AN14485

Using ADC in i.MX RT1180

Rev. 1.0 — 7 February 2025 Application note

Document information

Information	Content
Keywords	AN14485, i.MX RT1180, ADC, ADCK, DMA, SAR
Abstract	This document explains how to configure, implement, and use analog-to-digital converters on the NXP i.MX RT1180 microcontroller (MCU).

1 Introduction

This document explains how to configure, implement, and use analog-to-digital converters on the NXP i.MX RT1180 microcontroller (MCU).

The i.MX RT1180 MCU has two Analog-to-Digital Converter (ADC) modules: ADC1 and ADC2. Each ADC module has two successive approximation register (SAR) converter blocks: side A and side B. It means that each ADC module has a dual SAR converter, which can sample and convert (S/C) two single-ended signals at a time.

Each ADC module has eight trigger inputs, connected to the XBAR1 outputs. The signal on each trigger input can start execution of a conversion sequence. Signals on high-priority trigger inputs can interrupt conversion sequences triggered by signals on trigger inputs with lower priorities. Each ADC module has two FIFOs for storing conversion results, along with some flags. A conversion result can take up to 16 bits of a FIFO entry. Each FIFO is 32-bit wide and can store up to 16 entries.

ADC1 has 16 external channels (coming from SoC pins), and ADC2 has 14 external channels.

2 Implementing conversion sequences

In the i.MX RT1180 MCU, each ADC module has three main components (or concepts):

- **Trigger input (trigger or TRIG) ports**:
  These input ports are connected to the XBAR_OUT140 – XBAR_OUT147 signals of the i.MX RT1180 XBAR1 module. In this case, internal signals (for example, from a PWM or Timer module) or external signals (for example, from chip pins) can be used to trigger an analog-to-digital (AD) conversion conveniently. Each ADC module has eight TRIG ports: TRIG0 – TRIG7.
- **AD channels**:
  In i.MX RT1180, 16 pins can be assigned as ADC1 channels (CH0A – CH7A, CH0B – CH7B). Similarly, 14 pins can be assigned as ADC2 channels (CH0A – CH6A, CH0B – CH6B).
  Note:
  – *A channel with the suffix A belongs to the side A converter.*
  – *A channel with the suffix B belongs to the side B converter.*
- **Command buffers (CMDs)**:
  A command buffer is a 64-bit buffer that can store several configurations of an AD conversion. Each ADC module has 15 command buffers: CMD1 – CMD15. Each CMD is divided into two parts: CMDH (higher (upper) 32 bits of the CMD) and CMDL (lower 32 bits of the CMD).
  The following two register bit fields store important information related to an AD conversion:
  – CMDLm[ADCH]: The ADCH bit field of the current CMDL register indicates the AD channel to be converted.
  – CMDHm[NEXT]: The NEXT bit field of the current CMDH register indicates which CMD to use for the next conversion in the sequence, after the AD channel in the current CMD has been converted.

Each TRIG input can be associated with a CMD. When a rising edge is detected on a TRIG input port, the ADC module starts converting the AD channel contained in the CMD associated with the TRIG input port.

When the conversion of this channel completes, the ADC module moves to the next CMD pointed by the NEXT bit field of the current CMDH register, and starts converting the AD channel contained in that CMD.

The conversion process continues until the NEXT bit field value of the current CMDH register is 0. When this condition is met, the AD conversion stops. This sequence is known as the *conversion sequence* for a TRIG input port.

Figure 1 shows a conversion sequence.

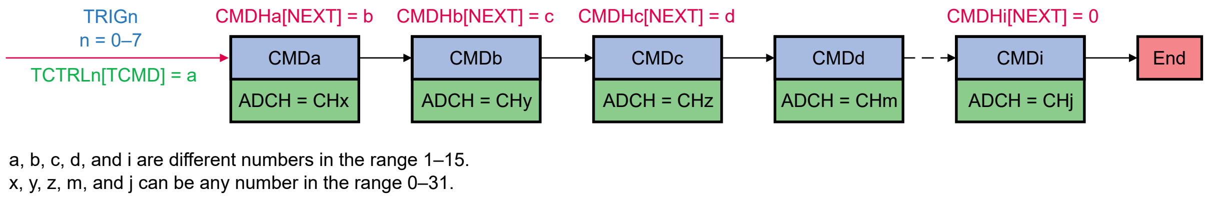

Figure 1. Conversion sequence

In Figure 1:

- TCTRLn (n = 0–7) is the register used to configure the TRIGn input. The TCTRLn[TCMD] bit field determines the first CMD associated with TRIGn. For example, TCTRL5[TCMD] = 3 indicates that when TRIG5 has a rising-edge signal, AD conversion starts with CMD3.
- CMDHm and CMDLm (m = 1–15) are CMD buffer registers, where:
  - CMDLm[ADCH] indicates the AD channel contained in CMDm. For example, CMDL3[ADCH] = 5 indicates that CMD3 contains channel CH5A, CH5B, or CH5A/CH5B pair. For more details, see Section 2.2.3.
  - CMDHm[NEXT] indicates which CMD to use after the channel in CMDm has been converted. For example, CMDH3[NEXT] = 1 indicates that CMD1 is used immediately after the channel in CMD3 has been converted.

2.1 AD conversion sequence types

Based on the value of CMDHm[NEXT], conversion sequences can be of several types, as described in the subsections that follow.

2.1.1 Sequential conversion

In a sequential conversion:

- None of the CMDs points to itself or a previously used CMD. For example, CMDH3[NEXT] = 3 indicates that CMD3 points to itself.
- The NEXT bit field value of the CMDH register in the last CMD of the sequence is 0.

A sequential conversion ends automatically after all CMDs have been executed. This execution flow is widely used for conversion sequences. Figure 2 shows an example of a sequential conversion.

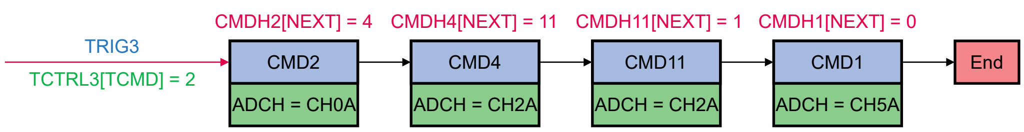

Figure 2. Sequential conversion

In Figure 2:

- If TRIG3 has a valid signal, the first CMD used is CMD2 because TCTRL3[TCMD] = 2. Because CH0A is configured in CMD2; therefore, the first converted channel in this sequence is CH0A.
- Through the configuration of the NEXT bit field of the CMDH register in each CMD, the control comes to CMD1 finally, where CMDH1[NEXT] = 0. Therefore, the conversion sequence ends execution after CH5A in CMD1 has been converted.

Each time a valid signal occurs on TRIG3, the conversion sequence described above is executed.

Note: *The order of execution for the CMDs in a conversion sequence is determined by the NEXT bit fields of the CMDH registers in the CMD buffers.*

2.1.2 Unending circular conversion

If the NEXT bit field of the CMDH register in the last CMD of a sequential conversion configuration points to the first CMD in the conversion sequence, the sequence runs indefinitely after getting triggered. This type of conversion is called *unending circular conversion*.

Figure 3 shows an example of an unending circular conversion.

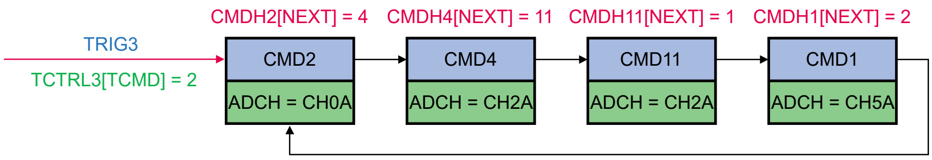

Figure 3. Unending circular conversion

In Figure 3, the last CMD is CMD1. For CMD1, the NEXT bit field value of its CMDH register (CMDH1) is 2, that is, the NEXT bit field is pointing to the first CMD in the sequence: CMD2. This sequence runs indefinitely until interrupted or aborted by a valid signal occurring on a higher-priority TRIG.

2.1.3 Continuous conversion

When a CMD points to itself (indicated by the NEXT bit field of the current CMDH register), the channel specified in the CMD is converted repeatedly until the CMD is interrupted or aborted. This type of conversion is called *continuous conversion*.

Figure 4 shows an example of a continuous conversion.

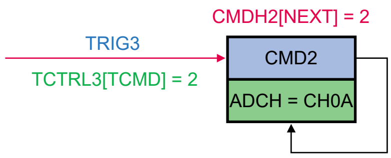

Figure 4. Continuous conversion

2.2 Configure each conversion in a sequence through CMD

In a conversion sequence, each conversion can be configured separately through a CMD. For example, the "hardware average" feature can be enabled for a channel specified in a CMD while the channels in other CMDs have this feature disabled.

Both the "hardware average" and loop features can be enabled for a channel in a CMD. In addition, some channels may need some other features, such as compare. All these settings can be configured in the CMDLm and CMDHm (m = 1–15) registers.

2.2.1 Hardware average

The "hardware average" feature can be enabled or disabled on a CMD in a conversion sequence. Figure 5 demonstrates the "hardware average" feature.

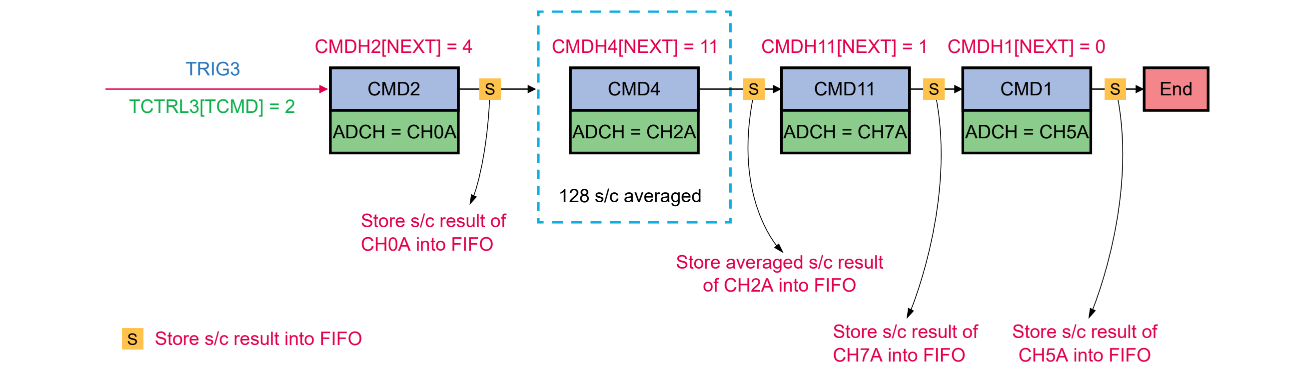

Figure 5. "Hardware average" feature demonstration

As shown in Figure 5, when conversion ends for channel CH0A in CMD2, conversion starts for CH2A in CMD4 because the NEXT bit field of the CMDH2 register points to CMD4.

