XMC4000

32-bit Microcontroller Series for Industrial Applications

Versatile Analog to Digital Converter (VADC)

AP32305

Application Note

About this document

Scope and purpose

This application note describes how to use the Versatile Analog to Digital Converter (VADC) with the XMC4000 product family. It describes various use cases related to conversion, triggers, results, events, interrupts and the comparator.

Applicable Products

- XMC4000 Microcontrollers Family

Usefull material and links

- Example code: http://www.infineon.com/XMC4000 Tab: Documents
- XMC Lib, http://www.infineon.com/DAVE
- DAVE™, http://www.infineon.com/DAVE
- XMC Reference Manual, http://www.infineon.com/XMC4000 Tab: Documents
- XMC Data Sheet, http://www.infineon.com/XMC4000 Tab: Documents

Table of Contents

Table of Contents

Table of Contents

1 Introduction

Measuring signals leads to the problem that the software must start conversions for each channel and then wait until the conversions are finished before the result can be read. This generates CPU load and reduces the CPU resources that are available for other tasks.

To reduce the load on the CPU, the XMC4000 family's VADC has a number of features which allow 'CPU free' background conversions, even for complex measurement sequences. The software or a DMA just has to read the conversion results.

Each VADC group provides three request sources. A request source pre-defines a sequence of channels that need to be converted. A trigger is fired when this sequence is converted.Each request source provides different features:

- Scan Request Source: This can request up to 8 channels in a fixed order (one after the other).
- Queue Request Source: This can be filled arbitrarily with up to 8 channels.
- Background Scan Request Source: This is common to all groups (1 per device).

A trigger is typically a hardware signal from another peripheral or a software command.

It is possible to repeat a sequence or just use it once ("Refill" for the Queue Request Source and "Autoscan" for the Scan Request Sources). A large number of trigger and gating signals ensure perfect timing control which is critical for real -time applications.

When more than one request source is used, a configurable arbitration process decides which conversion takes priority. If a request for a high priority conversion arrives while a low priority conversion is in progress, there are two Start Mode options available:

- wait-for-start : finish the current conversion
- cancel-inject-repeat: cancel the current conversion, inject the high-priority conversion, then repeat the cancelled conversion

These resources give the user full control of the conversion sequence including the interaction among different concurrent conversion requests.

Special sequencing modes, such as synchronizing and interleaving conversions between ADC groups or setting time-equidistant conversions, are easily configurable.

To relax the real-time critical reading of the result, there are more result registers (17) than channels (8). A powerful result handling mechanism helps de-couple the hardware related conversion timing from the software task scheduling.

Additionally, the XMC4000 microcontroller family has DMA channels that will safely read the conversion results.

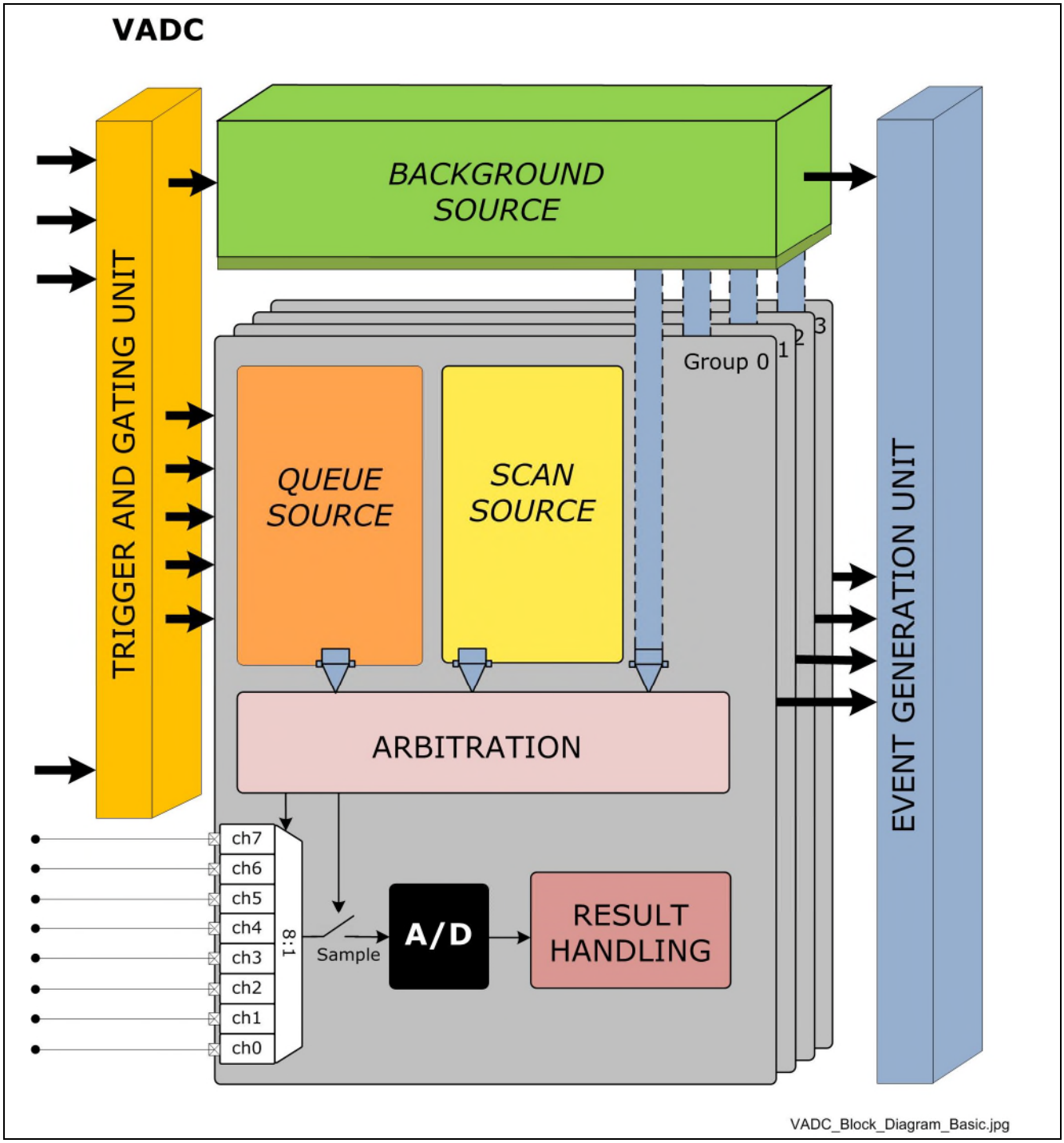

Figure 1 Basic VADC block diagram

2 Conversion Use Cases

This chapter describes common use cases for the VADC and how the microcontroller can be set up for each use case. In addition, there are examples of how to implement dedicated use cases with the Infineon XMC Library (XMC Lib).

A number of DAVE™ projects are available at www.infineon.com/XMC4000 under the Documents tab. For several use cases other peripherals are required as well as the VADC. The usage of these other peripherals is shown in the example projects, but it is not described in this document. Please see separate application notes for more detailed information on other peripherals.

2.1 Single Channel Conversion – Using the Background Request Source

This example shows how to convert a single channel using the background source. The background source is continuously requesting the conversion of channel 5 of group 0. The result is stored in the result register 5 of the group 0 and read out in the while loop of main.c.

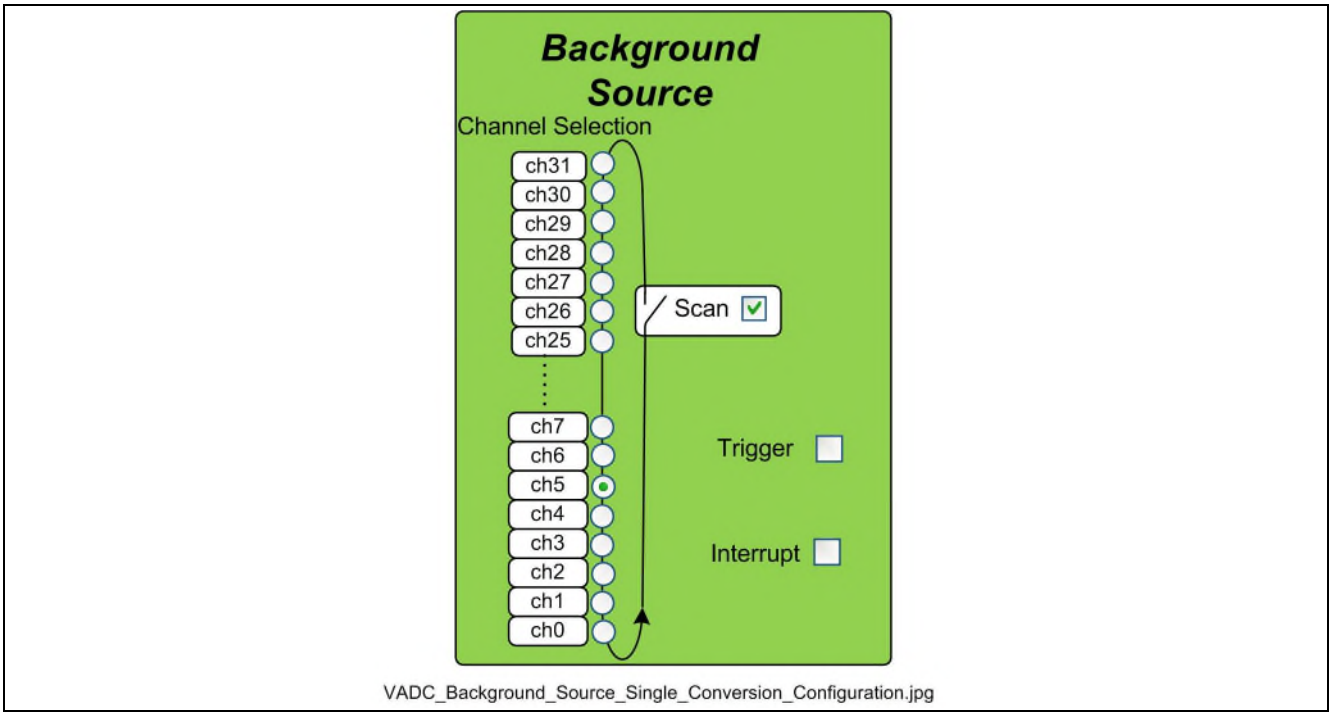

Figure 2 Source Configuration for Single Conversion

The next figure presents the conversion sequence generated by the configuration example.

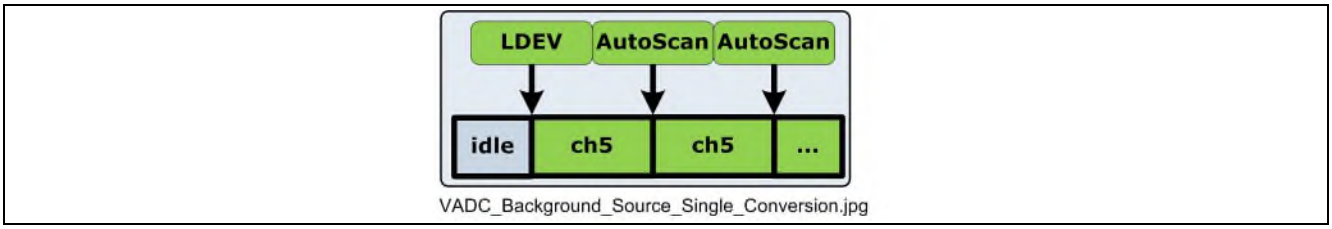

Figure 3 Channel 5 conversion after external trigger signal

A Load Event (LDEV) acts as a trigger, starting the conversion of the sequence, in this case 1 channel. After having converted channel 5 once, autoscan automatically requests the conversion of channel 5 each time the conversion is finished.

2.1.1 Register Configuration

The following steps need to be configured for a single channel conversion:

1. Configure the global VADC register and provide a clock.
2. Choose the channel number using the Background Request Source Channel Select Register (BRSSEL).
3. Configure the Background Request Source with registers BRSMR and BRSCTRL:
   - For continuous conversion, activate the autoscan and generate a load event (LDEV) to start request to the arbiter. Configure the gating for enabling conversions (ENGT = b01).
   - In case of discontinuous mode (time triggered conversions) configure the external trigger according to the triggering scheme. Configure the gating for enabling conversions (ENGT = b01).
4. In the Channel Control Register (GxCHCTRy) choose the Result Register(s) to store the conversion.
5. Configure interrupts if necessary.
6. Enable the Arbitration. Enable the arbitration slot for Background Request Source. Do this as late as possible to avoid triggering un-configured or half-configured conversions. The analog converter control has to be set as normal operation. This starts conversions in the corresponding ADC.
7. Include a Calibration safe loop. Wait until calibration is finished.

Figure 2,shows how to configure the Background Request Source to convert channel 5 in a repeated way (with autoscan).

2.1.2 XMC Lib Implementation

Configuration

In this example the analog clock divider is set to 3 to reach $f_{ADCI} = 30MHz$ (see also 3.9.3). For the group configuration and background source configuration the standard values are used. The alias feature of the channel configuration is disabled because it is not necessary here. The result register of the channel is set to register 5.

```
const XMC_VADC_GLOBAL_CONFIG_t g_global_handle = {
    .clock_config = {
        .analog_clock_divider = 3,
    },
};
const XMC_VADC_GROUP_CONFIG_t g_group_handle = { };
const XMC_VADC_BACKGROUND_CONFIG_t g_bgn_handle = { };
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch5_handle = {
```

```
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 5, };
```

Initialization and Function Implementation

This section shows how to use the VADC APIs to run a background conversion using the configuration above. First the global initialization is called to provide a clock to the VADC and to initialize the global VADC registers. Second the VADC group is initialized before the power mode is set to normal mode. Only after this is done, the startup calibration routine is called to ensure a correct calibration.

```
XMC_VADC_GLOBAL_Init(VADC, &g_global_handle);
XMC_VADC_GROUP_Init(VADC_G0, &g_group_handle);
XMC_VADC_GROUP_SetPowerMode(VADC_G0, XMC_VADC_GROUP_POWERMODE_NORMAL);
XMC_VADC_GLOBAL_StartupCalibration(VADC);
```

In the initializationphase of the background source of the background source of the background source of the background source of the background source the gating mode of the VADC is automatically set to “ignore”. The desired channel is initialized before the channel is added to the background source. The autoscan feature is enabled for continuous conversion. Finally the conversion is triggered by software:

```
XMC_VADC_GLOBAL_BackgroundInit(VADC, &g_bgn_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G0, 5, &g_g0_ch5_handle);
XMC_VADC_GLOBAL_BackgroundAddChannelToSequence(VADC, 0, 5);
XMC_VADC_GLOBAL_BackgroundEnableContinuousMode(VADC);
XMC_VADC_GLOBAL_BackgroundTriggerConversion(VADC);
```

The result can now be read out in a loop. Please keep in mind that the update interval of the result depends on the clock frequency and the clock divider factor.

```
while (1U) {
    result = XMC_VADC_GROUP_GetResult(VADC_G0, 5);
}
```

2.2 Linear Sequencing of Several Channels – Using the Scan Request Sources

When more than one channel needs to be converted, Background and Scan Request Sources can be used to convert a linear sequence of channels. Sometimes, several pins need to be sensed without a specific order. Normally the requirement is to convert these channels in a certain time frame.

Application characteristics impose which source is chosen in each case. This example shows how to convert multiple channels in a linear sequence using the scan source. The scan source requests the conversion of channel 7, channel 6 and channel 1 of group 0. The result is stored in the result registers 7, 6 and 1 of group 0. It is read out in the while loop of main.c. No autoscan is chosen, but CCU4 is used to trigger the conversion of the sequence periodically. An interrupt is generated after each sequence conversion. The results are read out in the corresponding interrupt service routine.

The following two figures show respectively the configuration and then an example sequence for the Scan Request Source.

According to the configuration, the sequence chosen in register GxASSEL is: channel 7, channel 6 and channel 1. This sequence is configured as No Autoscan (SCAN=b0), and with the Trigger and Interrupts enabled (ENTR=b1 and ENSI=b1). The conversion of every channel of the sequence is made every time that the trigger signal arrives (See Trigger Options).

At the end of each sequence an interrupt is generated, in this case every time after channel 1 is converted.

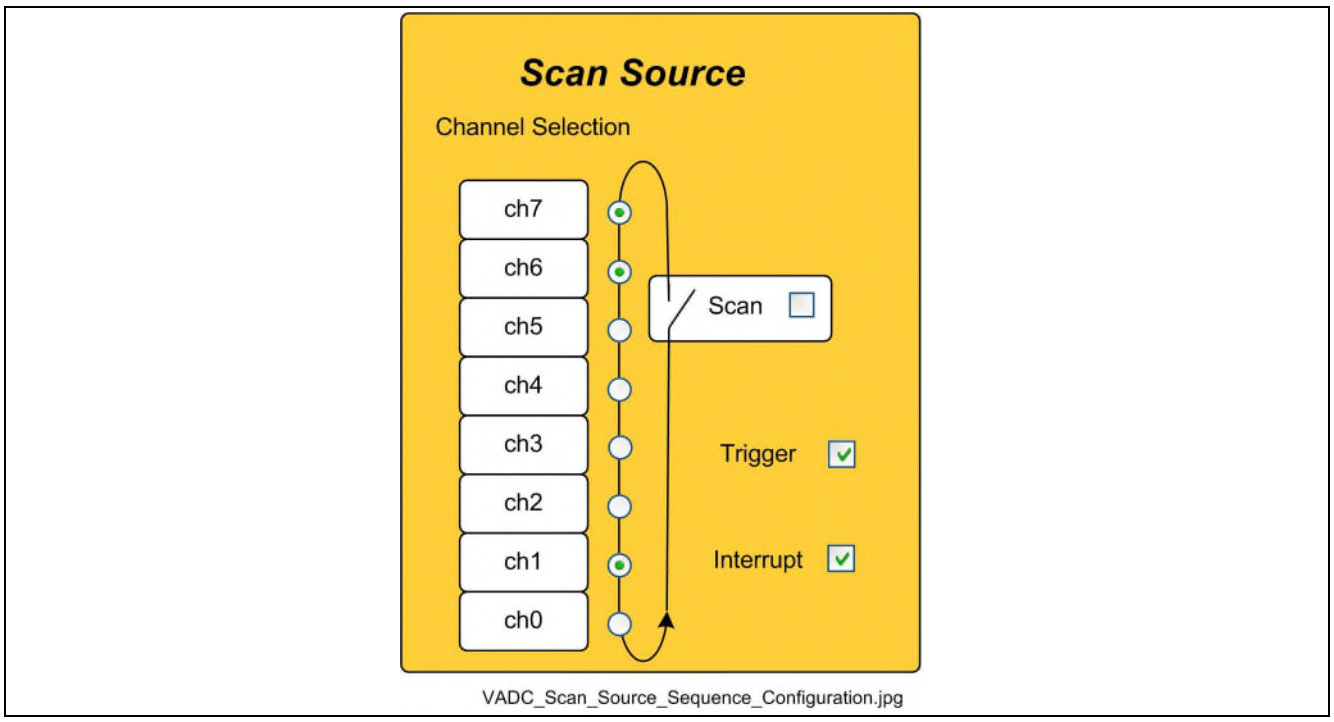

Figure 4 Scan Request Source sequence configuration

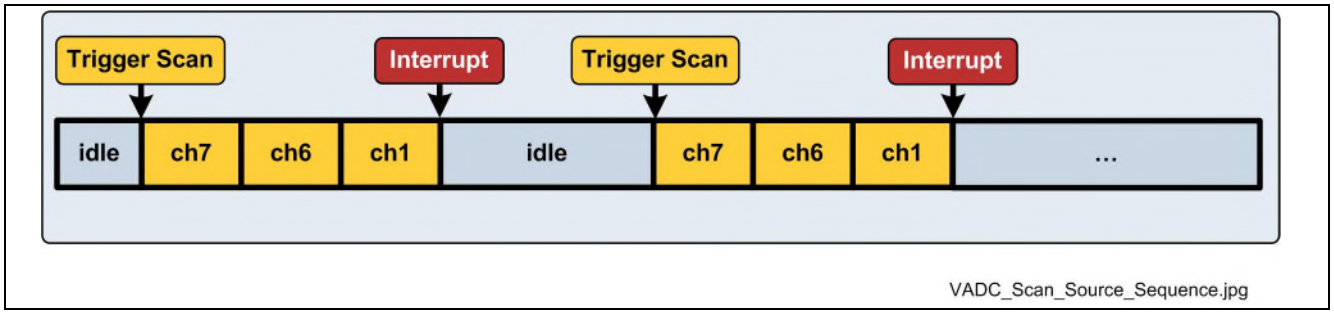

Figure 5 Scan Request Source sequence example

2.2.1 Register Configuration

The Scan Request Source is group specific, so only channels in the group are reachable. The conversion starts from the highest channel number to the lowest.

The following steps need to be configured for conversion of a linear sequence of channels with Scan Request Source:

1. Define the global VADC register and clock configuration.
2. Choose the channel sequence using the Scan Source Channel Select Register (GxASSEL).
3. Configure the Scan Request Source with registers GxASMR and GxASCTRL:

- For continuous conversion, activate the Autoscan and generate a load event (LDEV). Configure the gating for enabling conversions (ENGT = b01).
- For discontinuous mode (time triggered conversions) configure the external trigger according to the triggering scheme. Configure the gating for enabling conversions (ENGT = b01).

4. In the Channel Control Register (GxCHCTRy) choose the Result Register(s) to store the conversions.
5. Configure interrupts if necessary.
6. Enable the Arbitration:
   - Enable the arbitration slot for Scan Request Source (GxARBPR). Do this as late as possible to avoid triggering un-configured or half-configured conversions. The analog converter control has to be set as per normal operation. This starts conversions in the corresponding ADC.
7. Include a Calibration safe loop. Wait until calibration is finished.

2.2.2 XMC Lib Implementation

Configuration

In this example the global and group configurations are implemented as in 2.1.2. For the scan source the trigger is enabled on any edge and uses the trigger input A (see also "Digital Connections in the XMC4x00" in the Reference Manual). An event is generated after each sequence conversion. The alias feature of the channel configuration is disabled as it is not necessary here. The result registers of the used channels are set to the same as the corresponding channel numbers. (The CCU4 configuration is explained in a separate application note.)

```
const XMC_VADC_GLOBAL_CONFIG_t g_global_handle = {
    .clock_config = {
        .analog_clock_divider = 3,
    },
};
const XMC_VADC_GROUP_CONFIG_t g_group_handle = { };
const XMC_VADC_SCAN_CONFIG_t g_scan_handle = {
    .trigger_signal = XMC_VADC_REQ_TR_A,
    .trigger_edge = XMC_VADC_TRIGGER_EDGE_ANY,
    .external_trigger = 1,
    .req_src_interrupt = 1, };
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch7_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 7, };
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch6_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 6, };
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch1_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
```

Conversion Use Cases

```
    .result_reg_number = 1, };
const XMC_CCU4_SLICE_COMPARE_CONFIG_t g_timer_object = {
    .timer_mode = XMC_CCU4_SLICE_TIMER_COUNT_MODE_EA,
    .prescaler_initval = (uint32_t) 7, };
```

Initialization and Function Implementation

The global and group initialization, the power mode settings and the startup calibration are implemented as in 2.1.2.

```
    XMC_VADC_GLOBAL_Init(VADC, &g_global_handle);
    XMC_VADC_GROUP_Init(VADC_G0, &g_group_handle);
    XMC_VADC_GROUP_SetPowerMode(VADC_G0, XMC_VADC_GROUP_POWERMODE_NORMAL);
    XMC_VADC_GLOBAL_StartupCalibration(VADC);
```

The scan source and the desired channels are initialized before the channels are added to the scan source.

```
    XMC_VADC_GROUP_ScanInit(VADC_G0, &g_scan_handle);
    XMC_VADC_GROUP_ChannelInit(VADC_G0, 7, &g_g0_ch7_handle);
    XMC_VADC_GROUP_ChannelInit(VADC_G0, 6, &g_g0_ch6_handle);
    XMC_VADC_GROUP_ChannelInit(VADC_G0, 1, &g_g0_ch1_handle);
    XMC_VADC_GROUP_ScanAddChannelToSequence(VADC_G0, 7);
    XMC_VADC_GROUP_ScanAddChannelToSequence(VADC_G0, 6);
    XMC_VADC_GROUP_ScanAddChannelToSequence(VADC_G0, 1);
```

After the CCU4 and the NVIC are configured as desired (see respective application notes or example projects), the ISR is programmed to set a flag, if a result interrupt occurs.

```
void VADC0_G0_0_IRQHandler(void) {
    isr_flag = 1;
}
```

The result is read out in a loop if the flag is set.

```
    while (1U) {
        if (isr_flag == 1) {
            isr_flag = 0;
            result[7] = XMC_VADC_GROUP_GetResult(VADC_G0, 7);
            result[6] = XMC_VADC_GROUP_GetResult(VADC_G0, 6);
            result[1] = XMC_VADC_GROUP_GetResult(VADC_G0, 1);
        }
    }
```

2.3 Linear Sequencing of Several Channels – Using the Background Request Sources

This use case is very similar to 2.2, but instead of the scan source the background source is used.

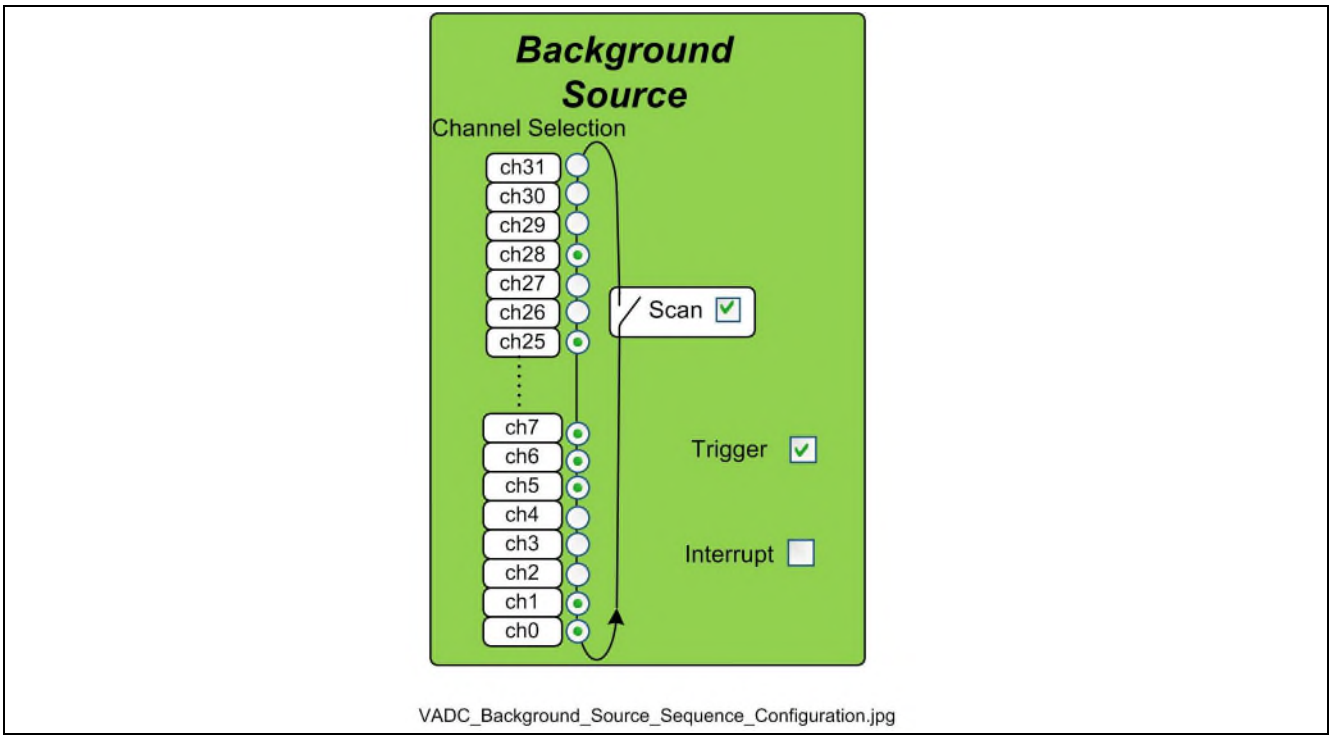

Figure 6 Background Request Source sequence configuration

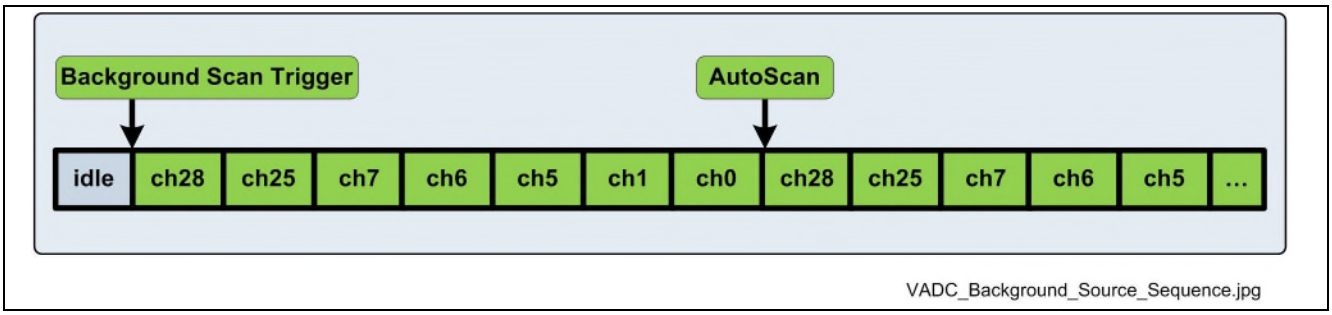

Figure 7 Background Request Source sequence example

2.3.1 Register Configuration

The register configuration can be done in a similar way to 2.1.1. According to the configuration, the sequence chosen in register BRSSELx (CHSELGy bitfield) is:

- channel 28, channel 25, channel 7, channel 6, channel 5, channel 1 and channel 0

The sequence is configured as Autoscan (SCAN=b1), and with the external Trigger enabled.

The conversion sequence starts after the trigger and is repeated continuously because Autoscan is enabled. That means that the trigger event starts a continuous conversion of the selected channels. No interrupt is generated after a conversion sequence.

2.3.2 XMC Lib Implementation

Configuration

In this example the configurations are implemented in a similar way to 2.1.2. In this case the configurations for more than 7 different channels are provided. The result registers of the used channels are set to the same as the corresponding channel numbers.

```
const XMC_VADC_GLOBAL_CONFIG_t g_global_handle = {
    .clock_config = {
        .analog_clock_divider = 3,
    },
};
const XMC_VADC_GROUP_CONFIG_t g_group_handle = { };
const XMC_VADC_BACKGROUND_CONFIG_t g_bgn_handle = { };
const XMC_VADC_CHANNEL_CONFIG_t g_g3_ch4_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 4, };
const XMC_VADC_CHANNEL_CONFIG_t g_g3_ch1_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 1, };
```

Note:The configuration of channel 7, 6, 5, 1 and 0 of group 0 are done the same as the two channel configurations above.

Initialization and Function Implementation

The initialization and function implementation is very similar to 2.1.2. All used groups (0 and 3) and all channels are initialized and the power mode is set to normal. The channels are then added to the background source. The result is read out in a loop.

```
XMC_VADC_GLOBAL_Init(VADC, &g_global_handle);
XMC_VADC_GROUP_Init(VADC_G3, &g_group_handle);
XMC_VADC_GROUP_Init(VADC_G0, &g_group_handle);
XMC_VADC_GROUP_SetPowerMode(VADC_G3, XMC_VADC_GROUP_POWERMODE_NORMAL);
XMC_VADC_GROUP_SetPowerMode(VADC_G0, XMC_VADC_GROUP_POWERMODE_NORMAL);
XMC_VADC_GLOBAL_StartupCalibration(VADC);
XMC_VADC_GLOBAL_BackgroundInit(VADC, &g_bgn_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G3, 4, &g_g3_ch4_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G3, 1, &g_g3_ch1_handle);
```

Note:Do the channel initialization for the rest of the channels (group 0 channels 7, 6, 5, 1, 0) accordingly.

```
XMC_VADC_GLOBAL_BackgroundAddChannelToSequence(VADC, 3, 4);
XMC_VADC_GLOBAL_BackgroundAddChannelToSequence(VADC, 3, 1);
```

Note:Add the rest of the channels (group 0 channels 7, 6, 5, 1, 0) to the background sequence accordingly.

```
XMC_VADC_GLOBAL_BackgroundEnableContinuousMode(VADC);
XMC_VADC_GLOBAL_BackgroundTriggerConversion(VADC);
while (1U) {
    result[3][4] = XMC_VADC_GROUP_GetResult(VADC_G3, 4);
    result[3][1] = XMC_VADC_GROUP_GetResult(VADC_G3, 1);
```

Note:Fetch the rest of the results (group 0 channels 7, 6, 5, 1, 0) from the result registers accordingly.

2.4 Dedicated Sequencing of Several Channels – Using the Queue Request Source

Some applications require more complex sequences for converting the analog inputs. The Queue Request Source allows the user to configure dedicated sequences of conversions, choosing arbitrarily from up to 8 channels in a group. Each independent channel (or queue entry) of the sequence can be configured separately as:

- Refill: to program its re-entrance into the queue or FIFO after being converted
- External Trigger Enable: to be converted only after an external trigger event occurs
- Interrupt Enable: to generate a source event after the conversion of this channel is finished

The following two figures show respectively the configuration and an example of a Queue Request Source application.

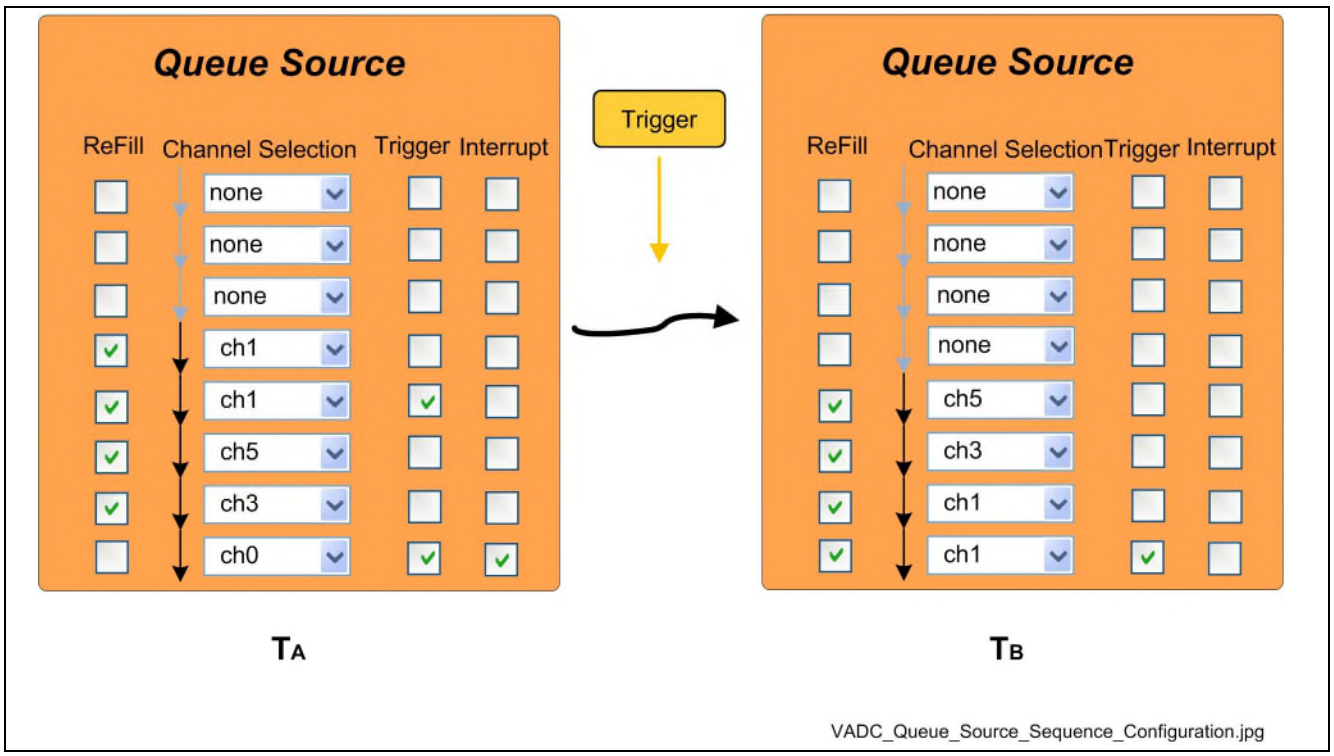

Figure 8 Queue Request Source sequence configuration

According to the configuration, the queue FIFO is:

- channel 0, channel 3, channel 5, channel 1 and channel 1

Channel 3, channel 5, channel 1 and channel 1 are configured as refill (RF is set to 1 in these channels) so the queue is fed again with these channels after the first conversion.

Channel 0 is configured as No Refill (RF=b0, it is converted just once) and to produce an interrupt after finishing its conversion (ENSI=b1).

Channel 0 and channel 1 are configured with external trigger enabled (EXTR=b1), the Queue Request Source does expect the trigger event in these channels to start the conversions.

The initial state of the queue (TA) is shown in the configuration. After the first conversion, the queue entries are channel 5, channel 3, channel 1 and channel 1, and the status of the queue changes to that presented as 'TB'.

The conversion sequence corresponding to this set-up is shown in the next figure:

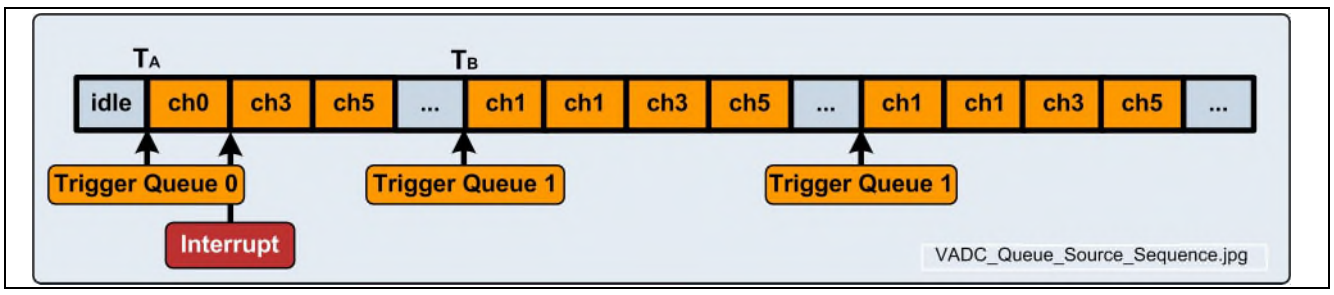

Figure 9 Queue Request Source sequence example

2.4.1 Register Configuration

The following steps need to be followed for converting a dedicated sequence of channels:

1. Define the global VADC register and clock configuration.
2. Choose and configure separately each queue entry that forms the sequence using the Queue Input Register (QINR0):
   - For continuous conversion, choose the channel requested in this entry and activate the Refill.
   - For discontinuous mode (time triggered conversions) choose the channel requested in this entry and enable the external trigger (ENTR).
3. Configure the Queue Request Source with registers GxQMR0 and GxQCTRL0:
   - For continuous conversion, configure the gating for enabling conversions (ENGT = b01).
   - For discontinuous mode (time triggered conversions) configure the external trigger according to the triggering scheme (select a hardware trigger or generate a trigger event (TREV) for software trigger) and configure the gating for enabling conversions (ENGT = b01).
4. In the Channel Control Register (GxCHCTRy) choose the Result Register(s) to store the conversions.
5. Configure interrupts if necessary.
6. Enable Arbitration.
   - Enable the arbitration slot for Scan Request Source (GxARBPR). Do this as late as possible to avoid triggering un-configured or half-configured conversions. The analog converter control has to be set as per normal operation. This starts conversions in the corresponding ADC.
7. Include a Calibration safe loop. Wait until calibration is finished.

2.4.2 XMC Lib Implementation

Configuration

In this example the global and group configurations are implemented as in 2.1.2. For the queue source, the trigger is enabled on any edge and uses the trigger input A (see also "Digital Connections in the XMC4x00" in the Reference Manual).

```
const XMC_VADC_GLOBAL_CONFIG_t g_global_handle = {
    .clock_config = {
        .analog_clock_divider = 3,
    },
};
const XMC_VADC_GROUP_CONFIG_t g_group_handle = { };
const XMC_VADC_QUEUE_CONFIG_t g_queue_handle = {
    .trigger_signal = XMC_VADC_REQ_TR_A,
    .trigger_edge = XMC_VADC_TRIGGER_EDGE_ANY,
    .external_trigger = 1 };
```

The alias feature of the channel configuration is disabled because it is not necessary here. The result registers of the used channels are set the same as the corresponding channel numbers. Every entry of the

queue source is configured as explained in the use case description above. For example, the first entry requests the conversion of channel 0 with no refill of the entry in the queue source. It needs to be triggered externally and generates an interrupt. (The CCU4 configuration is explained in a separate application note.)

```
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch0_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 0, };
```

Note: The configuration of channel 1, 3 and 5 of group 0 are done the same way as the channel configuration above.

```
const XMC_VADC_QUEUE_ENTRY_t g_queue_entry_0_handle = {
    .channel_num = 0,
    .refill_needed = 0,
    .external_trigger = 1,
    .generate_interrupt = 1, };
const XMC_VADC_QUEUE_ENTRY_t g_queue_entry_1_handle = {
    .channel_num = 3,
    .refill_needed = 1,
    .external_trigger = 0,
    .generate_interrupt = 0, };
```

Note: The configuration of queue entries 2, 3 and 4 are done in the same way.

