is enabled (STEN is 1 in the ADC0ASCF register), this will take as many triggers as the scan is configured to take samples. If Single Trigger is disabled, only one ADC trigger is required to complete the entire scan.
6. Process the data.

Performing a Continuous Scan

The ADC also has the capability of staying in autoscan mode after each scan. Once a scan is completed, the next scan's settings are then loaded into the autoscan's local registers from the ADC's registers. This requires that the settings for the next scan are written to the ADC's registers before the current scan completes. This can be accomplished by the following instructions:

1. Configure the autoscan in the ADC0ASAH, ADC0ASAL, AD0ASCNT and ADC0MX registers.
2. Write ASEN to 1. The autoscan settings are now loaded and the scan is enabled.
3. If different settings are desired for the next scan, configure them in the ADC0ASAH, ADC0ASAL, AD0ASCNT and ADC0MX registers.
4. Begin ADC Conversions.
5. Wait for the scan to be completed as indicated by the ADINT bit in the ADC0CN0 register. The new scan settings will now be loaded.
6. Process the data.
7. Write new autoscan settings to the ADC0ASAH, ADC0ASAL, AD0ASCNT and ADC0MX registers, if desired.
8. Repeat from #5.

Stopping a Continuous Scan

To stop a continuous scan, ASEN must be written to zero. This does not immediately disable autoscan mode, however - the currently running scan must first be completed. To stop a continuous scan:

1. Write ASEN to 0. Autoscan mode will be disabled after the current scan completes.
2. Wait for the scan to be completed as indicated by the ADINT bit in the ADC0CN0 register. The ADC will now be in normal mode.

3.2.1 Circular Buffer

Autoscan Example: Circular Buffer

This example shows the steps necessary to use autoscan mode to implement a 128-word ping-pong buffer for a single ADC channel in XDATA. The buffer will consist of two 64-word (128-byte) areas in XDATA, beginning at 0x0000 and 0x0080, and the firmware is responsible for changing the scanner hardware at the appropriate intervals to keep a continual flow of data into memory. This example assumes that the ADC will be triggered in multiple-trigger mode from a hardware source, such as a timer.

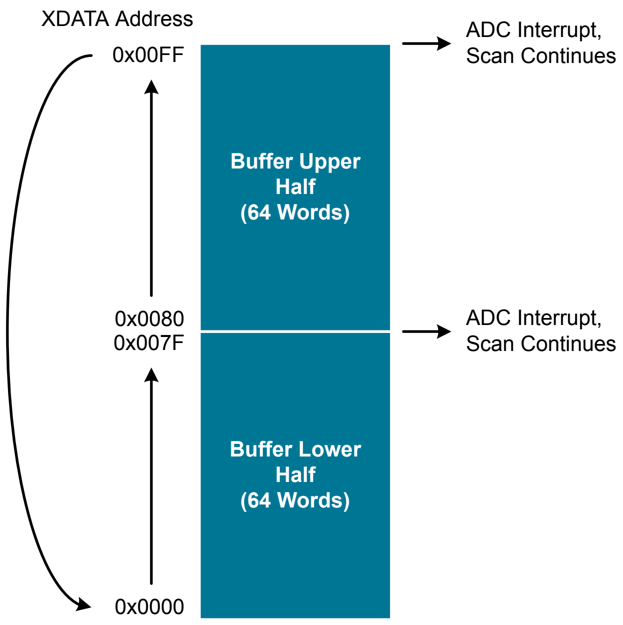

XDATA Address
0x00FF
0x0080
0x007F
0x0000
Buffer Upper Half (64 Words)
Buffer Lower Half (64 Words)
ADC Interrupt, Scan Continues
ADC Interrupt, Scan Continues

Figure 3.3. Circular Buffer Example

Initialization sequence:

1. Configure the ADC for no accumulation: Write ADRPT to 0.
2. Configure the input mux settings: Write ADC0MX to the desired channel, and write NASCH to 0.
3. Configure the starting address for the first half of the buffer: Write ADC0ASA[H:L] to 0x0000.
4. Configure to collect 64 samples: Write ADC0ASCNT to 63.
5. Initiate autoscan mode: Write ASEN to 1.
6. Configure the starting address for the second half of the buffer: Write ADC0ASA[H:L] to 0x0080.
7. Begin ADC conversions: Either start the conversion trigger source, or if the trigger source is already running, switch the ADC to use it).