Because the "hardware average" feature is enabled for CMD4 (CMDH4[AVGS] = 7), CH2A is converted 2^7 times = 128 times, and the average of the conversion results is stored in the result FIFO. For calculating the average, a channel can be converted up to 1024 times.

2.2.2 Loop feature

After a CMD is triggered in a conversion sequence, the LOOP bit field of the CMDH register (CMDHm[LOOP]) can be used to enable sampling and conversion of the channel specified in the CMD for multiple times. This feature is called *loop*.

When CMDHm[LOOP] = N, the channel in CMDm is sampled and converted N+1 times, and the result of each conversion is stored in the result FIFO. The maximum value for N is 15, it means that at most 16 loops are allowed.

One major difference between a loop and a hardware average is that in a loop, each conversion result is stored in the result FIFO, whereas in a hardware average, only the average result is stored in the FIFO. A hardware average can be considered an atomic behavior that normally cannot be interrupted and that produces only one result.

Figure 6 demonstrates the loop feature.

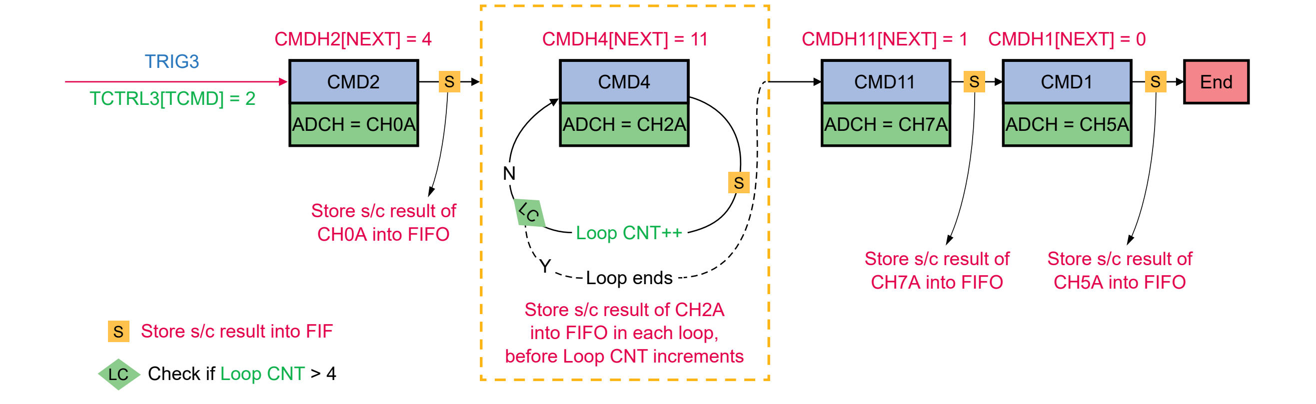

Figure 6. Loop feature demonstration

In Figure 6, the ADC module uses a hardware counter, *Loop CNT*, for counting the loop iterations. The loop feature is enabled for CMD4. Therefore, the channel specified in CMD4 is sampled and converted multiple times. After each conversion of the channel, the result is stored in the result FIFO, and the Loop CNT is incremented by 1.

When Loop CNT > CMDH4[LOOP], the loop ends and the sequence goes to CMD11. Because CMDH4[LOOP] = 4, the channel CH2A in CMD4 is sampled and converted five times, and the result of each conversion is stored in the FIFO. When CMDHm[LOOP] = 0, the channel in CMDHm is converted only once (the loop feature is disabled).

2.2.3 Conversion types

The ADC channels are named in pairs, such as CH0A/CH0B and CH1A/CH1B. For each CMD, four types of conversions can occur:

- **CMDLm[CTYPE] = 0**: Single-Ended mode with side A channel conversion
- **CMDLm[CTYPE] = 1**: Single-Ended mode with side B channel conversion.
  For this conversion type:
  - When CMDLm[ALTBEN] = 0, the channel for side B is selected by CMDLm[ADCH].
  - When CMDLm[ALTBEN] = 1, the channel for side B is selected by CMDLm[ALTB_ADCH].
- **CMDLm[CTYPE] = 2**: Differential mode. In this conversion type, the voltage between side A channel and side B channel (channel A to channel B voltage) is converted. In this case, side A and side B must have matching channel numbers, for example, CH0A/CH0B and CH1A/CH1B.
- **CMDLm[CTYPE] = 3**: Dual-Single-Ended mode. In this conversion type, both the side A and side B channels are converted simultaneously. In this case, side A and side B do not need to have matching channel numbers, such as CH1A and CH4B, CH2A and CH2B.
  For this conversion type:
  - When CMDLm[ALTBEN] = 0, the channels for side A and side B are selected by CMDLm[ADCH]. It means that side A and side B channels are paired, for example, CH0A/CH0B and CH1A/CH1B.
  - When CMDLm[ALTBEN] = 1, the channel for side A is selected by CMDLm[ADCH], whereas the channel for side B is selected by CMDLm[ALTB_ADCH]. In this case, side A and side B can have unmatched channel numbers, for example, CH1A/CH4B, CH2A/CH2B, and CH5A/CH3B.

Figure 7 shows different conversion types used for different CMDs in a conversion sequence.

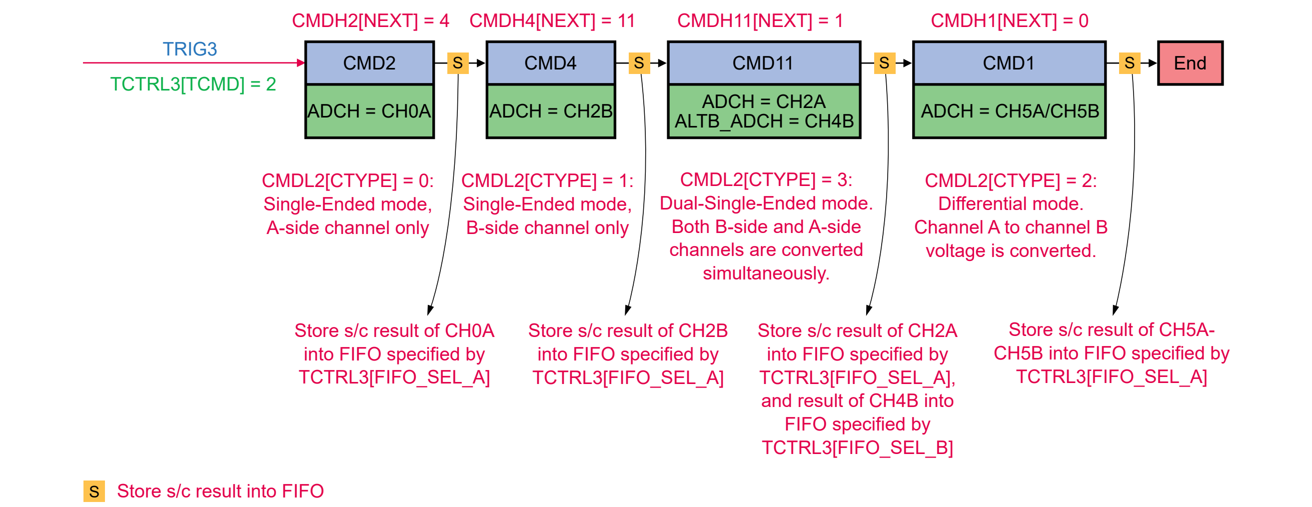

Figure 7. Conversion type demonstration

In Figure 7:

- The first CMD is CMD2, and the conversion type is Single-Ended mode with side A channel conversion (CTYPE = 0). The side A channel to be converted is CH0A.

- The second CMD is CMD4, and the conversion type is Single-Ended mode with side B channel conversion (CTYPE = 1). The side B channel to be converted is CH2B.
- The third CMD is CMD11, and the conversion type is Dual-Single-Ended mode (CTYPE = 3). The channels to be sampled and converted (simultaneously) are CH2A and CH4B. Because the channel numbers of side A and side B are unmatched; therefore:
  - CMDL11[ALTBEN] = 1
  - The channel for side B is selected by CMDL11[ALTB_ADCH].
- The last CMD is CMD1, and the conversion type is Differential mode (CTYPE = 2). In this case, the voltage difference between CH5A and CH5B is converted.

2.2.4 Result FIFO

Each ADC module has two 32-bit wide FIFOs for storing the conversion results: FIFO0 and FIFO1. Each FIFO can store 16 (32-bit) entries.

After a conversion sequence is triggered by the signal on TRIGn (n = 0–7), a FIFO is selected for storing the results of the conversion sequence by TCTRLn[FIFO_SEL_A] and TCTRLn[FIFO_SEL_B]:

- For a CMD with Dual-Single-Ended mode (CMDLm[CTYPE] = 3), the conversion result of the side A channel is stored in the FIFO specified by TCTRLn[FIFO_SEL_A], and the conversion result of the side B channel is stored in the FIFO specified by TCTRLn[FIFO_SEL_B].
  If FIFO_SEL_A = FIFO_SEL_B, channels of both side A and side B are stored in the same FIFO and the result of the side A channel is stored in the FIFO prior to the side B channel result.
- For a CMD other than Dual-Single-Ended mode (CMDLm[CTYPE] != 3), the conversion result is always stored in the FIFO specified by TCTRLn[FIFO_SEL_A].

Figure 7 shows which FIFO to use for each CMD in a conversion sequence. Because only two FIFOs are available to store all the conversion results, knowing the TRIG conversion and CMD associated with the result is necessary. Therefore, each FIFO entry contains information related to TRIGn, CMDm, and Loop CNT, along with the conversion result. It helps determine the conversion sequence, CMD, and loop time associated with the result.