Initialization and Function Implementation

The global and group initialization, the power mode settings and the startup calibration are implemented as in 2.1.2.

```
XMC_VADC_GLOBAL_Init(VADC, &g_global_handle);
XMC_VADC_GROUP_Init(VADC_G0, &g_group_handle);
XMC_VADC_GROUP_SetPowerMode(VADC_G0, XMC_VADC_GROUP_POWERMODE_NORMAL);
XMC_VADC_GLOBAL_StartupCalibration(VADC);
```

Afterwards the queue source and the desired channels are initialized.

```
XMC_VADC_GROUP_QueueInit(VADC_G0, &g_queue_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G0, 0, &g_g0_ch0_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G0, 1, &g_g0_ch1_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G0, 3, &g_g0_ch3_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G0, 5, &g_g0_ch5_handle);
```

Then the desired entries are inserted to the queue.

```
XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_0_handle);
XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_1_handle);
XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_2_handle);
```

```
    XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_3_handle);
    XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_4_handle);
```

After the CCU4 and the NVIC are configured as desired (see respective application notes or example project), the ISR is programmed to get the result of the channel 0 conversion.

```
void VADC0_G0_0_IRQHandler(void) {
    result[0] = XMC_VADC_GROUP_GetResult(VADC_G0, 0);
}
```

The other results are then read out in a loop.

```
    while (1U) {
        result[1] = XMC_VADC_GROUP_GetResult(VADC_G0, 1);
        result[3] = XMC_VADC_GROUP_GetResult(VADC_G0, 3);
        result[5] = XMC_VADC_GROUP_GetResult(VADC_G0, 5);
    }
```

2.5 Multiple Sequences – Multiple Request Sources Conversion

In many applications, complex sequences of conversions have to be executed by one device. For example, one application might need to convert some motor control current with a Queue Request Source, while some temperature sensors are measured with the Background Request Source. In these cases where a mixture of request sources is mandatory, the Arbiter must decide which conversion gets priority to access the A/D converter. If a request with high priority arrives while a low priority conversion is being processed, the way in which the highest priority conversion is injected into the pipeline depends on the configured conversion mode (arbiter configuration).

A configuration example for the VADC group 0 using all three request sources is shown in the next figure. The Queue Request Source has highest priority and interrupts the current conversion of the lower priority Scan Request Source.

Channel 4 is cancelled, and channel 2 is injected followed by the other high priority channels of the Queue Request Source.

Channel 4 is repeated after all Queue Request Source requests are finished.

The lowest priority Background Request Source triggers a request on channel 6 while a higher priority conversion is in process, but channel 6 has to wait until the completion of the higher priority Scan Request Source. This permits a full, 'independent-of-CPU', run of extremely complex schemes.

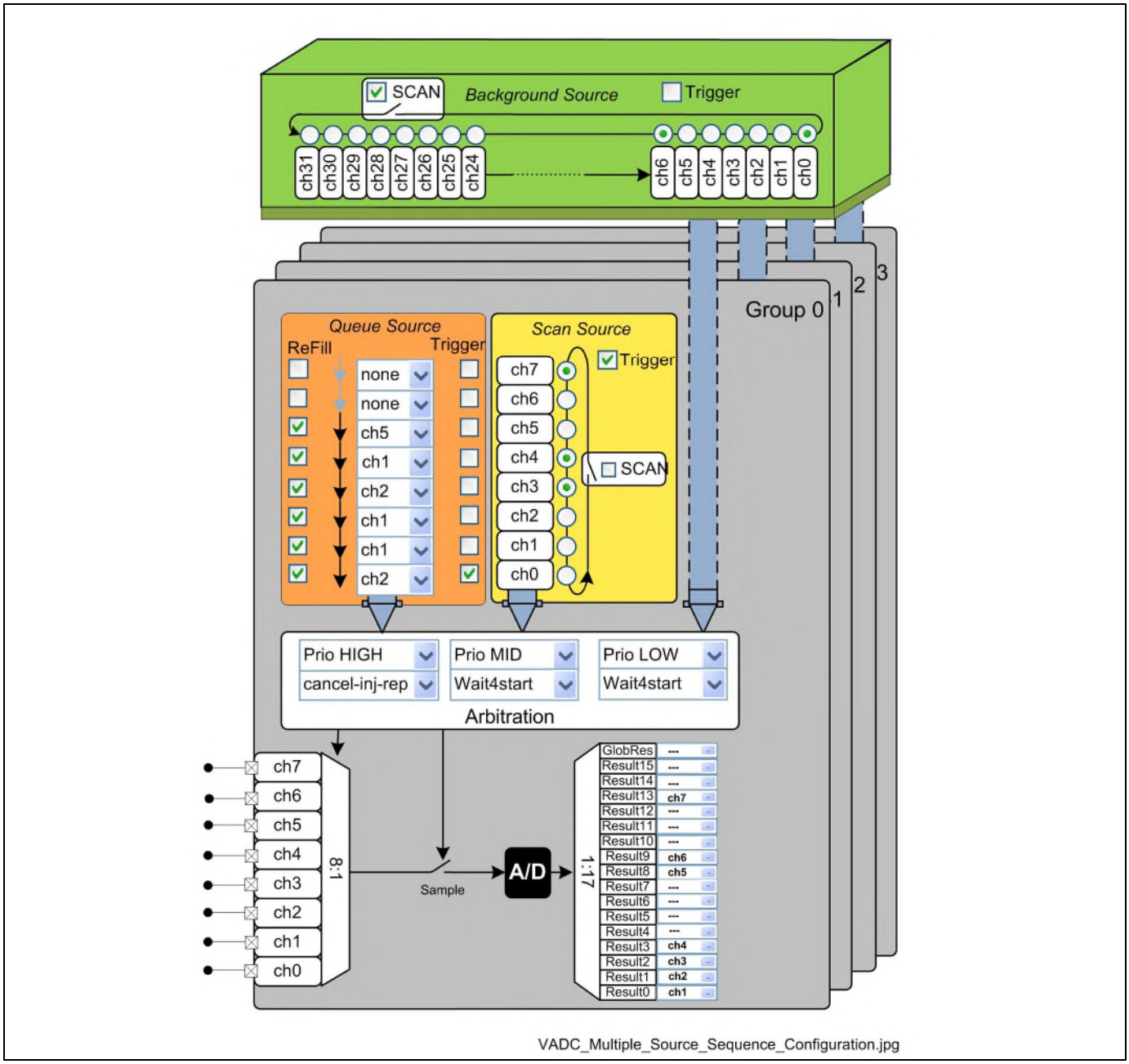

Figure 10 Sequence using all request sources for 'CPU free' background conversion

Here is the use case for the configuration described above

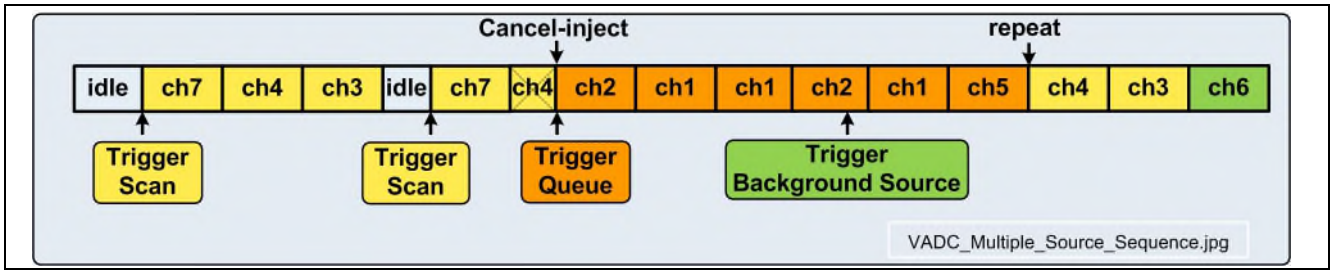

Figure 11 Example Sequence using all three request sources

The XMC Lib Implementation of this use case is available as a DAVE™ project at the website www.infineon.com/XMC4000 under the Documents tab.

2.6 Synchronous Conversion

Synchronous conversion can be very helpful in applications where two or more signals should be sampled at the same time, such as the measurement of voltage and current for power calculations, or current measurements in motor control for example.

Of course, synchronization can be achieved by means of using the same trigger signal in to two or more groups, and this is supported by the XMC4000 VADC (See Application Examples/Synchronization through the trigger). However, there is still some jitter between the sampling of the different groups in this method that is difficult to predict. This jitter depends on the load of a group. For example, the first group is free (no conversion is running) and the second group has an ongoing conversion. This conversion has to be finished or cancelled before the trigger can start the next conversion.

These and other effects that produce jitter are fully eliminated by the use of the synchronization mechanism in the VADC of the XMC4000 Microcontroller Family of products.

Synchronization in the XMC4000 VADC works with a master-slave concept. When the master starts a synchronous conversion, the slaves cancel the actual conversion and respond with a ready signal. When the master and the slaves are ready the conversion is started. After this conversion the slaves inject and repeat the canceled conversion.

The synchronous conversion has some constrains. The channel number from master and the slaves are the same. For example, if the master requests channel 5 the synchronous slave channel is also converting channel 5. This limit of free pin mapping can be solved with the ALIAS feature (See 3.5 Programmable Remapping of Analog Inputs – The Alias Feature). Also the 'Wait-for-read' mode is ignored in the slave. The request from the master is always handled with the highest priority and uses the cancel-inject-repeat mode. At last the slaves arbitration timing (ARBRND) must be configured according to the master timing, as well as the arbitration mode (ARBM) and the analog converter control status (ANONS) in the GxARBCFG register. However, the sampling time (GxICLASSy) and channel characteristics (GxCHCTRy) can be configured differently for the master and each slave group.

Note: The master group has to be initialized after the slave for arbiters running synchronously. Once started, a parallel conversion cannot be aborted.

An example for two signal synchronous sampling is shown below. This demonstrates that all the different conversions, even though started at the same time, can be configured in completely different ways, including distinct sampling time, resolution, and so on.

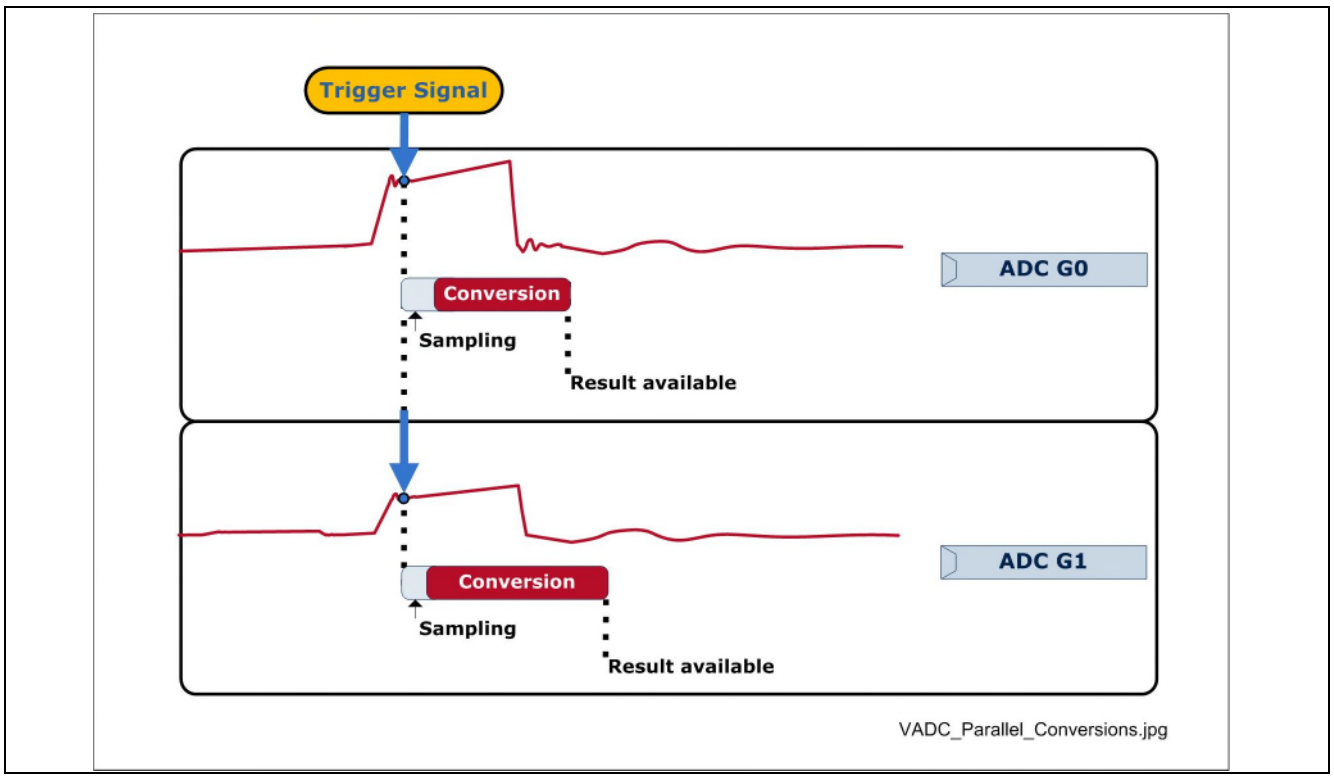

Figure 12 Parallel conversions

The number of groups available depends on the XMC4000 device. If four groups are available it is possible to synchronize groups in different ways. Two of them are shown in Figure 13 (1 master 3 slaves) and Figure 14 (2 masters 2 slaves).

In order to control the Synchronization mechanism several signals communicate between the master and slave groups.

As shown in Figure 13, the ANON signal connects the master to all slave groups with the Synchronization control information, while the READY signal from each slave is connected to the master with the slave information. The bitfield EVALR in the master defines from which slave a READY signal is expected. And the same bitfield (EVALR) is used for the slaves to define which master and slaves receive a READY signal.

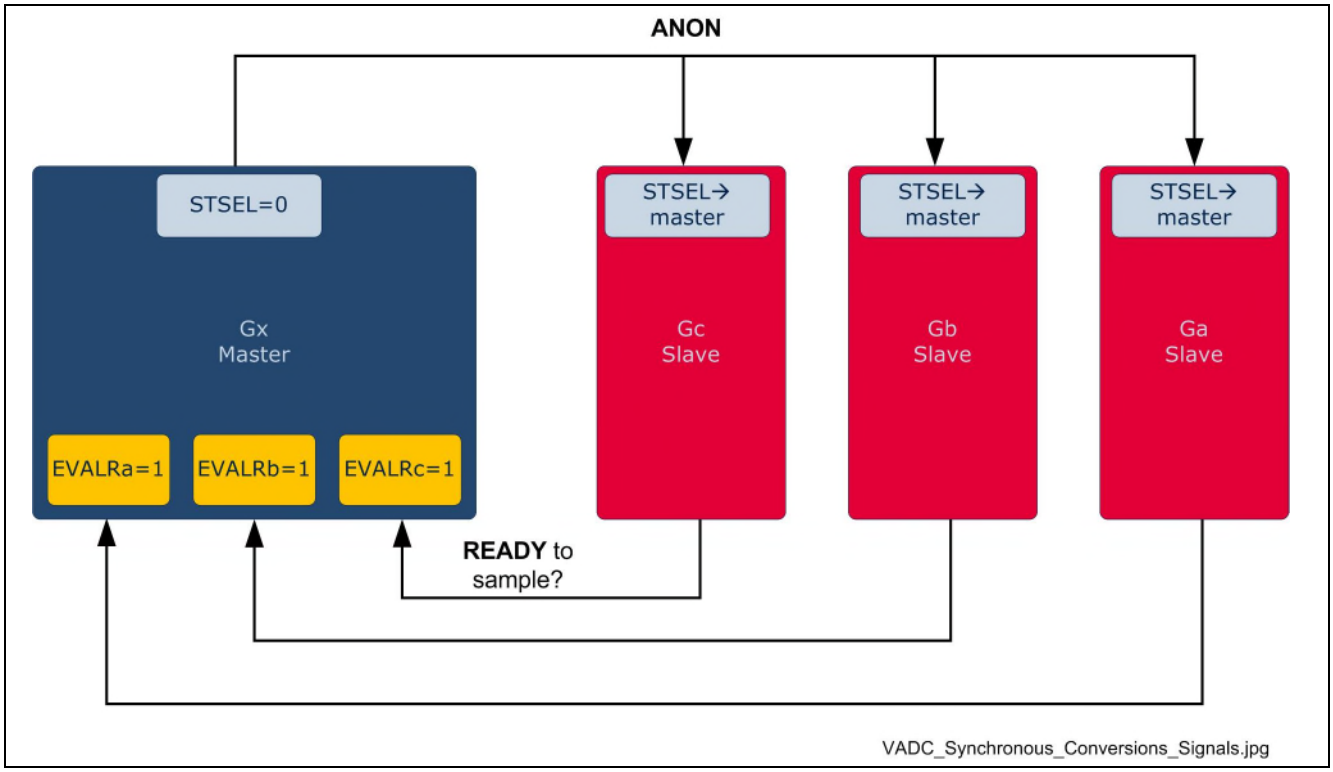

Figure 13 Synchronous conversions - One master, three slaves

In Figure 14, an example with 2 masters and 2 slaves is shown. This demonstrates something of the powerful performance possibilities of the XMC4000 family of products, with different combinations of sequencing with multiple request sources and synchronous sampling with different groups.

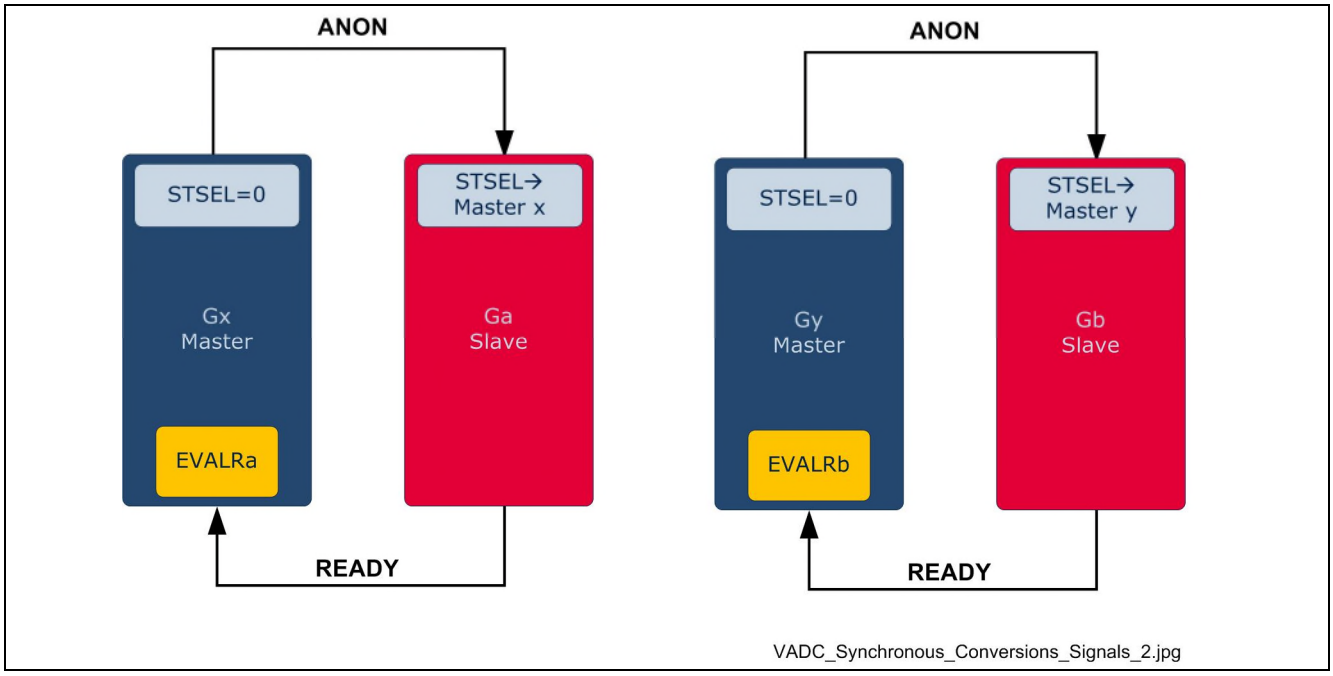

Figure 14 Synchronous conversions - Two masters, two slaves

2.6.1 Register Configuration

The following configurations are specific for the synchronized conversion:

1. Master group configuration:
   Configure in the Synchronization Control Register (GxSYNCTR) this group as master (Master, STSEL=b00) and enable the wait for slave READY signal in the EVALR bitfield (EVALR1-3) according to the slave groups.

2. Channel configuration for the master group:
   When the SYNC bit in the Channel Control Register (GxCHCTRy) is enabled in the channel of the master group, the same channels requested in all slave groups.

3. Slave groups synchronize configuration:
   Configure in the Synchronization Control Register (GxSYNCTR) this group as slave of master group x (STSEL: Slave of CI1 or CI2 or CI3). Provide the READY signal to the master and all slaves (EVALRx: EVALR1 to EVALR3)

4. Channel configuration for the slave group:
   The slave channels have to be configured by the Channel Control Register (GxCHCTRy) of the slave group. The SYNC bit has no influence in the channel of a slave.

2.6.2 Synchronous Conversion – Using the Queue Source

In Figure 15, a source (Queue Request Source) requests conversions in the group configured as master. A CCU8 timer is needed to generate the trigger signal.

Two trigger events are defined: compare match 1, and compare match 2 while counting up. On each of these events, the full queue is converted because of the queue configuration. However, just two of the channels within this sequence request the synchronized conversion of the slave channels. The remaining channel is converted only in the master group. This example demonstrates the flexibility that the ADC offers.

Channels 0 and 1 have to be configured as synchronized in the master register (GxCHCTR, SYNC bitfield) so these channels are converted automatically in the slave group after the master requests its conversions. The slave group and channels also need to be configured.

The Alias feature allows a different input pin for the channel 0 and 1. This feature is not used in the following example.

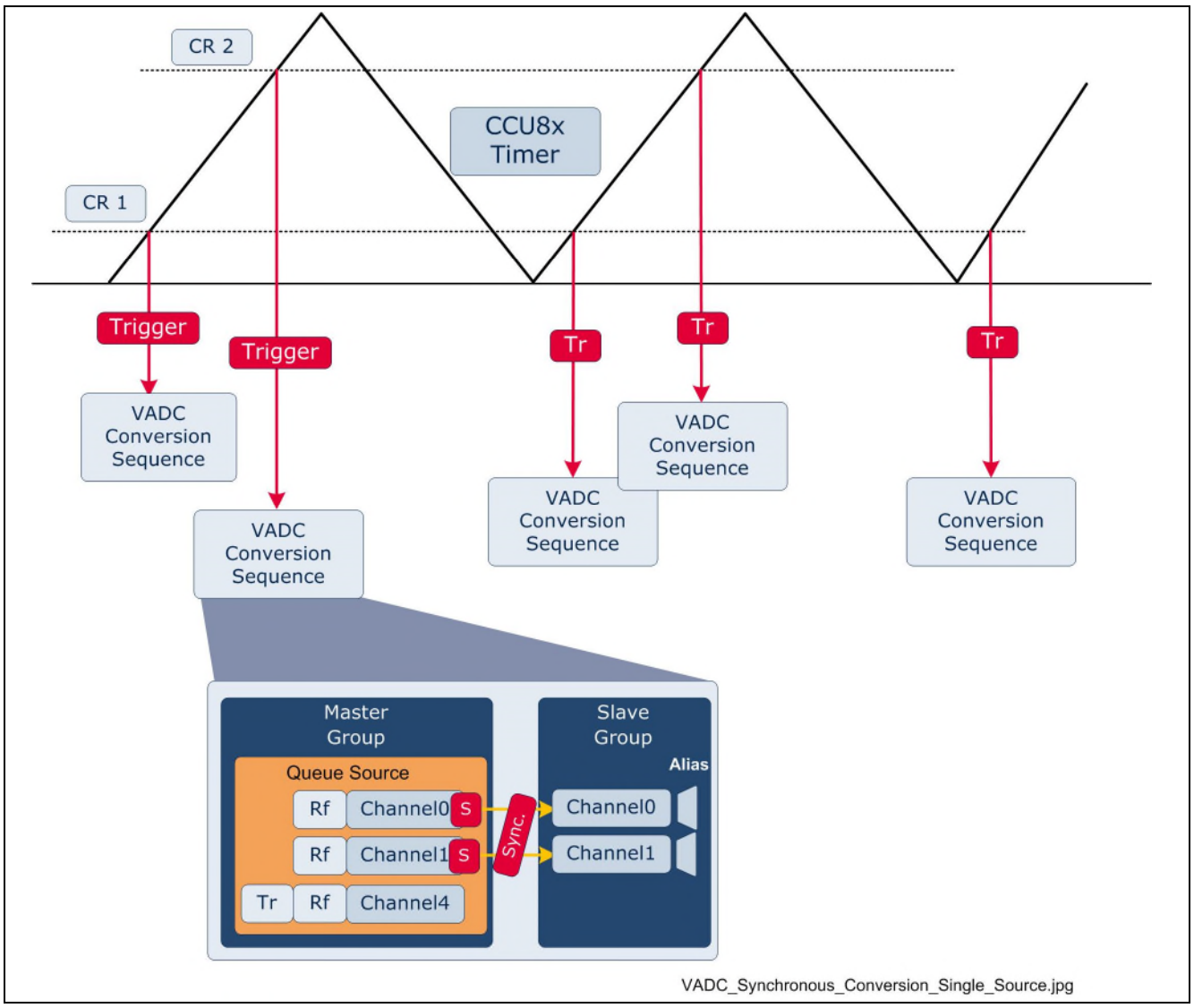

Figure 15 Single Source Synchronization Example

2.6.3 XMC Lib Implementation

This example shows the usage of the synchronization

Configuration

In this example the analog clock divider is set to 0 to reach $f_{ADCI} = 120MHz$. For the group configuration the standard values are used. The queue source trigger is enabled on any edge and uses the trigger input I (see also "Digital Connections in the XMC4x00" in the Reference Manual).

```
const XMC_VADC_GLOBAL_CONFIG_t g_global_handle = {
    .clock_config = {
        .analog_clock_divider = 0,
    },
```

```
};
const XMC_VADC_GROUP_CONFIG_t g_group_handle = { };
const XMC_VADC_QUEUE_CONFIG_t g_queue_handle = {
    .trigger_signal = XMC_VADC_REQ_TR_I,
    .trigger_edge = XMC_VADC_TRIGGER_EDGE_ANY,
    .external_trigger = 1 };
```

The alias feature of the channel configuration is disabled as it is not necessary here. The result registers and the synchrounous conversion of the used channels are set as desired. The sync conversion of group 0 is enabled for channel 0 and channel 1 and disabled vor channel 4. For the slave (group 1) it is not necessary to enable the sync conversion.

```
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch4_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 4,
    .sync_conversion = 0, };
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch1_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 1,
    .sync_conversion = 1, };
const XMC_VADC_CHANNEL_CONFIG_t g_g0_ch0_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 0,
    .sync_conversion = 1, };
const XMC_VADC_CHANNEL_CONFIG_t g_g1_ch1_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 1, };
const XMC_VADC_CHANNEL_CONFIG_t g_g1_ch0_handle = {
    .alias_channel = XMC_VADC_CHANNEL_ALIAS_DISABLED,
    .result_reg_number = 0, };
```

Every entry of the queue source is configured as explained in the use case description above. (The CCU8 configuration is explained in a separate application note.)

```
const XMC_VADC_QUEUE_ENTRY_t g_queue_entry_0_handle = {
    .channel_num = 4,
    .refill_needed = 1,
    .external_trigger = 1 };
const XMC_VADC_QUEUE_ENTRY_t g_queue_entry_1_handle = {
    .channel_num = 1,
    .refill_needed = 1,
    .external_trigger = 0 };
```

Conversion Use Cases

```
const XMC_VADC_QUEUE_ENTRY_t g_queue_entry_2_handle = {
    .channel_num = 0,
    .refill_needed = 1,
    .external_trigger = 0 };
const XMC_CCU8_SLICE_COMPARE_CONFIG_t g_timer_object = {
    .timer_mode = XMC_CCU8_SLICE_TIMER_COUNT_MODE_CA,
    .prescaler_initval = (uint32_t) 7, };
```

Initialization and Function Implementation

The initialization and function implementation is very similar to 2.1.2. All used groups (0 and 3) and all channels are initialized and the power mode is set to normal. After calibration the queue and all channels are initialized.

```
XMC_VADC_GLOBAL_Init(VADC, &g_global_handle);
XMC_VADC_GROUP_Init(VADC_G0, &g_group_handle);
XMC_VADC_GROUP_Init(VADC_G1, &g_group_handle);
XMC_VADC_GROUP_SetPowerMode(VADC_G0, XMC_VADC_GROUP_POWERMODE_NORMAL);
XMC_VADC_GROUP_SetPowerMode(VADC_G1, XMC_VADC_GROUP_POWERMODE_NORMAL);
XMC_VADC_GLOBAL_StartupCalibration(VADC);
XMC_VADC_GROUP_QueueInit(VADC_G0, &g_queue_handle);
XMC_VADC_GROUP_ChannelInit(VADC_G0, 4, &g_g0_ch4_handle);
```

Note:Do the channel initialization for the rest of the channels (group 0 and 1 each channel 0 and 1) accordingly.

Now the entries are inserted to group 0. Group 0 is set to be master and group 1 is set to be slave. Please notice that there is no request source necessary for the synchronized channels.

```
XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_0_handle);
XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_1_handle);
XMC_VADC_GROUP_QueueInsertChannel(VADC_G0, g_queue_entry_2_handle);
XMC_VADC_GROUP_SetSyncMaster(VADC_G0);
XMC_VADC_GROUP_SetSyncSlave(VADC_G1, 0, 1);
XMC_VADC_GROUP_CheckSlaveReadiness(VADC_G1, 1);
XMC_VADC_GROUP_CheckSlaveReadiness(VADC_G0, 1);
```

The CCU8 configuration is explained in a separate application note. After CCU8 is implemented, the results can be read in a loop.

```
while (1U) {
    result_0 = XMC_VADC_GROUP_GetResult(VADC_G0, 4);
```

Note:Fetch the rest of the results (group 0 and 1 each channel 0 and 1) from the result registers accordingly.

2.6.4 Synchronous Conversion – Using the Multiple Sources

In the following figure a 'Multi-Source with Synchronization' example is shown to clarify that the synchronized slave channels do not need any request source.

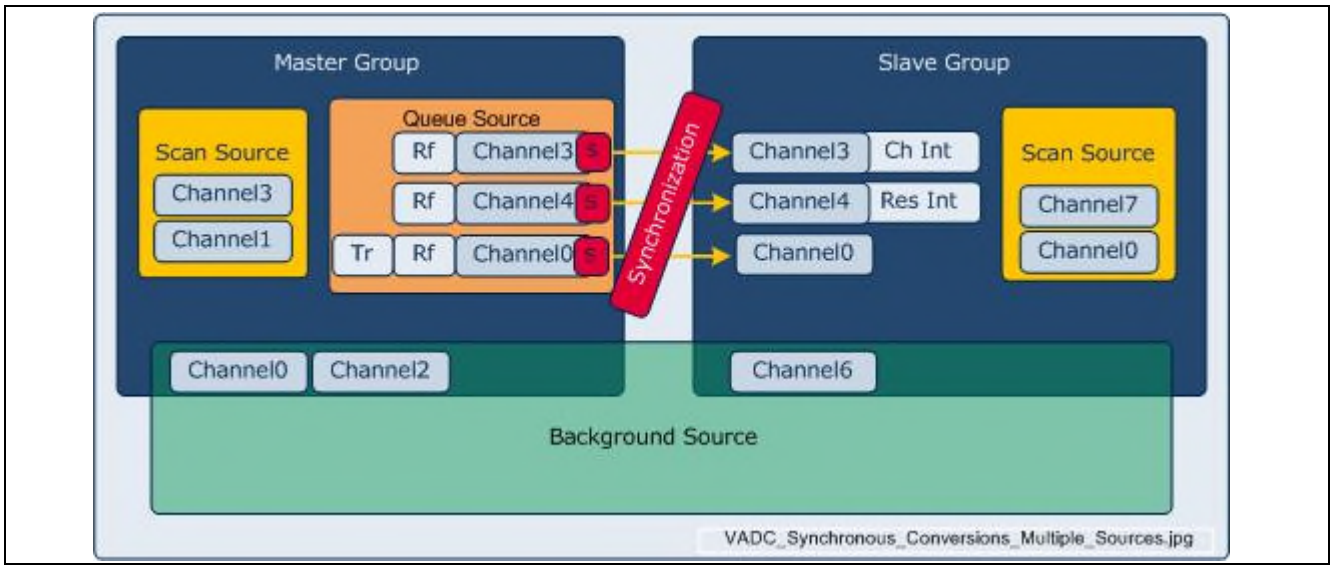

Figure 16 Multiple Source Synchronization example

- For the master group, a Scan Request Source sequence with channel 1 and channel 3; Queue Request Source is requesting 3 channels, all of them synchronized.
- For the slave, a Scan Request Source sequence with channel 0 and channel 7.
- The Background Request Source is requesting channel 0 and channel 2 of the master group, and channel 6 of the slave.

Channel 3 in the slave group is configured to produce a Channel Interrupt when its conversion has finished, while channel 3 in the master group is not configured for this.

A new result of channel 4 produces a Result Interrupt in the slave group.

Steps to configure the Multiple Source Synchronization example:

1. Define the global VADC and clock configuration.
2. Set the synchronization configuration for master group x, in the Synchronization Control Register. Select the role of the group (Master, TSEL=b00) and enable control by the “READY” signal (EVALRa=b1, being the slave group).
3. Set the synchronization configuration for slave group a, in the Synchronization Control Register. Select the role of the group (STSEL, Slave of CI1 (b01), CI2 (b10) or CI3 (b11))
4. Configure the Request Source for the master group x.
5. Configure the Request Source for the slave group a.
6. For the master group x, in the Channel Control Register (GxCHCTRy), choose the Result Register numbers to store the conversions and request a synchronized conversion of the chosen channels (SYNC = b1).
7. For the slave group a, in the Channel Control Register (GxCHCTRy), choose the Result Register numbers to store the conversions. A Channel event (CHEVMODE=b11) is configured in channel 3.
8. Enable and configure the ALIAS in Slave group.

9. Enable the Result Service Request generation.
10. Set up the event generation.
11. Enable the Arbitration. The analog converter control has to be set as per normal operation, it has to run permanently (ARBM=b0), and the Arbitration Slot for the Request Source must be enabled.

Note:The master group has to be initialized after the slave in order for arbiters to run synchronously.

12. Calibration safe loop. Wait until calibration is finished.
13. Configure a Timer in order to generate the trigger signal.

2.7 Time-Equidistant Sampling

In several types of applications, input data has to be optimized, for example, a filter or audio application. XMC4000 devices allow the user to define a fixed time raster to execute conversions. There are different approaches to achieve this, but the most accurate method is described here.

In the next figure a signal provided by the ADC (ARBCNT-one pulse per arbitration round) serves as the time base of the timer CCU4. In this way the maximum coupling between the ADC and the timer is achieved.

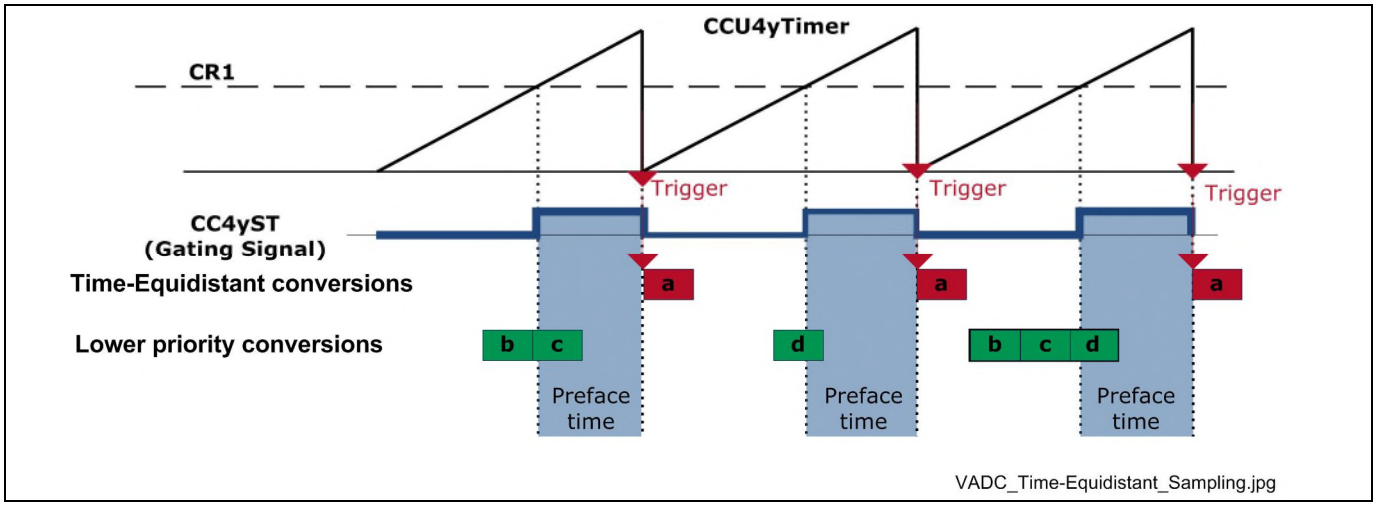

Figure 17 Timer Mode and Signal trigger for Equidistant Sampling

The timer is generating a level signal which gates conversions in the ADC, and a second pulse signal which might be the period match. The frequency of this timer (signal REQTRx) defines the sampling rate and its high time defines the preface time interval where the corresponding request source takes part in the arbitration. The time-equidistant conversion must be set to the highest priority. The preface time between the request trigger and conversion start must be long enough for a currently active conversion to finish.

Configuration steps

1. Define the global VADC register and clock configuration.
2. Configure the Request Source. In the Source Control Register (GxQCTRL0 or GxASCTR), enable the mode for equidistant sampling (TMEN=b1, TMWC=b1) and the standard gating must be enabled (ENGT = b10 or b11).Choose, as usual, the sequence needed.
3. Configure the request source to be gated by this level signal (CCU4x.ST) and be triggered by the service request generated by CCU4x period match (check interconnect section in reference manual). This is selected in control registers of each request source.
4. In the Channel Control Register choose the Result Register(s) to store the conversions.

5. Enable the Arbitration. Enable the arbitration slots as late as possible to avoid triggering of un-configured conversions. The analog converter control has to be set as normal operation and the Arbitration Slot for Queue Request Source enable.
6. Include a Calibration safe loop. Wait until calibration is finished.
7. Configure a timer (e.g CCU4) to be clocked by the GxARBCNT signal. This ensures thebest synchronicity. Generate a level signal with this timer that is high (respectively low) at least longer that the ADC conversion time of the channel to be equidistant. The period match can be used as a trigger for the ADC.

Note:Equidistant sampling is not supported by the Background Request Source. It is also possible to do equidistant sampling for more than one request source in parallel if the preface times and the equidistant conversions do not overlap.

2.8 High Conversion Rate

Some applications need a very high conversion rate. This can be continuous sampling (e.g. audio signals) or burst sampling (e.g. to analyze the dynamic behavior of a signal). The VADC in XMC4000 devices offers several ways of achieving a high conversion rate.

Some of the parameters that can improve the frequency rate are:

a) Analog clock frequency: The conversion clock frequency is configured with the DIVA parameter in the GLOBCFG Register, fADCI = fADC / (DIVA+1), where fADCI is the analog clock frequency and fADC is the module ADC frequency, the CPU clock frequency. The minimum value supported for the analog clock is 30MHz, corresponding to a value of DIVA=3. DIVA=1 is possible, but a loss of accuracy must be expected due to the “out of specification” condition.

b) Digital clock frequency: It is possible to define a divider for the arbiter clock frequency. Setting this to 0 ensures maximum arbiter frequency (bitfield DIVD in the GLOBCFG Register).

c) Sample time: It is possible to control the sampling time by adding clock cycles to the minimum sample phase of 2 analog clock cycles. For a high conversion rate the sample time has to be set to its minimum (STCS=b0).

d) Resolution

- Standard conversions: There are three possible resolutions in standard mode: 12, 10 or 8 bits. The higher the resolution, the longer the conversion takes.
- Fast compare mode: In this 1 bit resolution mode, the selected input voltage is directly compared with a digital 10 bit value that is stored in the corresponding result register. This is the fastest way to know if the compared input voltage is above or below the given reference value.

e) Post calibration: After converting a channel post calibration is executed. This can be disabled to reduce the conversion time, but this can also lead to a loss of accuracy from a long-term perspective.

f) Arbitration run mode: The arbitration can be configured to run permanently or to start after a request has arrived (GxARBCFG.ARBM). The fastest reaction is achieved when the arbitration starts after a request. This ensures that the request does not arrive in the middle of an arbitration round, and so it does not have to wait until the next arbitration round to be considered in the arbitration.

g) Arbitration round length: The number of arbitration slots per arbitration round can be modified in multiples of 4. The time spent per arbitration round can also be configured. For a high conversion rate, the bitfield ARBRND in the GxARBCFG register should be 0, to set a minimal arbitration round length.