Interrupt Service Routine:

1. Clear AD0INT.
2. Configure the starting address for the opposite buffer: Write ADC0ASA[H:L] to 0x0000 if it is 0x0080, or vice-versa.
3. Process the data in the most recent buffer, or optionally signal to the main thread that data is ready to be processed.

Software Example: ADC Lib Autoscan Circular Buffer

This example demonstrates using the autoscan feature of the ADC to implement a 128 sample circular buffer, allowing 64 samples to be processed while the other 64 samples are being acquired. Only one interrupt is generated per 64 samples, drastically reducing CPU overhead.

Two 64-sample buffers are created in XDATA space, one after another. Timer 2 is configured to trigger ADC conversions at 64 Hz. Each conversion, when completed, fills one place in the current buffer. After the buffer is full (one second of samples), a conversion complete interrupt is triggered, which swaps the full buffer with the previously inactive buffer. The contents of the full buffer are then printed out over UART.

3.2.2 Large Buffer

Autoscan Example: Large Buffer

This example shows the steps necessary to use autoscan mode to implement a 2048-word buffer for a single ADC channel in XDATA, starting at address 0x0000 and ending at 0x07FF. This buffer will be incrementally filled by 64 word scans, requiring the firmware to advance the location of the scans as they are completed. Using autoscan for this purpose reduces the requirement for CPU intervention from 2048 instances to 32 instances.

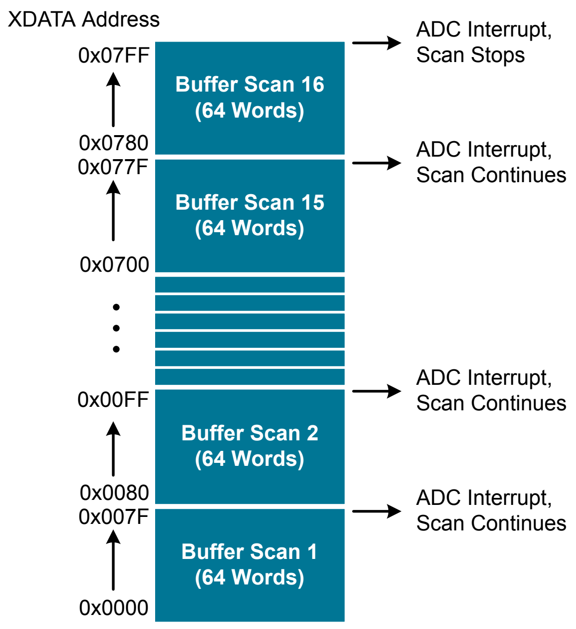

XDATA Address
0x07FF
0x0780
0x077F
0x0700
0x00FF
0x0080
0x007F
0x0000
Buffer Scan 16 (64 Words)
Buffer Scan 15 (64 Words)
Buffer Scan 2 (64 Words)
Buffer Scan 1 (64 Words)
ADC Interrupt, Scan Stops
ADC Interrupt, Scan Continues
ADC Interrupt, Scan Continues
ADC Interrupt, Scan Continues

Figure 3.4. Large Buffer Example

Initialization sequence:

1. Configure the ADC for no accumulation: Write ADRPT to 0.
2. Configure the input mux settings: Write ADC0MX to the desired channel, and write NASCH to 0.
3. Configure the starting address for the beginning of the buffer: Write ADC0ASA[H:L] to 0x0000.
4. Configure to collect 64 samples: Write ADC0ASCNT to 63.
5. Initiate autoscan mode: Write ASEN to 1.
6. Begin ADC conversions: Either start the conversion trigger source, or if the trigger source is already running, switch the ADC to use it).
7. Configure the starting address for the second scan, 64 samples ahead of the beginning: Write ADC0ASA[H:L] to 0x0080.

Interrupt Service Routine:

1. Clear AD0INT.
2. Configure the starting address for the next scan, 64 samples ahead of the current address: Write ADC0ASA[H:L] to ADC0ASA[H:L] +0x0080.
3. If the last scan is being configured, disable autoscan mode to prevent autoscan from continuing after the buffer is full: Write ASEN to 0.
4. If the buffer has been filled (the interrupt has been triggered after autoscan has been disabled), process the data, or optionally signal to the main thread that data is ready to be processed.

Software Example: ADC Lib Autoscan Large Buffer

This program demonstrates the use of the ADC's Autoscan feature to fill a 2048 sample buffer while only requiring 32 interrupts to be serviced by the CPU.