Figure 8 shows the structure of a result FIFO.

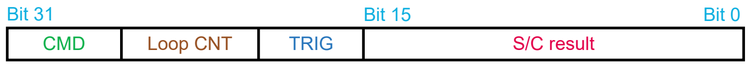

Figure 8. A result FIFO with one 32-bit entry

The lower 16 bits in a FIFO entry are used to store the conversion result. The higher (upper) 16 bits are used to store the associated TRIG number, CMD number, and the Loop CNT value. As described in Section 2.2.2, after each conversion in a loop completes and before the Loop CNT is incremented, the CMD number, Loop CNT, TRIG number, and the result are stored in the specified FIFO.

2.2.5 Channel number remains unchanged during loop if CMDHm[LWI] = 0

If CMDHm[LWI] = 0, the channel number specified by CMDHm remains unchanged during the loop. For example, if CMDH4[LWI] = 0 for the conversion sequence of Figure 6, the result FIFO has contents similar to Figure 9 (assuming that the sequence is not interrupted by a higher-priority TRIGn signal).

FIFO increases

...	...	...	...
CMD1	Loop CNT = 0	TRIG3	CH5A result
CMD11	Loop CNT = 0	TRIG3	CH7A result
CMD4	Loop CNT = 4	TRIG3	CH2A result
CMD4	Loop CNT = 3	TRIG3	CH2A result
CMD4	Loop CNT = 2	TRIG3	CH2A result
CMD4	Loop CNT = 1	TRIG3	CH2A result
CMD4	Loop CNT = 0	TRIG3	CH2A result
CMD2	Loop CNT = 0	TRIG3	CH0A result
...	...	...	...

Figure 9. A result FIFO with loop enabled and CMD4[LWI] = 0

2.2.6 Channel number is incremented during loop if CMDHm[LWI] = 1

When CMDHm[LWI] = 1, the channel number specified by CMDHm is incremented by 1 after each iteration of the loop (similar to the Loop CNT). For example, if CMDH4[LWI] = 1 for the conversion sequence of Figure 6, the result FIFO has contents similar to Figure 10 (assuming that the sequence is not interrupted by a higher-priority TRIGn signal).

FIFO increases

...	...	...	...
CMD1	Loop CNT = 0	TRIG3	CH5A result
CMD11	Loop CNT = 0	TRIG3	CH7A result
CMD4	Loop CNT = 4	TRIG3	CH6A result
CMD4	Loop CNT = 3	TRIG3	CH5A result
CMD4	Loop CNT = 2	TRIG3	CH4A result
CMD4	Loop CNT = 1	TRIG3	CH3A result
CMD4	Loop CNT = 0	TRIG3	CH2A result
CMD2	Loop CNT = 0	TRIG3	CH0A result
...	...	...	...

Figure 10. A result FIFO with loop enabled and CMD4[LWI] = 1

2.2.7 Enable loop and hardware average on same CMD

When CMDHm[AVGS] = M (where M != 0) and CMDHm[LOOP] = N (where N != 0), CMDm is executed N+1 times in a loop. For each iteration of the loop, the channel specified by CMDm is sampled and converted 2^M times and the average of the results is stored in the result FIFO.

In Figure 11, when the control comes to CMD4, because CMDH4[AVGS] = 7 and CMDH4[LOOP] = 4, CMD4 is executed 5 times due to the loop.

In each iteration during the loop, the channel CH2A specified by CMD4 is sampled and converted 2^7 times = 128 times and the average of the results is stored in the result FIFO. In the entire loop, the channel CH2A is sampled and converted 5 x 128 times = 640 times.

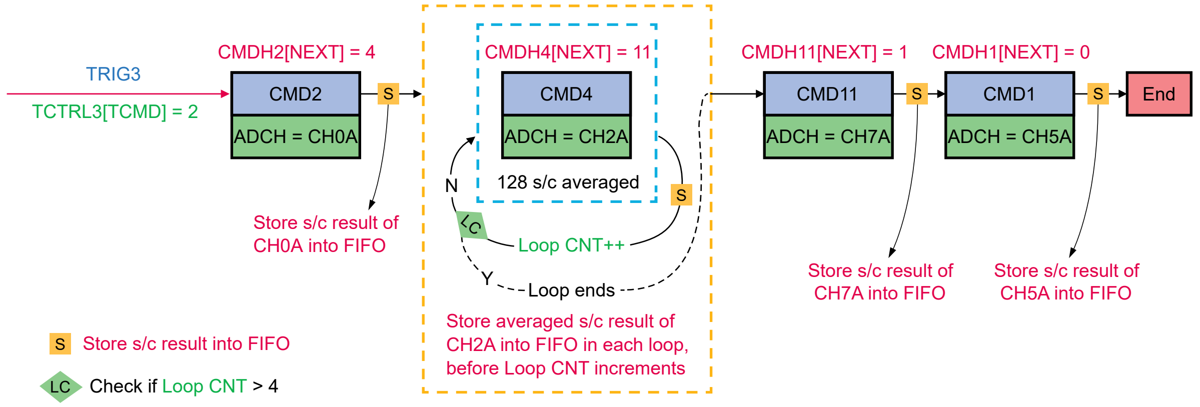

Figure 11. Loop and hardware average enabled on same CMD

Figure 12 shows the conversion results of Figure 11 stored in the result FIFO (assuming that this sequence is not interrupted by a higher-priority TRIGn signal). CMD4 occupies five FIFO entries with each entry storing an average of 128 conversion results.

FIFO increases ↑

...	...	...	...
CMD11	Loop CNT = 0	TRIG3	CH7A result
CMD4	Loop CNT = 4	TRIG3	128 averaged CH2A result
CMD4	Loop CNT = 3	TRIG3	128 averaged CH2A result
CMD4	Loop CNT = 2	TRIG3	128 averaged CH2A result
CMD4	Loop CNT = 1	TRIG3	128 averaged CH2A result
CMD4	Loop CNT = 0	TRIG3	128 averaged CH2A result
CMD2	Loop CNT = 0	TRIG3	CH0A result
...	...	...	...

Figure 12. A result FIFO with loop and hardware average enabled

2.2.8 Wait for a trigger signal during execution of a sequence

Sometimes, after getting triggered, a conversion sequence is required to be paused at some channel. The execution can be resumed from the paused channel later at a specific time point. This goal can be achieved by enabling the "wait for trigger" feature for a CMD containing the channel that has to be paused.

To enable the feature, configure the WAIT_TRIG bit field of the CMDHm register (CMDHm[WAIT_TRIG] = 1). Figure 13 demonstrates the "wait for trigger" feature.

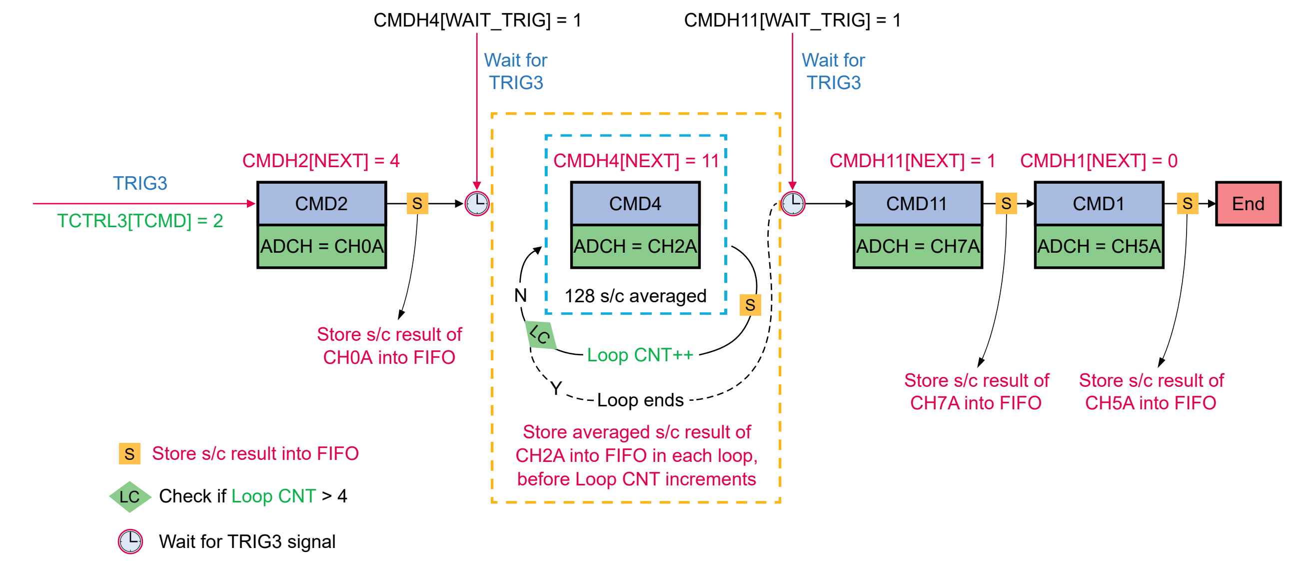

Figure 13. "Wait for trigger" feature demonstration

In Figure 13, a valid signal occurring on TRIG3 triggers a conversion sequence, which contains CMD2, CMD4, CMD11, and CMD1. Because CMDH4[WAIT_TRIG] = 1 and CMDH11[WAIT_TRIG] = 1:

- The conversion sequence pauses execution after CMD2, and it waits for a valid trigger signal on TRIG3 to resume execution from CMD4.
- After CMD4 is executed (including hardware average and loop), the sequence pauses execution again, and it waits for a valid trigger signal on TRIG3 to resume execution from CMD11.
- After CMD11 is executed, the sequence continues execution with CMD1. With CMD1 execution, the conversion sequence completes execution.

2.2.9 Compare feature

For CMD1–CMD4, a feature, called *compare*, is available. It works when the CMPEN bit field of a CMDHa register (where 'a' = 1–4) is set to 2 or 3.

When the compare feature is enabled for a CMD and the channel conversion (including hardware average) has been completed, the hardware compares the conversion result with the values in the CVa[CVH] and CVa[CVL] bit fields. Based on the comparison result, a decision is taken on whether to store the conversion result in the result FIFO.