In addition timing mode and the parameters described, the request sources can be configured to convert at maximum speed:

- If the application is using just One Request Source, it should be configured to start conversion automatically after a conversion is finished. Set Queue Request Source as 'Refill', and set Scan and Background Request Sources as 'Autoscan'.
- If the application is using Multiple Request Sources, the source in which the high conversion rate is required should be configured with the higher priority and cancel the inject-repeat-mode. Of course the priority is only needed if other request sources are configured in the same group. A dedicated group for this sequence helps to improve the performance in terms of conversion rate.

Note: Burst sampling of one signal through group chaining achieves the maximum sampling rate. The same signal can be routed to different VADC groups. Just after the first group sampling is finished, the second group is triggered by hardware. In this mode four signals can be sampled in approximately 200 ns. See Trigger Options/ Burst Sampling.

Problems with a high sampling rate and possible solutions

Fast conversions usually require a fast memory transfer from the result registers to memory buffer. Thiscan lead to tight real-time conditions for software, or even for the memory bus in the case of a DMA transfer. To relax this situation the XMC4000 VADC includes special result handling, which makes it possible to build up a result FIFO by queuing several result registers. See section: 5 Result Handling Use Cases.

In each ADC group the eight channels can be assigned individually to 17 result registers. For fast consecutive conversion of the same channel (for example, using the Queue Request Source), a result buffer can be implemented using a FIFO mechanism. Therefore several result registers can be linked together.

Another option is to assign one result register to several channels and use the 'wait-for-read' mode. This suspends the start of conversion for any channel which is assigned to this result register until the result has been read. This is helpful especially for conversion sequences with a mixture of time non-critical channels triggered by the Scan Request Source, and time critical channels triggered by the Queue Request Source. Both options avoid data loss by overwriting a result register before it is read. See the section: 5.2.2 Wait-for-Read.

A DMA can be used to transfer the content of the result register to any other location in RAM. The trigger of the DMA transfer can be easily linked to the result event (new result available) to ensure maximum throughput and software independent transfers (deterministic).

The following figure shows four high sampling rate conversions of channel 6 which are be buffered by a result FIFO.

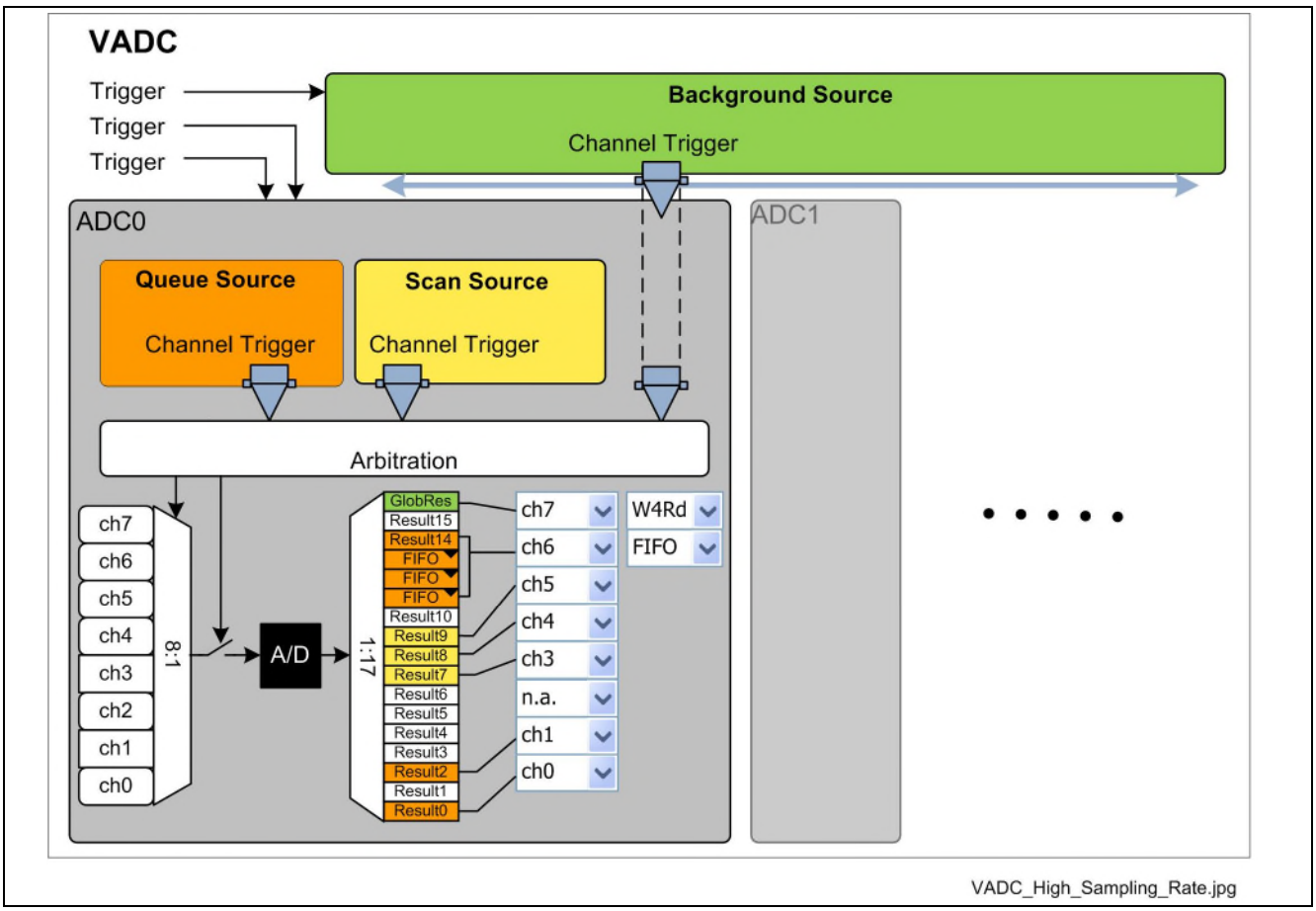

Figure 18 High sampling rate conversions using result buffering and FIFO

The reading of the result has to be from Result11 (the lowest FIFO element). The Background Request Source triggers channel 7 which is assigned to the Global Result Register and is set to wait-for-read mode. This prohibits the unintentional overwriting of the result when a new background conversion is requested but the result has not yet been read. Two more conversions (channel 0 and channel 1) are triggered by the Queue Request Source and can be stored in independent result registers. Additionally there are three conversions (channel 3, 4, 5) which are triggered by the Scan Request Source.

2.9 Runtime Handling of Conversions

2.9.1 Stopping Conversions

To stop a conversion or sequence of conversions, the recommendation is to stop the responsible request source. This ensures that the arbiter does not take over any other request from this source and at the same time keeps the pending bit registers unmodified. For more details on how to stop the sequences, please refer to the XMC4000 reference manual.

2.9.2 Re-Configuring Request Source Sequences at Runtime

By changing the GxQINR0, GxASSEL or BRSSEL registers, depending on the request source, it is possible to re-configure sequences during runtime for debugging purposes. It is also usefull for applications that need to re-configure the request sequence in real-time.

The ALIAS register can also be re-configured during runtime so the application can, at any time, change the requested sequences and the analog channel converted with the configuration fixed for the channel 1 and channel 0.

2.9.3 Flush the Queue Request Source

By setting the FLUSH bitfield of the Queue 0 Mode Register GxQMR0 to 1, it is possible to clear all queue entries, including the back-up stage and the event flag EV during runtime.

When stopping a conversion during runtime it is best to flush the queue before filling it with a new sequence. This ensures that there are no old conversion requests in the FIFO that could lead to an incorrect conversion sequence.

2.10 Tips, Tricks and Pitfalls

This section provides some hints and insights into more advanced features of the VADC.

2.10.1 Pending Channel Registers

The following figure demonstrates how the Scan mechanism (either 'Scan Request Source' or 'Background Request Source') works.

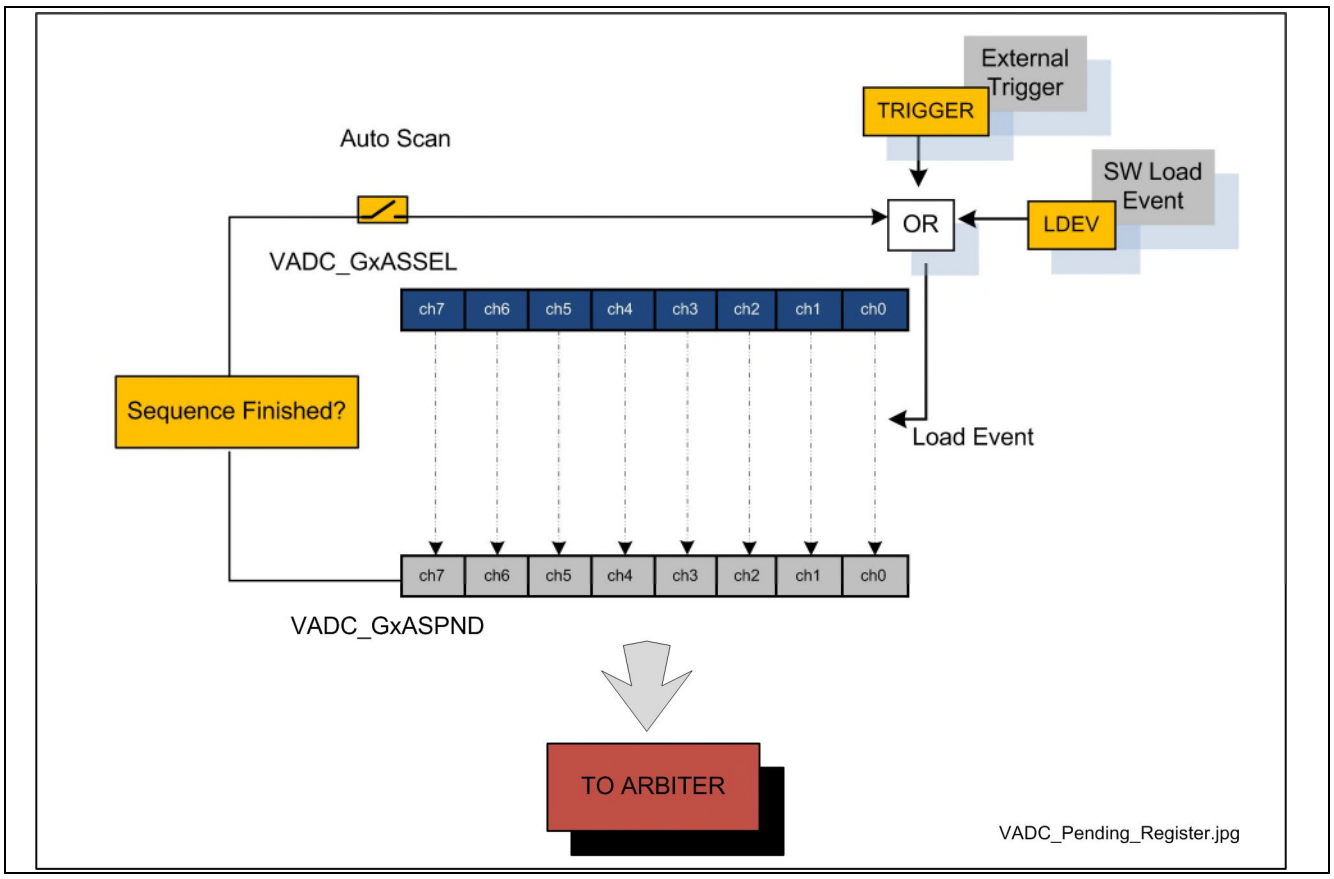

Figure 19 Pending GxASPND Register

After a load event (caused by a trigger, the autoscan mechanism, or a software instruction) the content of register GxASSEL is copied to register GxASPND (pending register), which is the real interface to the arbiter.

Whenever a channel in the sequence is converted, the pending register is updated by clearing the corresponding 1 bit.

Any new load event again copies register GxASSEL to GXASPND, even if the whole sequence has not yet been fully converted. The way in which this is done can be modified by the load mode bitfield. Figure 20 shows an example of this for a Scan Request Source sequence.

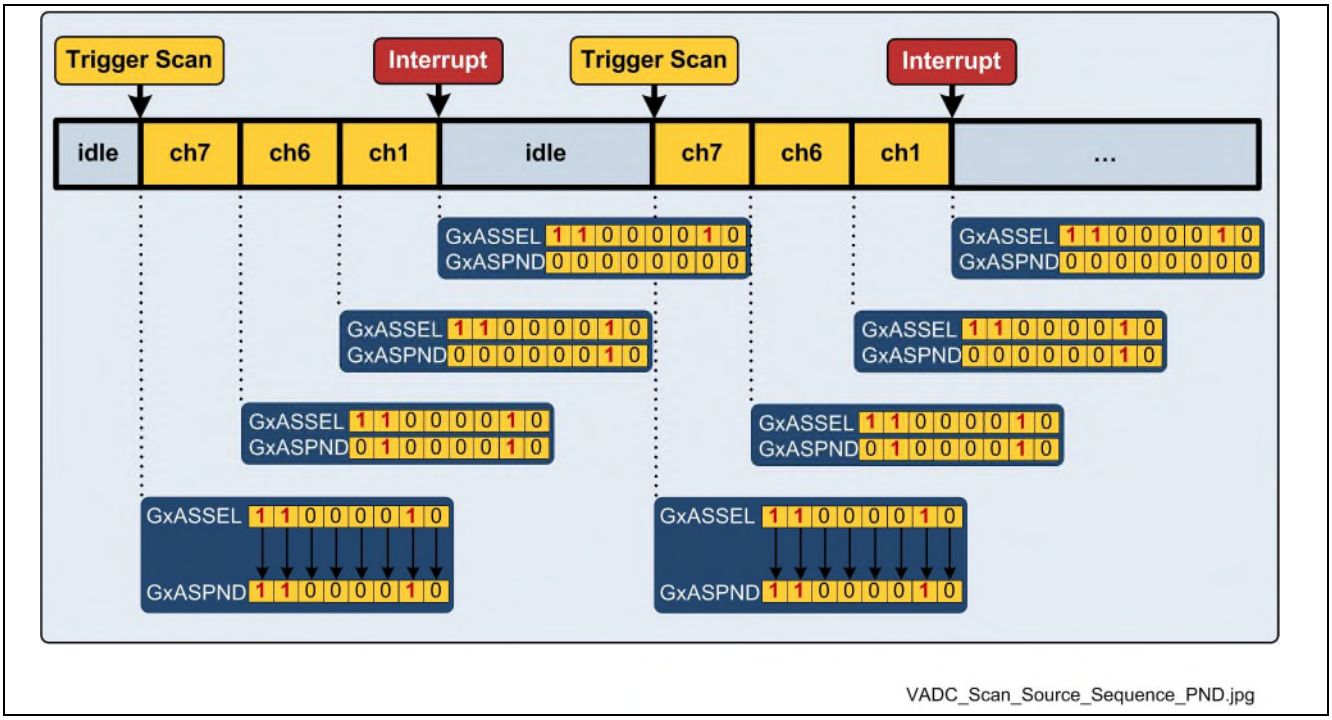

Figure 20 Pending Register in a Scan Request Source sequence

The Scan Request Source looks always to the highest channel number in the GxASPND register in order to make the request to the arbiter. When a load event occurs before the whole sequence is converted, the lowest channel numbers is not requested.

Pending register bitfields can be manually filled while the application is running, making it possible to convert a channel at any time without the need of a Load Event or a finished sequence signal with Autoscan. This is very useful for debugging purposes.

Given that the Background Request Source is based on the same mechanism as the Scan Request Source, the copy from the BRSSEL to BRSPND register happens in a similar way after a load event (software event, Autoscan or hardware through a trigger), as shown in the following figure:

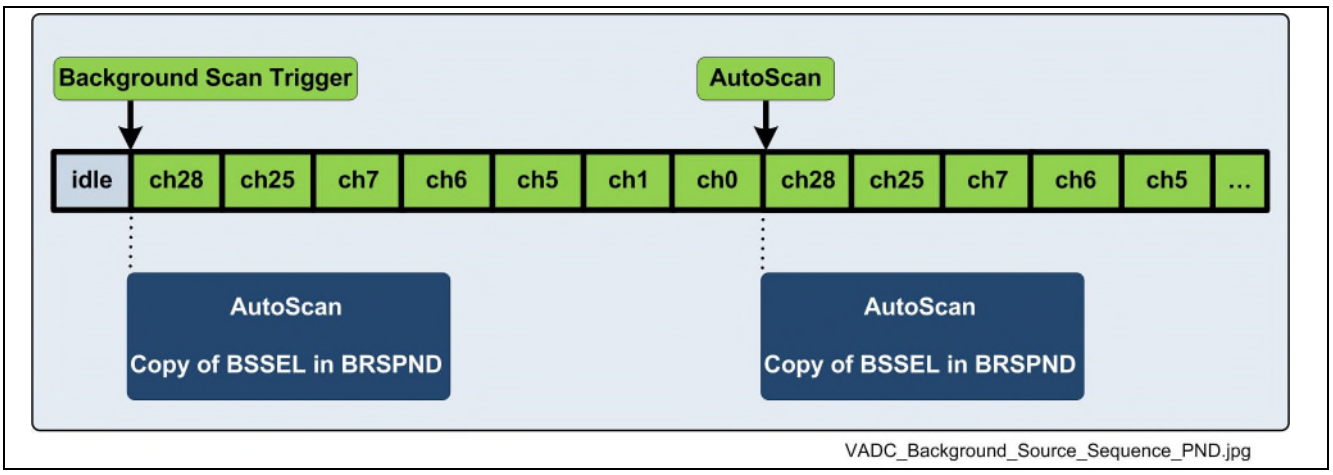

Figure 21 Pending Register in a Background Request Source sequence

3 The ADC Converter

3.1 Introduction

Many features of the ADC converter in the XMC4000 microcontroller family can be easily configured to adapt its operation to each applications needs.

In complex systems several signals and sensors often have to be measured. Usually the signals are not the same type: some are short pulses like shunt currents, some are high-ohmic such as temperature, light or pressure-sensor signals, and some others do not reach the full-scale but just a fraction. For best measurement performance it is necessary to adapt the channel to the signal characteristic:

- It is possible to assign each channel to a conversion input class. Different sample timings and resolution scan be configured for two group-specific classes and two global classes.
- An additional voltage reference lower than the standard VAREF can be used per group. This allows the use of different references for distinct signals.
- Each conversion can be easily assigned to a result register. There are more result registers (17) to channels (8) so this relaxes the real-time critical reading of the result.

When more than one request source is used, a configurable arbitration process takes place and decides which conversion gets priority. If a request for a high priority conversion arrives while a low priority conversion is in progress, there are two Conversion Start Modes:

- finish the current conversion (wait-for-start)
- cancel the current conversion, inject the high-priority conversion and then repeat the cancelled conversion afterwards (cancel-inject-repeat).

With these resources, the user has full control of the conversion sequence including the interaction among different concurrent conversion requests.

The alias feature increases the flexibility of the ADC and gives other advantages. It redirects conversion requests for channels 0 and/or 1 to other channel numbers but maintains the settings of channel 0 and/or 1.

Several safety features, such as **Broken Wire Detection** and other testing features are also available in the diverse configuration options of the ADC Converter in the XMC4000 family of devices.

3.2 Channel Configuration

3.2.1 Correspondence of Each Analog Pin to an ADC Channel

The correspondence between analog pins and ADC channels will vary depending on which XMC4000 device and package are chosen. The same pin can be connected to two different channels in distinct groups. The following figure illustrates the relation between pins and channels for the currently available members of the XMC4000 family.

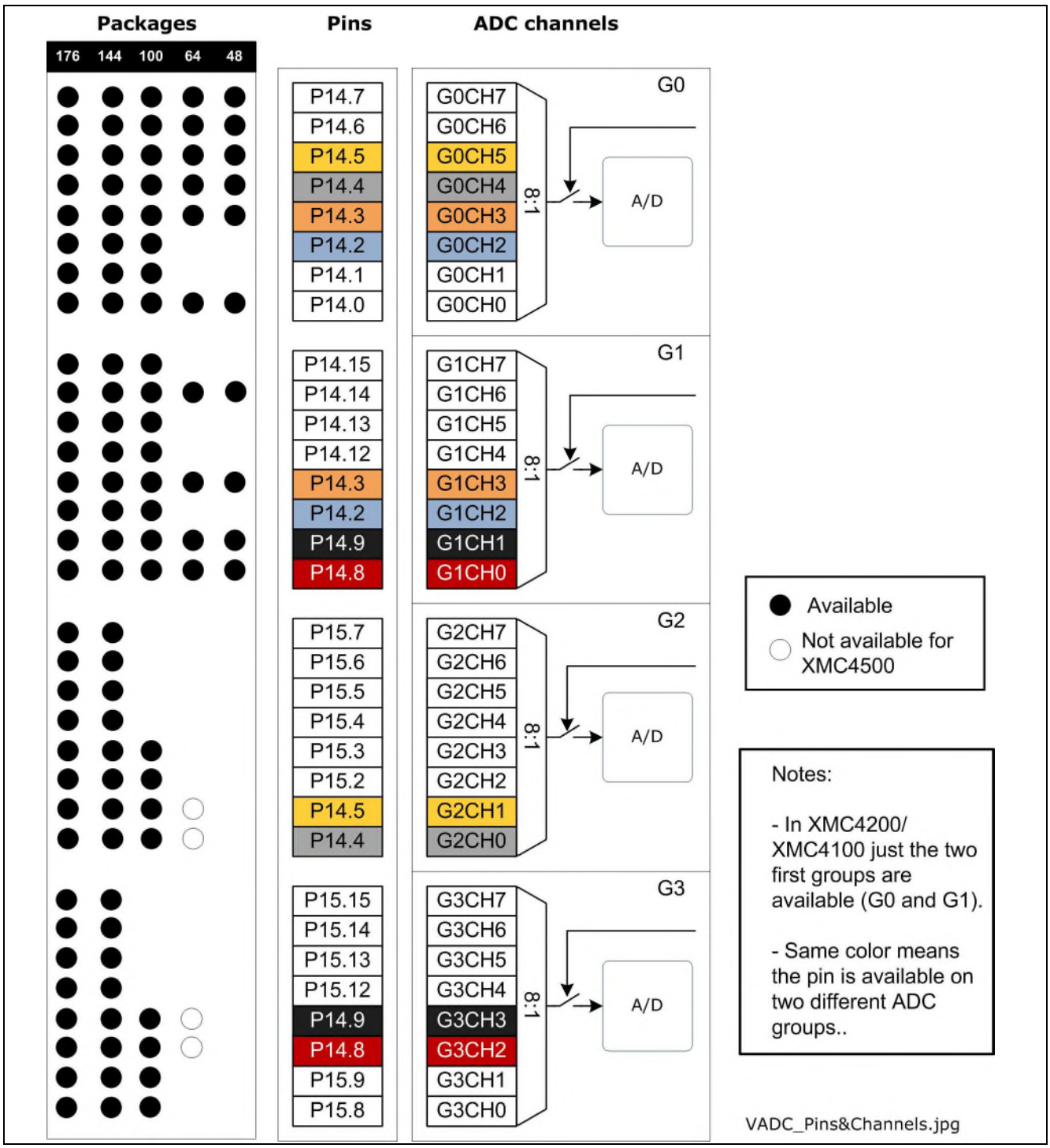

Figure 22 Correspondence between analog pins and ADC channels

3.2.2 Resolution and Sampling Time Configuration - Select and Configure Input Classes

To adapt the ADC configuration to distinct signal characteristics, different sample times and resolution must be configured. To support this XMC4000 devices provide two group-specific classes and two global classes.

Each channel must then be assigned to one of them. There is therefore no need for runtime configuration in order to adapt different signals.

To configure and use this feature, follow these steps:

1. Configure the resolution and sample time in one of the classes, either in group specific (GxICLASS0 and GxICLASS1 registers) or global classes (GLOBICLASS0 and GLOBICLASS1 registers):
   - resolution (CMS): select one of the available conversion modes; Standard conversion with 12, 10 or 8 bits of resolution, or 1 bit resolution in Fast Compare mode.
   - sample time (STCS): indicate the additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles.
2. Assign the channel to the selected Input Class in the ICLSEL bitfield of the respective Channel Control Register (GxCHCTRy).

The figure which follows shows three different signals being measured: temperature, light and shunt of current. A different class is defined for each sort of sensor, depending on its characteristics:

- Global Class 1 for the temperature sensor
- Input Class 0 of group 1 for the light sensor
- Global Class 0 for the shunt measurement

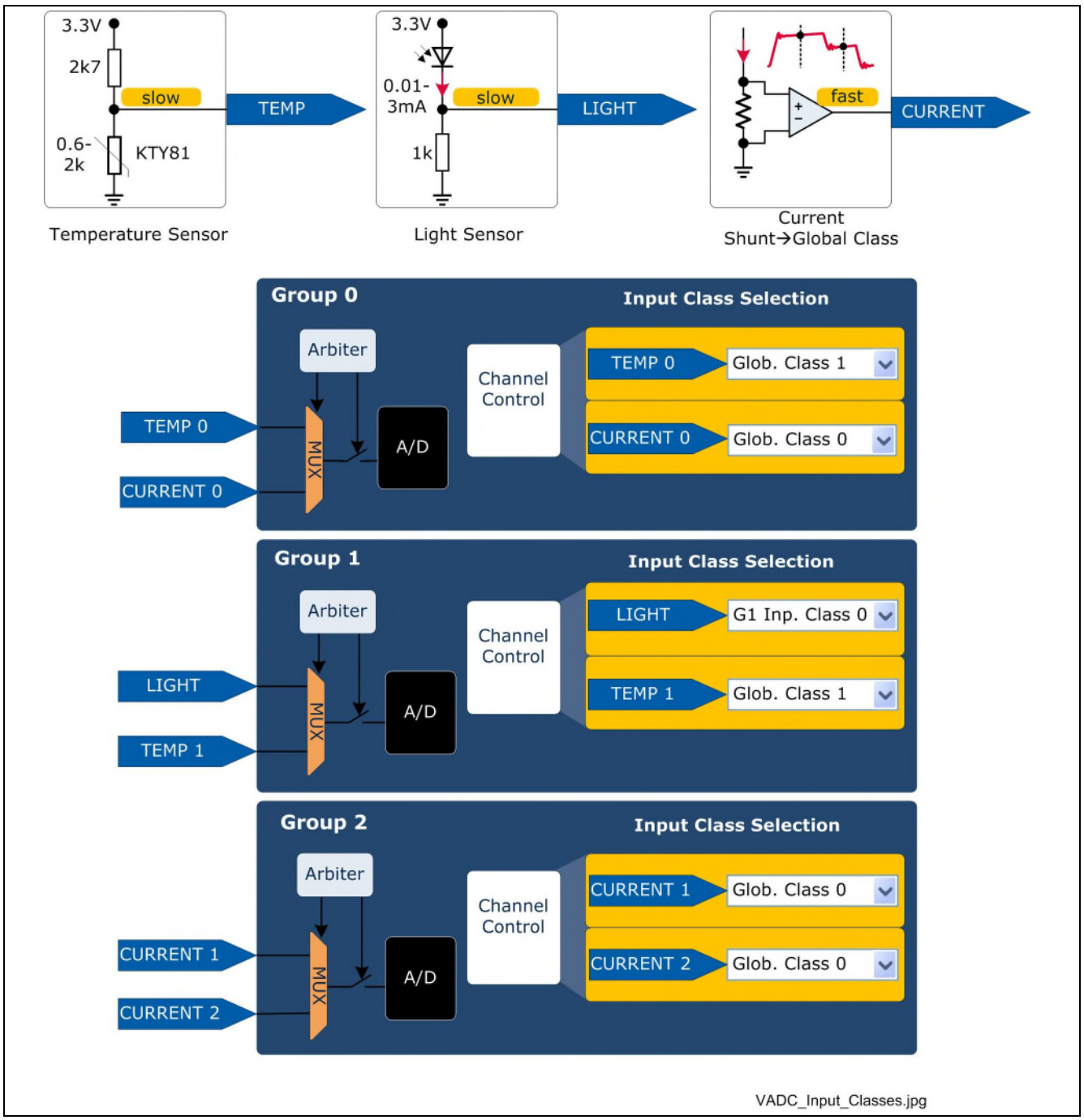

Figure 23 **Multiple sensor measurement with adaptation to the sensor characteristic via the input class configuration**

3.2.3 Selecting the Result Register

3.2.3.1 Group Specific and Global Registers

Every application needs access to the results of its A/D conversions, so these are stored in result registers. Each group of the VADC in XMC4000 products has 16 result registers.

The Global Result Register is not related to any group, but is used for results specific to the Background Request Source.

To choose the result register to store a channel conversion in:

- For each channel select a result register in the RESREG bitfield of the respective Channel Control Register (GxCHCTRy).
  - For a conversion requested by the Background Request Source, the RESTBS bitfield in GxCHCTRy is used to determine if the result is stored in the group-specific register (RESREG) or in the Global Result Register.

The next figure shows a possible configuration for the XCM4500. In this example:

- channel 2 and channel 6 conversions are stored in the Result Register 3 of group 0
- channel 3 is stored in Result Register 12 of group 0
- channel 1 is stored in Result Register 2 of group 0
- channel 0 is stored in the Global Result Register.

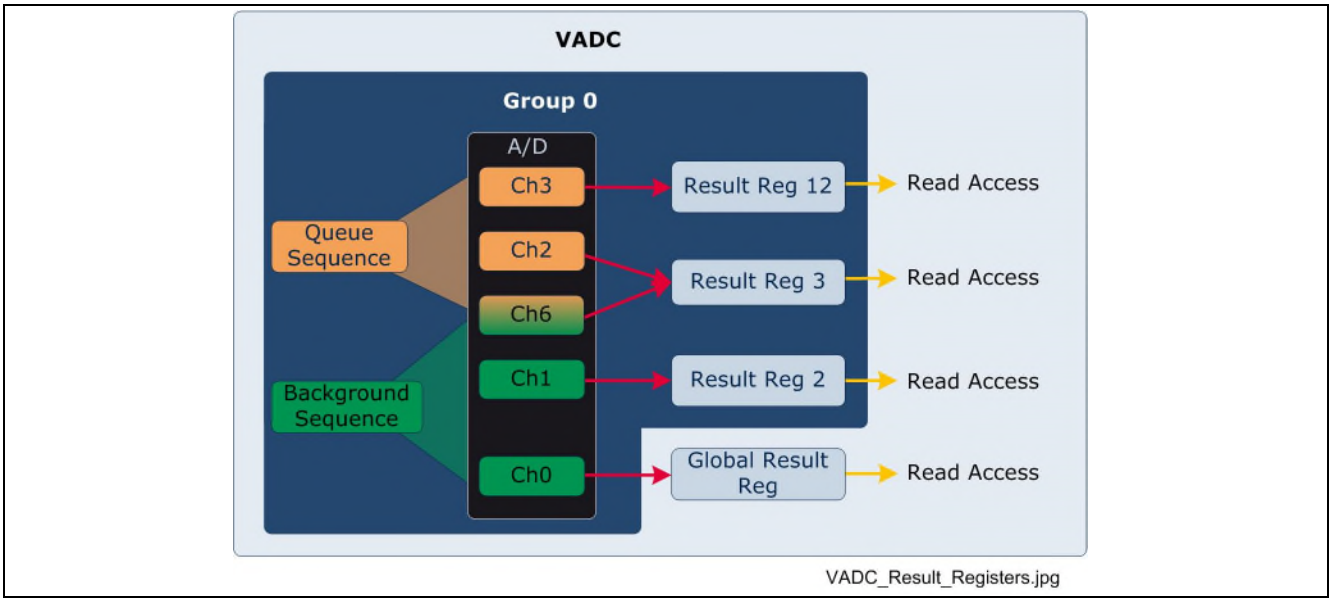

Figure 24 XMC4500 Result Registers example configuration

Note:Several channels can be configured to save the conversion result in the same Result Register but this can cause results to be overwritten.

3.2.3.2 Request Source Specific Result Register (XMC4100, 4200 and 4400)

The XMC4400, 4200, and 4100 devices additionally offer a 'Source–specific' Result Register that makes it possible to store the result of every conversion requested by this Source in a specific Result Register. The user can select the Result Register depending not only on the channel converted, but also on the source that requests this conversion (Request Source). This feature provides a high level of flexibility.

To use this feature, choose the Result Register that is to be used to store every result of a conversion requested by the same Request Source. This selection is made in the SRCRESREG bitfield of the Source Control Register for each Request Source (GxQCTRL0, GxASCTRL, and BRSCTRL).

As can be seen in the example in the next figure, every result of the conversion of channels requested by the 'Queue Request Source' (channel 3, channel 2 and channel 6) is stored in the Result Register 2.

Every result of the conversion of channels requested by the Background Request Source (channel 6, channel 1 and channel 0) is stored in the Result Register 3.

The conversion of channel 6 is allocated into Result Register 2 if converted by the Queue Request Source, or into Result Register 3 if converted by the Background Request Source.

If this feature is not required, it can be disabled by programming the SRCRESREG bitfield to b0000. The Result Register is then chosen by the setting of the RESREG bitfield of the GxCHTRy register.

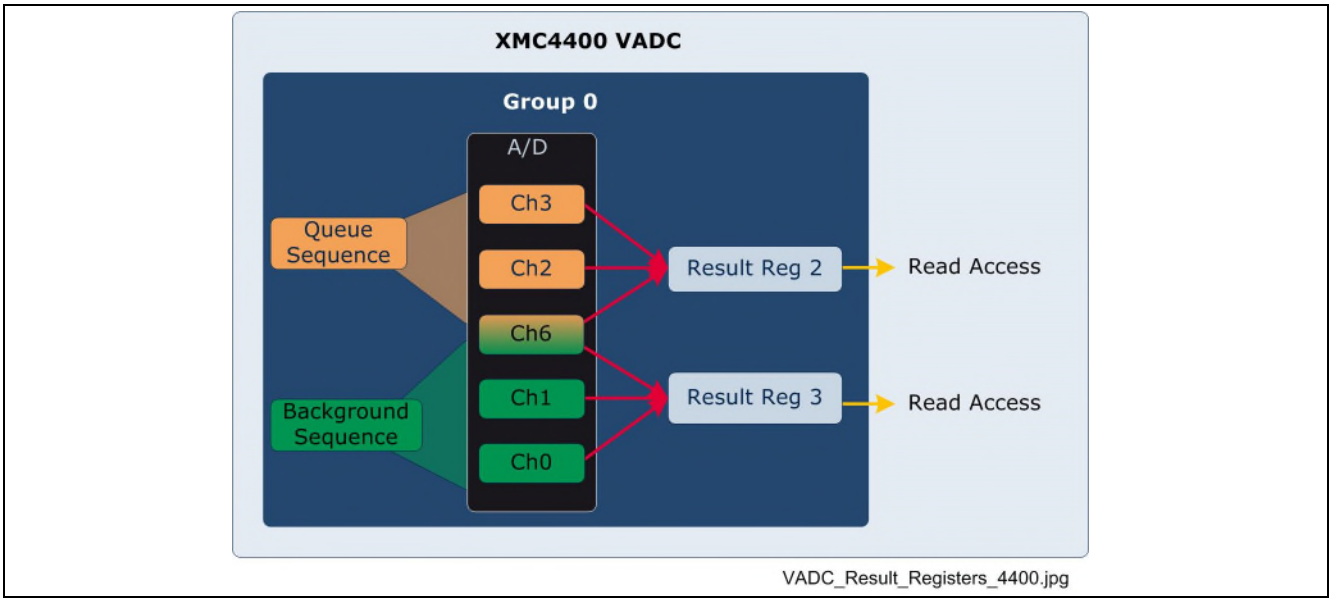

Figure 25 XMC4100, 4200 and 4400 devices: Result Register example configuration

3.2.4 Using Two Different Voltage References - Alternate Reference

In some applications the input signals to be measured never reach the full-scale of the reference voltage, but only a fraction of it. The ADC therefore allows an alternate reference voltage ALTREF on its analog input channel 0 in order to support such requirements.

The ALTREF is group-related which means each group can have an ALTREF.

Each channel in the group can select either the standard reference VAREF or the alternate one.

In order to configure an alternate reference in a group,the bitfield REFSEL in the GxCHCTRy Register has to be set to 1 (alternate reference input from channel 0), otherwise VAREF (standard reference) is chosen by default.

In case the alternate reference must be trimmed or calibrated for example using the XMC4000 Digital to Analog Converter (DAC), it is possible to measure it against the standard VAREF. Of course, the alternate reference consumes an input pin, channel 0, but the channel control logic on channel 0 can still be used. The alias feature is useful for this purpose.

In the next figure, the current through a shunt resistor is being measured:

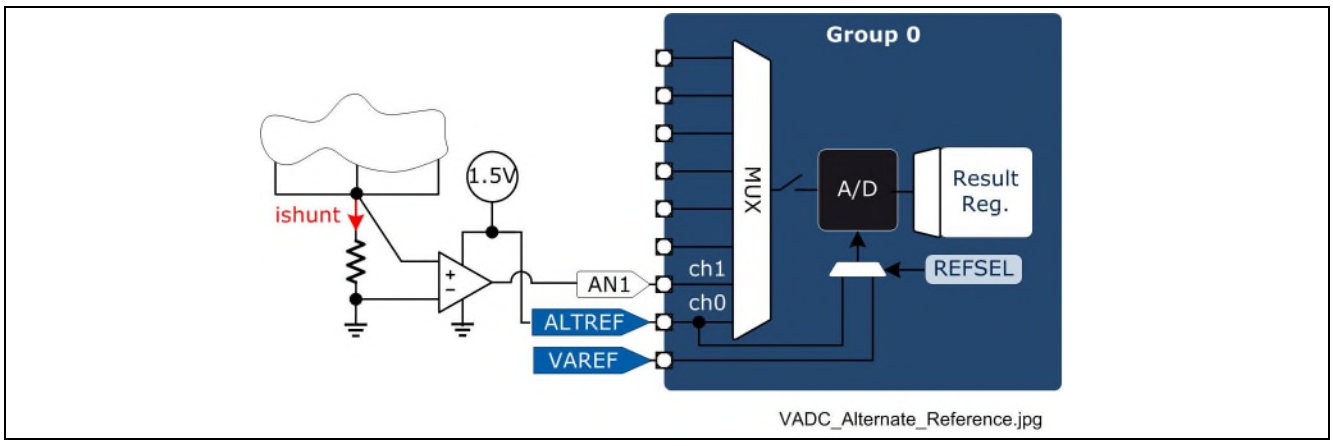

Figure 26 Selecting the Alternate Reference on ch0 via REFSEL

An Alternate Reference set to 1.5 Volts is connected to the pin for channel 0.

By modifying the REFSEL value to 0 or 1 in the GxCHCTRy register, the reference for this specific channel can be changed to VAREF or ALTREF.

Note:It is possible to connect the DAC generated signal in Channel 0 of every Group and, therefore, to use it as an alternate reference. This allows the user to configure the ALTREF as a dynamic signal, adapting it to the needs of every application without the need to use external devices to generate this voltage reference.

Note:Some channels cannot select an alternate reference. Please refer to the Reference Manual of the selected XMC4000 device to know which channel number supports conversions with an alternate reference.

3.2.4.1 Adjusting the alternate reference to provide gain

Changing the reference voltage can achieve gain. This is very useful for many applications, allowing for the use of cheaper (less accurate) external devices to obtain the same final gain.

The reference voltage defines the range of voltage inputs and therefore the size of the LSB. For the ADC of the XMC4000 devices, the LSB is defined as $VAREF/2^{12}$.

A smaller voltage for the voltage reference (ALTREF) will mean that all the output codes are constrained to a smaller voltage range and that the LSB will also be a smaller voltage ($ATREF/2^{12}$).

The dominant noise source is expected to be the quantization noise. Decreasing the voltage reference will decrease the LSB and therefore the quantization noise will contribute less to the overall result. The best noise performance will be achieved when the ALTREF matches the full-scale range of the input signal.

With a lower reference voltage, the majority of the output codes are used with the smaller range of input voltages. This is the same as providing gain in the ADC.

The next figure shows an example. Here the same voltage values have been measured using VAREF (3.3 Volts) and an ALTREF of 1.219 Volts. Almost the full range of the output codes are used in the ALTREF's measurements.