A 2048 sample buffer is created in XDATA space. The ADC is then configured to perform 64 samples per autoscan, then generate a conversion complete interrupt. Inside this interrupt, the autoscan is configured to store samples in the next 64 spots in the buffer, and a new scan is started. Once the buffer is entirely full, the autoscan is disabled. Pressing PB0 will repeat filling the buffer.

3.2.3 Multiple Inputs

Autoscan Example: Single Scan of Two Channels

This example shows the steps necessary to use autoscan mode to implement a single scan of two adjacent mux channels into a 64-word buffer (32 conversions per channel). In this example, a single software trigger is used to initiate the entire scan sequence.

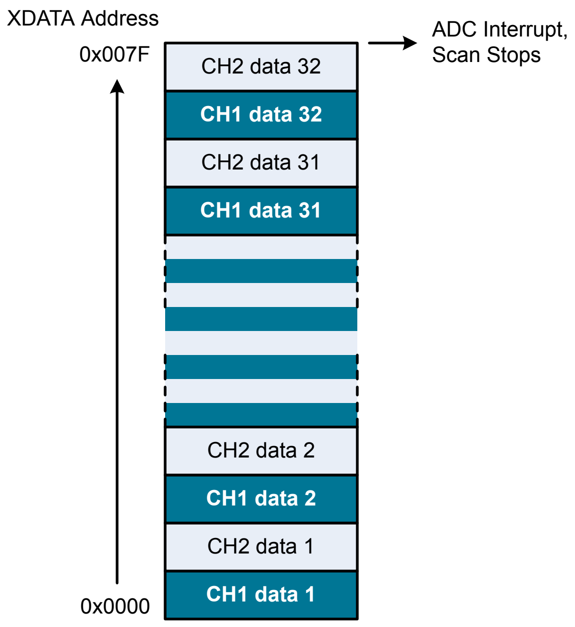


Figure 3.5. Circular Buffer Example

Initialization sequence:

1. Configure the ADC for no accumulation: Write ADRPT to 0.
2. Configure the ADC trigger source: Write ADCM to 0 for software triggers, and write STEN to 1 to enable a single-trigger autoscan.
3. Configure the input mux settings: Write ADC0MX to the starting (lowest-numbered) channel, and write NASCH to 1 (for two channels).
4. Configure the starting address for the memory output: Write ADC0ASA[H:L] to 0x0000.
5. Configure to collect 64 samples: Write ADC0ASCNT to 63.
6. Initiate autoscan mode: Write ASEN to 1.
7. Write ASEN to 0. This will instruct the scanner to stop upon scan completion.
8. Begin ADC conversions: Write ADBUSY to 1.

Interrupt Service Routine:

1. Clear AD0INT.
2. Process the data, or optionally signal to the main thread that data is ready to be processed.

Software Example: ADC Lib Autoscan Multiple Inputs

This example demonstrates using the autoscan feature of the ADC to sample multiple inputs without CPU intervention.

A 64-sample buffer is created in XDATA space and used by the ADC to sample two consecutive input channels, starting with pin P1.7, the STK's joystick. The second channel is the next consecutive channel, pin P2.1. Timer 3 is configured to trigger ADC conversions every two seconds. The ADC's single-trigger functionality is enabled, causing each trigger to complete the entire scan operation. Once the scan is completed, a conversion complete interrupt is triggered and the results of the 64 samples are printed over UART.

4. Window Comparator

4.1 Operation

The ADC's programmable window detector compares the ADC output registers to user-programmed limits, and notifies the system when a desired condition is detected. This is especially effective in an interrupt driven system, saving code space and CPU bandwidth while delivering faster system response times. The window detector interrupt flag (ADWINT) can also be used in polled mode. The ADC Greater-Than (ADC0GTH, ADC0GTL) and Less-Than (ADC0LTH, ADC0LTL) registers hold the comparison values. The window detector flag can be programmed to indicate when measured data is inside or outside of the user-programmed limits, depending on the contents of the ADC0GT and ADC0LT registers. The following tables show how the ADC0GT and ADC0LT registers may be configured to set the ADWINT flag when the ADC output code is above, below, between, or outside of specific values.