Depending on the CMDHa[CMPEN] value, the compare feature has two variants:

- **Compare with "store on true" option (CMDHa[CMPEN] = 2)**: In this variant, the conversion result is stored in the result FIFO if the comparison result is true:
  - When the comparison result is false, the hardware discards the conversion result (does not store it in the result FIFO), and increments the loop counter.
  - When the comparison result is true, the hardware stores the conversion result in the result FIFO, and increments the loop counter.

  Note: *The loop counter is incremented after each conversion, irrespective of the comparison result.*
- **Compare with "repeat until true" option (CMDHa[CMPEN] = 3)**: In this variant, the compare operation is repeated until the comparison result is true:
  - When the comparison result is false, the hardware discards the conversion result (does not store it in the result FIFO). The loop counter is kept unchanged, and the channel inside the current CMD is converted again.

– When the comparison result is true, the hardware stores the conversion result in the result FIFO, and increments the loop counter.

In Figure 14, the compare feature is enabled for CMD4 with the "store on true" option (CMDH4[CMPEN] = 2), along with the "hardware average" and loop features.

In each iteration of the loop, an average of 128 conversion/sampling results is calculated, and a comparison operation is performed. Irrespective of the outcome of the comparison operation, the loop counter is always incremented by 1. Only when the outcome of the comparison operation is true, the average of the 128 conversion results is stored in the result FIFO.

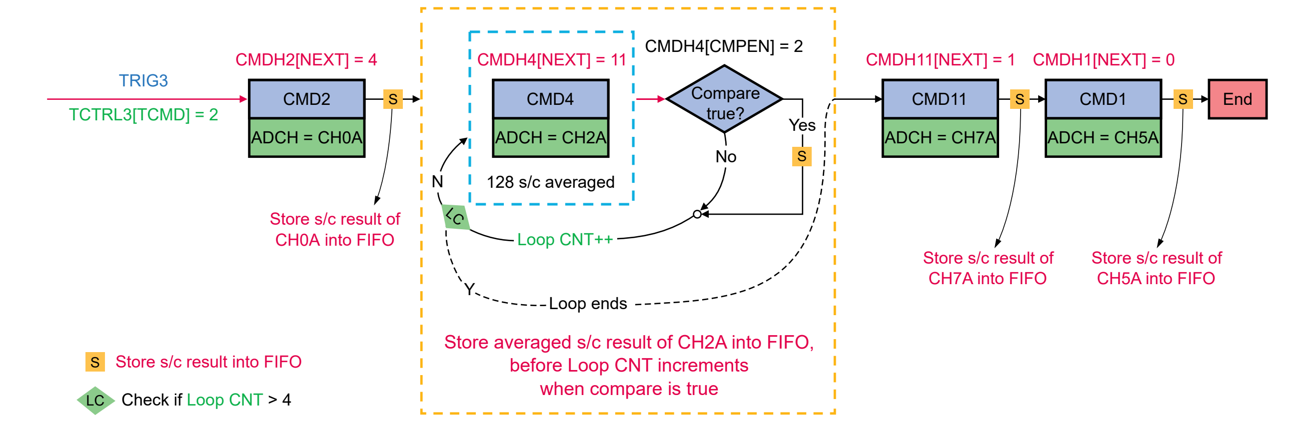

Figure 14. Compare feature with "store on true" option

In Figure 15, the compare feature is enabled for CMD4 with the "repeat until true" option (CMDH4[CMPEN] = 3), along with the "hardware average" and loop features.

If the comparison result is false for the average of the 128 conversion results, the loop counter remains unchanged, and channel conversion starts again for 128 times. Only when the comparison result is true, the averaged result is stored in the result FIFO, and the loop counter is incremented.

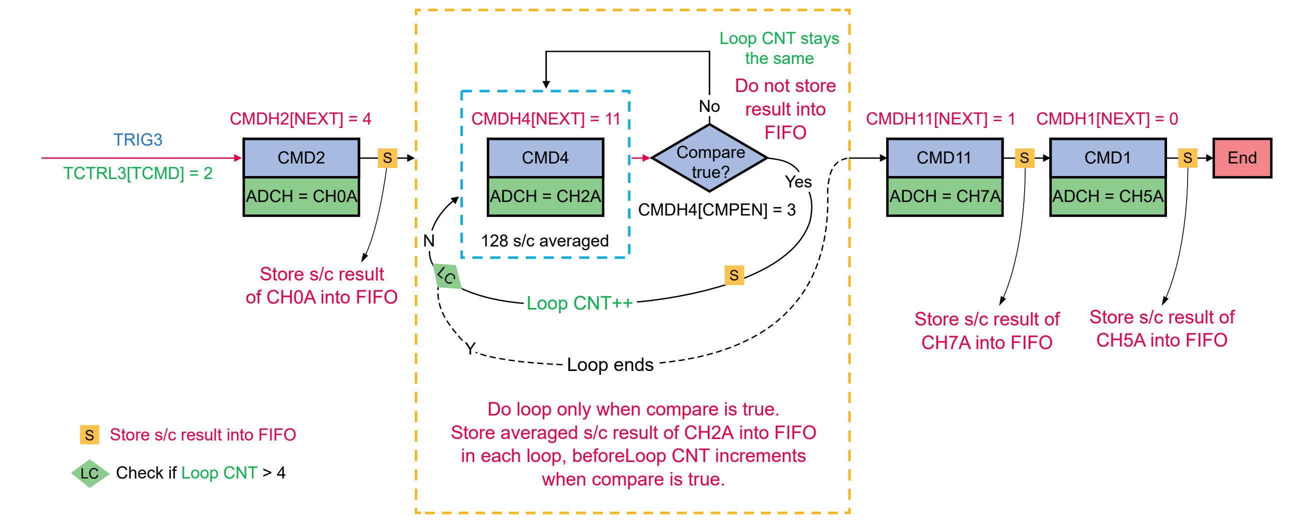

Figure 15. Compare feature with "repeat until true" option

In Dual-Single-Ended mode (CMDLm[CTYPE] = 3), the comparisons are made only for the conversion results of channels with suffix A.

A comparison result determines whether to store the current pair of conversion results in the result FIFO:

- If the comparison result for a channel with suffix A is true, the conversion results of both side A and side B channels are stored in the result FIFO. The loop behavior is determined by CMDHa[CMPEN].
- If the comparison result for a channel with suffix A is false, neither of the conversion results (side A channel or side B channel) is stored in the result FIFO. The loop behavior is determined by CMDHa[CMPEN].

2.2.9.1 Interpreting a comparison result

Based on the values in CVa[CVL] and CVa[CVH], the following four scenarios are possible:

- **min. value < CVL < CVH < max. value**: The comparison result is true when the conversion result is less than CVL **OR** greater than CVH.
- **min. value < CVL < CVH = max. value**: The comparison result is true when the conversion result is less than CVL.
- **min value = CVL < CVH < max. value**: The comparison result is true when the conversion result is greater than CVH.
- **CVL > CVH**: The comparison result is true when the conversion result is less than CVL **AND** greater than CVH.

2.2.10 Conversion result format

For each CMD, the conversion result can be configured in one of the following formats by setting the CMDLm[MODE] bit field value:

- **12-bit / 13-bit format (CMDLm[MODE] = 0)**:
  - For the Single-Ended mode conversion type, a 12-bit conversion result format is used.
  - For the Differential mode conversion type, a 13-bit conversion result format is used with 2's complement output.
- **16-bit format (CMDLm[MODE] = 1)**:
  - For the Single-Ended mode conversion type, a 16-bit conversion result format is used.
  - For the Differential mode conversion type, a 16-bit conversion result format is used with 2's complement output.

Note: *In Differential mode, the conversion result is stored in the result FIFO in 2's complement format.*

Figure 16 shows example conversion results in a result FIFO when CMDLm[MODE] = 0.

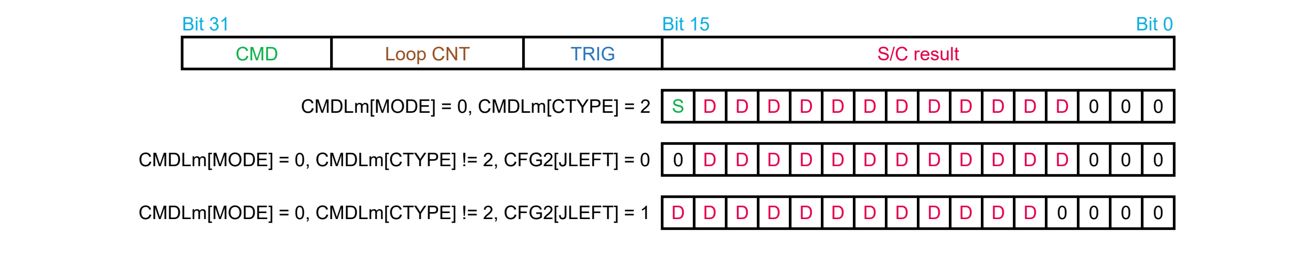

Figure 16. A result FIFO with 12-bit / 13-bit conversion result

In Figure 16, 'D' represents a data bit and 'S' represents a sign bit.

Figure 17 shows example conversion results a result FIFO when CMDLm[MODE] = 1.

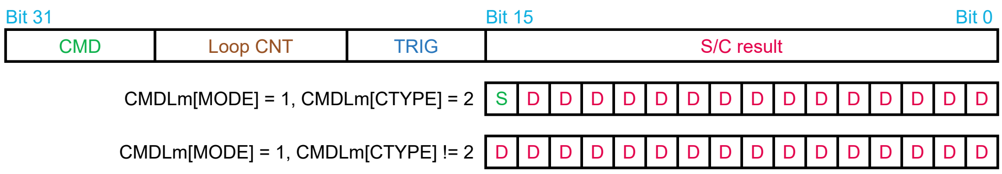

Figure 17. A result FIFO with 16-bit conversion result

In Figure 17, 'D' represents a data bit and 'S' represents a sign bit.

2.2.11 Channel scaling

For the ADC modules, the reference voltage is usually 1.8 V. When the input signal on a channel is at a voltage higher than 1.8 V, the signal must be reduced by a scale. To achieve this goal, the "channel scaling" feature can be used. This feature is available for all CMDs.