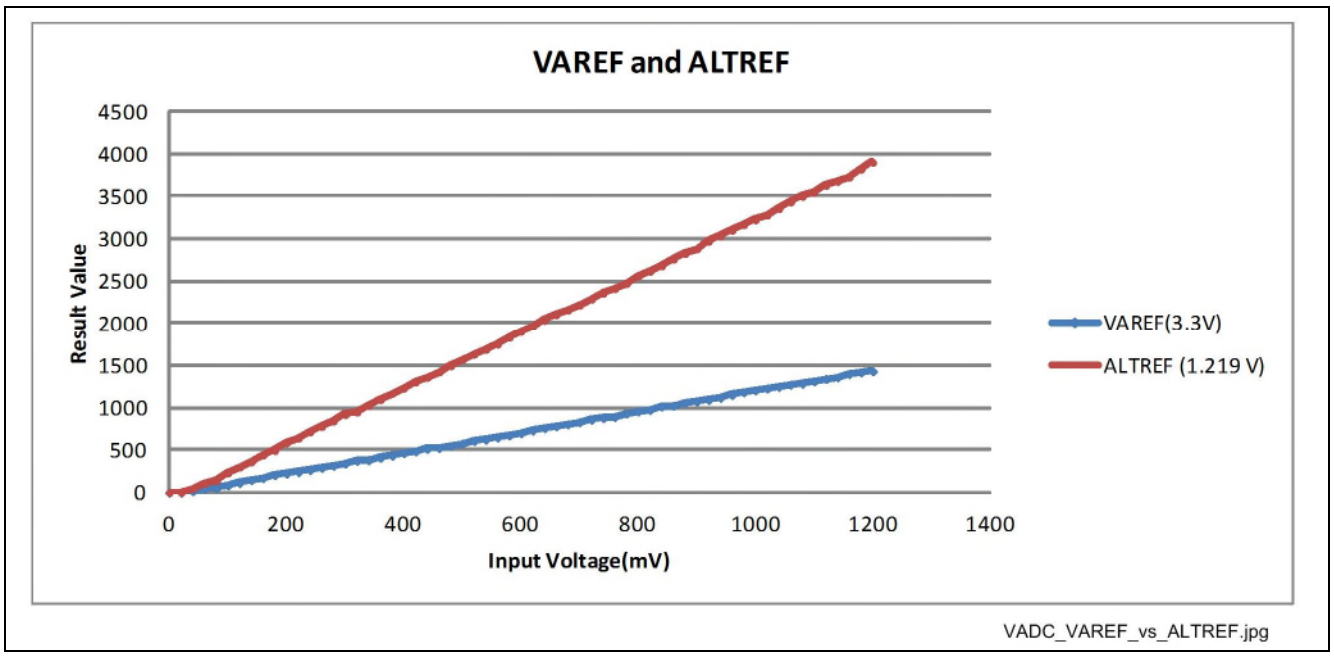

Figure 27 Providing gain using the ALTREF

3.2.5 Priority Channels

Each channel of a group can be defined as a Priority Channel. That means this channel can only be converted by its own group's request sources (Scan Request Source or Queue Request Source) but never by Background Request Source. A channel that is not assigned as a Priority Channel can be converted by the Background Request Source.

To program the Priority Channel, select the Priority Channel by setting the correspondent bitfield ASSCHy=b1 (where y = 0 to 7, the channel number) in the Channel Assignment Register (GxCHASS).

3.3 Understanding Arbitration

In many applications, complex sequences of conversions have to be executed by one device. For example, one application might need to use a Queue Request Source to convert a motor control current and some temperature sensors need to be measured with the Background Request Source. In these cases where a mixture of request sources is mandatory, the Arbiter must decide which conversion gets priority to access the A/D converter. If a request with high priority arrives while a low priority conversion is in progress, the highest priority is injected in the pipeline depending on the conversion mode (arbiter configuration).

In order to evaluate each Request Source, a certain time slot is assigned to each one (three slots for the Request Sources, and one extra slot for Synchronization). The sum of these arbitration slots forms the Arbitration Round.

As can be seen in the following figure, the arbiter outputs a pulse (ARBCNT) for each Arbitration Round. This can be used as the timing-base for the timer in time-equidistant sampling.

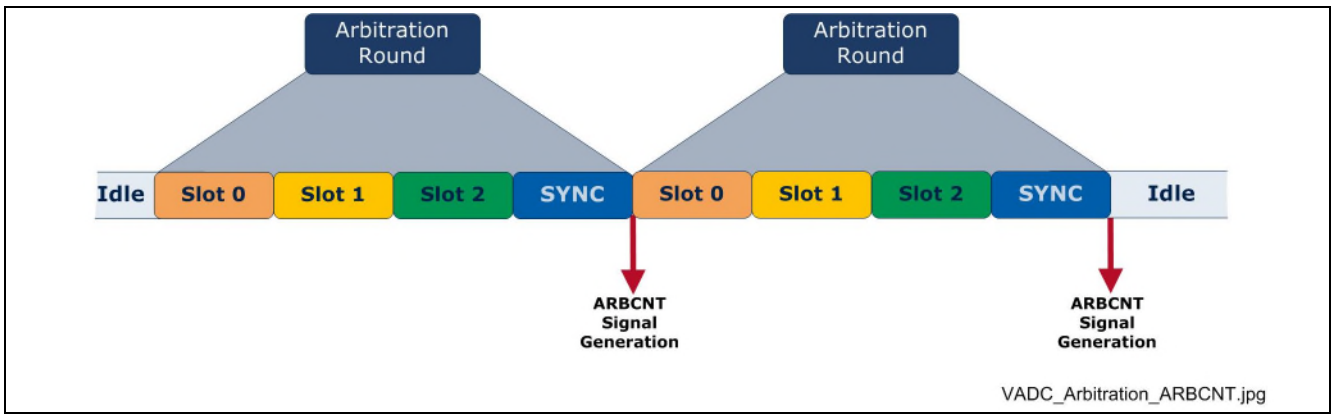

Figure 28 Arbitration round

For each arbitration round the arbiter outputs a 'winner' of the enabled sources, taking into account the Priority and the selected mode.

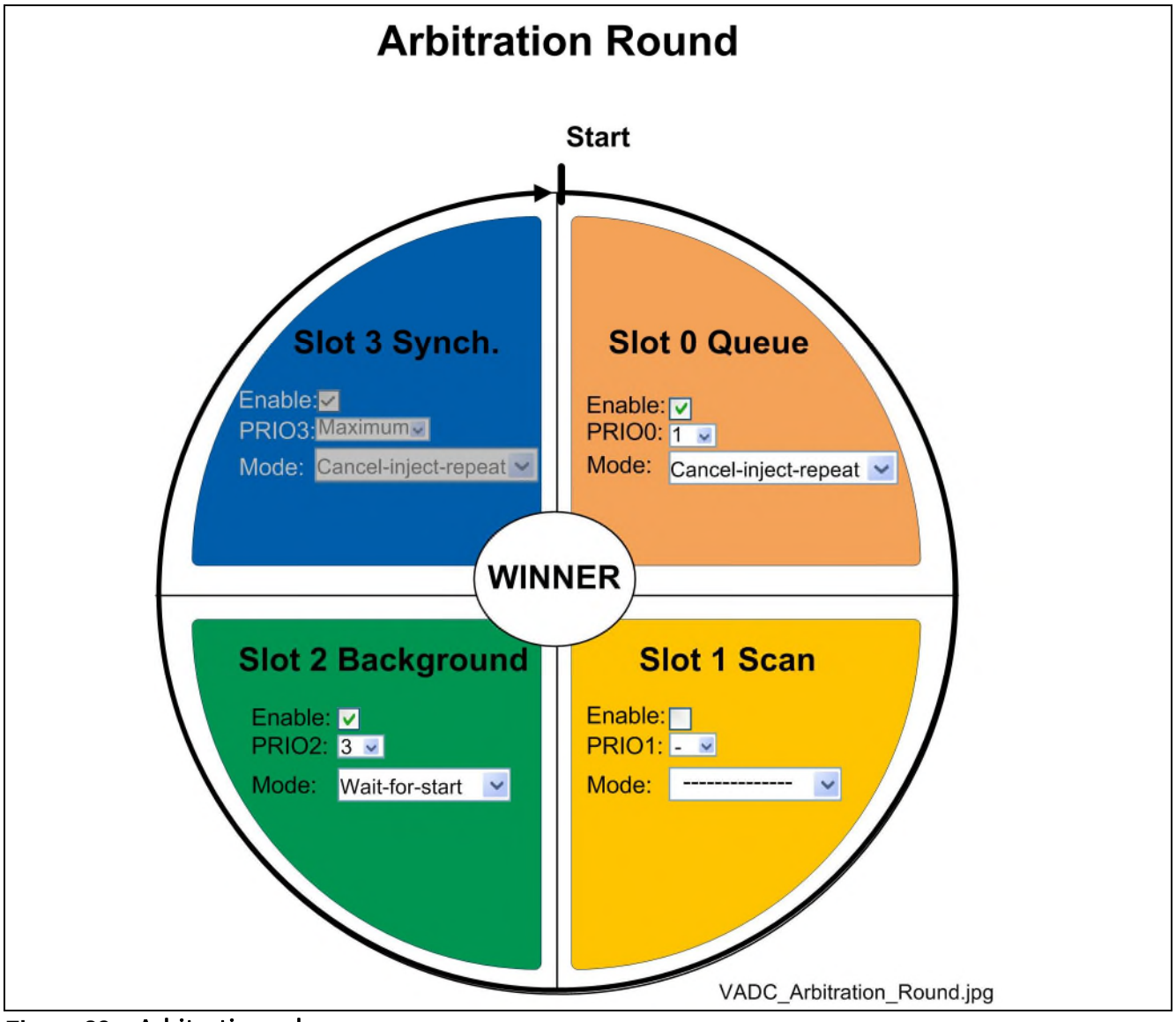

Figure 29 Arbitration schema

In Slot 3 (Synchronization), the priority and conversion mode cannot be configured because it is by default set to the highest priority. This Slot always wins the Arbitration Round. It is always set to cancel-inject, so synchronous conversions always cancel the current conversion.

Operation options for the Arbiter

- Arbiter running permanently
  - When the channel request arrives, the arbiter has to wait until the current Arbitration Round has finished before starting a new Arbitration Round where the request can be considered. The waiting time could be different for each new request as this is typically asynchronous to the arbitration process. This provokes a jitter effect. To ensure synchronicity, ARBCNT could be used.

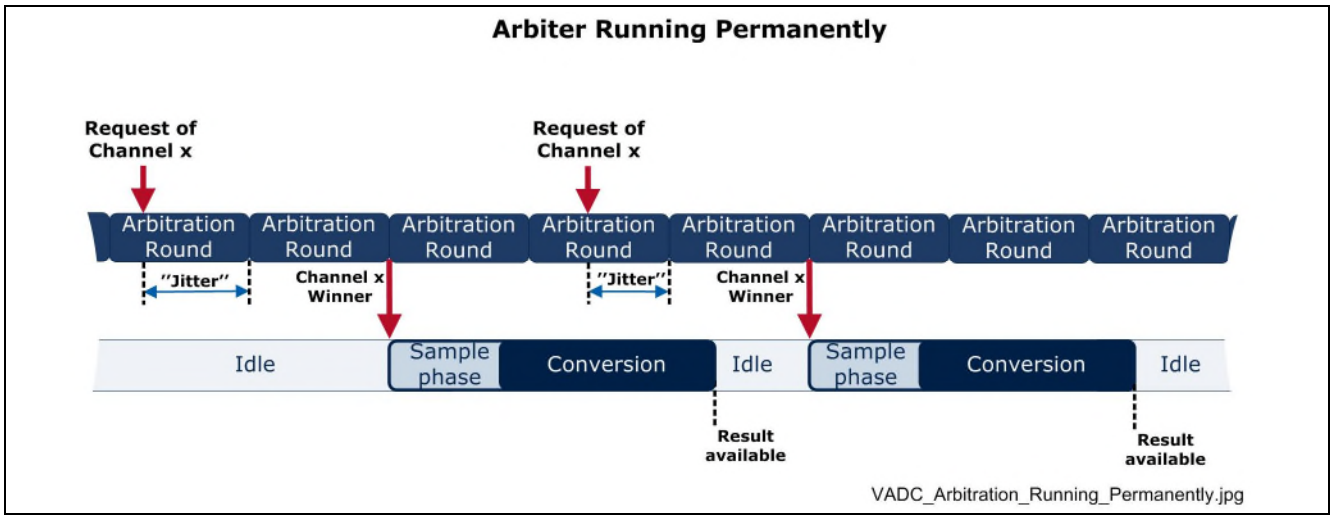

Figure 30 Arbiter running permanently

- Arbiter running after a request arrives
  - This option eliminates the jitter between conversions, starting a new Arbitration Round when a request arrives. In this mode, synchronized conversions between groups are not possible. This is because a parallel conversion in a slave group is not requested from any Request Source, so the Arbiter does not start running.

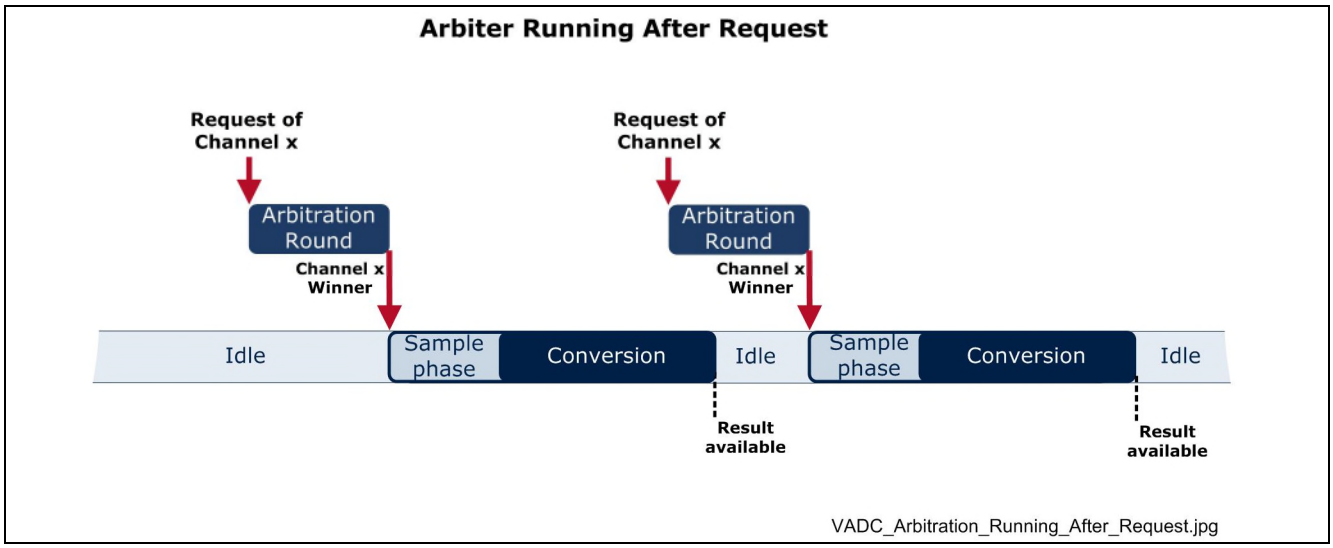

Figure 31 Arbiter running after a request arrives

3.4 Start Modes

The user can configure what happens when a running conversion must be aborted because a higher priority conversion request has been received. This configuration is made by selecting one of the following start modes:

- Wait-for-start mode
  - The Arbiter lets the current conversion finish and then the conversion with higher priority is converted after that. This mode provides maximum throughput, but can produce a jitter for the higher priority conversion. In the example below, the Higher Priority sequence (Queue Request Source) is configured as Wait-for-start mode, so it waits until the Lower Priority sequence (Scan Request Source) has finished its conversion.

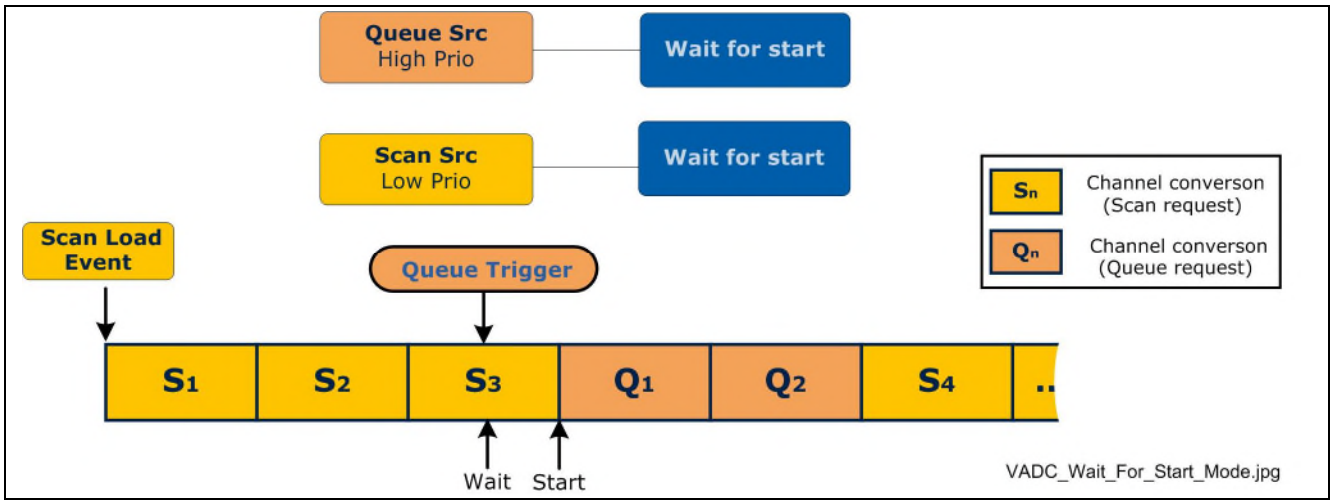

Figure 32 Wait-for-start mode

- Cancel-inject-repeat mode
  - The current conversion with lower priority is aborted and the conversion of the arbitration winner is started after the abortion (3f_{ADC} cycles). This mode provides minimum jitter for the higher priority conversions, but reduces the overall throughput. In the example below, the Higher Priority Queue Request Source is configured as Cancel-inject-repeat mode, so it cancels the current Lower Priority sequence Scan Request Source conversion. The S_3 cancelled conversion is repeated after the queue has finished its sequence.

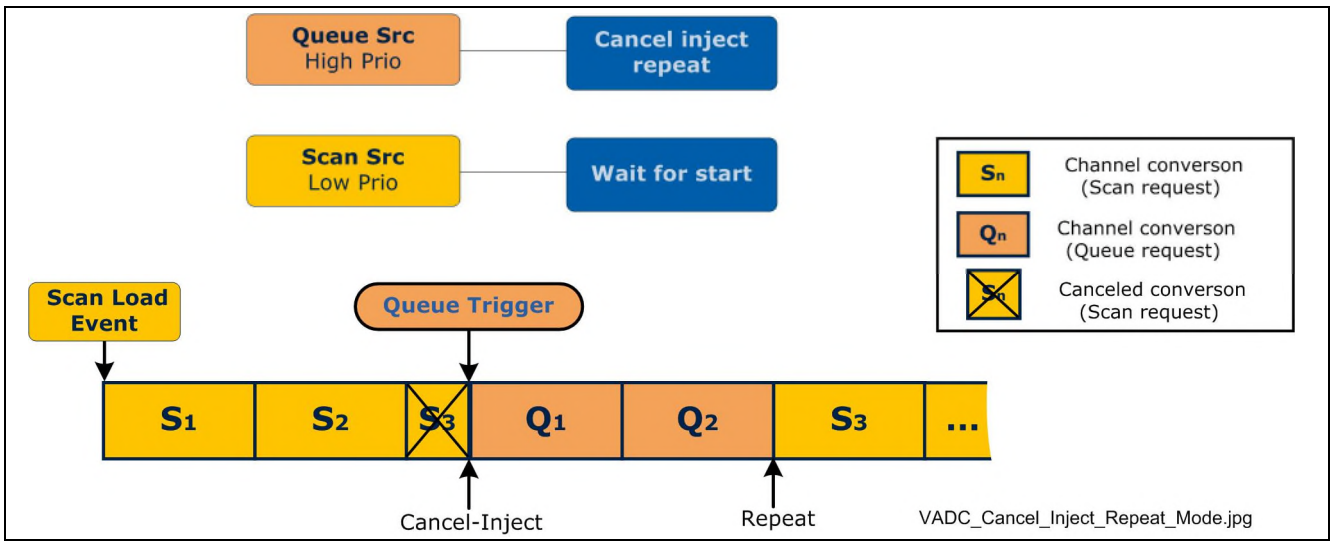

Figure 33 Cancel-inject-repeat mode

In the cancel-inject-repeat mode there is an additional variant to select whether the canceled conversion is repeated or not, after the higher priority has been finished. This is configured in the RPTDIS bitfield.

An example follows. Here, the S_3 conversion is cancelled when the Queue Trigger arrives, but is not repeated after the Queue sequence has finished.

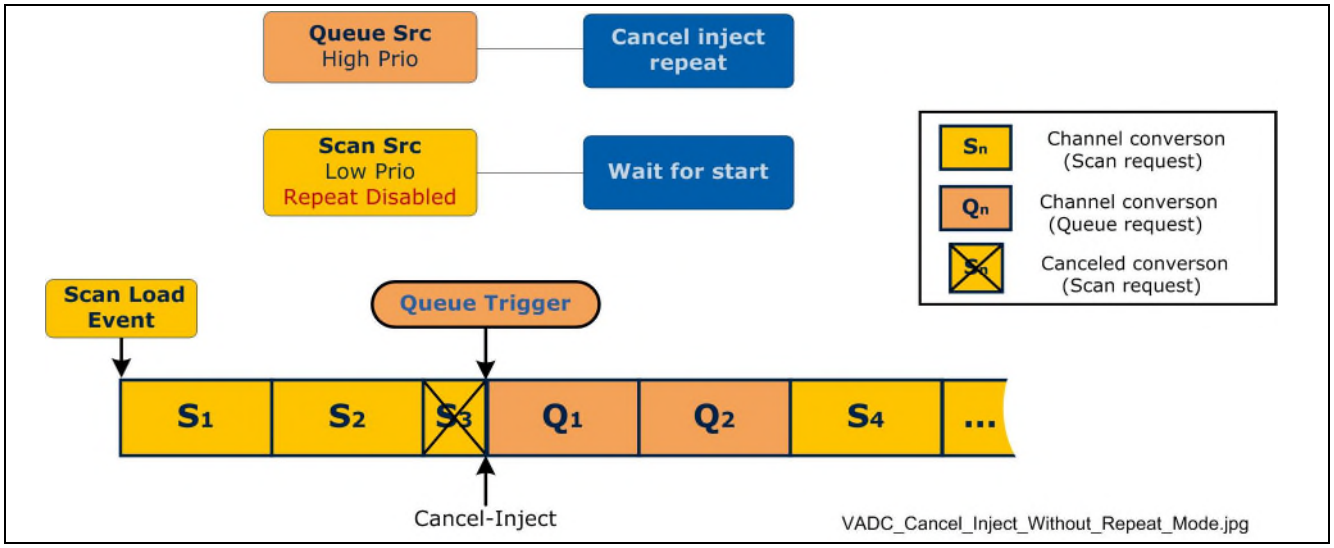

Figure 34 Cancel-inject without repeat mode

To configure the Start mode in each request source:

1. Choose the conversion start mode for each request source in the GxARBPR register bitfields:
   - CSM0 (Arbitration Slot 0, Queue Request Source)
   - CSM1 (Arbitration Slot 1, Scan Request Source)
   - CSM2 (Arbitration Slot 2, Background Request Source)
2. Disable, if necessary, the “Repeat” feature in the respective Source Mode Register (GxQMR0, GxASMR, BRSMR), bitfield RPTDIS=b1.

3.5 Programmable Remapping of Analog Inputs – The Alias Feature

The Alias feature allows the user to redirect conversion requests for channels 0 and/or 1 to other channel numbers.

Using the alias feature, a conversion request for channel 0 or channel1 leads to a conversion of the analog input channel “x” instead of channel 0 or channel 1. However, it still takes into account the settings for channel 0 or channel 1.

This allows for many different useful configurations, such as converting the same analog channel (pin) with 2 different configurations (sample time, resolution, result register, and so on).

3.5.1 Using the Alias Feature For Channel 0

To use the Alias feature for Channel 0, select the input channel that should be converted in the ALIAS0 bitfield of the GxALIAS register. When the conversion request is made, the pin connected to the selected channel is converted instead of the pin of channel 0. Channel configuration and result register assignment is the chosen in GxCHCTR0 (Channel Control Registers for channel 0).

3.5.2 Using the Alias Feature For Channel 1

To use the Alias feature for Channel 1, select the input channel that should be converted in the ALIAS1 bitfield of the GxALIAS register. When the conversion request is made, the pin connected to the selected channel is converted instead of the pin of channel 1. Channel configuration and result register assignment is in GxCHCTR1 (Channel Control Registers for channel 1).

3.5.3 Example Using the Alias Feature

In the following example, the Alias feature is used to convert the same input pin with different configurations. Analog input channel 5 is being converted twice, but each time with distinct conversion characteristics.

- The arbiter requests a conversion of channel 1 but, because of the Alias configuration, the analog channel 5 is converted with the channel 1 configuration. The Result is therefore stored in the channel 1 result register, Result Register 1.
- The arbiter requests the conversion of channel 5. The analog input channel 5 is now converted again, but with the channel settings of channel 5 and the result is stored in Result Register 5.

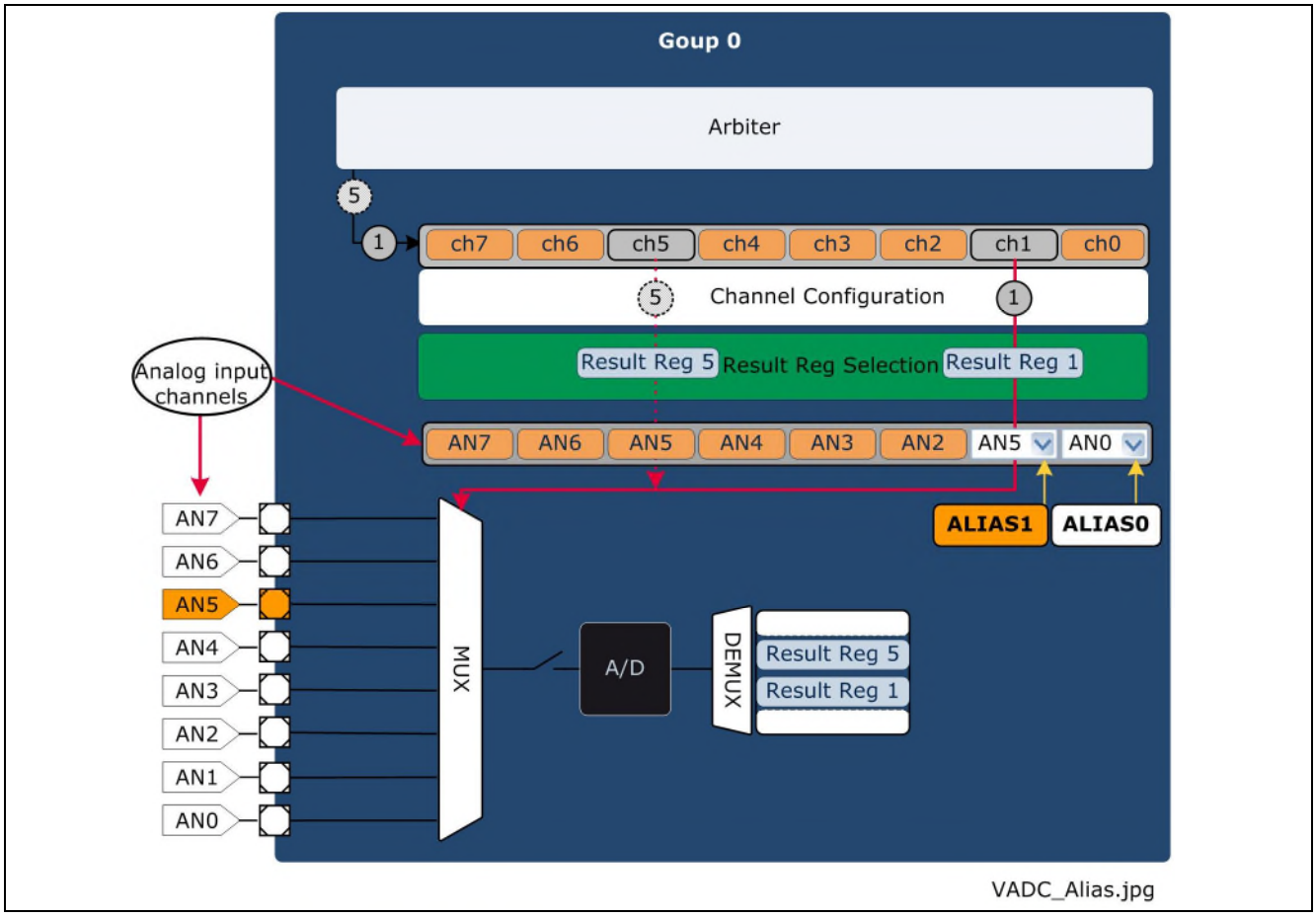

Figure 35 Using the Alias feature

3.6 Safety Features

Several safety features are available to verify the validity of the analog input signals of an application:

- Multiplexer Diagnosis
- Pull-Down Diagnostics
- Converter Diagnostics
- Broken Wire Detection

3.6.1 Diagnostic Features

Many testing mechanisms can be used to test the external and internal channel paths, applying additional loads to the signal path:

- **Multiplexer Diagnosis**
  - Additional pull-up and/or pull-down devices controlled via the port logic can be connected to a channel. Knowing also the external input signal, this makes it possible to test if the multiplexer connects the correct pin to the converter.
- **Pull-Down Diagnostics**

  - A strong pull-down can be activated in a single input channel in order to test the external connection. Channel 7 of every group has Pull-Down diagnosis.

- **Converter Diagnostics**
  - Test signals can be connected instead of or in parallel to the converted input in order to test the converter.

To define these tests, the GLOBTF Register has to be configured as follows:

1. Define the group number for Converter Diagnostics (CDGR)
2. Enable, if required, the Converter Diagnostics (CDEN)
3. Indicate where to connect the pull devices in the Converter Diagnostics (CDSEL):
   - VAREF, VAGND, 1/3rd VAREF or 2/3rd VAREF
4. Disable the Write Control for Converter Diagnostics configuration (CDWC=b1)
5. Enable the Pull-Down Diagnostics (PDD)
6. Disable the Write Control for Converter Diagnostics configuration (MDWC)

3.6.2 Broken Wire Detection

The Broken Wire Detection feature allows the converter's internal capacitor to be pre-charged in advance to the regular Sample Phase to either VAREF or VAGND, instead of the standard VAREF/2.

This feature can be used to detect a broken connection to a sensor. If there is continuous conversion of a sensor voltage and the connection with the sensor is interrupted, the conversion results become the pre-charged value (VAREF or VAGND) instead of the expected sensor value.

If Broken Wire Detection is not used, a conversion where the wire is broken would result in VAREF/2, which is not easily interpretable as a wrong wire connection. Nevertheless, the broken wire feature allows configuration of this value to either VAREF or VAGND according to the sensor characteristics.

This feature can be assigned to any channel independently.

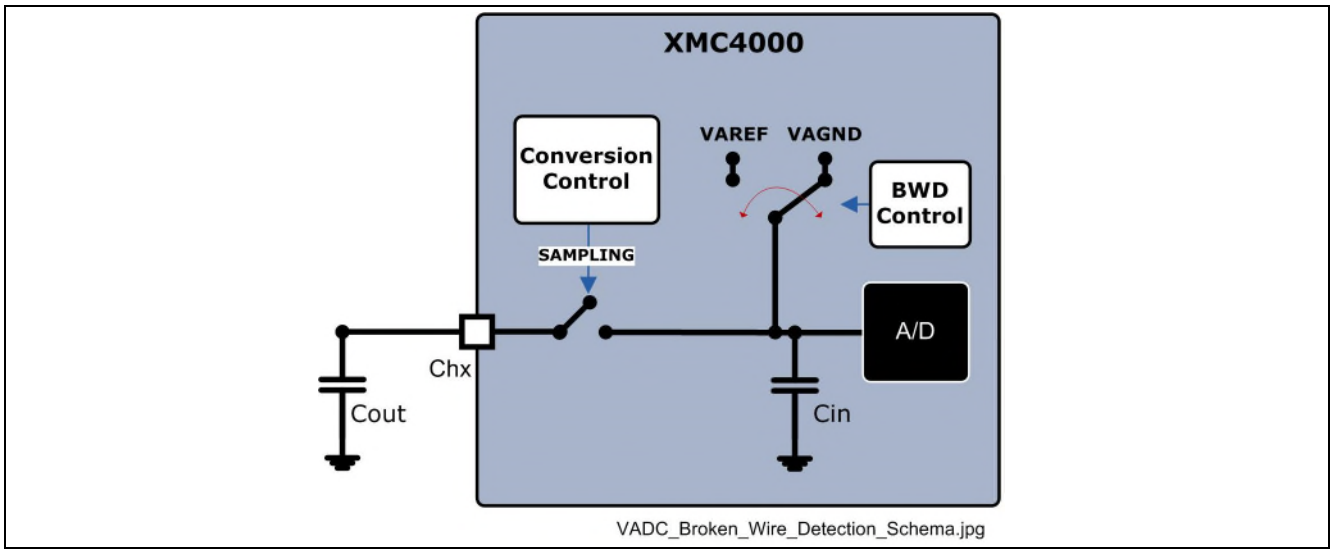

Figure 36 Broken Wire Detection schema

To use the Broken Wire Detection feature:

1. Enable the Broken Wire Detection in the Channel Control Register GxCHCTRy, bitfield BWDEN=b1.
2. Select the pre-charged value; BWDCH=b00 pre-charges to VGAND and BWDCH=b01 to VAREF.

Note:It is important to take into account that adding the Preparation Phase to pre-charge the converter's internal capacitor, increments the time spent in the conversion by exactly the same time as "Sample Time".

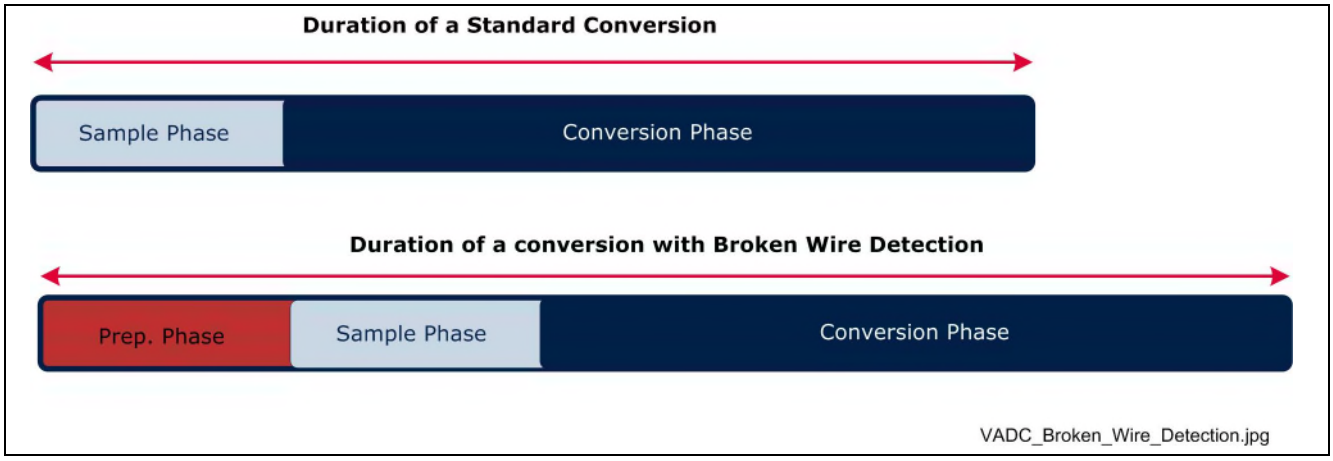

Figure 37 Duration of Standard versus Broken Wire Detection conversion

3.7 Is the ADC really sampling?

Broken Wire Detection is very useful not only for failure detection, but also for debugging purposes or sampling identification.

The converter's internal capacitor is pre-charged to a selected value so there is an easily detectable small voltage glitch every time a sampling phase starts. This is due to the fact that at that precise moment, a switch is closed connecting the internal capacitor and the pin, leading to a charge transfer between both capacitors. This voltage spike can be measured with the help of an oscilloscope, for example.

Depending on the pin voltage, a pre-charge to VAREF or VAGND generates more prominent spikes and is therefore easier to detect.

If the pin voltage is lower than VAREF/2, configure the pre-charge value to VAREF. If the pin voltage is higher than VAREF/2, set up the broken wire detection to pre-charge to VAGND. In this way the maximum voltage difference is ensured

The following figure shows an oscilloscope capture. Here, four channels are being converted and the result of every conversion is presented. A small voltage glitch appears in every signal, showing when each conversion starts.

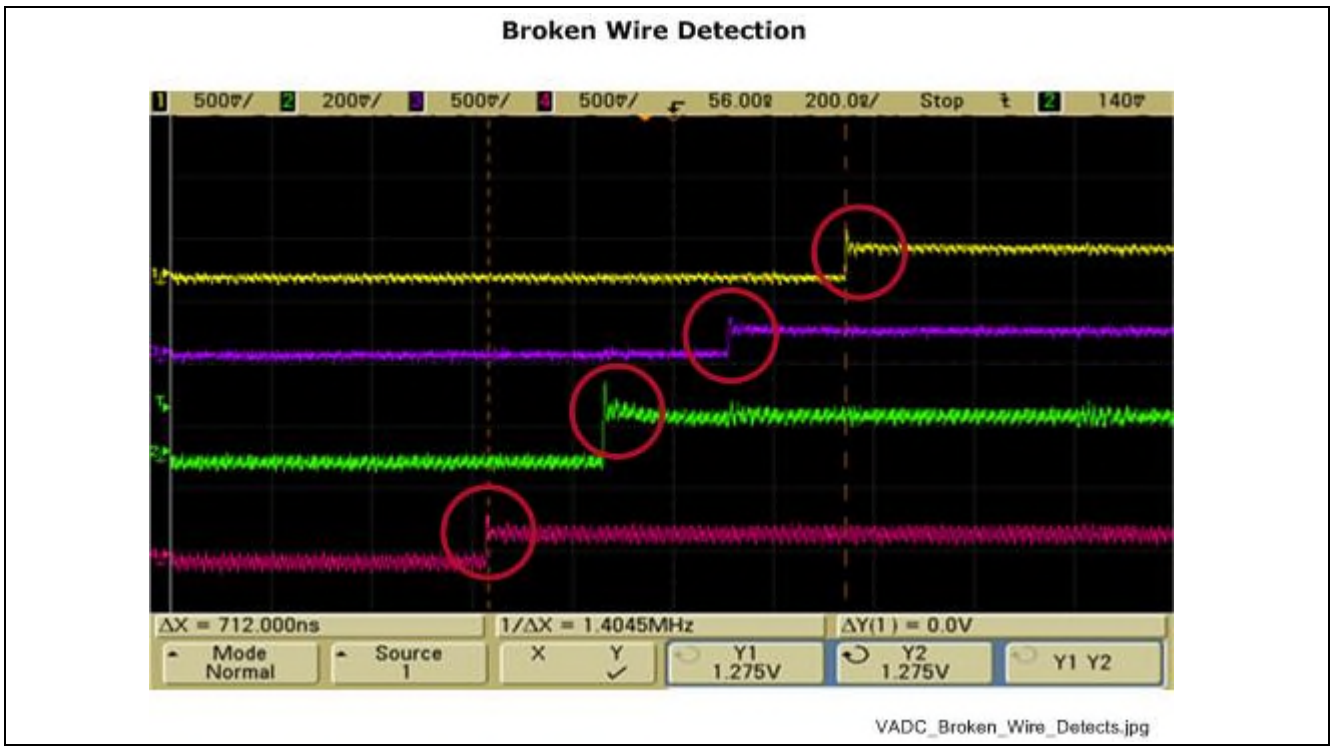

Figure 38 Broken Wire Detection Oscilloscope Capture

3.8 Increasing the Number of Analog Inputs: External Multiplexer Control

In order to increase the number of available input channels, external multiplexers can be connected to each channel. The VADC in the XMC4000 devices can be configured to control these multiplexers automatically by generating the respective control signals. These control signals are generated in standard binary or Gray code, by one of two available interfaces (Interface 0 and Interface 1). Using Gray code avoids intermediate multiplexer switching when selecting a sequence of channels, because only one bit changes at a time.

The VADC supports 1:8 multiplexers with the following modes:

- Sequence mode: Converts all external channels configured when the channel is encountered.
- Single-step mode: Converts just one different external channel each time the selected channel is encountered.
- Steady mode: Converts the configured external channel when the selected channel is encountered.

3.8.1 Steps to configure external multiplexer control

1. Define the VADC Group associated to Interface 0 in the bitfield EMUXGRP0 of the External Multiplexer Register (EMUXSEL), and/or the VADC Group associated to Interface 1 in the bitfield EMUXGRP1 of the EMUXSEL the register.
2. Configure the respective Request Source and the channel in which the multiplexer is connected for the Group chosen in the previous step.
3. Configure in the External Multiplexer Control Register (GxEMUXCTR):

- the initial setting for the external multiplexer in the EMUXSET bitfield:
  - the channel to which the external multiplexer control is applied in the EMUXCH bitfield

  - the external multiplexer mode between Sequence, Single-step and Steady in the EMUXMODE bitfield
  - the control signal coding scheme (Gray or binary) in the EMXCOD bitfield
  - use STCE whenever the setting changes (EMXT=b0) or use STCE for each conversion of an external channel (EMXST=b1)
  - the write control for EMUX configuration (EMXWC=b1)
- resolution (CME) - select one of the available conversion modes:
  - Standard conversion with 12, 10 or 8 bits of resolution
  - 1 bit resolution in Fast Compare mode
- sample time (STCE) - indicate the additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles.

4. Configure the resolution and sample time for the external multiplexer conversions in one of the classes either group specific (GxICLASS0 and GxICLASS1 registers) or global classes (GLOBICLASS0 and GLOBICLASS1 registers).
5. Assign the channel to the selected Input Class in the ICLSEL bitfield of the respective Channel Control Register (GxCHCTRy).

Note:XMC4400, 4200 and 4100 devices have an extra bitfield in the External Multiplexer Control Register (GxEMUXCTR): EMUXCSS. This bitfield allows for a choice of whether the EMUXCH bitfield selects an arbitrary channel (EMUXCSS=b0) or if each bit of bitfield EMUXCH selects the associated channel for EMUX control (EMUXCSS=b1).