Table 4.1. ADC Window Comparator Example (10-bit codes, Above 0x0080)

Comparison Register Settings	Output Code (ADC0H:L)	ADWINT Effects
	0x03FF	ADWINT = 1
	...	
	0x0081	
ADC0GTH:L = 0x0080	0x0080	ADWINT Not Affected
	0x007F	
	...	
	0x0001	
ADC0LTH:L = 0x0000	0x0000	

Table 4.2. ADC Window Comparator Example (10-bit codes, Below 0x0040)

Comparison Register Settings	Output Code (ADC0H:L)	ADWINT Effects
ADC0GTH:L = 0x03FF	0x03FF	ADWINT Not Affected
	0x03FE	
	...	
	0x0041	
ADC0LTH:L = 0x0040	0x0040	
	0x003F	ADWINT = 1
	...	
	0x0000	

Table 4.3. ADC Window Comparator Example (10-bit codes, Between 0x0040 and 0x0080)

Comparison Register Settings	Output Code (ADC0H:L)	ADWINT Effects
	0x03FF	ADWINT Not Affected
	...	
	0x0081	
ADC0LTH:L = 0x0080	0x0080	

Comparison Register Settings	Output Code (ADC0H:L)	ADWINT Effects
	0x007F	ADWINT = 1
	...	
	0x0041	
ADC0GTH:L = 0x0040	0x0040	ADWINT Not Affected
	0x003F	
	...	
	0x0000	

Table 4.4. ADC Window Comparator Example (10-bit codes, Outside the 0x0040 to 0x0080 range)

Comparison Register Settings	Output Code (ADC0H:L)	ADWINT Effects
	0x03FF	ADWINT = 1
	...	
	0x0081	
ADC0GTH:L = 0x0080	0x0080	ADWINT Not Affected
	0x007F	
	...	
	0x0041	
ADC0LTH:L = 0x0040	0x0040	
	0x003F	ADWINT = 1
	...	
	0x0000	

4.1.1 Monitoring an Input Voltage

A common use case for using the Window Compare feature is to monitor a particular voltage source and only act if it is outside a given range. For example, when monitoring a temperature sensor, the firmware may only want to be interrupted if the temperature is above an operating maximum or below an operating minimum. To use the Window Compare feature to monitor a voltage source:

1. Convert the desired boundary voltages into their ADC output representations. The output value for a given voltage for an ADC with *n* bits of resolution is: (Voltage / VREF) x 2^n. For example, if the desired voltage is 1V, VREF is 1.65V, and the ADC is set to 12 bit mode, the output code would be: (1/1.65) x 4096, or 0x09B2.
2. Set ADC0GT to the output code that the ADC sample must be greater than in order to trigger a window compare event. Similarly, set ADC0LT to the less-than output code.
3. Enable Window Compare interrupts. Set EWADC0 in EIE1 to 1.
4. Begin ADC conversions. Typically, this will be a periodic trigger, such as from a timer overflow. A Window Compare interrupt will only be triggered if the ADC's result is greater than the ADC0GT value or less than the ADC0LT value.

The Window Compare feature can also be combined with the Autoscan feature to autonomously monitor the voltage up to four consecutive ADC input channels. To monitor four channels (with the samples stored in XDATA starting at address 0x0000), setup the Autoscan as follows:

1. Configure the autoscan trigger function: Write STEN to 1 to enable a single-trigger autoscan, clear STEN to 0 to require one trigger per sample in the autoscan.
2. Configure the input mux settings: Write ADC0MX to the starting (lowest-numbered) channel, and write NASCH to 3 (for four channels).
3. Configure the starting address for the memory output: Write ADC0ASA[H:L] to 0x0000.
4. Configure to collect 4 samples: Write ADC0ASCNT to 3.
5. Initiate autoscan mode: Write ASEN to 1.

From here, follow the previous instructions on how to monitor a voltage source with Window Compare. With this setup, the ADC will automatically cycle through sampling each of the four input sources. After every conversion, if the result is outside the Window Compare boundaries, a Window Compare interrupt will be triggered. From here, the samples can be read from XDATA to see which channel triggered the interrupt.

Software Example: ADC Lib Interrupt Window Compare

This example demonstrates using the ADC to sample a voltage using the Window Compare feature, which is used to only interrupt the CPU when the ADC sampled voltage falls outside of the programmed window.