The "channel scaling" feature can be configured for the channel of a CMD using the CSCALE bit field of the CMDL register (CMDLm[CSCALE]), as follows:

- **CMDLm[CSCALE] = 1**: Full scale, no reduction in channel signal
- **CMDLm[CSCALE] = 0**: Scale the channel with a factor of ½, the channel signal is halved.
  - When CMDLm[ALTBEN] = 0, CMDLm[CSCALE] controls both side A and side B channels for the CMD.
  - When CMDLm[ALTBEN] = 1, CMDLm[CSCALE] controls the side A channel for the CMD, whereas CMDLm[ALTB_CSCALE] controls the side B channel for the CMD.

Note: *When the i.MX RT1180 MCU is out of reset, CMDLm[ALTB_CSCALE] = 1 and CMDLm[CSCALE] = 1. It indicates that no signal reduction happens for the selected channel.*

2.3 Add a delay between two conversions in a sequence through PAUSE register

When PAUSE[PAUSEEN] = 1, a delay occurs between two consecutive AD conversions in a conversion sequence. The delay is 4 x PAUSE[PAUSEDLY] ADC clock cycles.

Figure 18 shows where the delay caused by the PAUSE register is located in a sequence, with the "store on true" compare option configured for CMD4 (CMDH4[CMPEN] = 2).

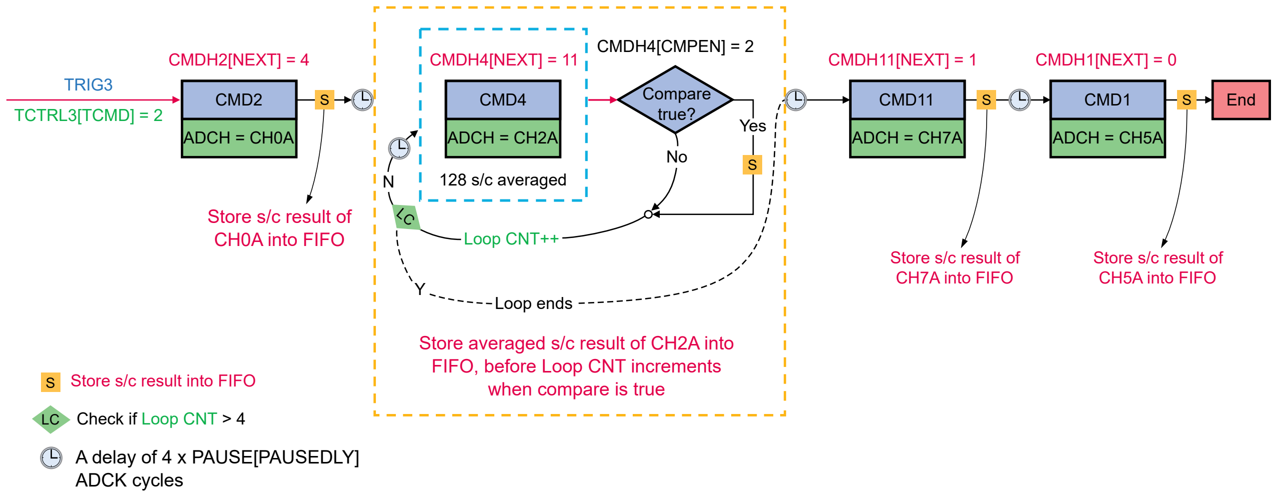

Figure 18. Using PAUSE register for delay with "store on true" compare

Figure 19 shows where the delay caused by the PAUSE register is located in a sequence, with the "repeat until true" compare option configured for CMD4 (CMDH4[CMPEN] = 3).

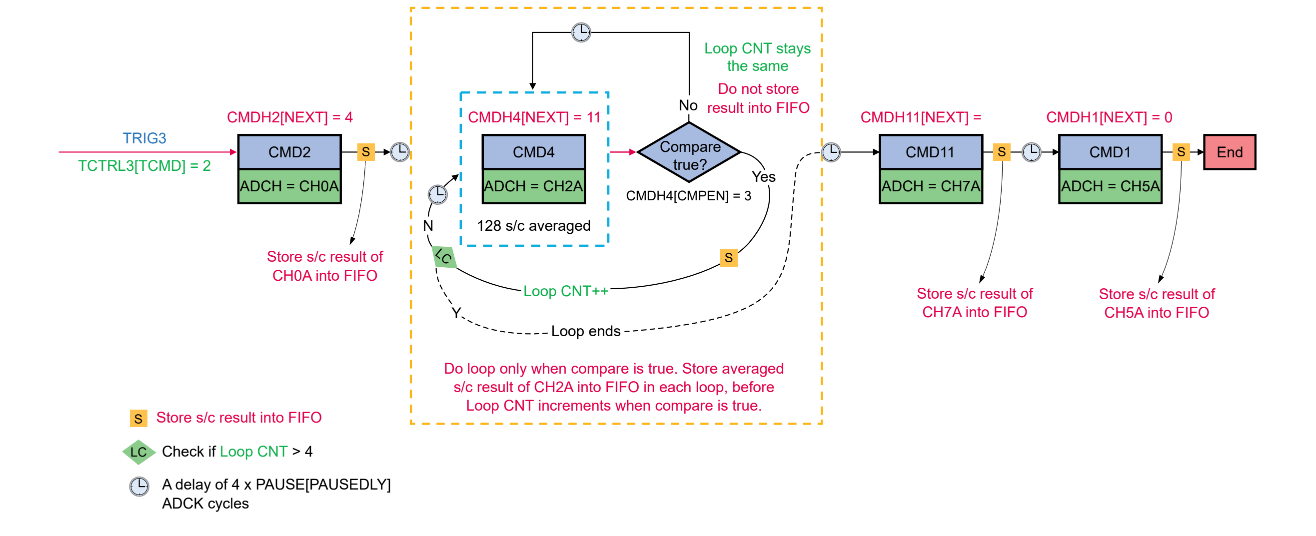

Figure 19. Using PAUSE register for delay with "repeat until true" compare

2.3.1 PAUSE register does not work with "wait for trigger" feature

Both the "wait for trigger" feature and the PAUSE register are used to add delays in a conversion sequence. If both are used together (CMDHm[WAIT_TRIG] = 1 and PAUSE[PAUSEEN] = 1) on a CMD of a conversion sequence, only the "wait for trigger" feature makes an impact, the PAUSE register has no impact.

Figure 20 shows the conversion sequence of Figure 18 modified by enabling the "wait for trigger" feature for CMD4 and CMD1.

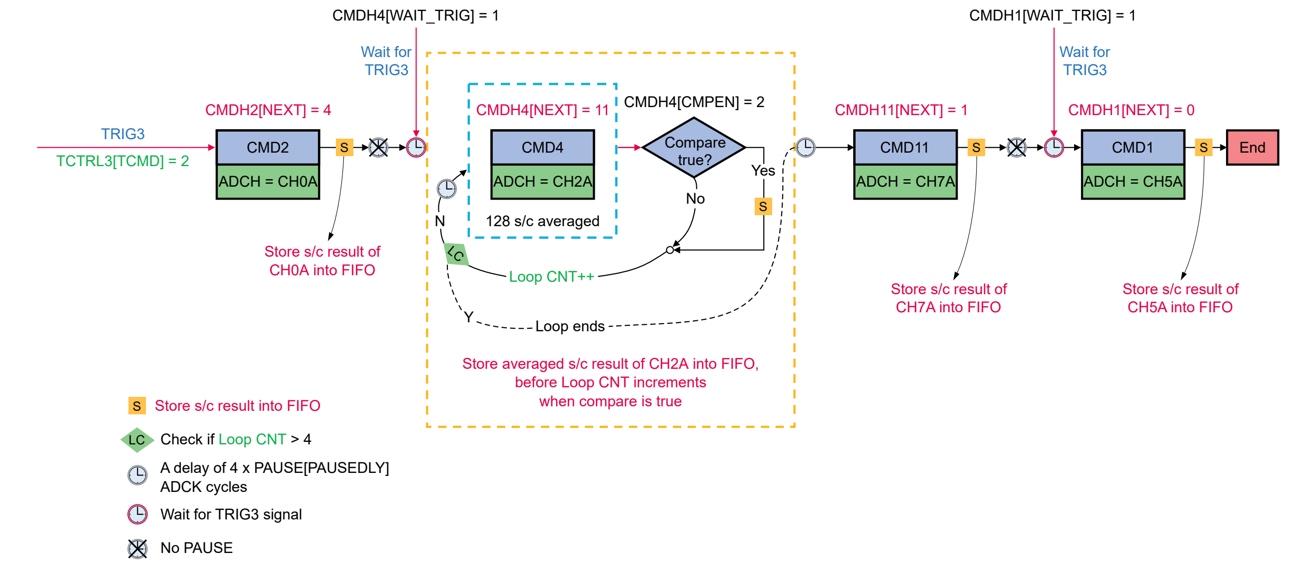

Figure 20. PAUSE register has no impact when enabled with "wait for trigger" feature

After completion of CMD2, the sequence pauses and waits for another valid signal on TRIG3 to continue execution with CMD4. The delay configured with the PAUSE register is ignored.

After completion of CMD4, a time delay caused by the PAUSE register occurs before CMD11 is executed. After CMD11 execution, the sequence pauses and waits for another valid signal on TRIG3 to continue execution with CMD1. The delay configured with the PAUSE register is ignored.

2.4 Configure whole sequence through trigger control register TCTRLn

A Trigger Control register (TCTRLn) can be used to configure the following settings for a conversion sequence:

- CMD to be used for the first AD conversion in the sequence
- Maximum delay allowed between the trigger signal and the first AD conversion
- Priority of the sequence
- FIFO to be used for storing the conversion results
- Whether a hardware signal can trigger the sequence

The different bit fields of the TCTRLn register serve the following purposes:

- **TCTRLn[TCMD]**: Determines the first CMD in a conversion sequence triggered by TRIGn. In Figure 2, TCTRL3[TCMD] = 2, it means that the channel specified in CMD2 is the first channel to be converted for the conversion sequence triggered by TRIG3.
  When TCTRLn[TCMD] = 0, the TRIGn does not trigger any conversion sequence.
- **TCTRLn[TDLY]**: Indicates the delay between the occurrence of a valid trigger signal on TRIGn and the execution of the first CMD in the conversion sequence.
  The delay is 2^{TDLY} ADC clock (ADCK) cycles:
  - For the ADC1 module, ADCK refers to adc1_clk_root, which is controlled by CLOCK_ROOT44 related registers.
  - For the ADC2 module, ADCK refers to adc2_clk_root, which is controlled by CLOCK_ROOT45 related registers.