For debugging, the following bitfields could be useful in applications using external multiplexers:

- EMUX bitfield: Indicates the setting of the external multiplexer corresponding to the value in bitfield RESULT, available in GxRES0 registers.
- EMUX ACT bitfield: Indicates the current value for the external multiplexer selection, available in the EMUXCTR register.

3.9 Tips, Tricks and Pitfalls

This section provides some hints and insights into more advanced features of the VADC.

3.9.1 VAREF Series Resistor

In some cases, it is desirable to introduce a series resistance to the VAREF pin. This is done to stabilize the reference. However, if the resistance value is too big, the voltage drop across it will increase, for example while executing a conversion or calibrating the internal capacitors, and therefore the VAREF value seen by the microcontroller will be lower than the initially expected value, leading to less accurate conversions.

3.9.2 Ratiometric Configuration

In the majority of measurement systems, the reference and the input analog signal are the principal factors that establish the accuracy. Any change in the supply voltage of the analog source results in a change at the analog input voltage seen by the A/D converter. In order to improve the accuracy of the system, it is desirable to use a Ratiometric Configuration of the connection for measurements.

In ratiometric measurements there is a relation between the voltage of the analog source and the reference voltage at pin VAREF. The same voltage reference source is used for the analog source excitation and the reference voltage input VAREF. So changes do not cause measurement errors. Hence, the A/D converter conversion result is independent to variations in the analog source excitation or variations in the reference voltage input VAREF. Because of that it is not necessary to have a stable voltage reference to achieve an accurate measurement result.

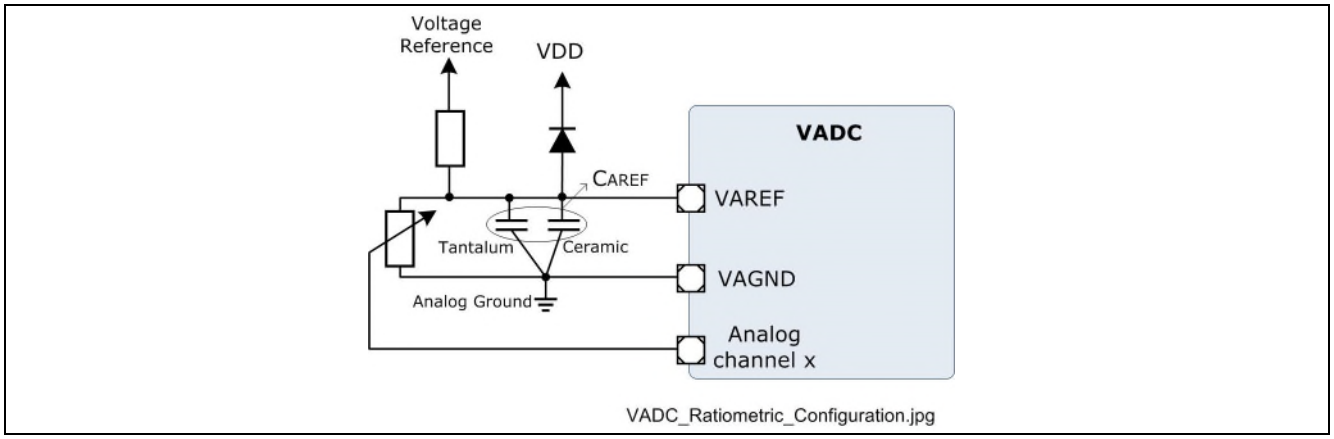

Figure 39 Ratiometric Configuration

3.9.3 Maximum Clock Frequency

A DIVA value of 0 is not forbidden. This provides an analog clock frequency of 60 MHz This value is 'out of specification' according to the data sheets. However conversions will be executed as usual. In some cases, where the sensor current supply to the internal ADC capacitor is not high enough, or under special impedance values, this 'out of specification frequency' might lead to incorrect shifted result values given that the internal capacitors have no time to recover from the charge transfer. It is likely that the TUE figure will degrade.

4 Trigger Options

4.1 Introduction

In every application, numerous trigger signals are needed in order to control when conversions start. In the XMC4000 VADC, up to 16 trigger events (pulse signals) originating in other XMC4000 peripherals, can be connected to the different Request Sources. On a trigger event, a pre-configured sequence is started leading to the conversion of a certain analog input or a sequence of inputs. This ensures perfect real-time control as no CPU intervention is required.

Trigger events can be generated in two different ways:

- Software:
  - The trigger pulse event is generated via user code
- Hardware:
  - The trigger pulse is generated directly by a signal from a peripheral
  - The trigger pulse is generated indirectly, after being processed by the ERU (Event Request Unit)

The two types of triggers are shown in the following figure.

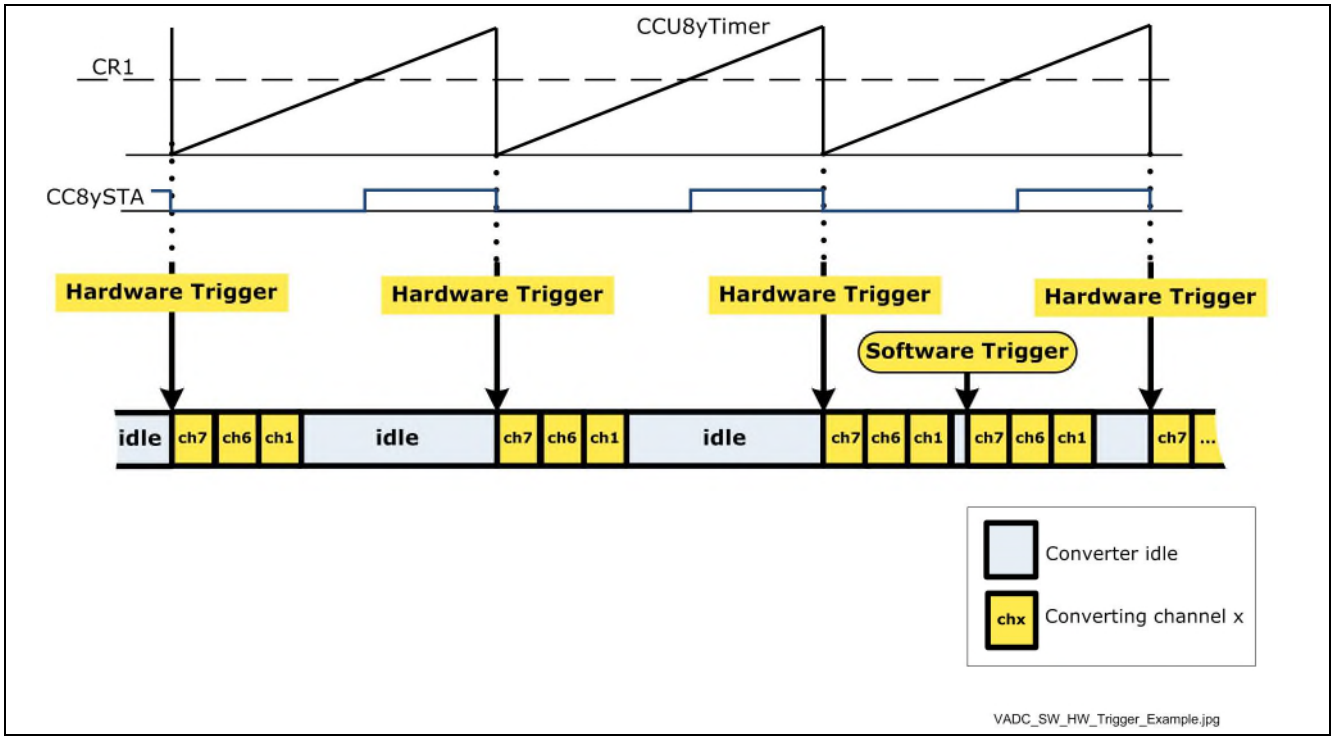

Figure 40 Hardware and Software Triggers

A timer (CCU8y in this instance) generates a hardware trigger on its period match, so the sequence of conversion programmed in the Scan Request Source (channel 7, channel 6 and channel 1) is synchronously repeated each time the trigger is detected. Additionally, another trigger event is generated by software (after the third hardware trigger in the figure) causing the repetition of the same sequence asynchronously.

Very flexible hardware logic allows pre-configuration of how the trigger event requests the conversion of channels. Gating logic permits the use of up to 16 additional signals (level signals from other peripherals in the microcontroller) to gate or hinder the conversion of channels during a period of time. Those level signals can be input as triggers, increasing the total number of possible VADC trigger inputs to 32.

Burst sampling is easily programmed by using the information in the signal GxSAMPLE, which contains the sampling instant of one channel.

4.2 Software Trigger

A software trigger is generated by user code and is typically asynchronous to the application. Depending on the request source, there are different ways to generate the software trigger (load event in Scan Request Sources):

- Queue Source:
  - By setting the TREV bitfield in the GxQMR0 register to 1. A valid queue entry is required (GxQINR0 register programmed at least once properly), so that a conversion is requested.
- Scan Request Source:
  - By setting the LDEV bitfield in the GxASMR register to 1.
- Background Request Source:
  - By setting the LDEV bitfield in the BRSMR register to 1.

4.3 Hardware Trigger

A hardware triggering scheme ensures real-time conditions and synchronicity with the application. A hardware trigger can be generated in a peripheral (direct trigger) or can be generated in a pin and be indirectly routed to the ADC through the ERU peripheral (indirect trigger).

4.3.1 From a Peripheral

A hardware trigger can come from different peripherals. The trigger sources available for XMC4500 devices are shown in the next figure.

Note:Please refer to the specific device Reference Manual for the relevant connectivity information.

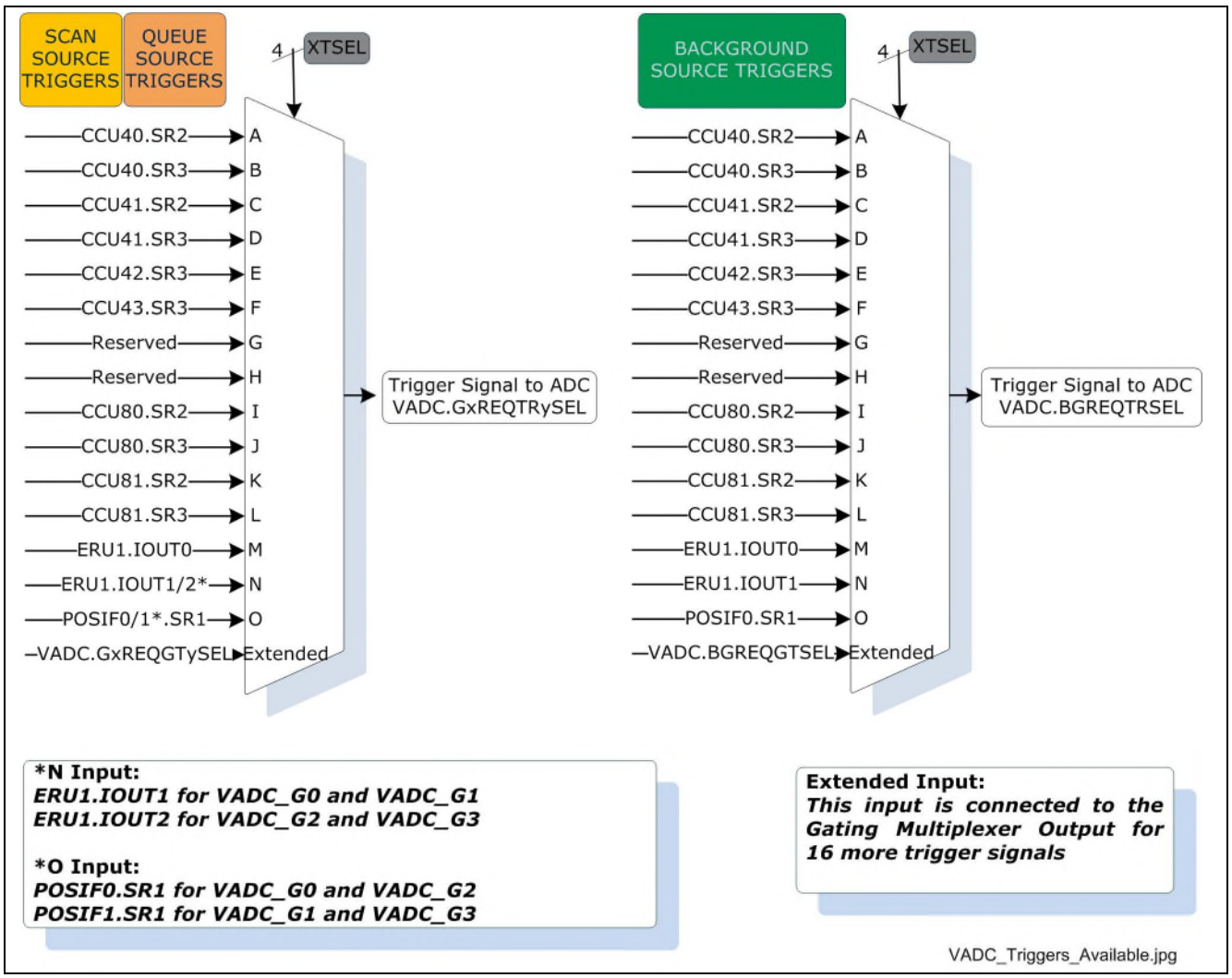

Figure 41 Hardware Triggers available for each Request Source in XMC4500 devices

The two figures which follow show schematic views of the trigger and gating logic for Scan and Queue Request Sources respectively.

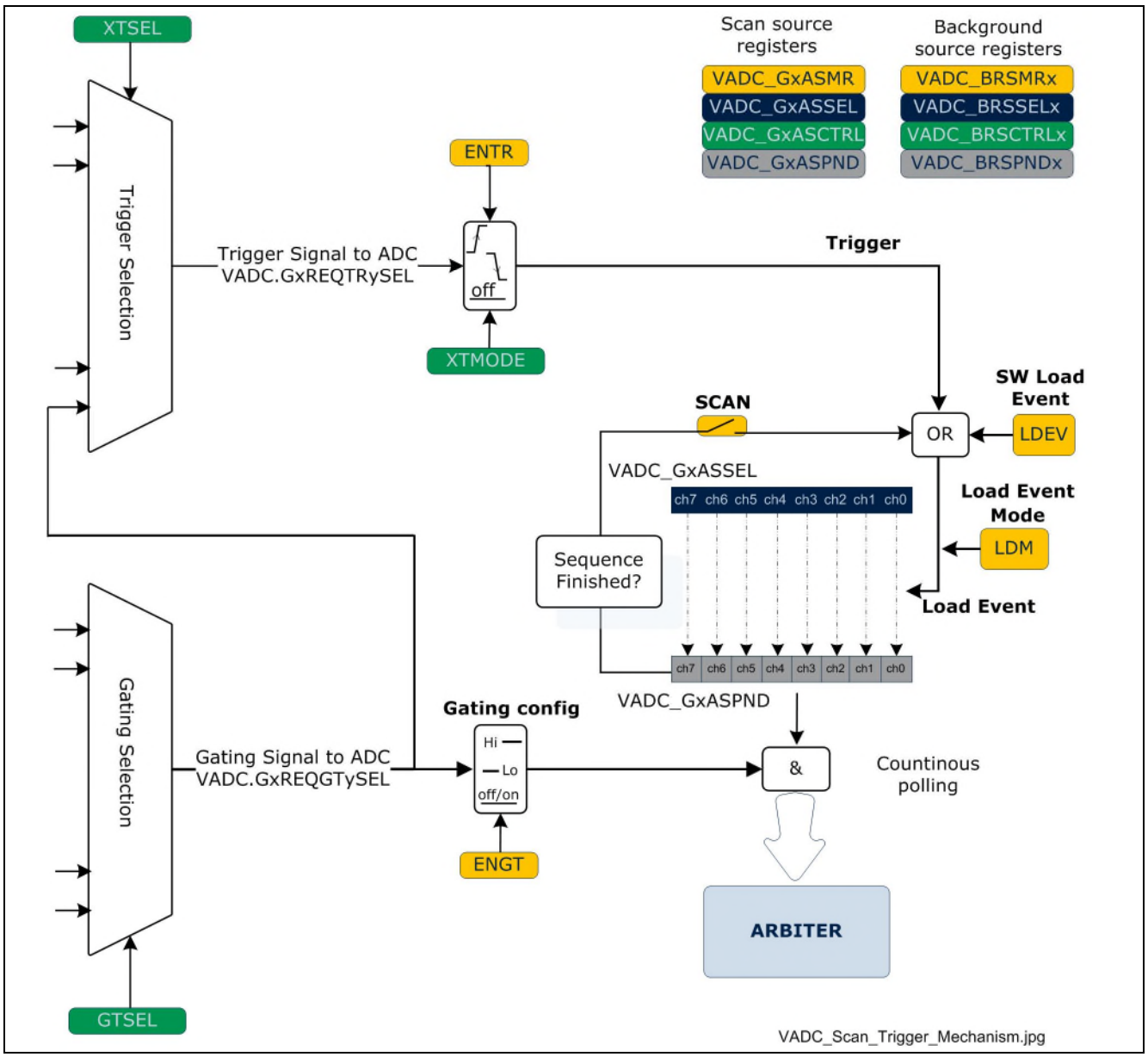

Figure 42 Scan Request Source trigger mechanism

Trigger Options

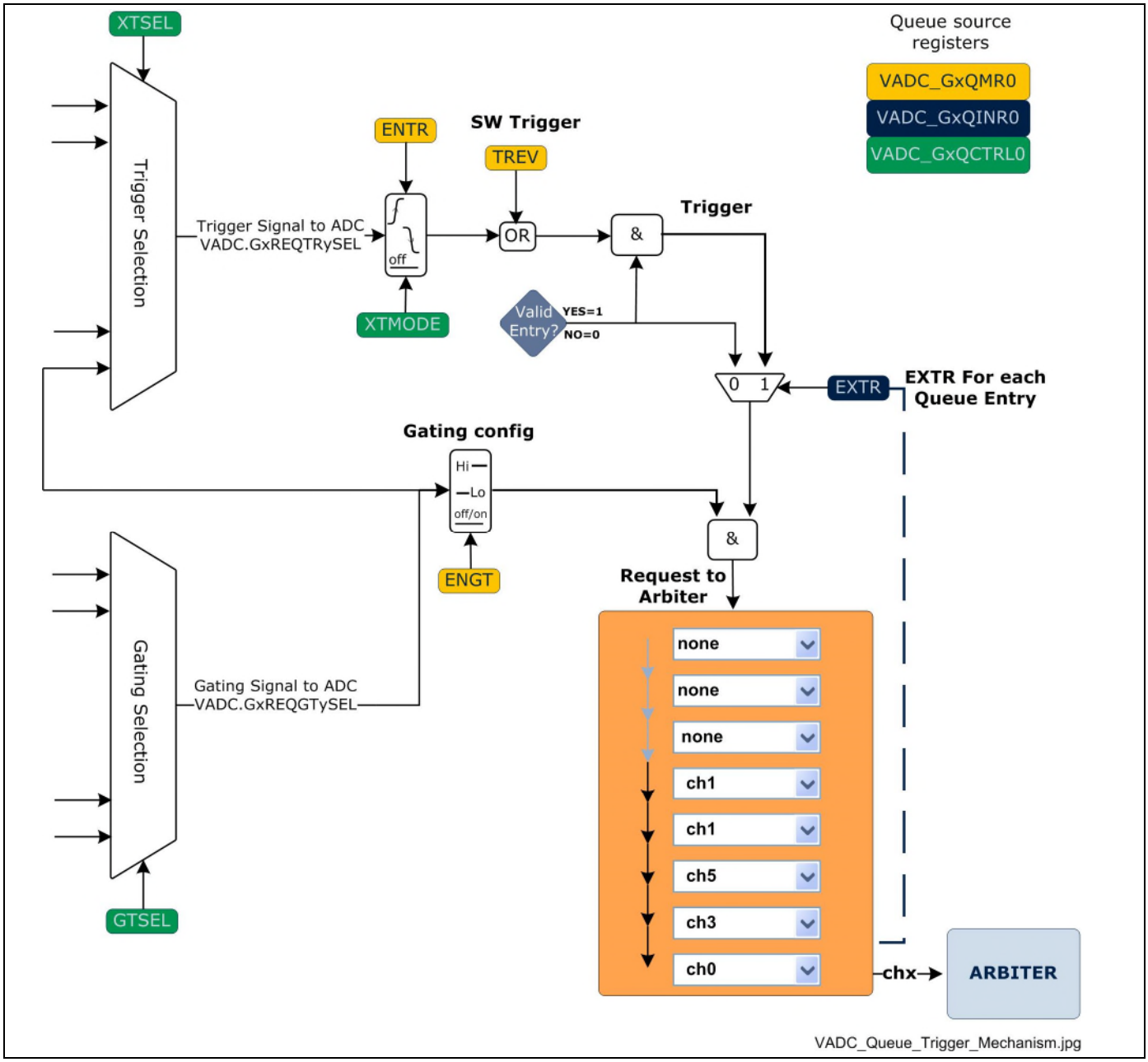

Figure 43 Queue Source trigger mechanism

As the previous two figures show, the trigger mechanism is different for the different request sources. The Scan Request Source generates a load event, while the queue source requests the next channel in the queue configured as an external trigger.

The following steps are required to program the trigger configuration.

Note:Refer to the specific device Reference Manual for the available connectivity.

Trigger for Scan Request Sources (Scan and Background)

1. Configure the respective Request Source Mode Register (BRSMR, GxASMR).

Trigger Options

- Enable the External Trigger (ENTR=b1) and set the gating for enabling conversions when a pending request is available (ENGT = b01).
- Configure the Autoscan source load event mode (LDM) to define the way the SEL channels (selected channels) are copied into the pending (PND) channels. If autoscan (SCAN=b1) is configured, the load event automatically happens after all channels have been requested.

2. In the respective Request Source Control Register (GxASCTRL and BRSCTRL):
   - Enable the Write Control for Trigger Configuration (XTWC=b1)
   - Configure the External Trigger Operating Mode (XTMODE)
   - Select the External Trigger Input (XTSEL).

Trigger for Queue Source

1. Configure the Queue Request Source Mode Register (GxQMR0):
   - Enable the External Trigger (ENTR=1)
   - Set the gating for enabling conversions (ENGT != b0)
2. In the Queue 0 Source Control Register (GxQCTRL):
   - Enable the Write Control for Trigger Configuration (XTWC=b1)
   - Configure the External Trigger Operating Mode (XTMODE)
   - Select the External Trigger Input (XTSEL)
3. Select the channel to convert and enable the External trigger (ENTR=b1) for each required entry in the Queue in the GxQINR0 Register.

4.3.2 Trigger from a Pin Signal (through ERU)

A signal from an analog pin can be used to trigger ADC conversions. In order to achieve this, the pin signal has to be directed to the **ERU (Event Request Unit)** where any 'FPGA-type' logic operations can be implemented. The resulting output signal must then be redirected to the VADC by selecting the appropriate trigger source.

The steps to use a pin signal as a trigger source are as follows:

1. Configure the ERU according to the specific logic operation:
   - Choose a pin from one of the PORTS inputs for the ERU in the EXISEL Register.
   - Enable the Output Trigger Pulse, Rising or Falling Edge Detection, the Gating Selection for Pattern Detection Result and the combination desired to be applied to the input signal in the EXICON Register
2. Configure the VADC, choosing in the XTSEL bitfield the output signal configured in the previous step from the two External Trigger signals coming from the ERU (ERU1.OUT1 or ERU1.IOUT0).

In the following example figure, two general purpose pins are ORed in the ERU module. The resulting output is routed to the ADC as a trigger signal.

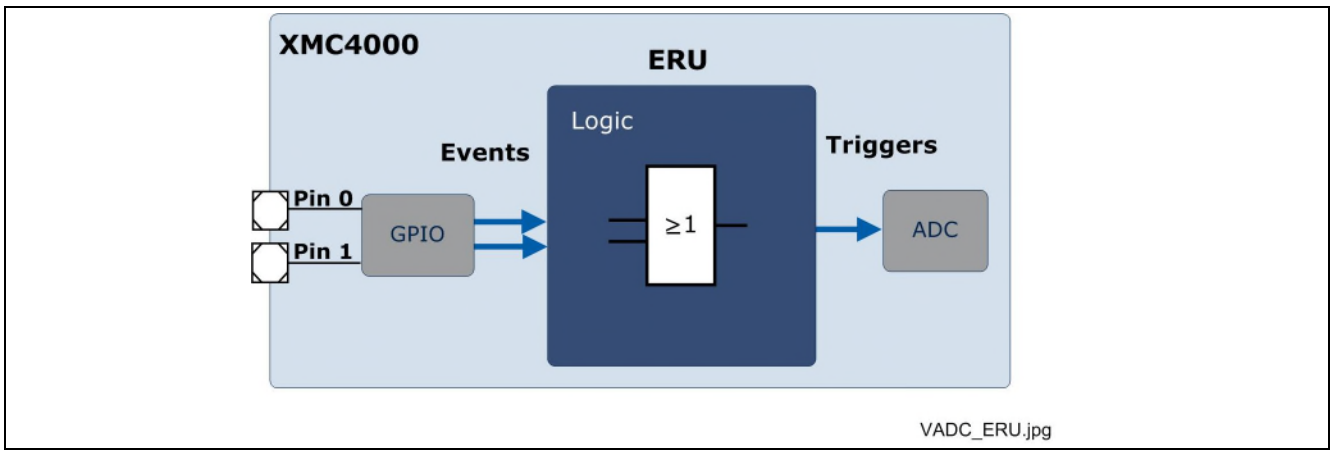

Figure 44 Triggering the VADC with the result of an OR operation of two pins in the ERU

4.3.3 Gate Signals as a Trigger

Usually, trigger signals are pulses whereas gate signals are level signals. In the XMC4000 devices, the VADC allows the routing of all gating signals as trigger inputs in order to double the number of available input trigger signals. This is achieved by connecting the output of the gating multiplexer to the input of the trigger multiplexer (figure below).

In order to use a gate signal as trigger, it is necessary to program the gating mechanism by setting ENGT to 1 (enable always) and to choose the GxREQGTySEL signal as XTSEL entry (0xFF). After this, the trigger signal can be selected from the gating multiplexer with the GTSEL bitfield.

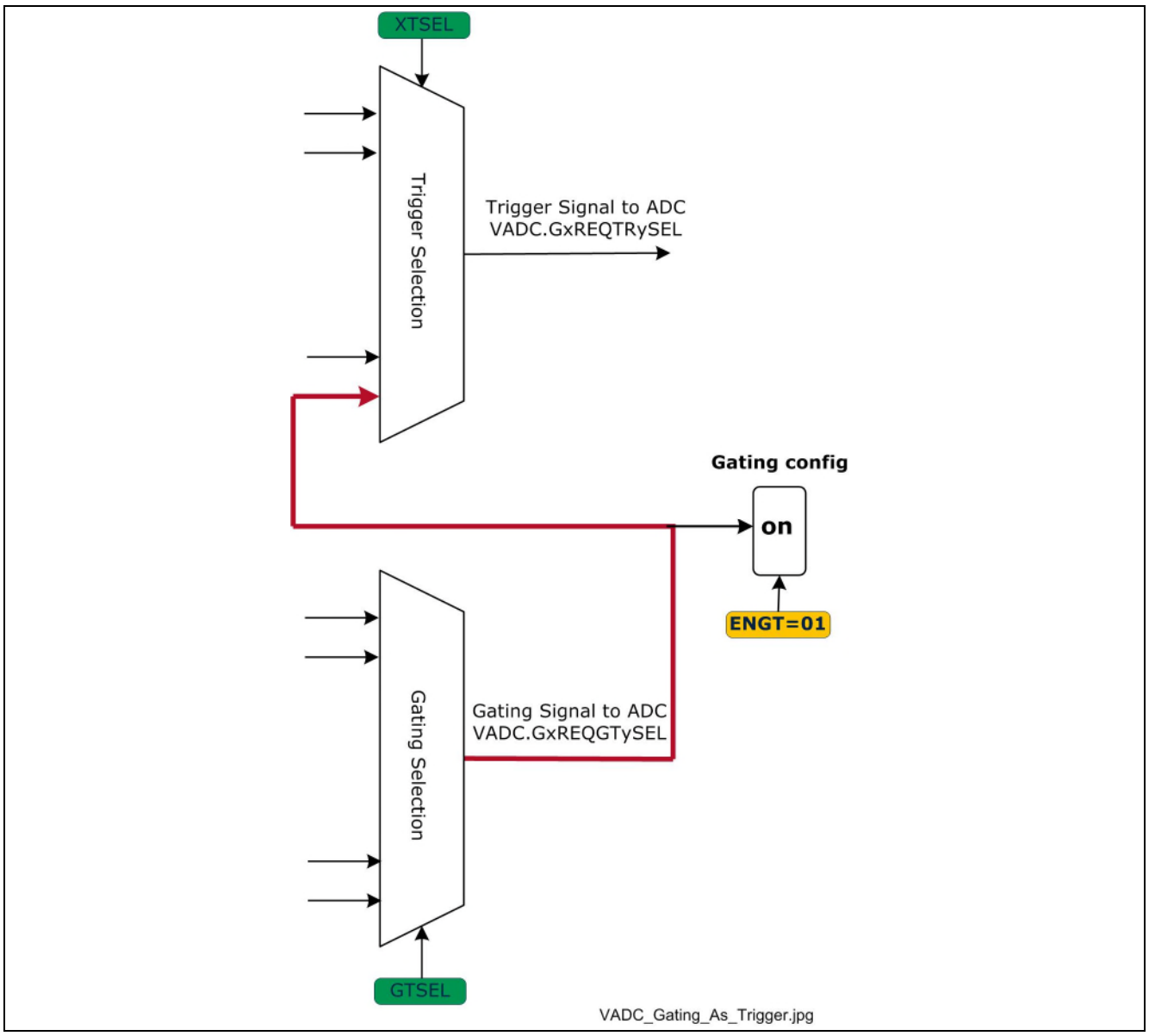

Figure 45 Gate signal as Trigger

4.4 Gating the Trigger

Signals from peripherals can be used to gate trigger events that are hindering the trigger during the low or high value of that extra signal. This provides an additional flexibility level.

The following figure gives an example of gated triggers. A timer in the CCU8 unit creates the desired PWM pattern and the resulting current waveform (I_DC_Link) presents some ringing during the switching of the PWM pattern.

If the VADC is configured to convert this signal continuously, a CCU4 timer can be used to generate a level signal (CC4yST) to gate the continuous conversions of the current waveform. The VADC does not convert a channel during the ringing when the current value is not providing relevant information.

The start of the CCU4 timer can be easily synchronized to the CCU8 timer events, such as CompareRegister1 match and/or CompareRegister2 match, to ensure full, real-time compliance.

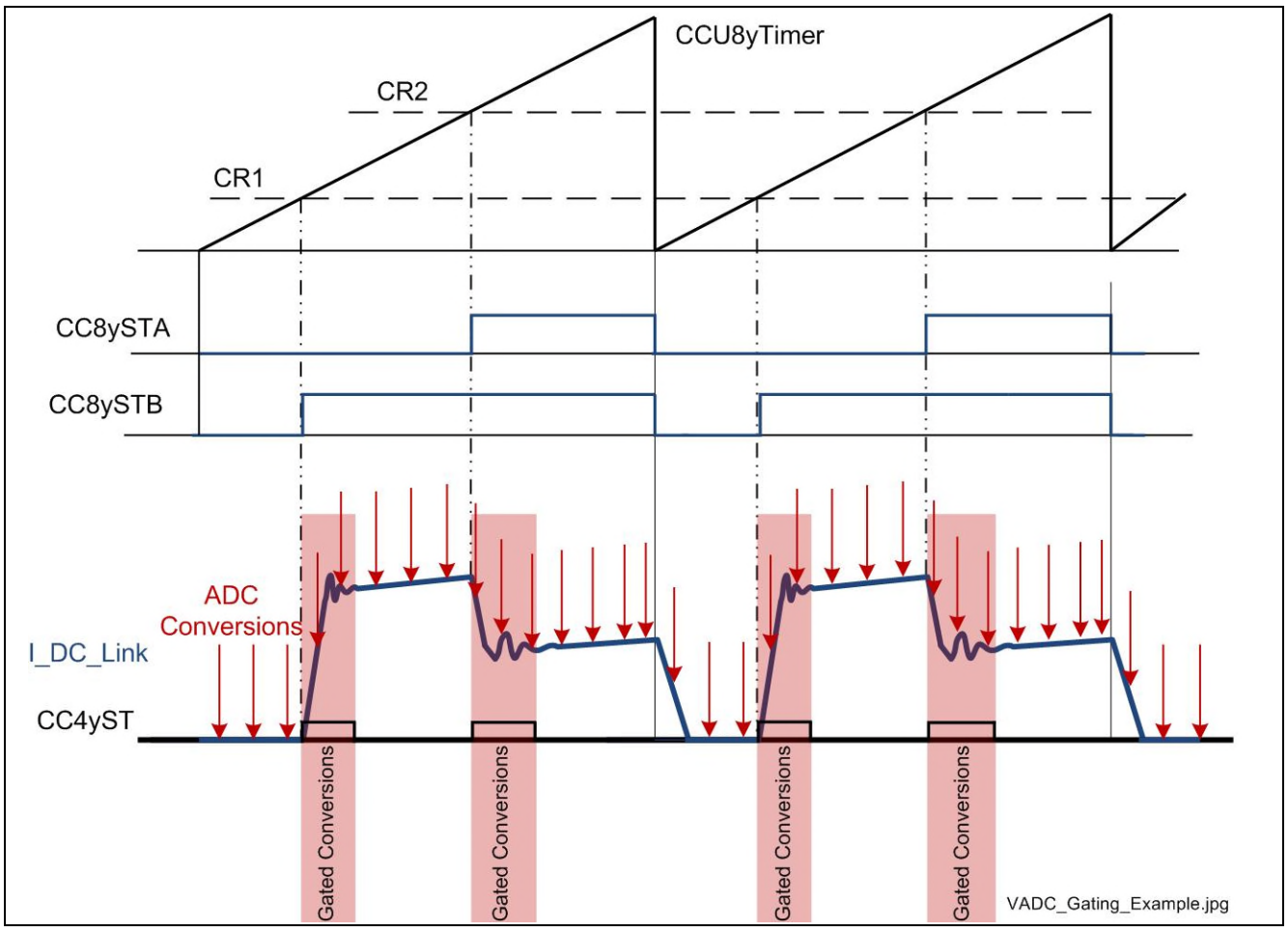

Figure 46 Example of gating a continuous conversion of a current waveform

A timer in CCU8 generates the PWM pattern while a timer in CCU4 generates a gating signal on CCU8 events.

Gating configuration steps

Note:Continuous trigger configuration is not considered here.

1. Configure the respective Request Source Mode Register (BRSMR, GxASMR, GxQMR0). Select the gating mode with bitfield ENGT. This selects whether gating is to be active always, never, or when the input gate signal REQGTx is high or low.
2. In the respective Request Source Control Register (GxASCTRL and BRSCTRL):
   - Enable the Write Control for Gating Configuration (GTWC=b1)
   - Select the External Gate Input (GTSEL) from the inputs

The available gating signals sources for the XMC4500 devices are shown below.

Note:Please refer to the specific device Reference Manual for connectivity information.

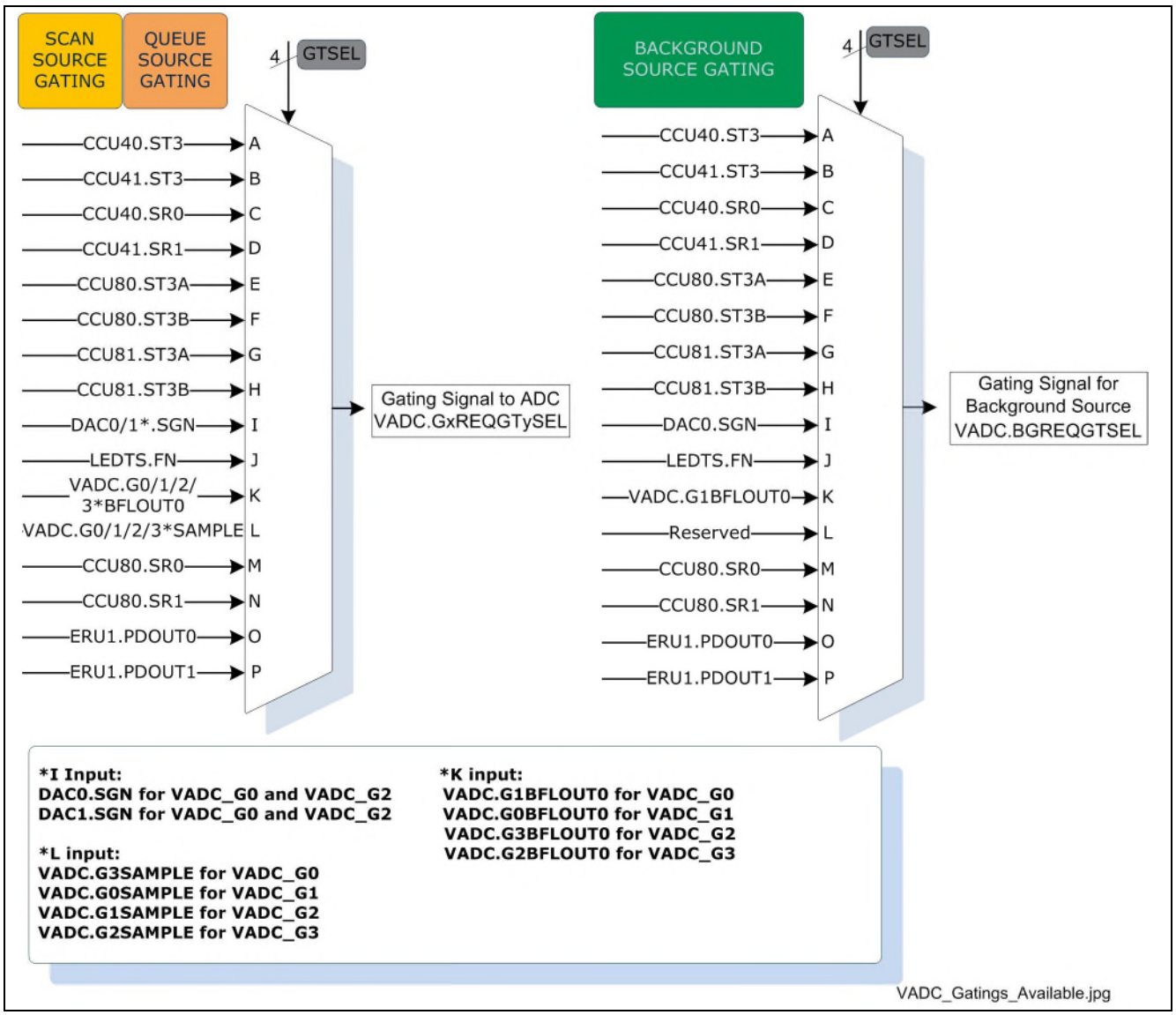

Figure 47 Available Gating signals for each request source

4.5 Burst Sampling

Some analog signals change very quickly and the dynamic behavior has to be analyzed. In the XMC4000 VADC, each time that the sampling of a group has finished, a pulse signal (GySAMPLE) is generated. This signal can be used to internally trigger a different ADC group, creating a new sample and therefore a new GzSAMPLE signal in this second group.

Using this mechanism it is easy to build up a burst sampling scheme. Burst sampling of one signal in group chaining achieves the maximum sampling rate (one signal can be sampled in about 200 ns).

The Sample signal is a gating signal, so in order to use it as a trigger the Gate Signals as a Trigger procedure must be followed.

In order to activate the sample signal generation, follow these steps:

Trigger Options

1. Enable the Write control for TGB and TGS in the TG_P bitfield of the OCS Register, and enable the generation of the GySAMPLE signal in each chosen group. TGS=b01 in OCS Register. Select the OTGB0 Trigger Bus, TGB=b0 in the OCS Register.
2. Set the Sample signal as a trigger as explained in Gate Signals as a Trigger.

As an example use case, 4 groups can be chained in the following way. After sampling channel x in each group the signal GySAMPLE (“y” being the group number) is automatically triggering the next group conversions. When group 3 has finished sampling, its GySAMPLE signal is sent back to group 0, closing the loop. This configuration is automatically converting all 4 channels continuously with no intervention needed.

The XMC4000 VADC allows internally the connection of 2 analog pins of 2 different groups (please refer to the specific device reference manual for further details). Using 2 of these connections and connecting 2 pins physically outside the microcontroller, one can measure the same voltage with a sampling rate of 4 times the ADC sampling rate; i.e. 8MSamples/sec.