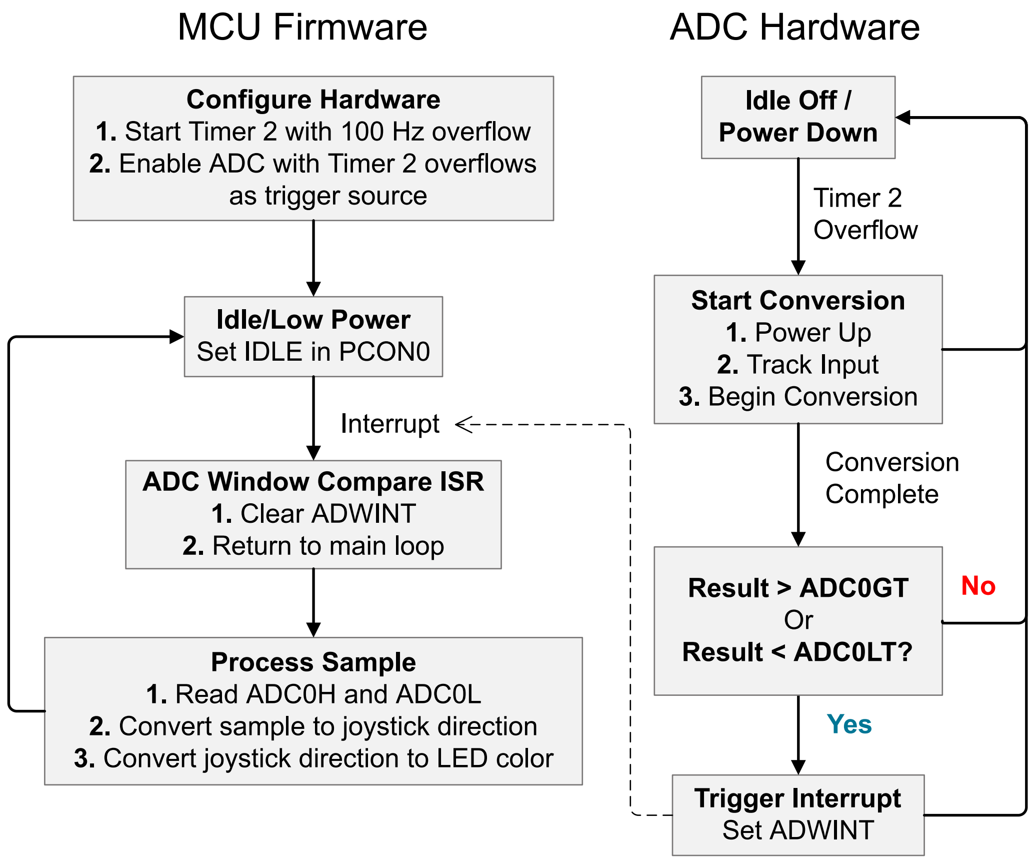


Figure 4.1. Window Compare Example

Like the previous interrupt examples, Timer 2 is configured to overflow at approximately 100 Hz and the ADC is configured to start a conversion on each Timer 2 overflow. When a conversion is completed, if it is greater than or less than the ADC's respective window compare values, it will trigger an interrupt. Inside this interrupt, these window compare values are adjusted to be slightly above and below the most recently sampled value. This effectively means that the ADC will only trigger an interrupt when the voltage being sampled changes significantly. This drastically reduces the number of interrupts serviced compared to the normal interrupt example.

5. Output Formatting and Accumulation

5.1 Operation

The registers ADC0H and ADC0L contain the high and low bytes of the output conversion code from the ADC at the completion of each conversion. Data may be accumulated over multiple conversions and the final output may be shifted right by a selectable amount, effectively providing an "accumulate and average" function. In the following examples, 1 LSBn refers to the voltage of one LSB of the converter at the specified resolution, calculated as VREF x 1 / 2^n. An LSB12 would be calculated as VREF x 1/4096.

When the repeat count ADRPT is configured for a single conversion and the ADSJST field is configured for no shifting, output conversion codes are represented in the selected resolution of the converter. Example codes are shown below for the different data formats with ADRPT = 0, ADSJST = 0, and a gain setting of 1x (ADGN = 0). Unused bits in the ADC0H and ADC0L registers are set to 0.

Table 5.1. Output Coding, ADRPT = 0, ADSJST = 0

Input Voltage	10-bit	12-bit	14-bit (EFM8LB1 only)
	ADC0H:L	ADC0H:L	ADC0H:L
VREF - 1 LSBn	0x03FF	0x0FFF	0x3FFF
VREF / 2	0x0200	0x0800	0x2000
VREF / 4	0x0100	0x0400	0x1000
0	0x0000	0x0000	0x0000

When the repeat count is greater than 1, the output conversion code represents the accumulated result of the conversions performed and is updated after the last conversion in the series is finished. Sets of 4, 8, 16, or 32 consecutive samples can be accumulated and represented in unsigned integer format. The repeat count can be selected using the ADRPT bit field. Unused bits in the ADC0H and ADC0L registers are set to 0. The example below shows the right-justified result for various input voltages and repeat counts for 12-bit conversions. Notice that accumulating 2^n samples is equivalent to left-shifting by n bit positions when all samples returned from the ADC have the same value.