  When TCTRLn[TDLY] = 0, no such delay occurs.
  The ADC1 and ADC2 root clocks can be configured in Config Tool, as shown in Figure 21.

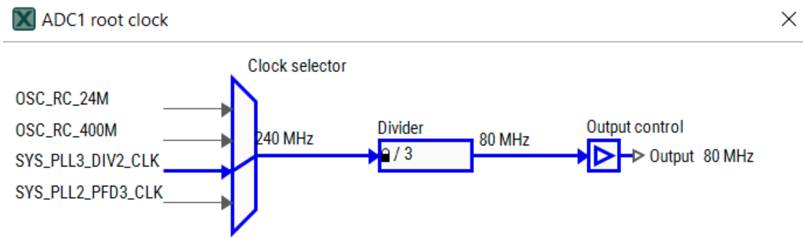

Figure 21. ADC1 root clock configuration in Config Tool

- **TCTRLn[TPRI]**: Indicates the priority of the conversion sequence triggered by the TRIGn signal. Eight priority levels are available: from level 1 (highest, TCTRLn[TPRI] = 0) to level 8 (lowest, TCTRLn[TPRI] = 7).
  If two or more trigger signals have the same priority level, the trigger event with the smallest number takes preference. For example, when TCTRL0[TPRI] = TCTRL1[TPRI], TRIG0 takes preference over TRIG1.
- **TCTRLn[FIFO_SEL_A] and TCTRLn[FIFO_SEL_B]**: These bit fields help determine which result FIFO to use for the sequence associated with the TRIGn signal. For more details, see Section 2.2.4.
- **TCTRLn[HTEN]**: Controls whether a hardware signal can trigger a conversion sequence:
  - If TCTRLn[HTEN] = 1, a rising-edge signal on TRIGn triggers the associated conversion sequence.
  - If TCTRLn[HTEN] = 0, signals on TRIGn do not trigger any conversion sequence. However, the sequence associated with TRIGn can be triggered manually by configuring the SWTRIG[SWTn] bit field, regardless of the TCTRLn[HTEN] bit field value.

2.4.1 Conversion sequence priority

Each ADC module has eight TRIG inputs, which can be configured with a priority level from 1 to 8. When multiple conversion sequences associated with different TRIGn are triggered at the same time, the conversion sequence associated with the highest priority TRIGn takes preference over other sequences.

2.4.1.1 Trigger exceptions are disabled if CFG[HPT_EXDI] = 1

The ADC modules support a feature, called *trigger exception*, which allows a higher-priority trigger signal to interrupt an ongoing lower-priority conversion sequence. This feature is similar to the nested interrupt of the CPU core.

The "trigger exception" feature can be configured using the CFG[HPT_EXDI] bit field:

- If CFG[HPT_EXDI] = 0 (default setting), the conversion sequence associated with a higher-priority TRIGn signal can interrupt a sequence associated with a lower priority.
- If CFG[HPT_EXDI] = 1, a conversion sequence with a higher-priority TRIGn signal cannot interrupt any sequence with a lower priority.

2.4.2 Breakpoint options for lower-priority sequences

When the trigger exception feature is enabled (CFG[HPT_EXDI] = 0), a valid TRIGn signal with a higher priority can interrupt an ongoing lower-priority conversion sequence, making its associated conversion sequence begin execution. The lower-priority conversion sequence may also contain a hardware average or loop.

The point where the lower-priority sequence is interrupted (breakpoint) is decided based on the CFG[TPRICTRL] bit field value:

- **If CFG[TPRICTRL] = 0**: If a higher-priority TRIGn signal occurs, the ongoing lower-priority sequence is aborted immediately (without waiting for the ongoing conversion to complete). The sequence associated with the higher-priority TRIGn signal starts execution after 10–11 ADC clock cycles from the time when the higher-priority TRIGn signal occurred.
  For more details on the CFG[TPRICTRL] = 0 use case, see Section 2.4.3.
- **If CFG[TPRICTRL] = 1**: If a higher-priority TRIGn signal occurs, the ongoing lower-priority sequence is aborted only after the currently executing CMD (including the hardware average but not the loop) completes. Then, the sequence associated with the higher-priority TRIGn signal starts execution.
  For more details on the CFG[TPRICTRL] = 1 use case, see Section 2.4.4.
- **If CFG[TPRICTRL] = 2**: If a higher-priority TRIGn signal occurs, the ongoing lower-priority sequence is aborted only after the currently executing CMD (including both hardware average and loop) completes. Then, the sequence associated with the higher-priority TRIGn signal starts execution.
  For more details on the CFG[TPRICTRL] = 2 use case, see Section 2.4.5.

2.4.2.1 Resumption options for interrupted lower-priority sequences

The resumption of a lower-priority sequence that was interrupted earlier by a higher-priority TRIGn signal depends on the CFG[TRES] bit field value:

- **If CFG[TRES] = 0**: The interrupted lower-priority sequence does not resume execution or does not restart.
  If a trigger exception interrupt is enabled (IE[TEXC_IE] = 1):
  - The STAT[TEXC_INT] flag can be used to determine if any trigger sequence is interrupted.
  - The TSTAT[TEXC_NUM] flag can be used to determine which trigger sequence is interrupted.
- **If CFG[TRES] = 1**: The interrupted lower-priority sequence resumes execution after the higher-priority sequence completes execution.

The resumption point is determined by the CFG[TCMDRES] bit field value:

– **If CFG[TCMDRES] = 0**: The lower-priority sequence resumes execution from the first CMD of the sequence. It means that the lower-priority sequence is restarted.
– **If CFG[TCMDRES] = 1**: The lower-priority sequence resumes execution from the CMD that was executing when the sequence was interrupted. If the interrupted CMD contains a loop, the loop starts again.

2.4.3 Exception triggering with breakpoint option CFG[TPRICTRL] = 0 and resumption option CFG[TRES] = 0

In Figure 22, TRIG6 has a higher priority than TRIG5. A valid signal occurring on TRIG6 aborts the ongoing conversion sequence of TRIG5 immediately because CFG[TPRICTRL] = 0. Then, the sequence of TRIG6 starts execution.

When the TRIG6 sequence completes execution, the sequence of TRIG5 does not resume execution because CFG[TRES] = 0.

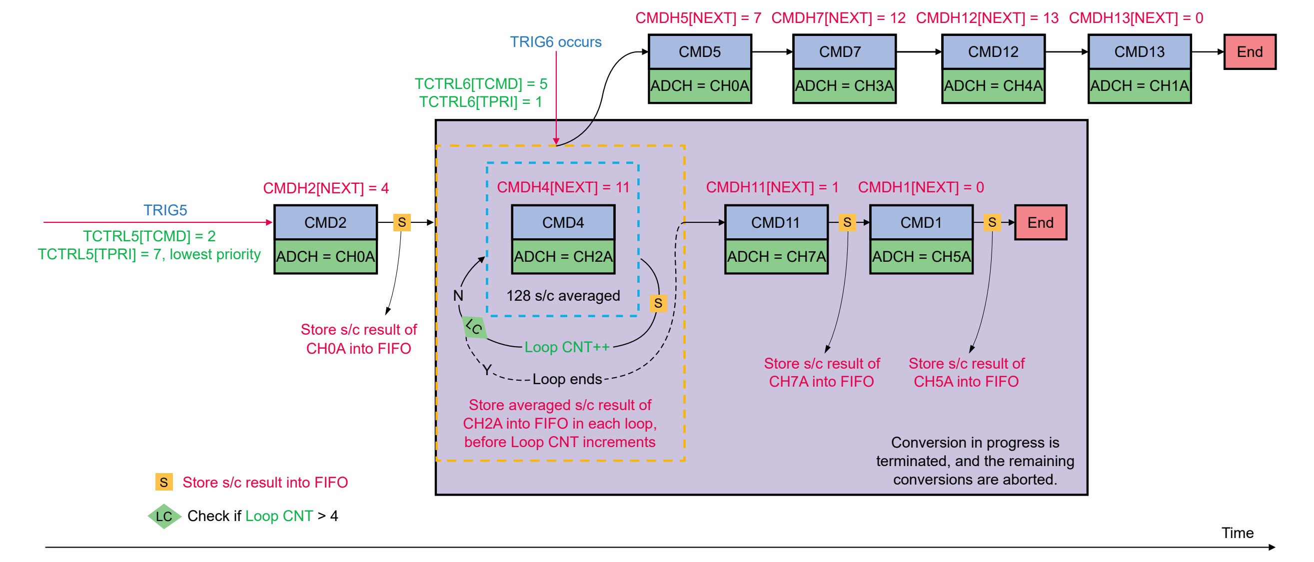

Figure 22. A higher-priority trigger signal aborts a lower-priority sequence immediately

2.4.4 Exception triggering with breakpoint option CFG[TPRICTRL] = 1 and resumption options CFG[TRES] = 1 and CFG[TCMDRES] = 1

In Figure 23, TRIG6 has a higher priority than TRIG5. A valid signal occurs on TRIG6 when CMD4 is executed in the TRIG5 sequence. Because CFG[TPRICTRL] = 1; therefore, the sequence of TRIG6 waits for a hardware average in CMD4 of the TRIG5 sequence to complete, before starting its execution. However, it does not wait for the loop in CMD4 to complete.

After completion of the TRIG6 sequence, the TRIG5 sequence resumes execution automatically from CMD4 (which was executing when the sequence was interrupted) (if no other higher-priority sequence is pending) because CFG[TRES] = 1 and CFG[TCMDRES] = 1. In this case, the loop on CMD4 is restarted.