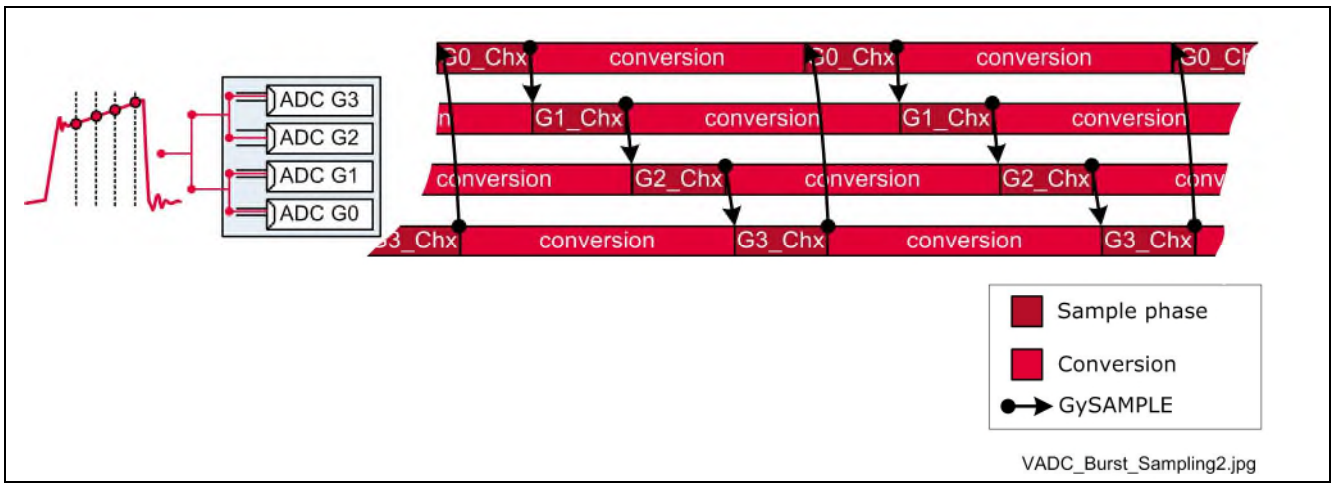

Figure 48 Burst sampling

The next figure shows another use case for bust sampling. Again a burst sampling of four groups is made, but in this instance G3SAMPLE is not enabled and group 0 has configured an external trigger coming from CCU8. Now there are four different analog pins connected, one per group. The burst of four groups is started each time the trigger event is received. With this set up, time triggered burst sequences are easily created.

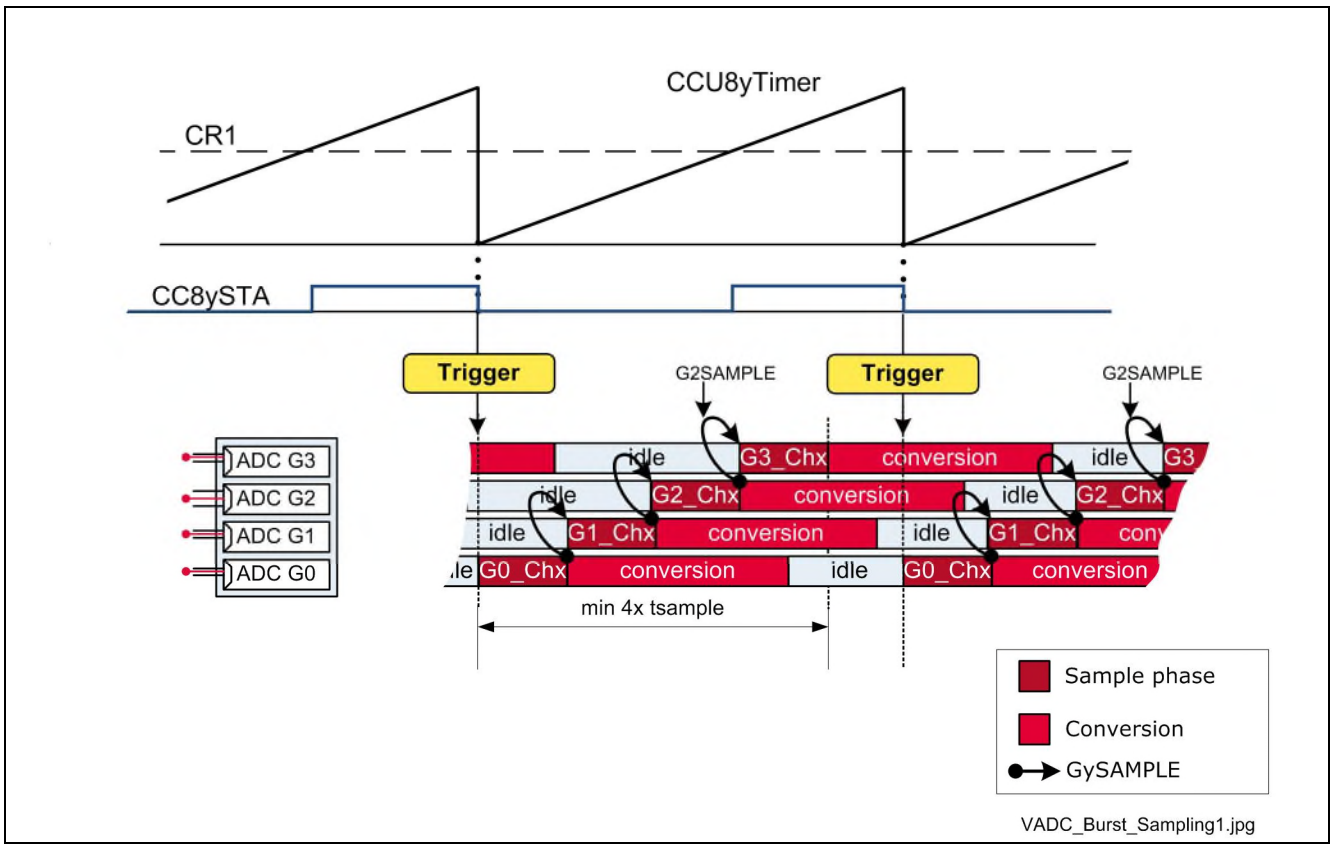

Figure 49 Burst Sampling and Hardware Trigger

The next figure demonstrates an additional use case, where a synchronization and burst sampling mechanism enable a complex sequencing.

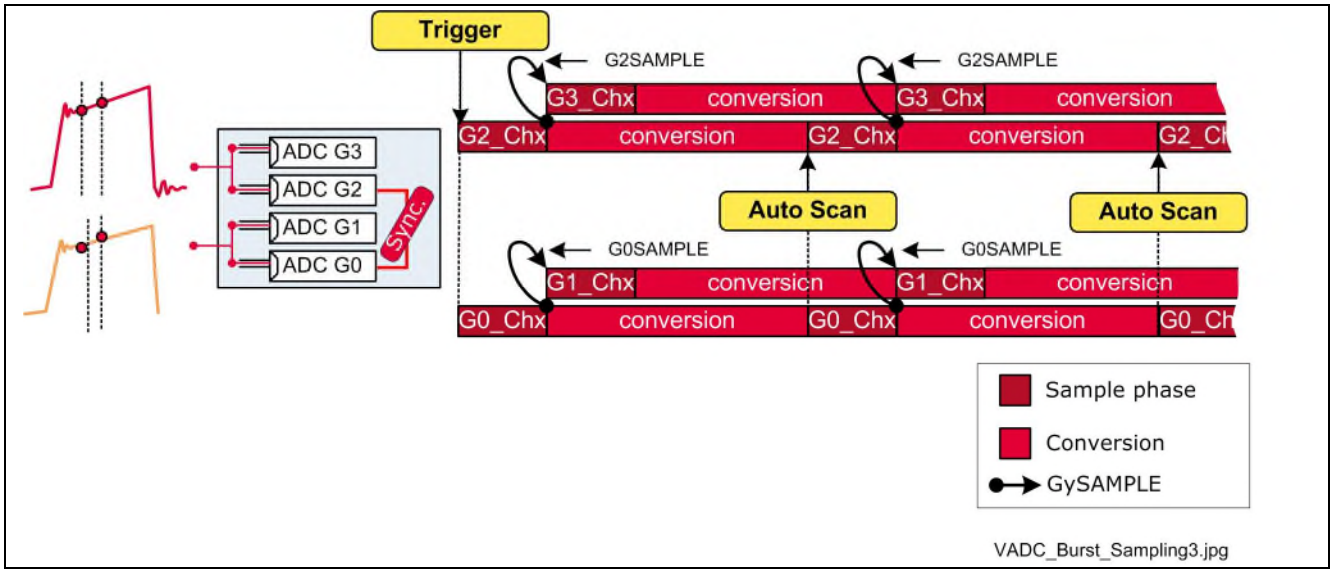

Figure 50 Burst sampling and Synchronization

Group 2 is configured as master and group 0 as its slave in the synchronization mechanism, so every time group 2 starts sampling, group 0 also starts sampling. When a sampling of group 2 finishes, it generates the G2SAMPLE signal to trigger the group 3 sampling. When a sampling of group 0 finishes, it generates the

Trigger Options

G0SAMPLE signal to trigger the group 1 sampling. Group 2 is configured as Autoscan, so the sampling of group 2 will start again when the previous conversion finishes.

It is important to notice that group 2 and group 0 will start their sampling synchronously, but the start of group 3 and group 1 will depend on the group 2 and group 0 sampling time respectively.

5 Result Handling Use Cases

5.1 Introduction

Some applications need a very high sampling rate. This usually requires a fast memory transfer from the result registers to the memory buffer, which may lead to tight real-time conditions for software or even for the memory bus if a DMA transfer is used. To ease this situation, the XMC4000 VADC has a special result handling feature.

In each VADC group the eight channels can be assigned individually to 17 result registers.

Note:For a better understanding of how to assign results to each Result Register, see 3.2.3 Selecting the Result Register in the ADC Converter chapter.

The format and alignment for each result register is configurable. This saves software handling for the application.

Usually, sensor measurements and results processing are not synchronous. For these cases a FIFO structure is helpful to allow the A/D conversions to continue. Then, as soon as processing is required, the data can be easily read out. Up to 16 result registers can be linked together.

Another option is to assign one result register to several channels and use the wait-for-read mode. This suspends the start of conversion for any channel which is assigned to this result register until the result has been read. This is especially helpful for conversion sequences with a mixture of time non-critical channels triggered by the Scan Request Source, and time-critical channels triggered by the Queue Request. A DMA channel can be used to move a result to a RAM location as soon as the result is available.

These options avoid possible data loss caused by overwriting a result register before it is read.

Sometimes oversampling of a signal is necessary in order to increase the resolution or to get better accuracy by averaging. The result handling of the VADC group offers a data reduction feature which can be used independently on different channels.

Finite and infinite impulse response filters, and a difference mode are also available. This allows signals to be processed by Hardware, so saving CPU load.

Result Handling Features

Not every result register can implement the handling features described. Table 1 details which register can execute these features.

Table 1 Result Handling Features

Result Handling Feature	Result Register
FIR and IIR Filters	7 and 15
Subtrahend in the Difference Mode	0
Minuend in the Difference Mode	1 to 15
Data Reduction	0 to 15
FIFO	0 to 15

The following figure shows some of the result handling features available in the XMC4000 VADC. This example uses data reduction achieved by a four-time accumulation on channel 7, a FIR filter on channel 2, and an IIR filter on channel 3. Offset-compensation is made by subtracting channel 1 from channel 0 and a result FIFO is built for channel 6.

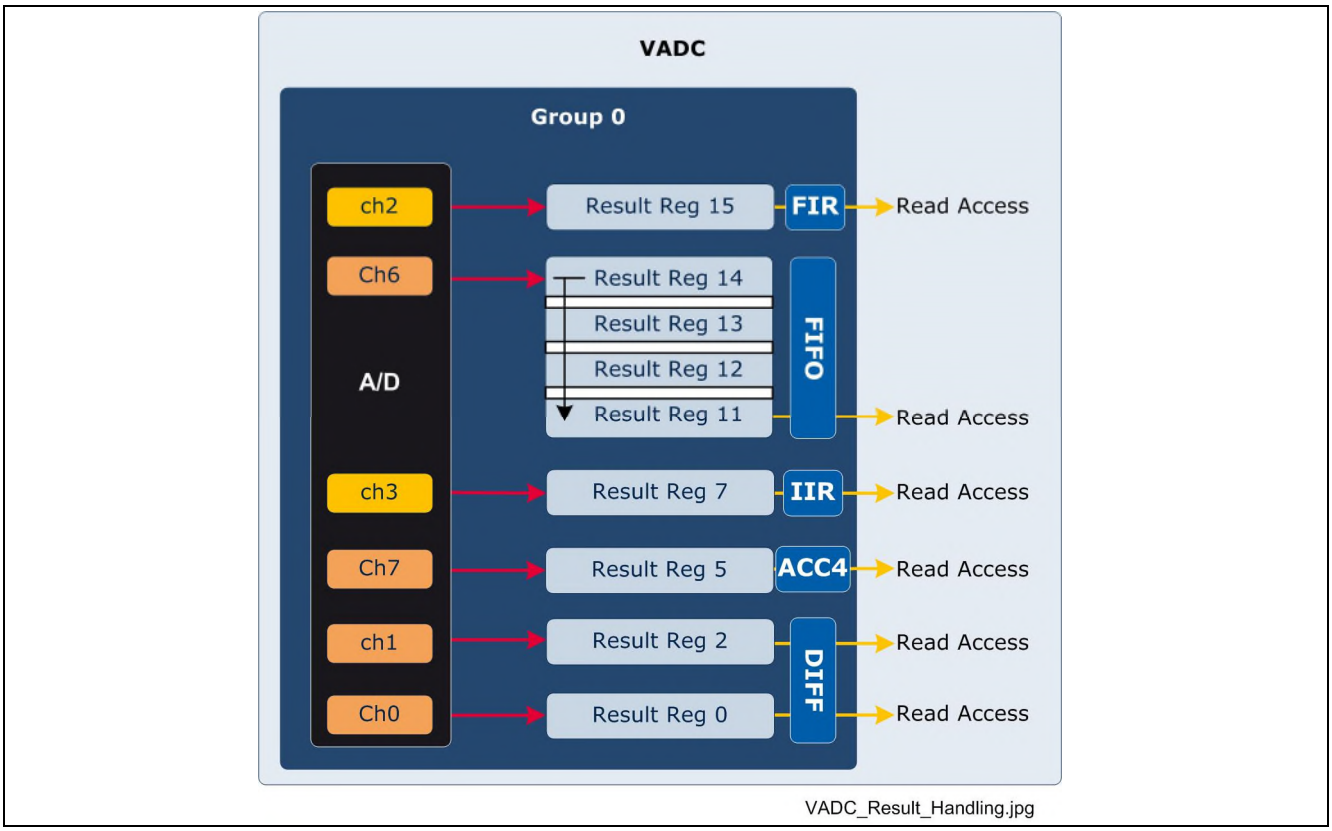

Figure 51 Result Handling Methods

5.2 Results Handling for Safe Read

To relax the response time required to transfer results from the result registers to memory, the XMC4000 VADC can implement several solutions:

- Result buffering using a FIFO mechanism to assign one result register to several channels
- Wait-for-read mode
- GPDMA

5.2.1 Result Register FIFO

The VADC of the XMC4000 device allows a result FIFO to be built by queuing up to 16 result registers. The FIFO is filled from the highest FIFO element and copied to the lowest free FIFO element. This helps to relax strict real-time conditions for software or the memory bus in high sampling rate applications, such as continuous sampling or burst sampling.

The FIFO should be read only from the lowest FIFO element. If the FIFO contains more than one valid result, the content is automatically copied down after the lowest FIFO element is read. The read process can be done by interrupts. For example the result event of the lowest FIFO element is enabled and used to read out

the lowest FIFO element once. In this case the FIFO is not reducing the number of interrupts but relaxing the strict real-time conditions.

Note:Only FIFO elements with a valid flag contain a valid result. The automatic copy process clears the valid flag but not the content of the copied result register.

To configure a FIFO queue:

1. Select in the RESREG bit field of the chosen Channel Control Register (GxCHCTRy), the Result Register input stage of the FIFO queue.
2. Enable the FIFO feature (FEN=b01) in the Result Control Register (GxRCRy) for the intermediate and output stages of the FIFO.
3. Enable interrupts, if required, and read results by the application in the output stage.

The next figure shows a possible scenario: Four high sampling rate conversions of channel 6 which have been buffered by a result FIFO.

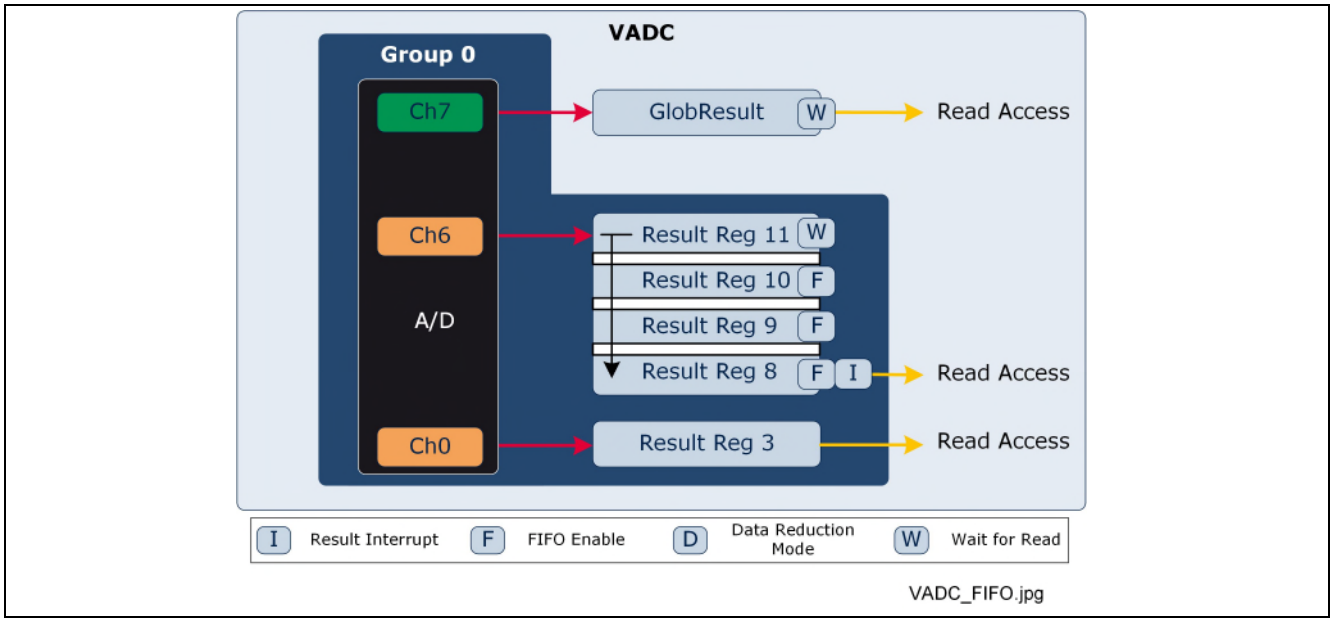

Figure 52 High sampling rate conversions using result buffering and FIFO

The result has to be read from the lowest FIFO element (Result Register 8). The input stage of the FIFO (Result Register 11) is set to wait-for-read and data-reduction mode. Intermediate stages of the FIFO (Result Register 10 and Result Register 8) are set to FIFO Enable. The output stage (Result Register 8) is set to FIFO Enable and generates an interrupt every time a new valid result is stored in this result register. On each interrupt the Result Register 8 is read once. If the interrupt is not executed before the next conversion is done, the new result is stored in the FIFO and copied to Result Register 8. When the interrupt is executed and Result Register 8 read, the content from Result Register 8 is automatically copied to Result Register 8. A new valid content in Result Register 8 generates a new interrupt.

The Background Request Source triggers channel 8 which is assigned to the Global Result Register and is set to wait-for-read mode. One more conversion (channel 0) is triggered by the Queue Request Source and can be stored in independent result registers.

Note:Special care has to be taken when debugging. The debugger can perform read operations on registers and the ADC hardware sometimes reacts to these read operations. For example when the debugger

reads from a FIFO, the FIFO logic is activated, shifting down all registers. This means a further read by the application software may read incorrect results. Therefore the debugger must always read from the dedicated debug view, using registers that do not trigger the ADC hardware, namely GxRESy instead of GxRESDy.

5.2.2 Wait-for-Read

This operation mode automatically suspends the start of a conversion for any channel which is assigned to a result register until the result has been read. That means, if wait-for-read is enabled for a result register, a request source does not generate a conversion request until the result register has been read (indicated by the valid flag VF = 0 of GxRESy).

Cancel-inject-repeat mode does not work when wait-for-read mode is enabled. A higher priority request cannot interrupt a current lower priority conversion request if both request sources target the same result register.

To enable the wait for read mode:

- Bitfield WFR of GxRCRy has to be set to 1.

The following figure demonstrates how, with 'wait-for-read' mode enabled, the channel conversion requests can be made only when the result has been read (Valid Flag equal to 0).

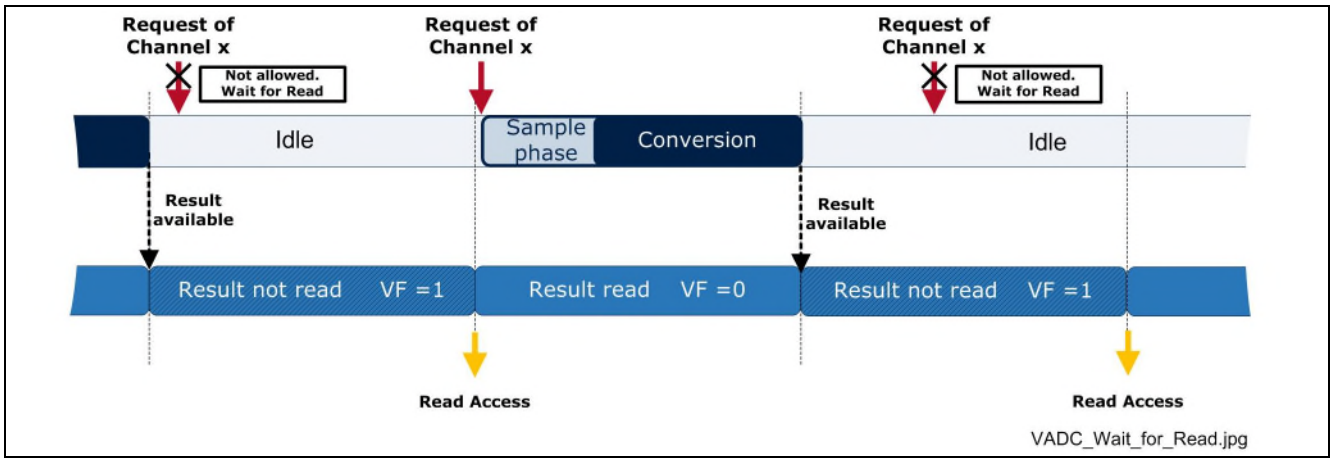

Figure 53 Wait for Read mode

Note: Wait-for-read mode is ignored for synchronized conversions of synchronization slaves. For more details, see 2.6 Synchronous Conversion.

5.2.3 Reading with the DMA

The GPDMA is a highly configurable DMA controller that allows high-speed data transfers between peripherals, memories, and peripherals and memories. Complex data transfers can be executed with minimal processor intervention, keeping CPU resources free for other operations.

The GPDMA peripheral is closely linked to the VADC. Events such as Result or Channel Events can trigger the DMA transfer.

To configure the DMA transfer trigger by a VADC generated event:

Result Handling Use Cases

1. Read the Channel Enable register to choose a free (disabled) DMA channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers.
3. Program the following channel registers for the selected channel:

- Write the starting source address (The Result Register selected in the VADC configuration).
- Write the starting destination address (The variable where to store your conversion).
- Program the Control Register and the LLP register with 0.
- Write the control information for the DMA transfer.
- Write the channel configuration information:
  - Designate the handshaking interface type for the source and destination peripherals. In this case hardware handshaking.
  - Assign a handshaking interface to the source and destination peripheral.
- If gather is enabled, program the Source Gather register.
- If scatter is enabled, program the Destination Scatter register.

4. After the GPDMA-selected channel has been programmed, enable the channel.
5. Configure the DLR (DMA Line Router) block to map the DMA requests from the peripherals to the required DMA request lines with the DLR_SRSELx and DLR_LNEN registers.
6. Configure the VADC.
7. Enable the Service Request Generation in the VADC.
8. Select the service request line chosen in the DLR configuration.

In the next figure an application example is presented. Here, a result FIFO queue is being implemented in the VADC. The DMA module is being triggered by the Valid Flag of the output FIFO stage (Result Register 11). When a new result appears in Result Register 11 (VF=b1), the DMA transfers the value from the source address “Result Register 11” to the destination address in the RAM.

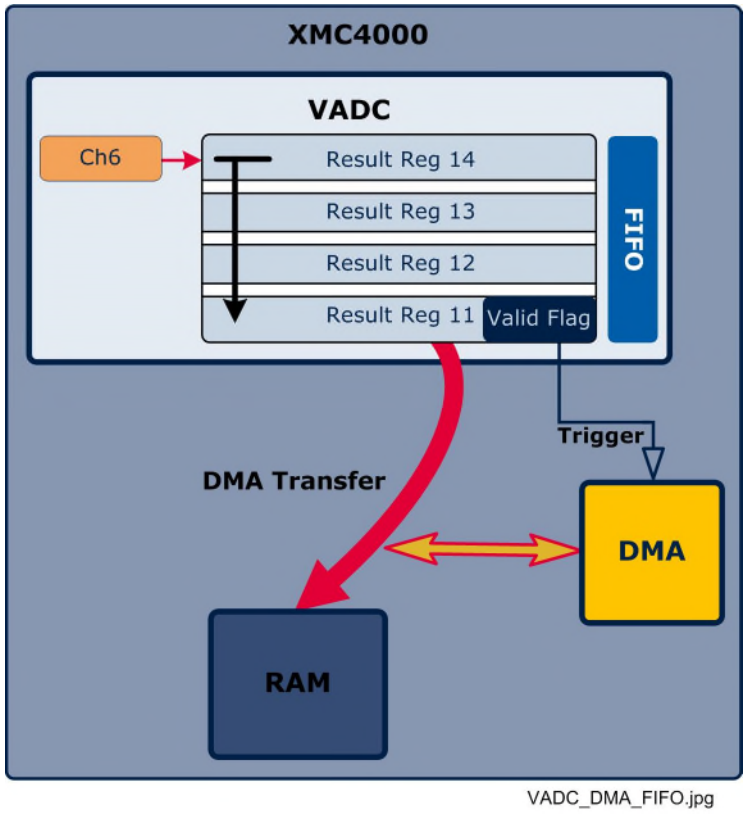

Figure 54 DMA Transfer from Result to RAM

5.3 Result Post-Processing

5.3.1 Result Format and Alignment

In order to minimize the data accommodation in the application software, it is possible to define a determinate format for each result value inside the Result Register.

In standard conversions (no data reduction) the results are stored in bits 0 to 11 in the RESULT bitfield of the selected GxRESy result register. However, if the resolution chosen is lower than 12 bits (10 or 8 bits) the alignment of the result inside the first 12 bits of the Result Register can be chosen as follows:

- **Select the alignment** between left-aligned (RESPOS=b0) or right-aligned (REPOS=b1) in the respective Channel Control Register (GxCHCTRy).

For data accumulation mode (data reduction) results are stored in bits 0 to 15 of the RESULT bitfield because of the additional data size. Here, the alignment is made as in normal conversion but taking into account the length of the results. See Data Reduction.

Note: The bitfield RESULT can be written by software to provide the reference value for Fast Compare Mode. In this mode, bits 11 to 2 are evaluated and the other bits are ignored. For more details, see: 2.8 High Conversion Rate.

5.3.2 Result Filtering

Both third order Finite Impulse Response (FIR) filters and first order Infinite Impulse Response (IIR) filters are available. These can be applied to the Result Register 7 and Result Register 15 of each group. This functionality allows for the averaging or low pass filtering of a signal without CPU overhead.

The filter coefficients are selectable. 14 different combinations for the FIR filter and 2 for the IIR filter coefficients can be selected. In the following example, the Magnitude and Phase Bode Diagram for two possible combinations are presented: an FIR filter in the first figure and an IIR filter in the second:

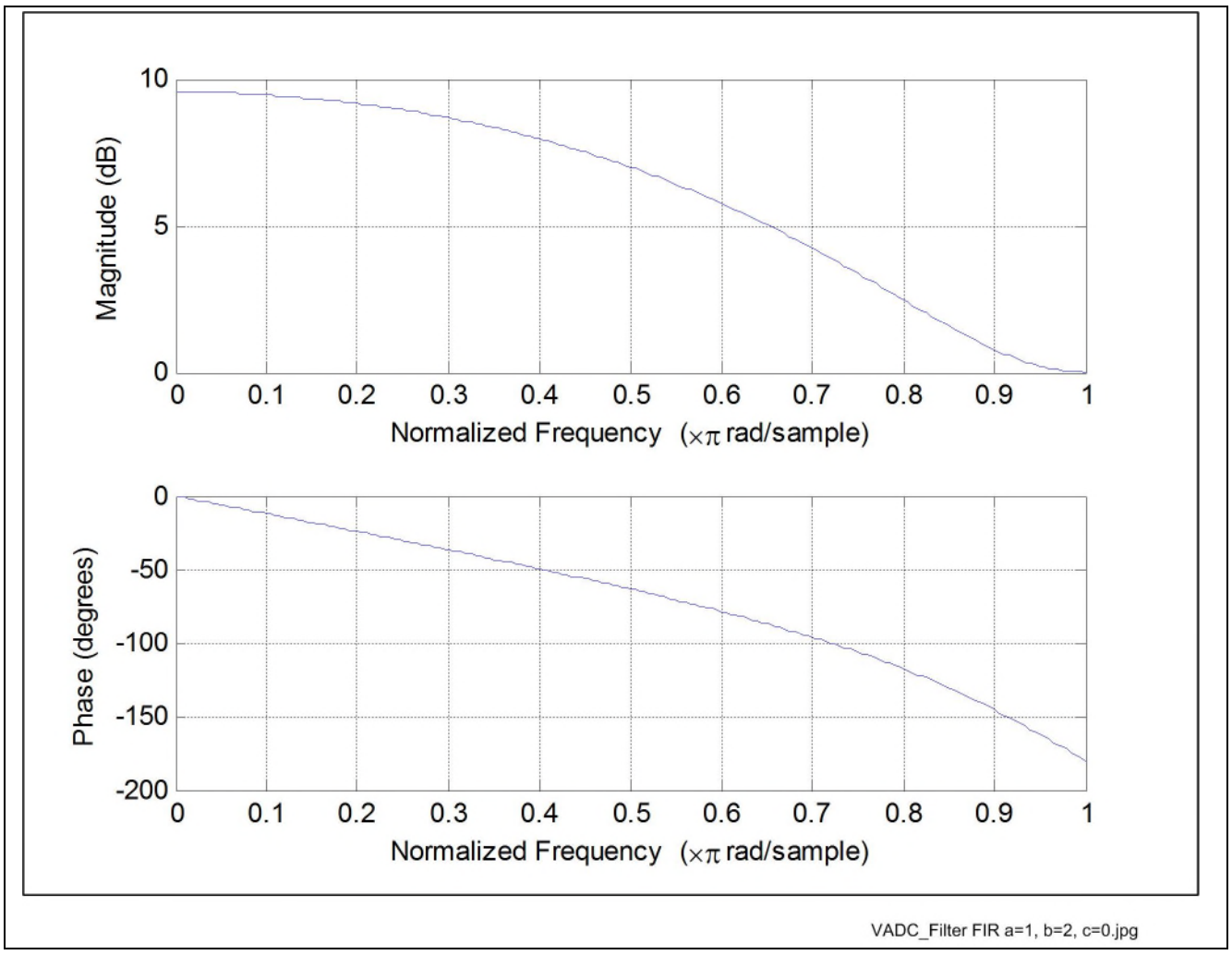

Figure 55 FIR filter with coefficients a=1, b=2 and c=0

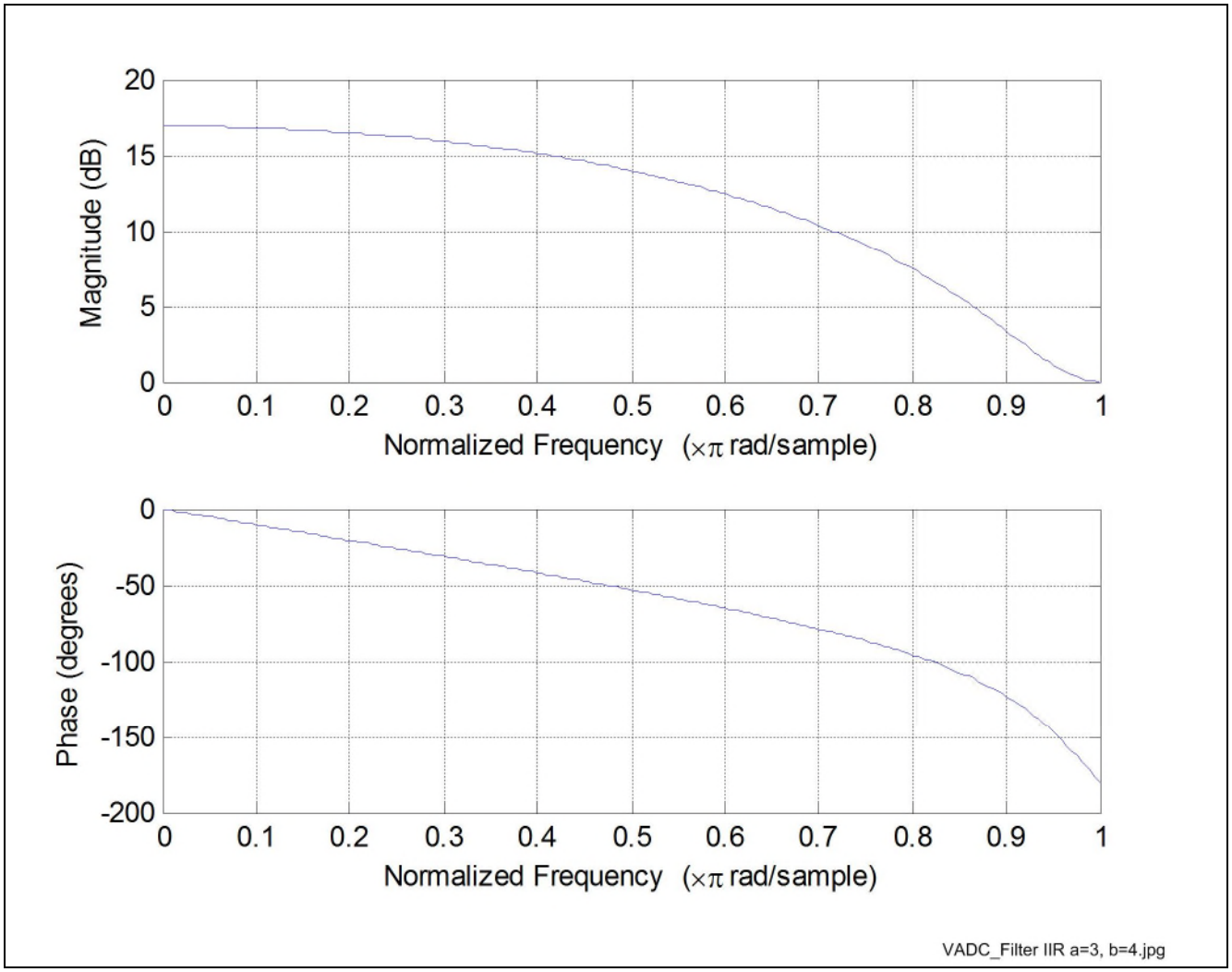

Figure 56 IIR filter with coefficients a=3, b=4

To program these filters

1. Select the Result filtering mode (DMM=b01) in the GxRCRy register.
2. Choose the kind of filter and the coefficients in the DRCTR bitfield of the GxRCRy register.

In addition, a difference mode can be used to subtract the Result Register 0 to any other result register (except the Global Result Register). This is useful for offset compensation or for emulating a differential conversion.

To program the difference mode

- Select the Difference mode (DMM=b10) in the GxRCRy register.

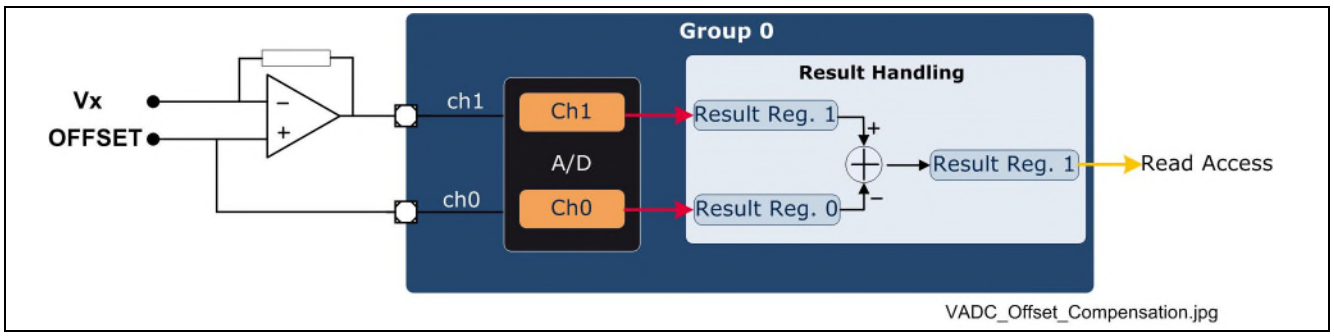

Figure 57 Offset Compensation using the Difference Mode

5.3.3 Data Reduction

In order to increase the resolution or remove some noise from the input signal, the XMC4000 VADC implements three Standard Data Reduction modes. A simple built-in accumulator can sum up two, three or four consecutive conversions into one Result Register. This applies to all available result registers except the Global Result Register. This can increase the resolution without any software computation.

To program the Data Reduction modes:

1. Select the Standard Data Reduction Mode (DMM=b00) in the GxRCRy register.
2. Choose between accumulate 2 result values (DRCTR=b0001), 3 result values (DRCTR=b0010) or 4 result values (DRCTR=b0011) in the GxRCRy register.

*Note: The data reduction counter **DRC** bitfield in the **GxRESy** register indicates the number of values still to be accumulated for the final result. So the final result is available and the 'valid flag' VF is set when bitfield DRC becomes zero (by decrementing or by reload).*

6 Events and Interrupts

6.1 Introduction

The VADC of the XMC4000 family of devices is able to generate signals that indicate the occurrence of certain events in other peripherals. For example, a new result is available, a conversion sequence has finished, the conversion result is above a boundary, and so on. These signals can be used by timers and DMAs to trigger specific actions such as starting or stopping a timer, or the transfer of some data. These signals can also be connected to the ERU for extended connectivity and, of course, can be connected to an NVIC node for deterministic code execution.

There are three sorts of events that can be tracked:

- Request source events: These indicate that a request source has completed the requested sequence of conversions.
  - For a Scan Request Source (group or Background), the event is generated when the complete defined set of channels (pending bits) has been converted.
  - For a Queue Request Source (group), the event is generated when a channel with source interrupt enabled, has been converted.
- Channel events: These can be generated when the conversion of a single channel is finished. Optionally, channel events can be restricted to result values within a programmable value range (similar to a comparator).
- Result events: These indicate that a new valid result is available in a result register. Usually, this triggers a read action by the CPU. Optionally, result events can be generated only at a reduced rate if data reduction is active.

All these events can be assigned to a service request output. Two different kinds of request outputs are available:

- Group specific (GxSRy) request outputs are only driven from the group where the event is generated. Background Request Source events are not included here.
- Common request lines (CxSRy) can be accessed from any group and even from the Background Request source events.

Each service request can generate an Interrupt where software can be executed, a DMA transfer made, or a Peripheral action.

In the XMC4000 devices, the Nested Vectored Interrupt Controller (NVIC) peripheral is in charge of controlling and handling the interrupts. It includes a Non-Maskable Interrupt (NMI) and provides up to 64 interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of Interrupt Service Routines (ISRs), dramatically reducing the interrupt latency.

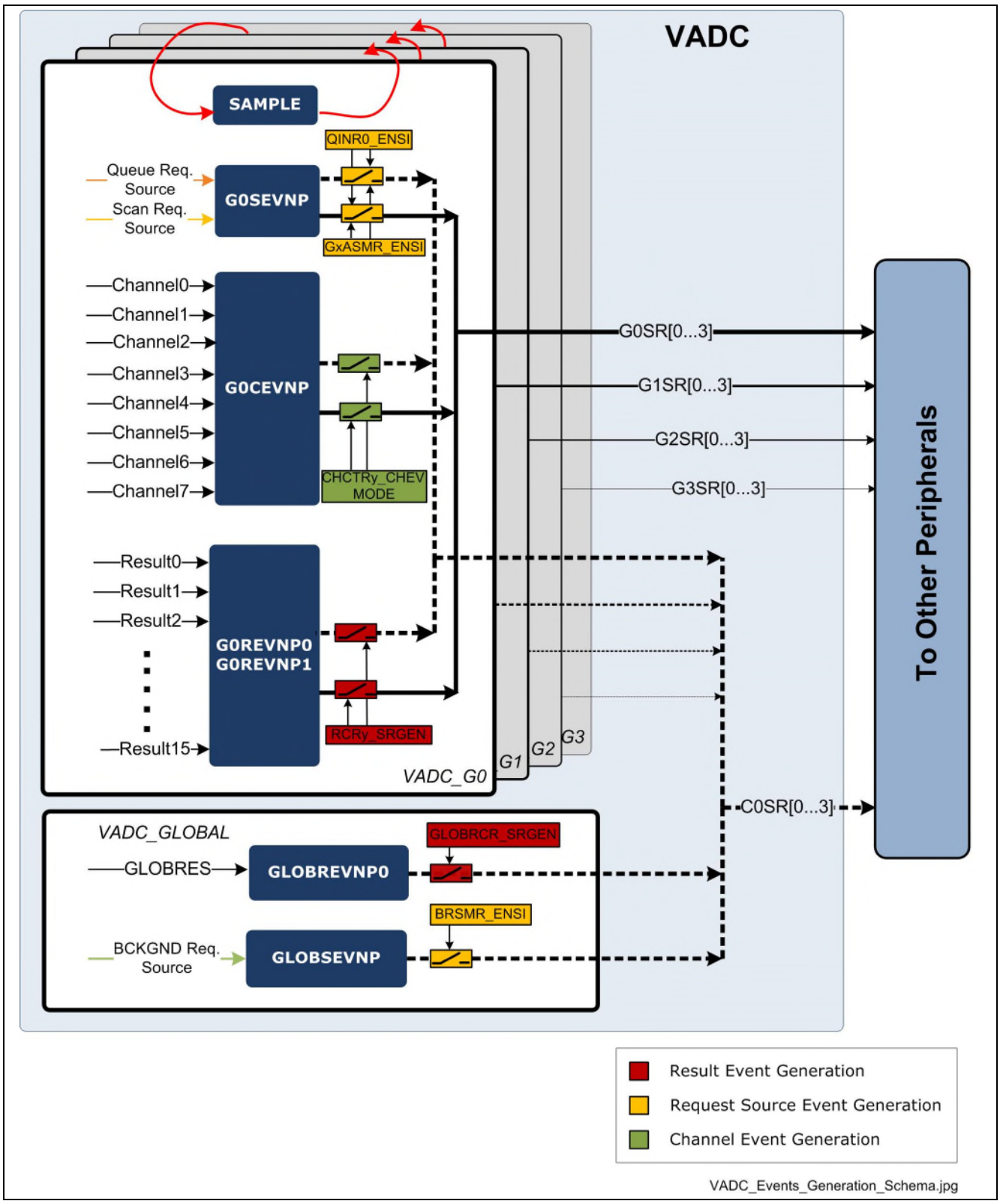

Figure 58 Service Request Generation Schema

6.2 Generating Events

Each A/D Converter can activate up to 4 group-specific Service Request output signals and up to 4 common service request output signals to issue an interrupt or to trigger a DMA channel.