Table 5.2. Effects of ADRPT on Output Code (12-bit conversions, ADSJST = 0)

Input Voltage	Repeat Count = 4	Repeat Count = 8	Repeat Count = 16
VREF - 1 LSB12	0x3FFC	0x7FF8	0xFFF0
VREF / 2	0x2000	0x4000	0x8000
(VREF / 2) - 1 LSB12	0x1FFC	0x3FF8	0x7FF0
0	0x0000	0x0000	0x0000

Additionally, the ADSJST bit field can be used to format the contents of the 16-bit accumulator. The accumulated result can be shifted right by 1, 2, or 3 bit positions, effectively dividing the output by 2, 4, or 8. The example below shows the effects of using ADSJST on a 12-bit sample.

Table 5.3. Using ADSJST for Output Formatting (12-bit conversions, ADRPT = 8)

Input Voltage	ADSJST = 0	ADSJST = 1	ADSJST = 3
	(no shift)	(shift right 1 bit)	(shift right 3 bits)
VREF - 1 LSB12	0x7FF8	0x3FFC	0x0FFF
VREF / 2	0x4000	0x2000	0x0800
(VREF / 2) - 1 LSB12	0x3FF8	0x1FFC	0x07FF
0	0x0000	0x0000	0x0000

5.1.1 Improving Noise Performance

The hardware accumulation feature can be used to effectively oversample the signal source. The hardware output right-shifting can effectively divide, or decimate, the ADC's output word. Together, these features can be used to perform oversampling and decimation solely in hardware, which can be used to improve the ADC's SNR and its resolution (i.e. increase the effective number of bits of the ADC measurement in the presence of noise) without CPU intervention.

The effectiveness of oversampling and averaging depends on the characteristics of the dominant noise sources in the system. The rest of this section assumes that the noise can be modeled as white noise (i.e. noise with a gaussian distribution). If the noise follows a different distribution, the techniques provided may be less effective and the results may not match the given equations. Key points to consider are:

- The noise must approximate white noise with uniform power spectral density over the frequency band of interest.
- The noise amplitude must be sufficient to cause the input signal to change randomly from sample to sample by amounts comparable to at least the distance between two adjacent codes (i.e., 1 LSB)
- The input signal can be represented as a random variable that has equal probability of existing at any value between two adjacent ADC codes.

Note: Oversampling and averaging techniques will not compensate for ADC integral non-linearity (INL).

To increase the effective number of bits (ENOB), the signal is oversampled, or sampled by the ADC at a rate that is higher than the system's required sampling rate, f_s.

For each additional bit of resolution, the signal must be oversampled by a factor of four. For W additional bits of resolution, the formula for determining the required oversampling frequency f_{os} given the original sampling frequency f_s:

$$f_{\mathrm{os}} = 4^{W} \times f_s$$

Accumulating four samples to achieve one bit of resolution will result in two additional bits of data in the output word, the least of which is not significant. The result can be right-shifted once (effectively divided by two) to remove this bit. Therefore, for every n extra bits of resolution, the source must be oversampled by 4^n, but can then be divided by 2^n to remove insignificant bits. This process is called decimation.

To use the ADC's hardware to perform this oversampling and decimation to acquire one additional effective bit of resolution:

1. Write ADRPT to accumulate four samples per conversion trigger.
2. Write ADSJST to right shift the result by 1 bit. This effectively divides the output by two.

For more information on this subject, please see AN118: Improving ADC Resolution by Oversampling and Averaging.

Software Example: ADC Lib Accumulate and Average

This example demonstrates using the ADC to sample a voltage in interrupt mode, accumulating and averaging thirty-two samples in hardware to improve noise performance, increasing the effective number of bits of the result by two.

This example uses the internal temperature sensor as a voltage source. The ADC is initially configured to accumulate 32 samples and divide the result by eight (right shift by 3). A second configuration is used to sample the temperature sensor only once. 64 samples are taken with each configuration, and the average and standard deviation values (in volts) are calculated. These results are then printed out over UART.

6. Additional Resources

For more information on ADC operation and theory, please see the following application notes:

- AN119: Calculating Settling Time For Switched Capacitor ADC’s
- AN118: Improving ADC Resolution by Oversampling and Averaging

Simplicity Studio

One-click access to MCU tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

www.silabs.com/simplicity

MCU Portfolio
www.silabs.com/mcu

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com