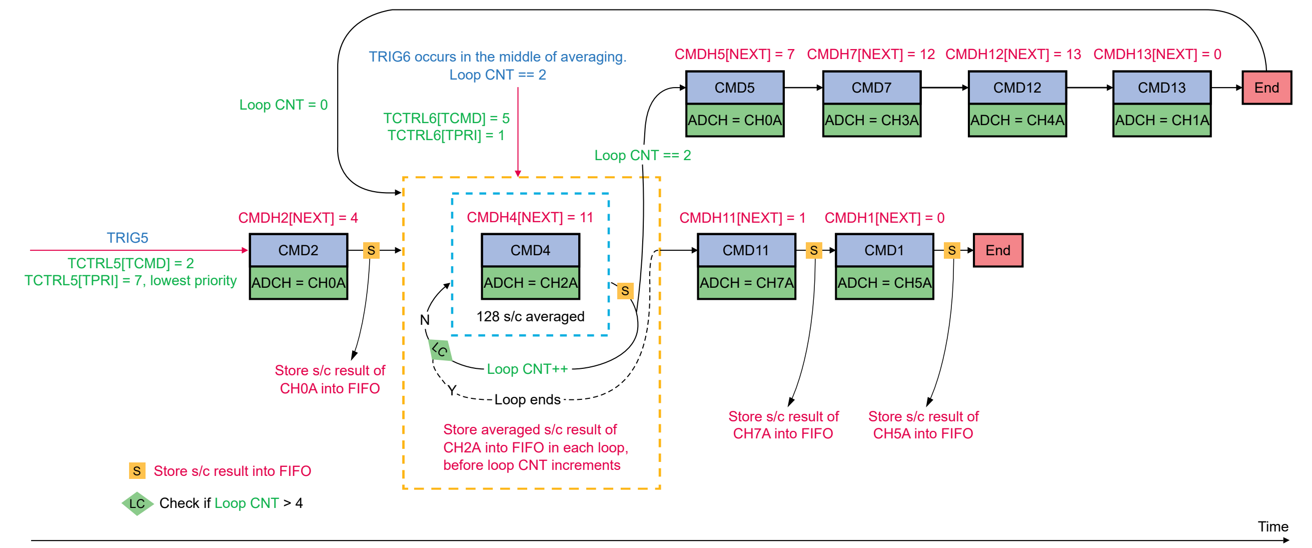

Figure 23. A higher-priority trigger signal aborts a lower-priority sequence after completion of hardware average in current CMD

Figure 24 shows the result FIFO entries when the sequence of TRIG5 is interrupted and when it is resumed.

FIFO increases

...	...	...	...
CMD11	Loop CNT = 0	TRIG5	CH7A result
CMD4	Loop CNT = 4	TRIG5	128 averaged CH2A result
CMD4	Loop CNT = 3	TRIG5	128 averaged CH2A result
CMD4	Loop CNT = 2	TRIG5	128 averaged CH2A result
CMD4	Loop CNT = 1	TRIG5	128 averaged CH2A result
CMD4	Loop CNT = 0	TRIG5	128 averaged CH2A result
CMD4	Loop CNT = 2	TRIG5	128 averaged CH2A result
CMD4	Loop CNT = 1	TRIG5	128 averaged CH2A result
CMD4	Loop CNT = 0	TRIG5	128 averaged CH2A result
CMD2	Loop CNT = 0	TRIG5	CH0A result
...	...	...	...

TRIG5 sequence resumes

TRIG6 sequence

Interrupted by TRIG6

Figure 24. A result FIFO showing interruption and resumption of a conversion sequence when CFG[TCMDRES] = 1

2.4.5 Exception triggering with breakpoint option CFG[TPRICTRL] = 2 and resumption options CFG[TRES] = 1 and CFG[TCMDRES] = 1

In Figure 25, TRIG6 has a higher priority than TRIG5. A valid signal occurs on TRIG6 when CMD4 is executed in the TRIG5 sequence. Because CFG[TPRICTRL] = 2, the sequence of TRIG6 starts execution after CMD4 of the TRIG5 sequence (including hardware average and loop) completes.

After the TRIG6 sequence completes execution, the TRIG5 sequence resumes execution automatically from CMD11 (if no other higher-priority sequence is pending) because CFG[TRES] = 1 and CFG[TCMDRES] = 1. In this case, the TRIG5 sequence resumes execution from CMD11 because CMD4 has been executed already.

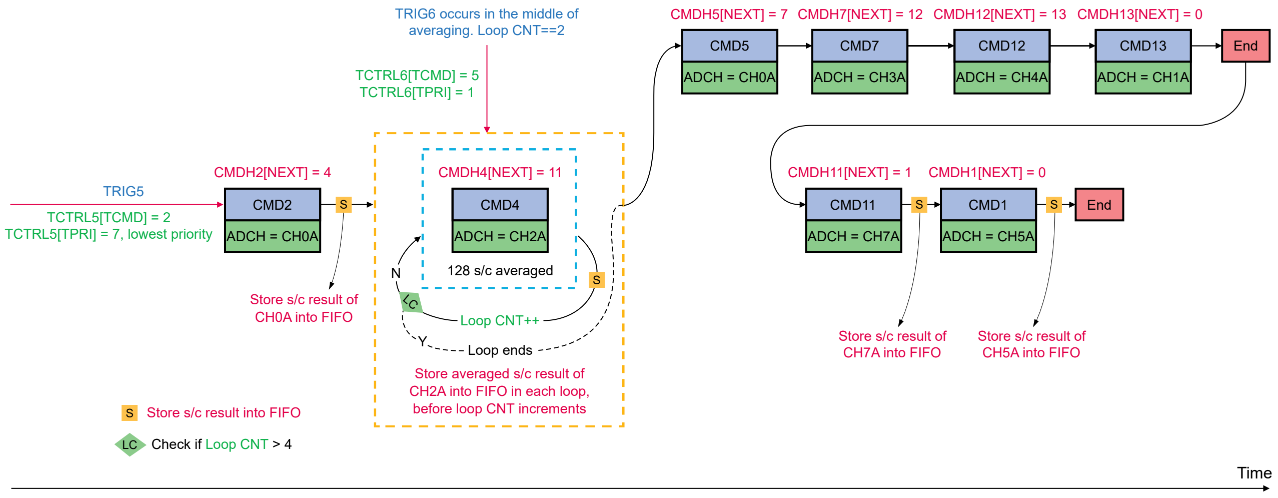

Figure 25. A higher-priority trigger signal aborts a lower-priority sequence after completion of hardware average and loop in current CMD

2.4.6 Trigger delay and power-up delay

As mentioned in Section 2.4, the TCTRLn[TDLY] bit field can be used to configure a delay between the occurrence of a valid trigger signal on TRIGn and the execution of the first CMD in the conversion sequence.

Besides this initial delay (trigger delay) for a conversion sequence, a power-up delay may be needed for the ADC module before it can start executing the sequence. The power-up delay is 4 x CFG[PUDLY] ADC clock cycles. The ADC power-up delay can be configured using the CFG[PWREN] bit field:

- If CFG[PWREN] = 0, the ADC module is enabled only when it performs a conversion. It means that a power-up delay is required each time the ADC module starts executing a sequence.
- If CFG[PWREN] = 1, the ADC module is always enabled, after it is enabled once by setting the CFG[PWREN] bit field value.
  To get enabled initially, the ADC module requires a power-up delay from the time the CFG[PWREN] bit field is set to 1. After that, the ADC module is always enabled. In this case, the ADC module begins execution of a conversion sequence immediately after a valid signal occurs on the associated TRIGn (assuming that TCTRLn[TDLY] = 0).

Figure 26 demonstrates the trigger delay and the power-up delay.

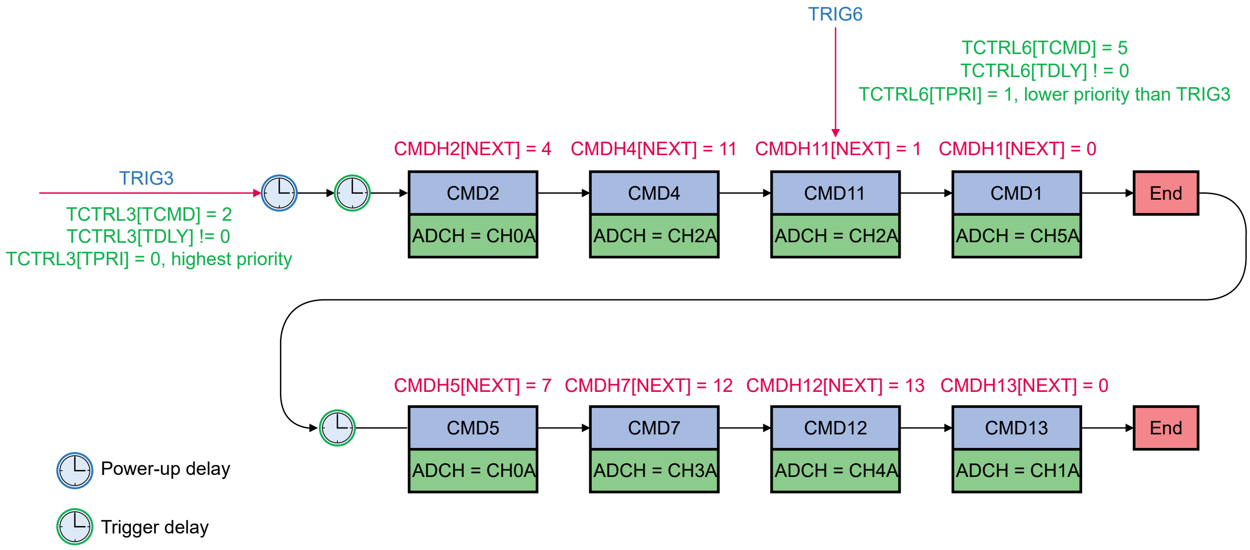

Figure 26. Initial trigger delay and power-up delay

In Figure 26, a valid signal occurs on TRIG3 immediately after the CFG[PWREN] bit field is set to 1. Therefore, a power-up delay (due to the ADC module power up) and a trigger delay (because TCTRL3[TDLY] != 0) occur before execution begins for the sequence associated with TRIG3.

A valid signal with a lower priority occurs on TRIG6 during the execution of the TRIG3 sequence. As the priority of the TRIG6 signal is lower than the priority of the TRIG3 signal, it waits for the TRIG3 sequence to complete execution. Also, because TCTRL6[TDLY] != 0, a trigger delay occurs at the beginning of the TRIG6 sequence.