These service requests are generated mainly by four types of events. Each event occurrence is indicated by a flag. The flag can be set by software by writing 1 to the respective bitfield in each Event Flag Register (GxSEFLAG, GxCEFLAG, GxRFLAG or GLOBEFLAG). The flag can be cleared by writing 1 to the respective bitfield in each Event Flag Clear Register (GxSEFCLR, GxCEFCLR or GxREFCLR). However, the ADC event can generate a service request without the need to clear it. That means every new event generates a service request.

Note:Each service request can be activated via software by setting the corresponding bit in the register GxSRACT. This can be used for evaluation and testing purposes.

6.2.1 Request Source Events

To generate Background Source events:

3. Enable the Source Interrupt (ENSI=b1) in the Background Source Mode Register (BRSMR).
4. Select a service request line in the Service Request Node Pointer (SEV0NP) bitfield of the Global Event Node Pointer Register (GLOBEVNP).
5. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

To generate Scan Request Source events:

1. Enable the Source Interrupt (ENSI=b1) in the respective Scan Source Mode Register (GxASMR).
2. Select a service request line in one of the available Service Request Node Pointers (SEV0NP and SEV1NP) in the Source Event Node Pointer Register (GxSEVNP).
3. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

To generate Queue Source events:

1. Enable the Source Interrupt by setting the ENSI=b1 in the respective Queue 0 Input Register (GxQINR0).
2. Select a service request line in one of the two Service Request Node Pointers available (SEV0NP and SEV1NP) in the Source Event Node Pointer Register (GxSEVNP).
3. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

6.2.2 Channel Events

A channel event is generated when the conversion of the channel has finished or if the conversion value is inside or outside a user-configurable range (See 7.2 Limit Checking: Signal Monitoring). Nevertheless, the generated event does not indicate if a valid result is available or not, just that the channel conversion has finished. Further result handling can delay the result availability.

How to configure and generate a channel event

1. Define the channel event generation mode in the CHEVMODE bitfield of the respective Channel Control Register (GxCHCTRy). Choose between:
   - Generate an event in Standard Conversion (b00=Never, b11=Always)
   - Generate an event in Standard Conversion restricted to result values within a programmable value range (b01=Result is inside the boundary band, b10=Result is outside the boundary band)

- Generate an event in Fast Compare Mode (b00=Never, b01=Result is above comp. value, b10= Result is below comp. value, b11=Result switched to either level)

2. Select a service request line in one of the eight Service Request Node Pointer Channel Events available (CEVxNP, with x=0 to 7), in the Channel Event Node Pointer Register 0(GxCEVNP0)
3. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

An application example is presented in the next figure:

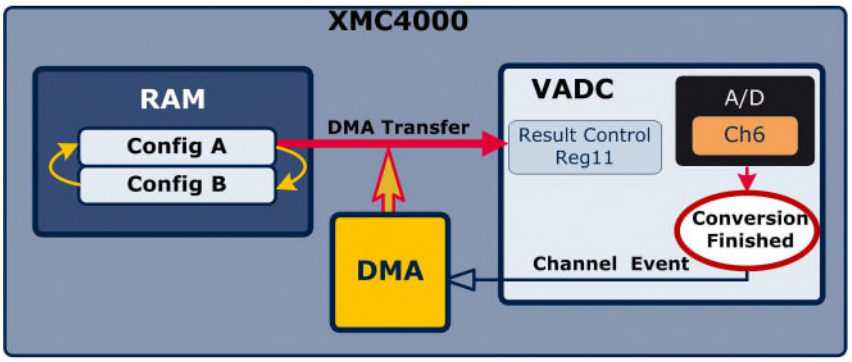

Figure 59 Channel Event application example

In this example a channel event is configured for Channel 6. Every time the Channel Event is generated, the DMA triggers a transfer from RAM to the Result Control Register of the Result Register 11. The DMA changes after each channel event the source address of the transfer; i.e. after every conversion of Channel 6, the configuration of the Result Register 11 automatically changes.

6.2.3 Result Events

The steps listed below are required to define and generate a result event every time a valid result value is available in the respective Result Register.

Group-specific registers

1. Enable the Service Request Generation after a result event (SRGEN=b1) in the respective Result Control Register (GxRCRy).
2. Select a service request line in one of the first eight Service Request Node Pointer Result Events available (REVxNP, with x=0 to 7) in the Result Event Node Pointer Register 0(GxREVNP0), or in one of the next eight Service Request Node Pointer Result Events available (REVyNP, with y, 8 to 15) in the Result Event Node Pointer Register 1(GxREVNP1).
3. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

Global Result registers

1. Enable the Service Request Generation after a result event (SRGEN=b1) in the Global Result Control Register (GLOBRCRy).
2. Select a service request line in the Service Request Node Pointer Background Result (REV0NP) of the Global Event Node Pointer Register GLOBEVNP).
3. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

In the example figure below, a DMA transfer between Address A to Address B in RAM is being made after a new valid result register is available (result event).

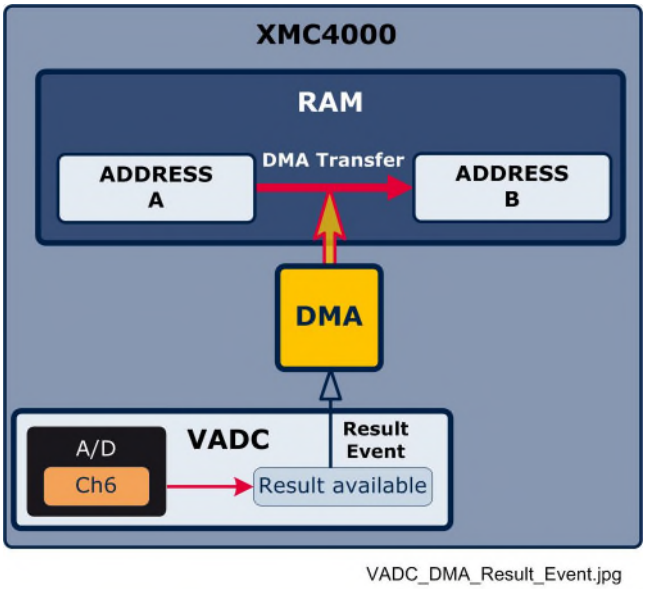

Figure 60 Result event application example

6.2.4 Boundary Flag Events in XMC4400, 4200 and 4100 Devices

XMC4400, 4200 and 4100 devices are able to generate events after a change in a boundary flag. For more details, see 7.4 Boundary Flags.

To generate and handle an event provoked by a boundary flag change:

1. Enable and configure the comparison mode (Fast Compare Mode or Limit Checker) and its respective boundary flag. (See Use Cases Comparator)
2. Select a service request line in the respective Boundary Flag Node Pointer bitfield (BFLyNP) of the Boundary Flag Node Pointer Register (GxBFLNP).
3. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

6.3 Interrupts

The Nested Vectored Interrupt Controller (NVIC) is the peripheral in charge of controlling interrupts in the XMC4000 family of products. The Cortex-M4 NVIC facilitates interrupt handling with low-latency exception.

In order to handle the interrupts generated by the events previously described:

1. Enable the required event generation.
2. Select a service request line.
3. Use the CMSIS function to configure the NVIC node by enabling the NVIC node for the request line chosen in the previous step (NVIC_EnableIRQ(IRQn_t IRQnunmber)), or disable the NVIC node (NVIC_DisableIRQ(IRQn_t IRQnumber)).
4. Define the Interrupt Handlers and the software actions required during the interrupts:

```
void NVICnode_IRQHandler (void){
//Operations made during the interrupt
}
```

The following table lists the service request sources per on-chip unit, and their assignment to NVIC IRQ numbers.

Table 2 Service Request to IRQ Number Assignment

Service Request	IRQ Number	Description
VADC.C0SR0- VADC.C0SR3	14…17	Analog to Digital Converter Common Block 0
VADC.G0SR0- VADC.G0SR3	18…21	Analog to Digital Converter Group 0
VADC.G1SR0- VADC.G1SR3	22…25	Analog to Digital Converter Group 1
VADC.G2SR0- VADC.G2SR3	26…29	Analog to Digital Converter Group 2(XMC4400 and XMC 4500)
VADC.G3SR0- VADC.G3SR3	30…33	Analog to Digital Converter Group 3(XMC4400 and XMC 4500)

7 Comparator Use Cases

7.1 Introduction

In many applications signals need to be monitored to compare the values with user-defined boundaries. The VADC in the XMC4000 devices implement several mechanisms to generate comparisons.

XMC4000 Compare Mechanisms

- Limit Checking is for monitoring if a signal is inside or outside a fixed band in standard conversions. For this compare operation, one or two boundaries have to be defined. Comparison time using this feature is the same as in a standard conversion (i.e. 550ns in a 12-bit conversion).
- Fast Compare Mode is for applications that need time-critical comparison of signals. In this mode, the input signal is directly compared to a value in the associated result register. This compare operation returns a binary value indicating if the compared input voltage is above or below the given reference value. This saves time if the exact conversion result value is not required. Comparison in Fast Compare Mode can last a minimum of 150ns, but depends on the sampling time chosen.
- Out of Range Comparator is an analog comparator that serves the purpose of over-voltage monitoring (voltage higher than VAREF) for the analog input pins of the chip. Please refer to the appropriate Data Sheet for the specific values.

In the following figure, Limit Checking is implemented with a user-defined band:

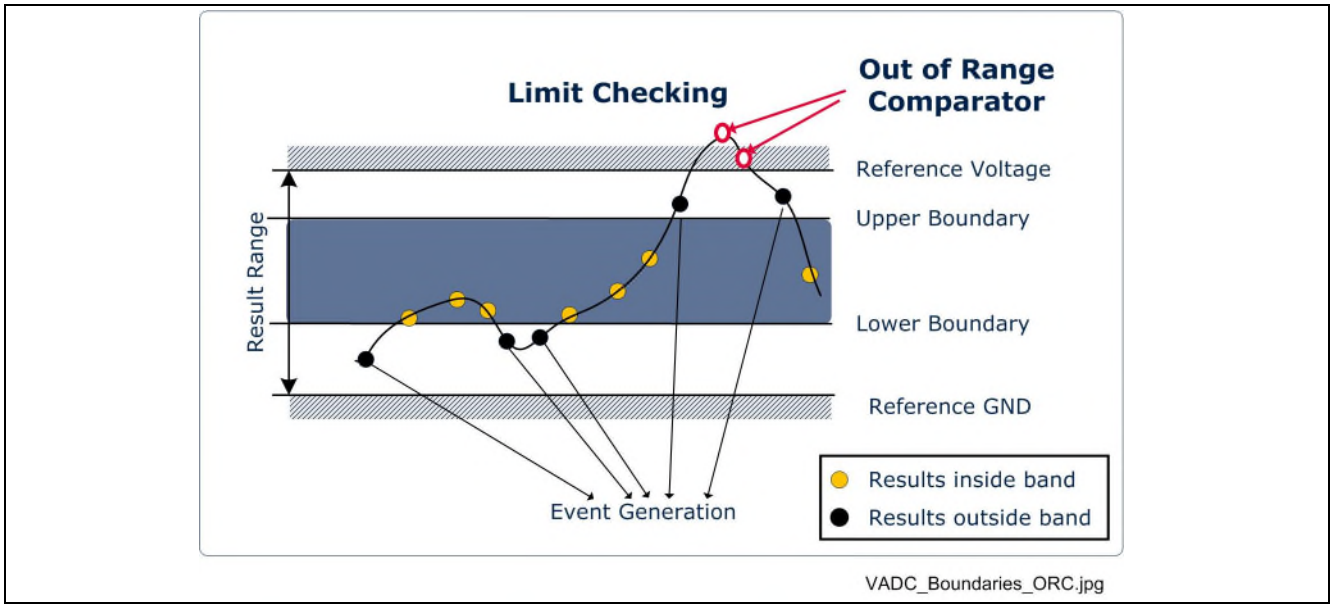

Figure 61 Limit Checking and Out of Range Comparator

In the figure above, conversion results outside the band cause events to be generated.

In the next figure, Fast Compare is used to generate a channel event every time the conversion value is below the compare value.

In both figures the Out of Range Comparator is being used to detect overvoltage (values greater than VAREF).

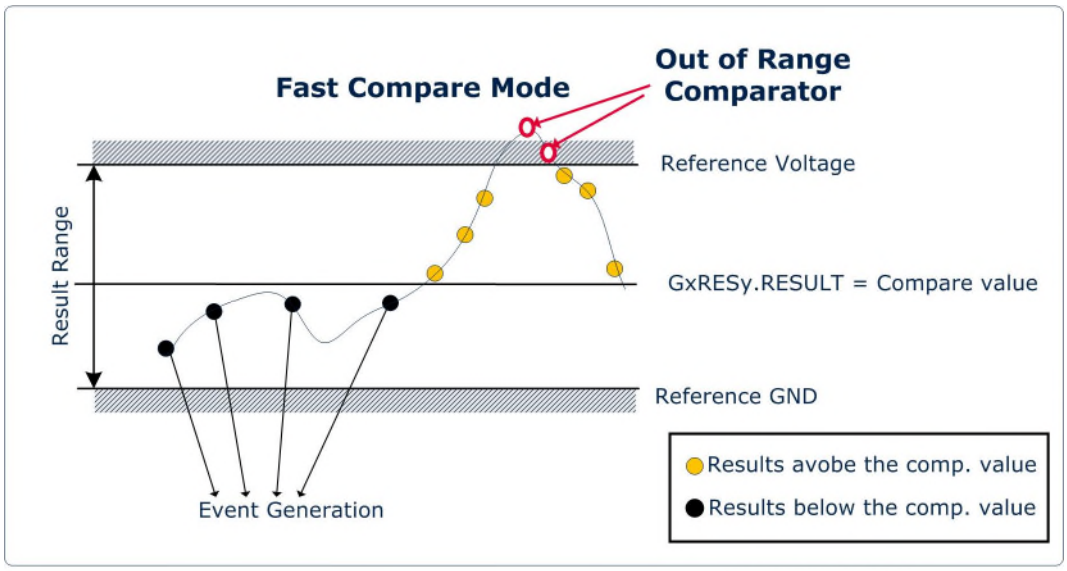

Figure 62 Fast Compare and Out of Range Comparator

Boundary flags show the comparison results in Limit Checking and Fast Compare modes. These can be set or cleared when the defined level is exceeded and are also available as control signals for other modules (such as trigger, event generation, and so on).

Boundary flags are not associated to every channel and result register. The following table shows to which channels or result registers these flags are assigned, depending on the comparison mode and the selected XMC4000 device.

Table 3 Boundary flags availability

XMC Device	With Fast Compare Mode	With Limit Checker
4400, 4200 and 4100	Associated to the lower 4 result registers. **BFLy → GxRESy (y, 0 to 3)**	Associated to the lower 4 result registers. **BFLy → GxRESy (y, 0 to 3)**
4500	Associated to the lower 4 result registers. **BFLy → GxRESy (y, 0 to 3)**	Associated to the lower 4 channels. **BFLy → Channely (y, 0 to 3)**

7.2 Limit Checking: Signal Monitoring

With Limit Checking, every digital conversion result can be automatically compared to an Upper and a Lower Boundary value. A channel event can be generated when the result of a conversion is inside or outside of a user-defined band, enabling service requests to only be issued under certain pre-defined conditions (depending on the boundary definition) and, therefore, saving CPU execution time.

If a value is equal to one of the boundaries, it is considered to be inside the boundary band. This means, in a comparison, the value equal to the Lower Boundary is considered above the Lower Boundary, and the value equal to the Upper Boundary is considered below the Upper Boundary.

Each VADC group can have user defined boundaries for Limit Checking and one additional Global Limit Checker can be selected by all VADC groups and their channels.

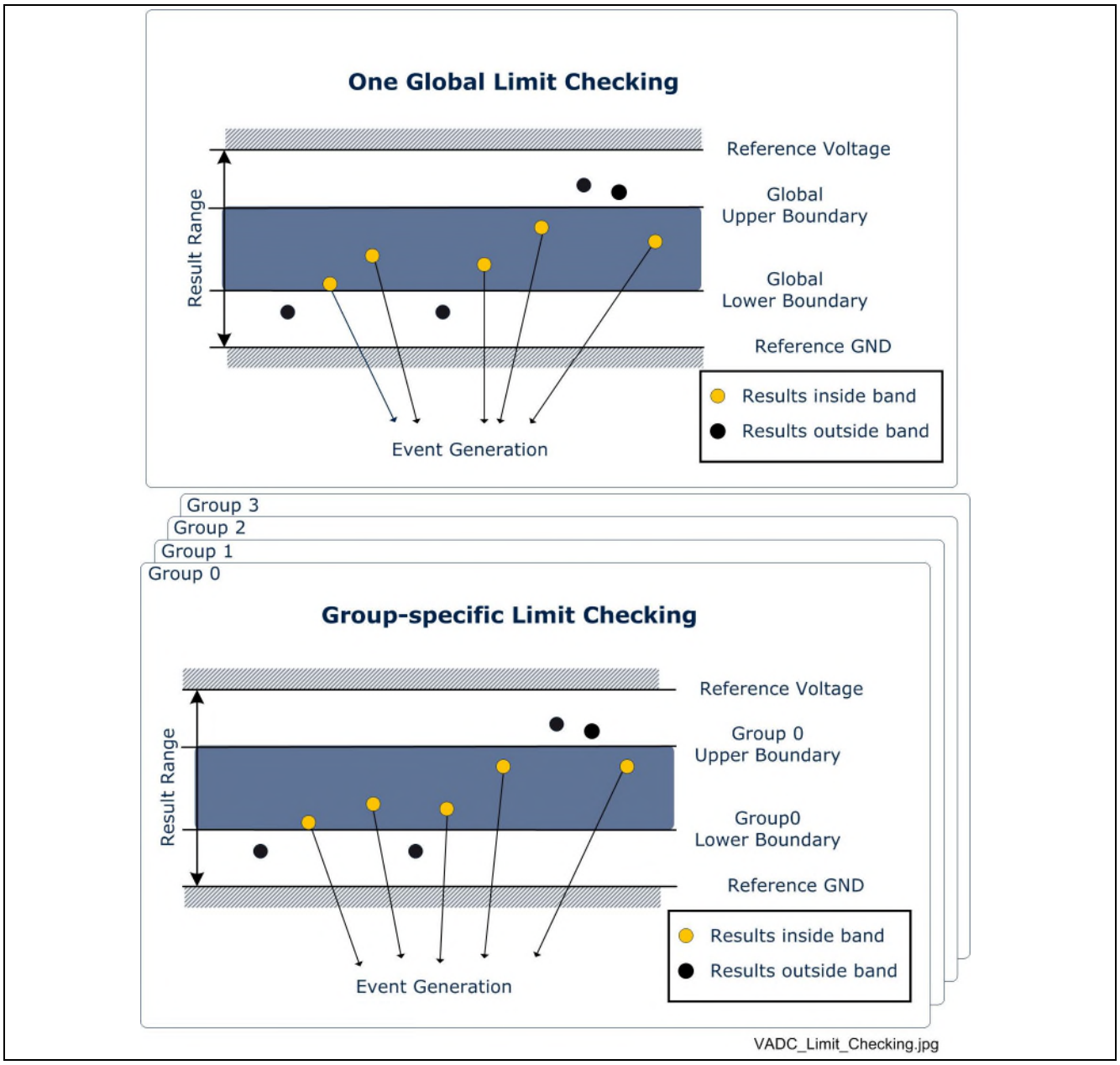

Figure 63 Global and Group-specific Limit Checking

There is no need to compare using two references (Upper and Lower Boundaries). By setting the Upper Boundary to the maximum value of the result range or the Lower Boundary to the minimum, the voltage space of two regions can be efficiently monitored.

The next figure shows how values can be compared to one boundary in standard conversions. Here, all the values are below the Upper Boundary or above the Lower Boundary.

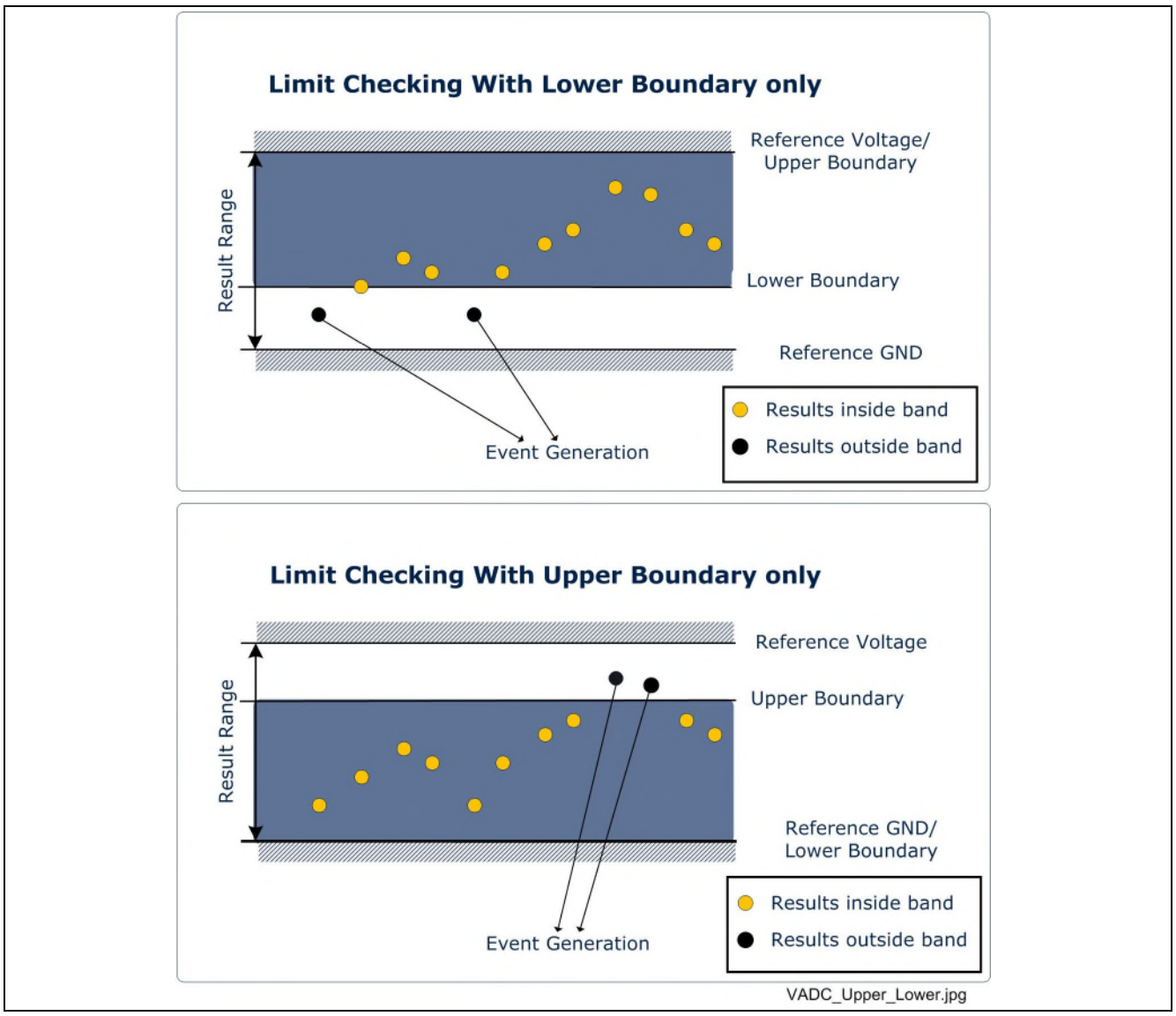

Figure 64 Limit Checking with just one boundary

Configuring the Limit Checking feature

5. Define the boundaries for the group-specific Limit Checking (GxBOUND) and/or for the global Limit Checking (GLOBBOUND) in the BOUNDARY0 and BOUNDARY1 bitfields.
6. In the BNDSELL bitfield of the GxCHCTRy Register, select the Lower Boundary for the comparison:
   - b00=Use group specific BOUNDARY 0
   - b01=Use group specific BOUNDARY 1
   - b10=Use global BOUNDARY 0
   - b11=Use global BOUNDARY 1
7. In the BNDSELU bitfield of the GxCHCTRy Register, select the Upper Boundary for the comparison:
   - being b00=Use group specific BOUNDARY 0
   - b01=Use group specific BOUNDARY 1
   - b10=Use global BOUNDARY 0

- b11=Use global BOUNDARY 1
- Configure the event generation. In the CHEVMODE bitfield of the GxCHCTRy Register, choose if the channel event is generated for each channel when:
- b01=Result inside the boundary band
- b10=Result outside the boundary band

8. Select a service request line in one of the eight Service Request Node Pointer Channel Events available (CEVxNP, with x=0 to 7), in the Channel Event Node Pointer Register 0(GxCEVNP0)
9. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

7.3 Fast Compare Mode

The integrated Fast Compare Mode allows for a conversion more than 3 times faster than a regular conversion. Within 150ns, an ADC can determine whether a result is above or below a user defined reference value.

Each channel of each group can have Fast Compare Mode enabled. Conversion is automatically compared to the value stored in the RESULT bitfield of the selected GxRESy register. Only bits 11 to 2 of this bitfield are evaluated, the other bits are ignored.

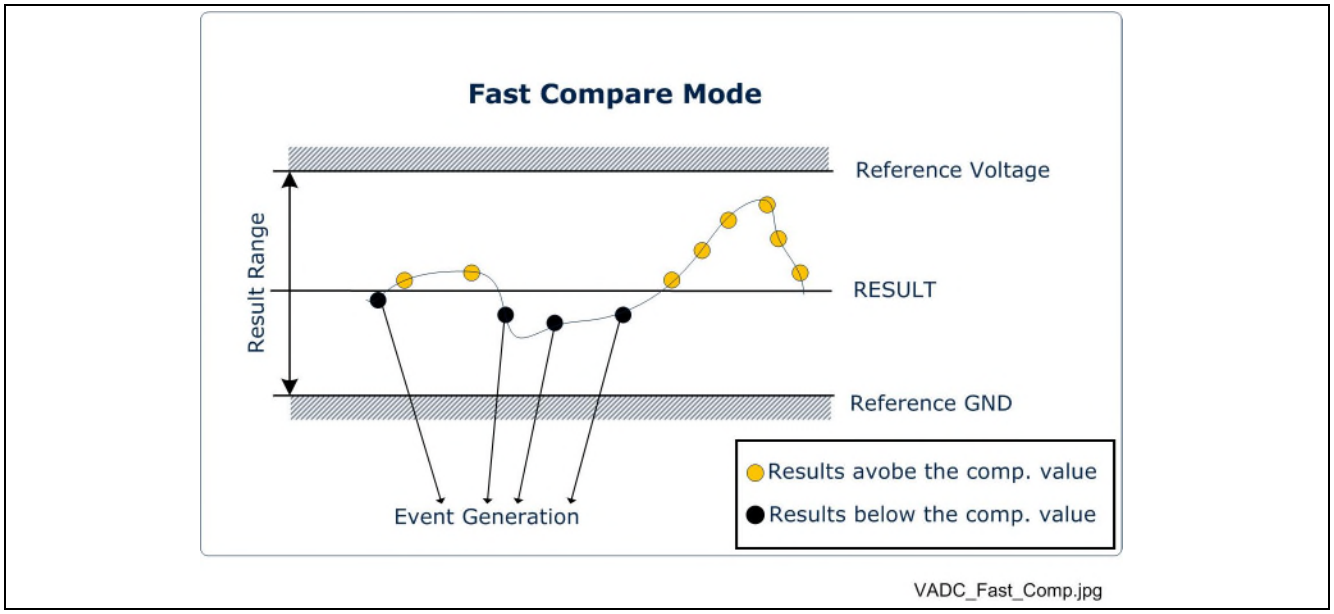

Figure 65 Fast Compare Mode

7.3.1 Steps for Fast Compare Mode

1. In the CMS bitfield of one of the available classes (GxICLASS0, GxICLASS1, GLOBICLASS0 and GLOBICLASS1), choose the 10-bit Fast Compare Mode (CMS=b101). See 3.2.2Resolution and Sampling Time Configuration - Select and Configure Input Classes
2. In the GxCHCTRy Register, choose the Result Register to which the conversion value is compared in the RESREG bitfield. Select the Class chosen in step 1 in the ICLSEL bitfield and define in the CHEVMODE bitfield if the channel event is generated for each channel when:
   - b01=The result is above the compare value

- b10=The result is below the compare value
- b11=The result switches to either level

3. Provide the reference value for Fast Compare mode in the RESULT bitfield of the GxRESy Register (the one chosen in step 2).
4. Select a service request line in one of the eight Service Request Node Pointer Channel Events available (CEVxNP, with x=0 to 7) in the Channel Event Node Pointer Register 0(GxCEVNP0)
5. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

Note: The bitfield FCR in the GxRESy Register is very useful for debugging purposes. It indicates the result of an operation in Fast Compare Mode. If the value is 0, the signal level is below the compare value. If the value is 1, the signal level is above the compare value.

7.3.2 Hysteresis in Fast Compare Mode

XMC4100, 4200 and 4400 devices allow the definition of delta limits for the Fast Compare Mode. These deltas define a hysteresis band to hinder bounces:

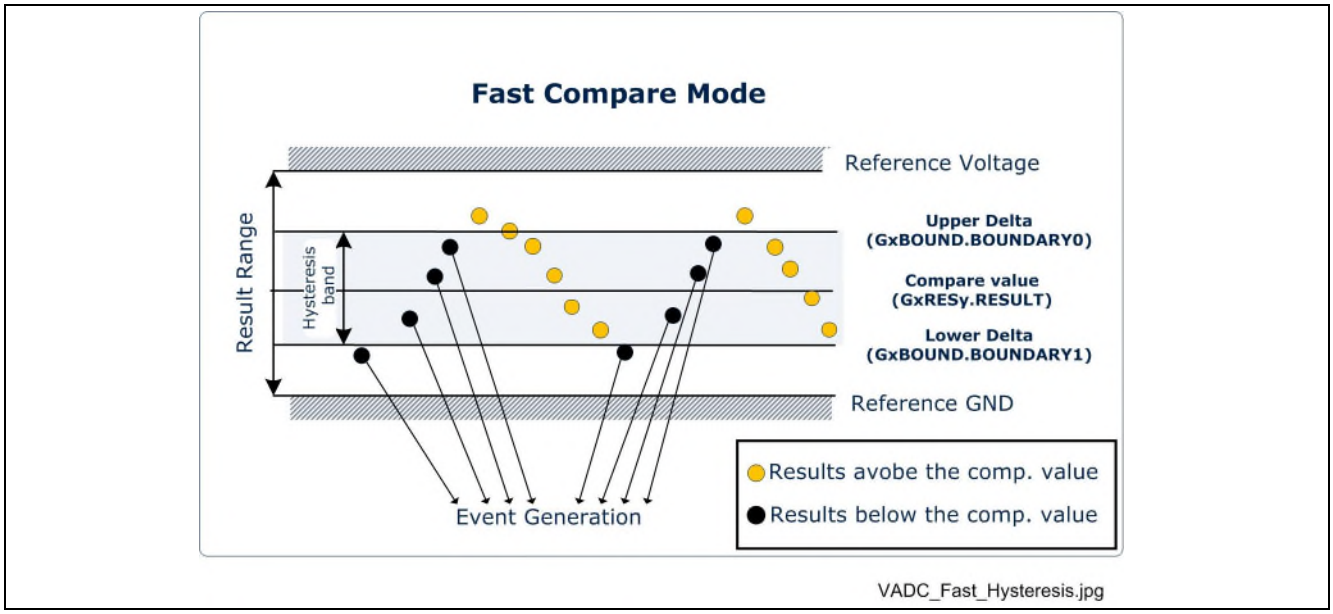

Figure 66 Fast Compare Mode Hysteresis

How to program deltas

- Define delta limits in bitfields BOUNDARY1 and BOUNDARY0 of register GxBOUND

For the XMC4500 microcontroller, the hysteresis in Fast Compare Mode has to be defined by software. An application example is depicted below. Here, the DMA switches the comparison value from Result Register 0 (RES0) to Result Register 1 (RES1) and vice-versa. Each DMA switch is triggered by the channel event generated if the conversion result changes to either level, above or below the compare value.

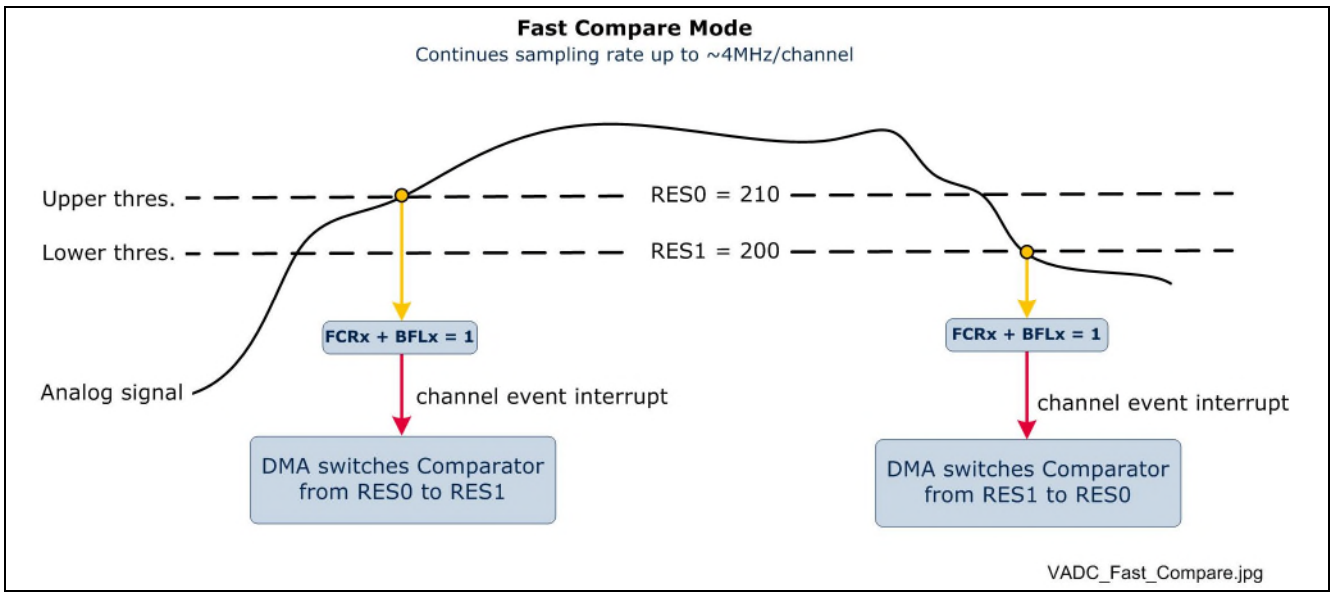

Figure 67 Fast Compare use case

7.4 Boundary Flags

Boundary flags indicate if a value has crossed the activation boundary. These flags can be represented as a change in the bitfield BFLy of the Boundary Flag Register (GxBFL), and can also act as trigger signals for other modules to signal events.

In Limit Checking mode, the band between the two boundary values defines a hysteresis for setting/clearing the boundary flags:

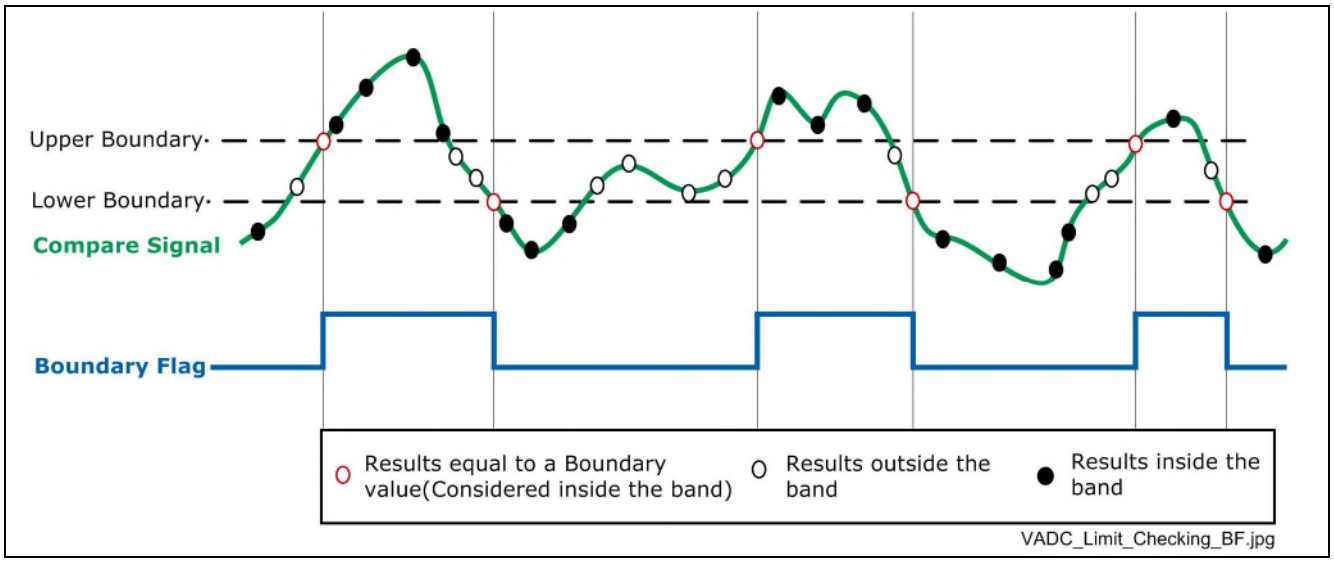

Figure 68 Boundary Flag in Limit Checking

In the next figure a boundary flag is used to monitor the comparisons in the**Fast Compare mode**. In this mode, the boundary flags are associated with the lower 4 result registers.

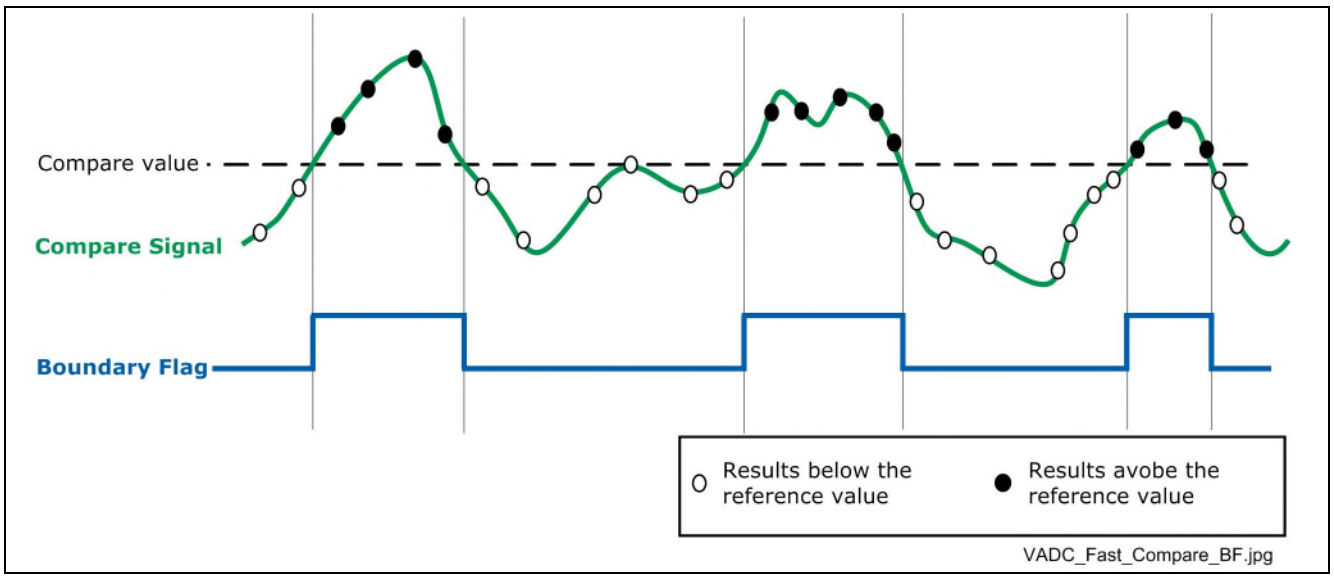

Figure 69 Boundary Flag in Fast Compare mode

Depending on the XMC4000 device, the Boundary Flag will be defined differently.