3 ADC reference voltage

In the i.MX RT1180 MCU, the reference voltage for the ADC is selected based on the CFG[REFSEL] bit field value:

- If CFG[REFSEL] = 0, ADC uses the voltage from its VREFH pad as a reference. The VREF_OUT pad from the VREF module and the VREFH pad from the ADC module are bonded on the same pin (ADC_VREFH) of the SoC package.
  Note: *When using VREFH as a reference, ensure that the ADC clock frequency is less than 20 MHz. Otherwise, ADC performance described in the i.MX RT1180 data sheet cannot be guaranteed.*
- If CFG[REFSEL] = 1 or 2, ADC uses the voltage from the VDDA_ADC_1P8 pin as a reference.

To receive ADC reference voltage through the VREFH pad, use one of the following options:

Option 1: ADC reference voltage from an internal source

In this option, the ADC reference voltage is received from an internal source (VREF module), as shown in Figure 27. To use this option, the VREF module must be configured and enabled.

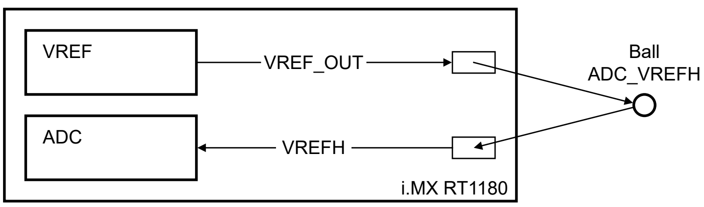

Figure 27. Using internal VREF for ADC reference

Option 2: ADC reference voltage from an external source

In this option, the ADC reference voltage is received from an external source, such as an LDO regulator, as shown in Figure 28. To use this option, disable the VREF module.

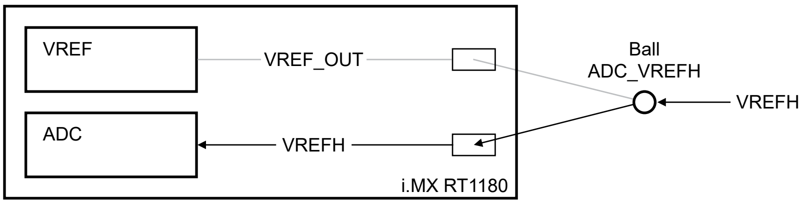

Figure 28. Using external VREFH for ADC reference with internal VREF disabled

4 Sampling+conversion time for a CMD channel

The time taken for sampling and AD conversion can be different for different CMD channels. The sampling time depends on the CMDHm[STS] bit field value, whereas the conversion time depends on the format used for the conversion result (12-bit / 13-bit format or 16-bit format).

Table 1 shows the total sampling+conversion time for a CMD channel with different sampling and conversion settings.

Table 1. Sampling+conversion time

CMDHm[STS] value	Conversion result format (CMDLm[MODE] value)	Total sampling+conversion time (in ADC clock cycles)
CMDHm[STS] = 0	12-bit / 13-bit format (CMDLm[MODE] = 0)	3.5 + 15.5
	16-bit format (CMDLm[MODE] = 1)	3.5 + 20.5
CMDHm[STS] > 0	12-bit / 13-bit format (CMDLm[MODE] = 0)	$3.5 + 2^{STS} + 15.5$
	16-bit format (CMDLm[MODE] = 1)	$3.5 + 2^{STS} + 20.5$

5 ADC interrupts

In the i.MX RT1180 MCU, an ADC module can trigger an interrupt in the following scenarios:

- **When the result FIFOn (n = 0 or 1) contains data, and the number of valid data words is greater than the watermark level set in the FCTRL[FWMARK] bit field**: This interrupt is enabled when the IE[FWMIEn] bit field is set to 1. When the number of valid data words is greater than the watermark (the FCTRL[FWMARK] bit field value), a 1-bit flag (the STAT[RDYn] bit field) is set to 1; otherwise, the flag is cleared. The STAT[RDYn] bit field cannot be set or cleared through the software.
  Note: *If the number of data words in the result FIFO is equal to the watermark, the STAT[RDYn] flag is not set.*
- **When the result FIFOn (n = 0 or 1) overflows**: This interrupt is enabled when the IE[FOFIEn] bit field is set to 1. When the FIFO overflows, a 1-bit flag (the STAT[FOFn] bit field) is set to 1, and the FIFO does not receive new data; however, the existing data remains intact. The STAT[FOFn] bit field can be cleared manually by writing 1 to it.
- **When a conversion sequence associated with TRIGn (n = 0–7) completes execution**: To enable/disable sequence completion interrupts for sequences associated with TRIG0-TRIG7, the IE[TCOMP_IE] bit field of 8-bit size is used. Each bit of the bit field enables/disables the corresponding sequence completion interrupt. When a conversion sequence completes execution, a 1-bit flag (the STAT[TCOMP_INT] bit field) is set to 1. To determine which TRIG sequence completed execution, read the TSTAT[TCOMP_FLAG] bit field of 8-bit size. To clear both the used bits in the STAT[TCOMP_INT] and TSTAT[TCOMP_FLAG] bit fields, write 1 in each bit.

- **When a higher-priority trigger signal interrupts an ongoing conversion sequence while the CFG[TRES] bit field is set to 0 (an interrupted lower-priority sequence does not resume execution or does not restart)**: This interrupt is enabled when the IE[TEXC_IE] bit field is set to 1. When the interrupt is triggered, a 1-bit flag (the STAT[TEXC_INT] bit field) is set to 1. The STAT[TEXC_INT] bit field can be cleared manually by writing 1 to it. To determine which sequence was interrupted, read the TSTAT[TEXC_NUM] bit field of 8-bit size.

For each ADC module, these four interrupts share one interrupt vector on the core side.

6 DMA interrupts triggered by ADC

When the number of valid data words in a result FIFO (FIFO0 or FIFO1) of an ADC module is greater than the watermark, the result FIFO can trigger a DMA interrupt, as described in Table 2.

Table 2. DMA interrupts

ADC module	Result FIFO	Triggered DMA interrupt
ADC1	FIFO0	eDMA4 through source 57
	FIFO1	eDMA4 through source 220
ADC2	FIFO0	eDMA4 through source 158
	FIFO1	eDMA4 through source 221

7 ADC calibration for offset and gain errors

To achieve the specified accuracy during initialization, each ADC module must perform calibration on offset and gain. The offset error has two values stored in the OFSTRIM16 and OFSTRIM12 registers, whose sizes are 16 bits and 12 bits, respectively. The gain errors of side A and side B are stored in the GCR0[GCALR] and GCR1[GCALR] bit fields.

For more details on the calibration process, refer to the "Initialization" section of the "Analog-to-Digital Converter (ADC)" chapter in *i.MX RT1180 Reference Manual*.

8 ADC temperature sensor

Each ADC module has an integrated temperature sensor, which is connected to its CH26A and CH26B channels. To get the correct temperature, use the following configurations:

- For the CMD that is used for temperature sensing:
  - The channel number must be 26 (CMDLm[ADCH] = 26).
  - The CMD must be in Differential mode (CMDLm[CTYPE] = 2).
- For a CMD, the conversion result can be in either 12-bit / 13-bit format or 16-bit format.
- Use the maximum hardware average time (CMDHm[AVGS] = 10).
- Use the longest sampling time (CMDHm[STS] = 7).
- Loop twice (CMDHm[LOOP] = 1).
- Keep the channel number in the CMD unchanged during the loop, for example, CMDHm[LWI] = 0.
- Disable the compare feature (CMDHm[CMPEN] = 0).

The result FIFO has two results corresponding to this CMD. The results can be used to calculate the ambient temperature as follows:

$$Temp = A * \left[\frac{\alpha * (Vbe8 - Vbe1)}{Vbe8 + (\alpha * (Vbe8 - Vbe1))}\right] - B$$

where:

- $Vbe1$ is the first result stored in the FIFO.
- $Vbe8$ is the second result stored in the FIFO.
- A is the slope factor and its value is 789.2.
- B is the offset factor and its value is 319.2.
- α is the band gap coefficient and its value is 11.2.

The calculated temperature is in °C. To achieve maximum temperature calculation accuracy, set the CMDLm[CSCALE] bit field to 1.

9 Points to remember

- When ADC_VREFH is selected as the ADC reference voltage, the ADC clock frequency must be less than 20 MHz to achieve the performance mentioned in the data sheet.
- When VDDA_ADC_1P8 is selected as the ADC reference voltage, the maximum ADC clock frequency is 88 MHz.
- The CTRL[RST] bit field can be used to reset the ADC internal logic and registers. However, it cannot be used to reset the CTRL register itself.
- If CTRL[ADCEN] = 1, the CFG register bit field values cannot be changed. Before writing the CFG register, clear the CTRL[ADCEN] bit field.
- The STAT[TCOMP_INT] flag is asserted when a sequence completes execution, only if IE[TCOMP_IE] = 1. The TSTAT[TCOMP_FLAG] bits are asserted when the corresponding sequences complete execution, only if IE[TCOMP_IE] = 1.
- The STAT[TEXC_INT] flag is asserted when a higher-priority trigger interrupt occurs, only if IE[TEXC_IE] = 1 and CFG[TRES] = 0.
- Calibration must be done before using ADC; otherwise, the performance cannot be guaranteed.
- The CSCALE bit must be cleared when the corresponding CMD channel signal is at a voltage higher than 1.8 V.

10 References

The following are some additional documents that you can refer to for more information on the i.MX RT1180 MCU:

- i.MX RT1180 Reference Manual (IMXRT1180RM)
- i.MX RT1180 Crossover Processors Data Sheet for Industrial Products (IMXRT1180IEC)
- i.MX RT1180 Crossover Processors Data Sheet for Extended Industrial Products (IMXRT1180XEC)

11 Acronyms

Table 3 lists the acronyms used in this document.

Table 3. Acronyms

Acronym	Description
AD	Analog-to-digital
ADC	Analog-to-Digital Converter
ADCK	ADC clock
CMD	Command buffer

Table 3. Acronyms...*continued*

Acronym	Description
LDO	Low dropout
SAR	Successive approximation register
S/C	Sampling+conversion
SoC	System-on-chip
XBAR	Inter-Peripheral Crossbar Switch

12 Revision history

Table 4 summarizes the revisions to this document.

Table 4. Revision history

Document ID	Release date	Description
AN14485 v.1.0	7 February 2025	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

Date of release: 7 February 2025
Document identifier: AN14485