XMC4400, 4200 and 4100 devices

1. In the GxBFL Register, choose between setting the boundary flag if a result has crossed the defined band (or compare value) (BFAy=b0). Or set it if the result is below the defined band (or compare value) (BFAy=b1).
2. Select in the BFIy bitfield of the GxBFL register if the bitfield BFLy (Boundary Flag) is used directly (BFIy=b0) or inverted (BFIy=b1).
3. It is possible to restrict the influence of compare operations to the active phases of the corresponding request source gate signal. This is configured in the BFMy bitfields of the Boundary Flag Control Register (GxBFLC) by choosing between:
   - disable boundary flag (BFMy=b0000)
   - always enable boundary flag (BFMy=b0001)
   - enable boundary flag while gate of source 0 is active (BFMy=b0010)
   - enable boundary flag while gate of source 1 is active (BFMy=b0011)
4. Enable, if required, the respective Boundary Flag Node Pointer bitfield (BFLyNP) in the Boundary Flag Node Pointer Register (GxBFLNP), and choose the respective service request line.
5. Enable the Interrupt, DMA transfer or Peripheral action for the service request line selected.

Note:In XMC4400, 4200 and 4100 devices, Boundary Flags (bitfields BFLy) can be set or cleared by software using the GxBFLS Register: BFCy=b1, clear bit BFLy; and BFSy=b1, set bit BFLy.

XMC4500 device

6. Enable the boundary flag by setting the BFEy bitfield in the in the Boundary Flag Register (GxBFL) to 1.
7. Define if required, the boundary flag generated as the input of another peripheral.

7.5 Overvoltage Detection: Out of Range Comparator

The Out of Range Comparator (ORC) is in charge of detecting voltage overshoot of the reference voltage (VAREF).

The output signal of the ORC is available at the input multiplexer of the event request unit ERU0. This can redirect the signal to the Nested Vectored Interrupt Controller (NVIC) of the Cortex M4.

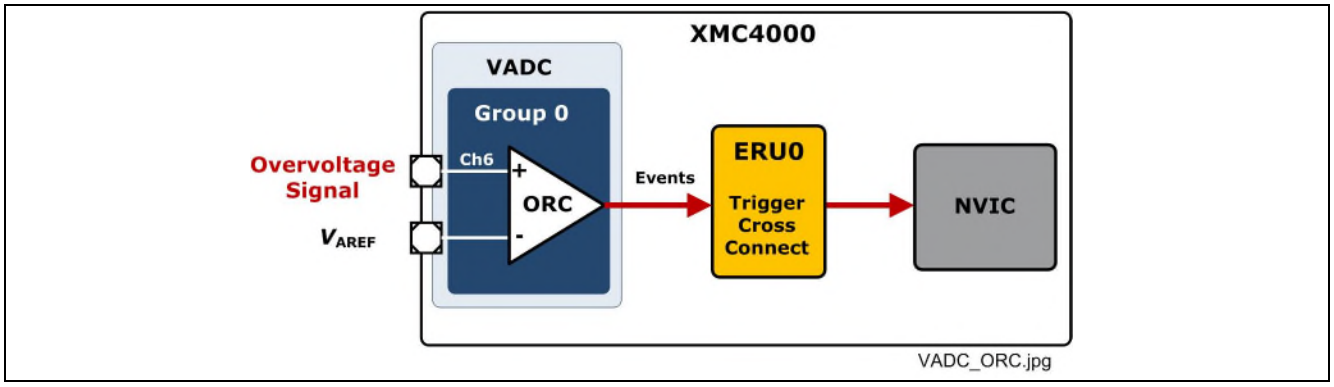

Figure 70 Out of Range Comparator

There are four ORCs available in the VADC of the XMC4000 devices: GxORC6 in channel 6 and GxORC7 in channel 7 for groups 0 and 1.

To program the ORC:

- Enable the respective Out of Range Comparator in the bitfield ENORCy (being y the channel number, 6 or 7) of the register GxORCEN (being x the group, 0 or 1).

8 Application Examples

8.1 Timer as a Trigger

Each sequence described in the Use Cases Conversion chapter and in Trigger Options > Burst sampling can be successfully implemented using an intelligent timer scheme without using any specific feature of the VADC. Two application examples demonstrating this are presented here.

8.1.1 Interleaved Triggering

In this example the same pin (14.2 in the CPU_45A-V2 board) is being converted by two different groups of the VADC: group 1 and group 2 convert the selected analog channel alternately. This means that the channel can be converted much faster than with just one ADC.

Here this is done using the configured trigger from the CCU8 timer instead of using the **GxSAMPLE Signal**. The CCU8 timer is configured to generate two triggers, one for group 0 and one for group 1. The one for group 0 is a CR1, whereas for group 1 it is a Period Match.

If both conversions are configured to last the same time, then CR1=Period/2. Otherwise, the timing scheme can be adapted.

In order to avoid the loss of data, a DMA transfer moves the results from the ADC result registers to a RAM location.

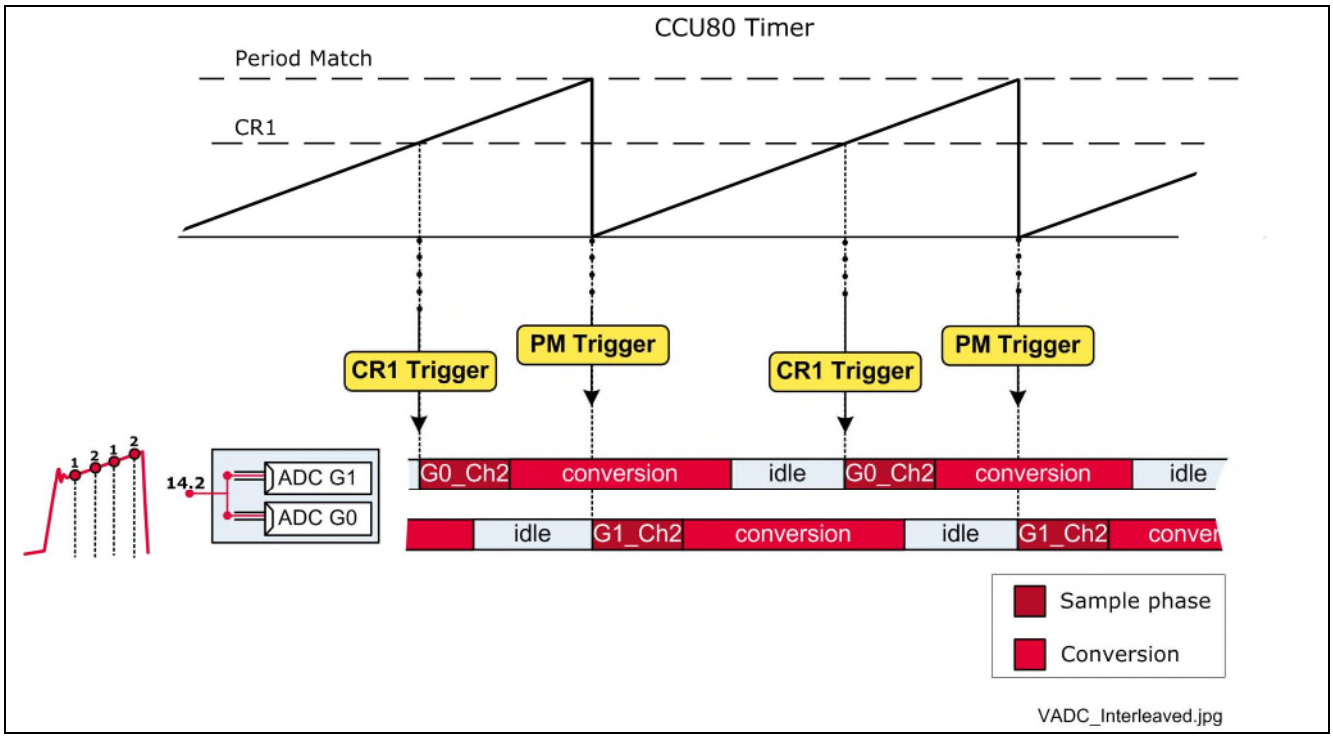

Figure 71 Interleaved conversion of 14.2 Pin

8.1.2 Synchronization through the Trigger

Two or more signals can be sampled at the same time using the same trigger signal to start every conversion.

As an example, the Service Request (SR2) generated by CCU8 for triggering, can be used to start the channel 1 in group 0 and channel 1 in group 1 conversions at the same time.

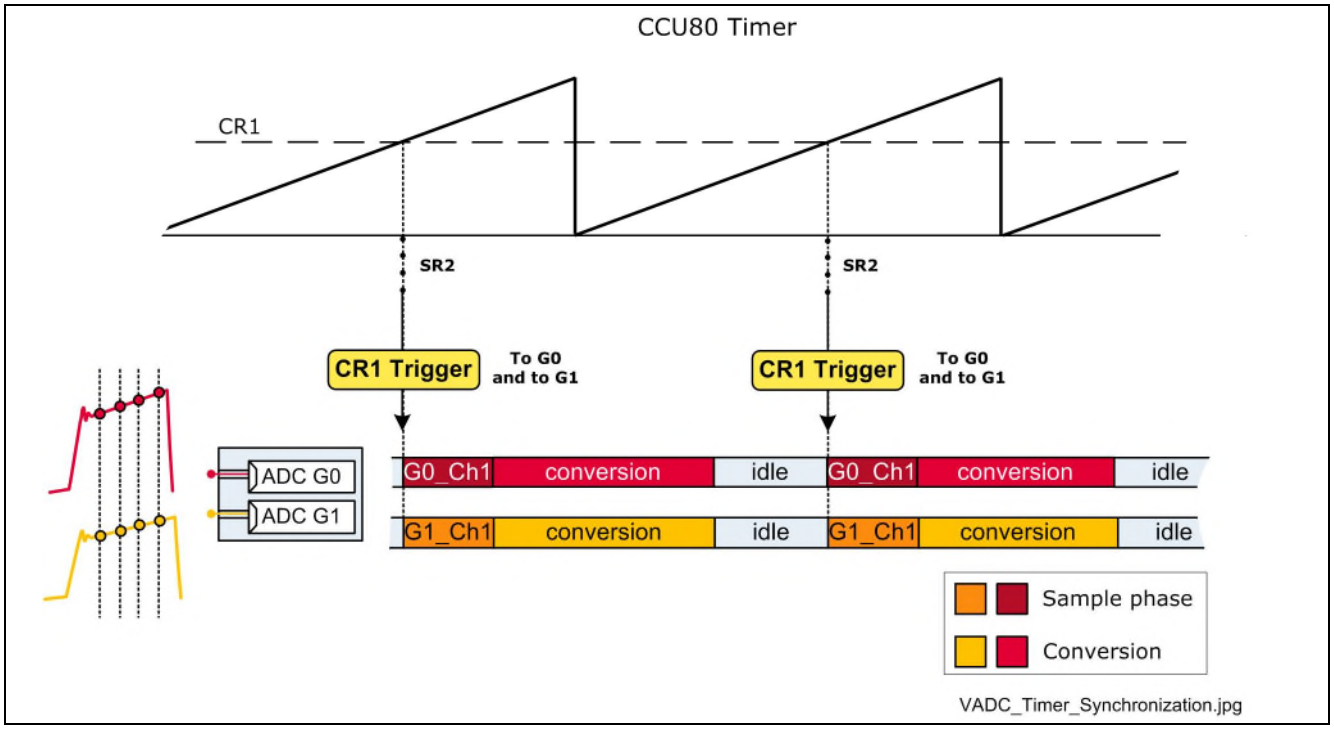

Figure 72 Synchronization through the trigger

8.2 Converting More than 200 Signals: External Multiplexer Control

The number of analog input channels in the VADC of the XMC4000 family of devices can be easily incremented by using external analog multiplexers in each input pin. There are two EMUX Control Interfaces that can select a group separately to automatically control the external multiplexer operation.

In the figure below, more than 200 channels are being converted by switching the EMUX Control Interfaces from group 0 to group 1 (Interface 0), and group 2 to group 3 (Interface 1), and vice-versa, in a hypothetical interrupt.

Emux control signals from Interface 0 are connected to group 0 and group 1 and those from Interface 1, to group 2 and group 3.

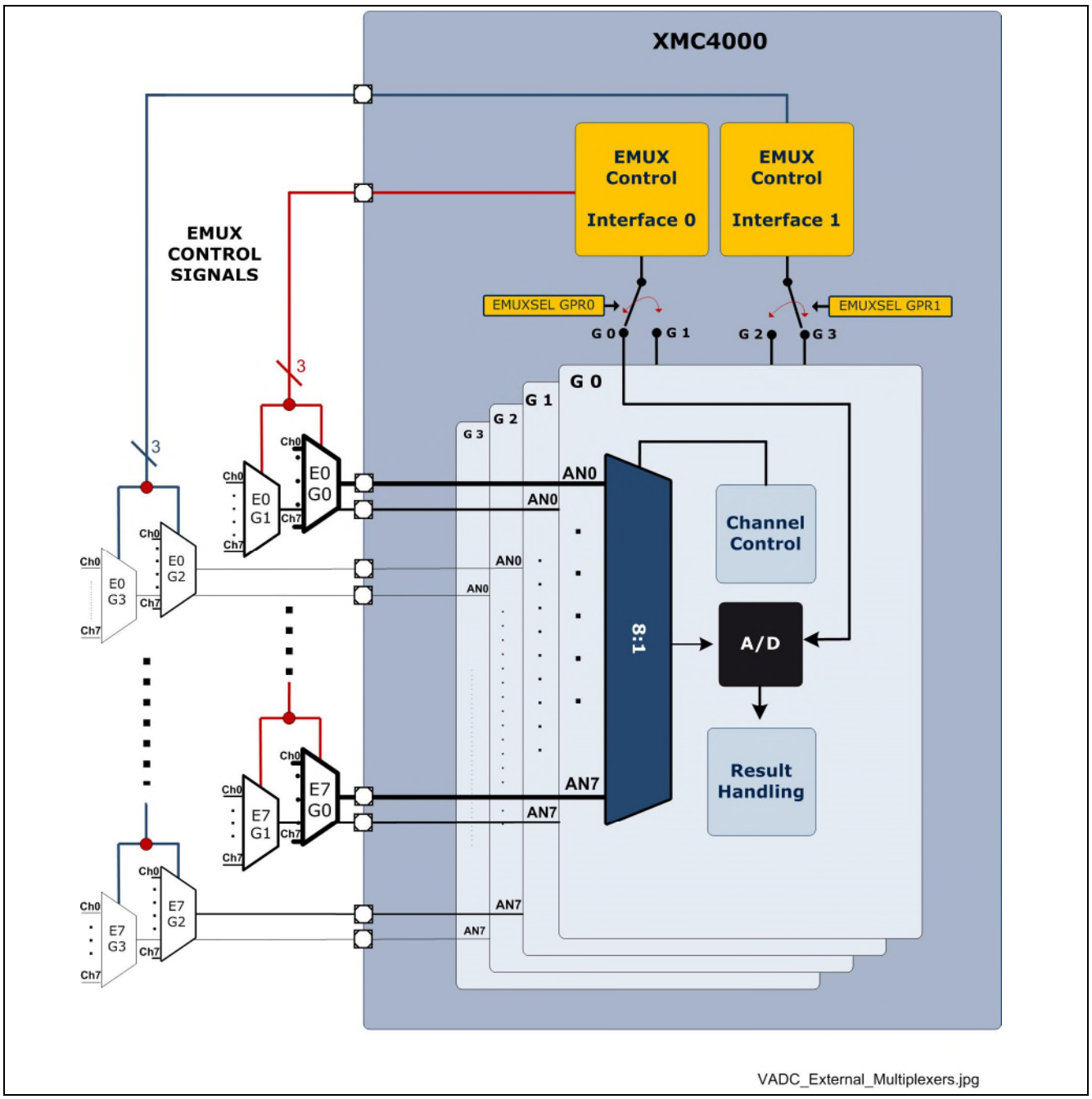

Figure 73 External Multiplexers

In the XMC4100, 4200 and 4400 devices, it is possible to control external multiplexers connected to more than one analog channel of the microcontroller.

In XMC4500 microcontrollers, only one channel can be selected to be controlled. Here the Alias feature is very useful because it allows varying the analog input channel that is being converted without modifying the Channel or EMUX Control configuration.

In the provided code example for XMC4500 microcontrollers, the Alias feature is configured for channel 0 and is being modified from 0 to 8, after every multiplexer sequence. 8 analog inputs are converted per VADC channel, and 8 channels per group make up a total of 64 analog inputs per group.

A result FIFO has been configured in every group for channel 0 to store the 8 results of every multiplexer. When the last result of the FIFO is available, an interrupt is generated. Inside this interrupt, the Alias is changed to convert the next channel.

After 8 interrupts (all the multiplexer's channels have been converted), EMUX Control Interface 0 switches from G0 to G1, and EMUX Control Interface 1 from G2 to G3. The group that is currently converting is disabled (setting the respective Arbitration slot of the request source in the respective group to 0) and the group selected now in the EMUX Control Interface is enabled.

Using this configuration, more than 200 analog channels can be monitored.

Note: The exact number also depends on the available package. Please refer to the specific data sheet for more details.

The flexibility of the VADC together with the options available with the EMUX Control, provide many different ways for converting more channels. Additionally, software and GPIOs can be used to increase the control options.

8.3 Current Measurement in Motor Control

Motor control systems require real-time critical signal measurements such as three phase current measurement. The information contained in the current flowing through the motor coils allows the use of advanced motor control algorithms (i.e. Field-Oriented Control). The simplest method of obtaining the motor current information is to measure the phase shunts directly. Using this method requires at least two sensors.

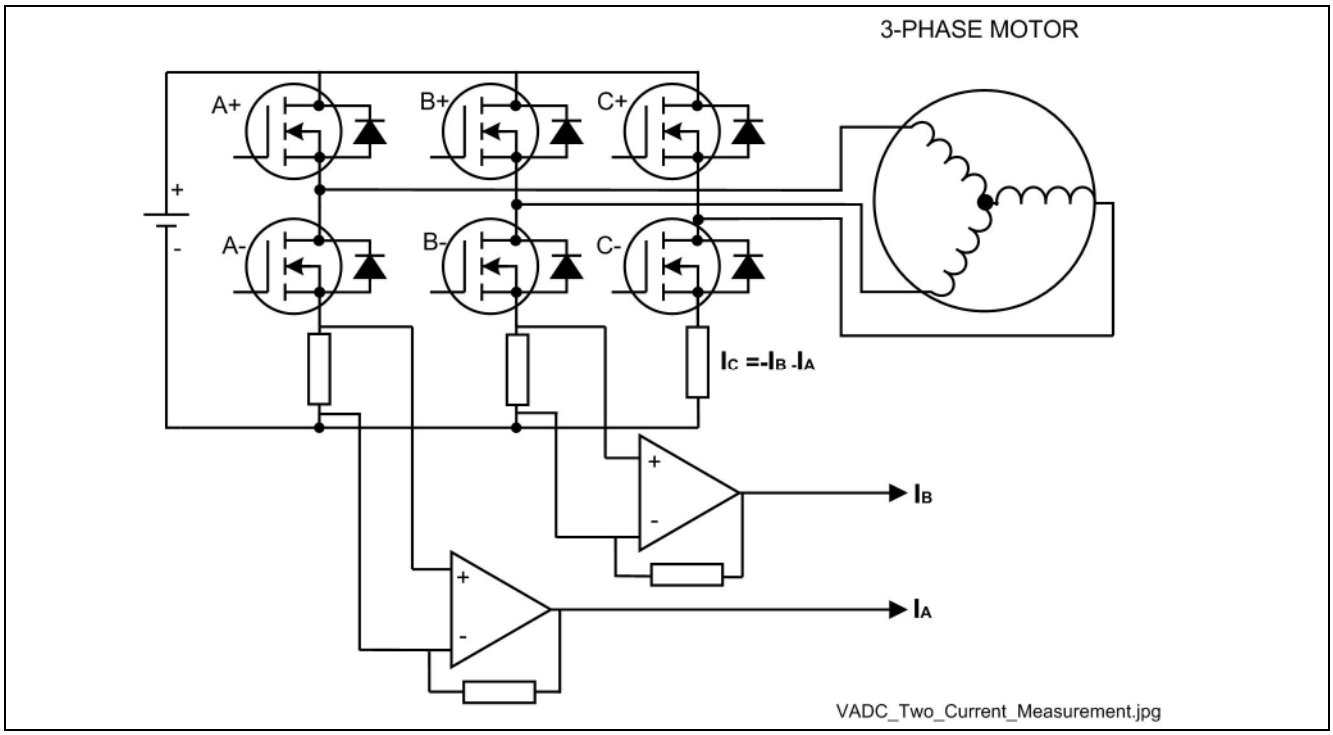

Figure 74 Two current measurements

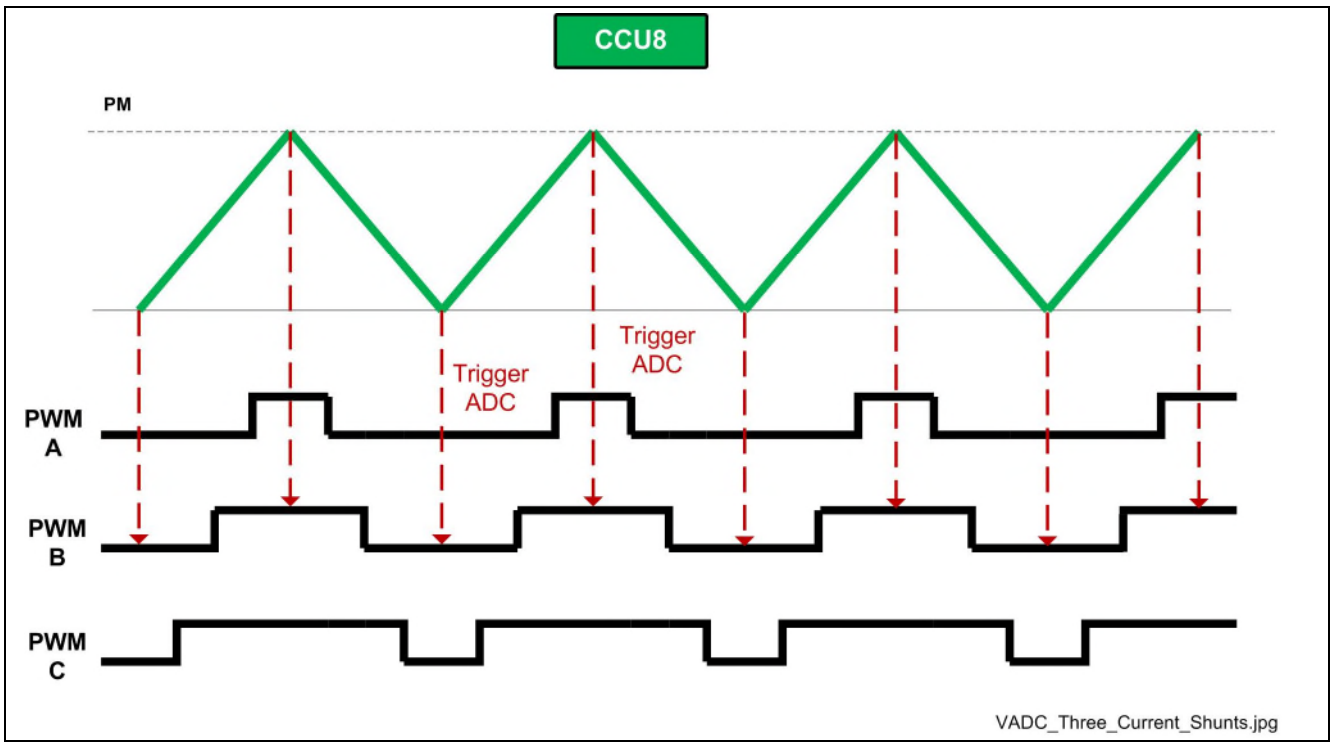

Figure 75 Two shunts measurement, twice per period

There are many different ways to measure motor current. Another common method uses a single shunt resistor for the three current phase's measurement.

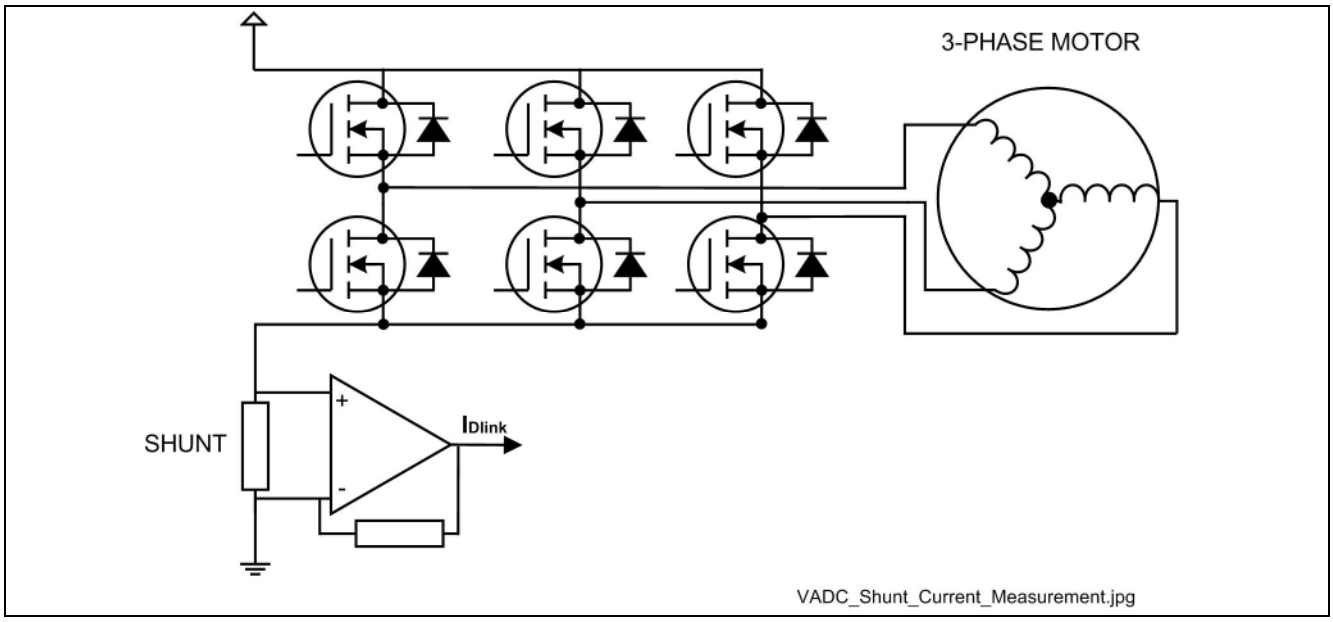

Figure 76 Shunt current measurement

However, this 'cheaper' method can become difficult under certain conditions. Since the current flows through the shunt resistor when one of the bottom transistors is switched on, the ADC sampling must be precisely defined and synchronized with the switching of the inverter's transistors and therefore with the PWM module, in this case the CCU8 timer.

The sample point must be controlled via a hardware trigger, otherwise the measurement may be erroneous, especially when the current pulses are too short to be measured accurately.

The XMC4000 family of products supports these kinds of measurements using time triggered sampling.

The following figure shows an example implementation of a single shunt solution using two ADC Groups triggered by a Capture and Compare Unit 8 (CCU8). This measures twice I_A and I_B per period. This can be used to improve data accuracy. The triggering timer is started after a PWM timer event. In this way a full interaction between the PWM signals and the ADC sampling is achieved.

A delay might be included to overcome the dead-time and switch/propagation delays in the inverter switches.

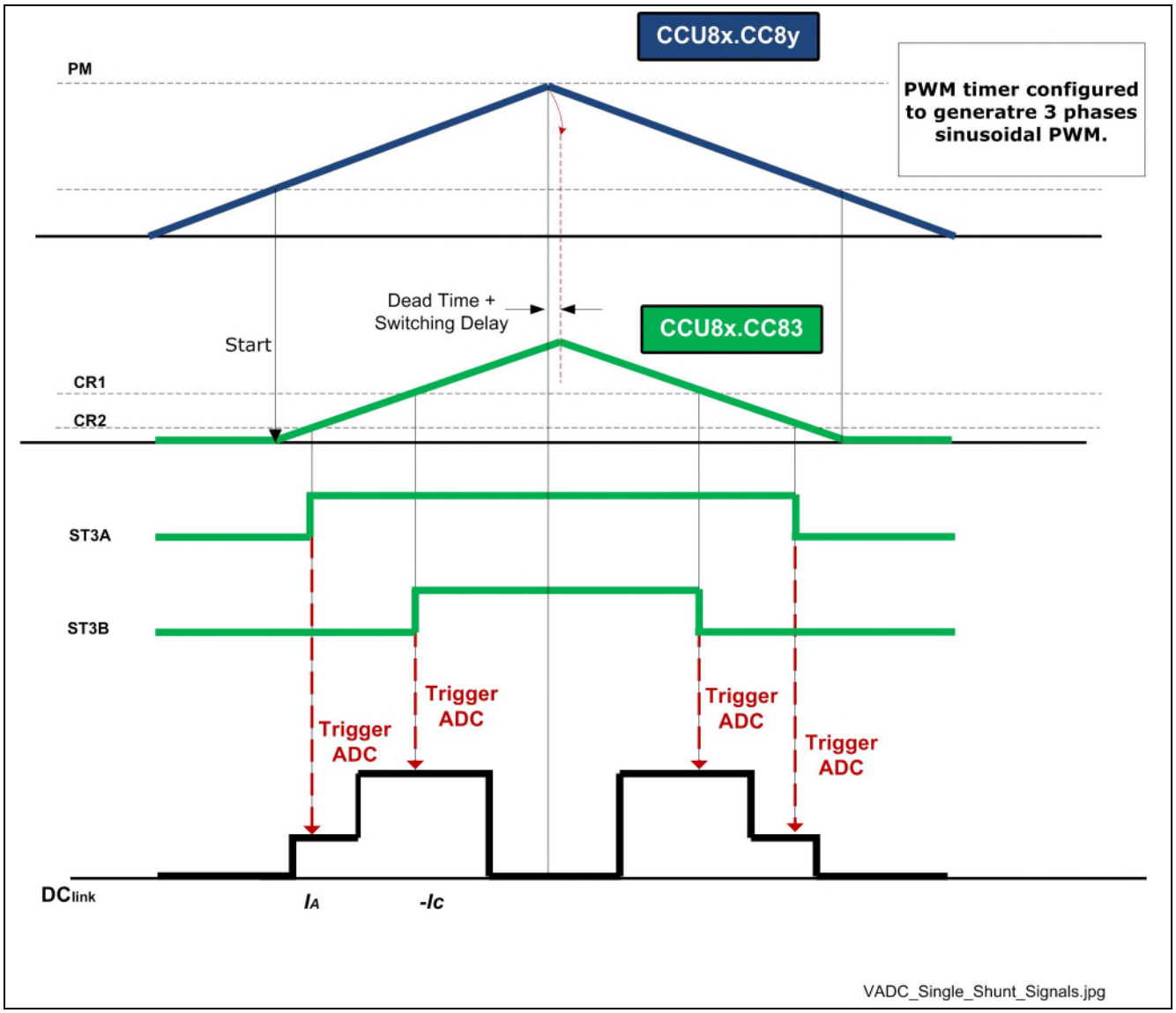

Figure 77 Single Shunt current measurement

8.4 Increasing the ADC Resolution: Oversampling and Averaging

8.4.1 Oversampling and Averaging

Many applications have resolution requirements based on the signal's dynamic range and the Signal-to-Noise Ratio (SNR).

XMC4000 devices have 12-bit resolution ADCs. However it is possible to achieve higher resolution and SNR. The noise caused by quantization error (quantization noise) defines the best SNR of a data converter. Under the correct conditions, oversampling and averaging will reduce this noise and improve the SNR. This will effectively increase the number of bits of resolution.

The following figure shows the Power Spectral Density of the quantization noise with and without oversampling. The quantization noise power can be considered as white noise, and is uniformly distributed across the spectrum between DC and half the sample rate. This quantization noise power is independent of the sample rate. By increasing the sampling frequency (K higher than the Nyquist Frequency), the same Noise power is spread over a bandwidth equal to the sampling frequency which is much greater than the signal bandwidth. The noise power outside the signal band can be easily low-pass filtered (averaged), so the quantization noise is reduced.

Moreover, using a sampling frequency significantly higher than the Nyquist Frequency limit helps to relax the analog anti-aliasing filter constraints.

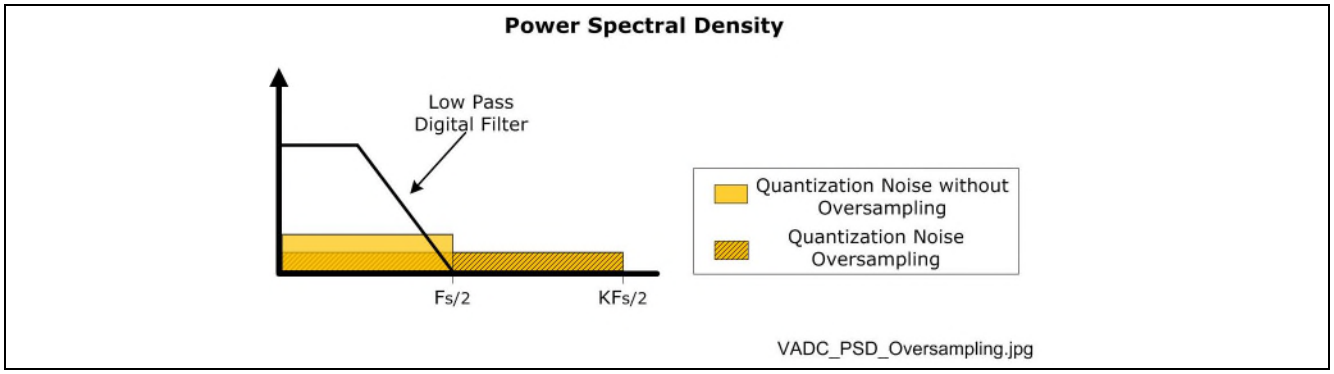

Figure 78 Quantization Noise Power Spectral Density

To increase the effective number of bits, the signal is sampled by the ADC at a frequency that is higher than the system's required sampling rate. The required sampling rate may be determined by how often the system requires a parameter to be measured (output word rate), or may be the Nyquist Frequency of the signal.

The following equation relates the desired number of additional bits of resolution and the oversampling frequency:

$$Fos = 4^b Fs$$

- Fos is the oversampling frequency
- b is the required number of additional bits of resolution
- Fs is the required sampling rate

Because quantization noise depends on the number of bits of resolution of the ADC, the best case SNR is calculated as a function of the number of bits of a data conversion as follows:

$$SNR(dB) \cong (6 \cdot N) + 1.76$$

- N is the effective number of bits of the measurement

This equation is valid for a full-scale input.

If the dynamic range of the input signal does not match the reference voltage of the ADC, the SNR will be lower.

In order to reduce the noise power outside the signal relevant band, the signal can be averaged.

One of the simplest methods of averaging is the decimation. This is just to accumulate $m \cdot (4^b)$ consecutive samples together and then to divide the total to m .

The VADC in XMC4000 devices offers several means of achieving a high sampling rate starting with an analog converter with 2MSamples/secs of sampling frequency. So it will be easy to achieve the desired oversampling resolution and implement the averaging process by software or using the data reduction mode. See Use Cases Result Handling > Data Reduction.

8.4.2 Restrictions

The noise must approximate white noise with uniform power spectral density over the frequency band of interest.

Furthermore, 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. Sometimes even the internal noise of the ADC is sufficient and there is no need for an external noise source.

If a signal is oversampled and averaged to achieve higher resolution, throughput (output data words per unit time) will be reduced by a factor of the oversampling ratio, OSR (OSR=Fs/Fos). There is therefore a tradeoff between resolution and throughput for a given sampling rate.

Another tradeoff is the reduced CPU bandwidth during each sampling period (1/Fos) due to the additional sampling and computations required to achieve the additional resolution.

Note:Oversampling and averaging techniques will not compensate for ADC Integral Non-Linearity (INL).

8.5 Capacitive Touch-Sensing Using Broken Wire Detection

Capacitive sensing is based on measuring a relatively very small variation of the capacitance in the presence of a pressure source (a finger), in a noisy environment. The following figure depicts a typical capacitive sensor implementation in a PCB (Printed Circuit Board).

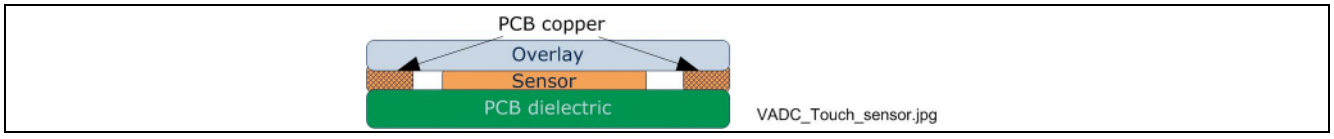

Figure 79 Typical structure of a capacitive touch-sensor

When no conductive object is close to the sensor the capacitance value will be lower than when a conductive object approaches to it. This is due to the variation in the space permeability.

The challenge of capacitive sensing is to detect this relatively small variation and differentiate it from environmental noise.

In the XMC4000 VADC, it is possible to use the Broken Wire Detection feature to detect capacitance variation and, therefore, to implement the capacitive touch-sensing.

Using Broken Wire Detection, the converter's internal capacitor is pre-charged to a selected value (See The ADC Converter chapter, Is it the ADC really sampling?).

As seen in the next figure, if the external capacitance connected to this pin becomes higher (a finger approaches the touch sensor), a large amount of the internal capacitor's charge will be transferred to the external capacitor. The overall voltage level will vary and this difference will indicate the finger approximation. This is easy to recognize. As part of the Broken Wire procedure, the voltage in the capacitor is measured. The variation can be calculated by software.

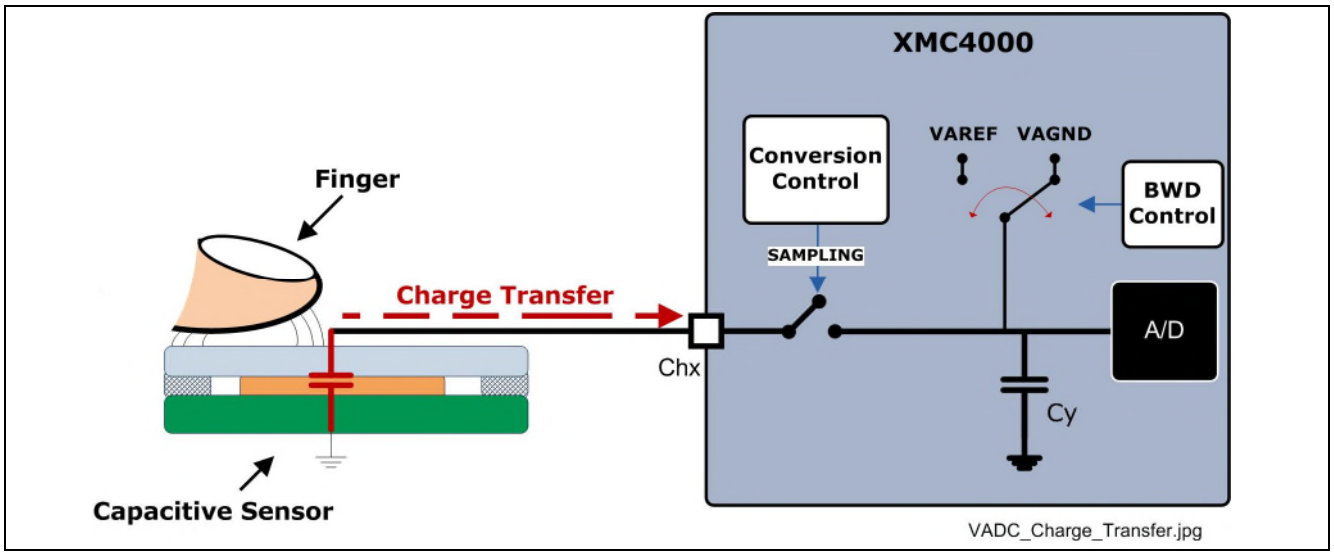

Figure 80 Touch sensing using Broken Wire Detection with the XMC4000 device schema

An example has been developed to show this voltage variation. 8 queue entries have been programmed converting channel 1 of group 1. Every entry has been configured as refill. The first entry has the external trigger enabled and the last entry generates an interrupt.

Depending on the pin voltage, a pre-charge to VAREF or VAGND will generate more prominent spikes and is therefore easier to be detected. So the pre-charged value for the Broken Wire is modified in the interrupt generated at the end of every sequence. A Software Trigger is also made in the interrupt, starting the sequence again after the change of the internal capacitor pre-charged value.

The following figure shows an oscilloscope capture of the voltage signal in the programmed channel.

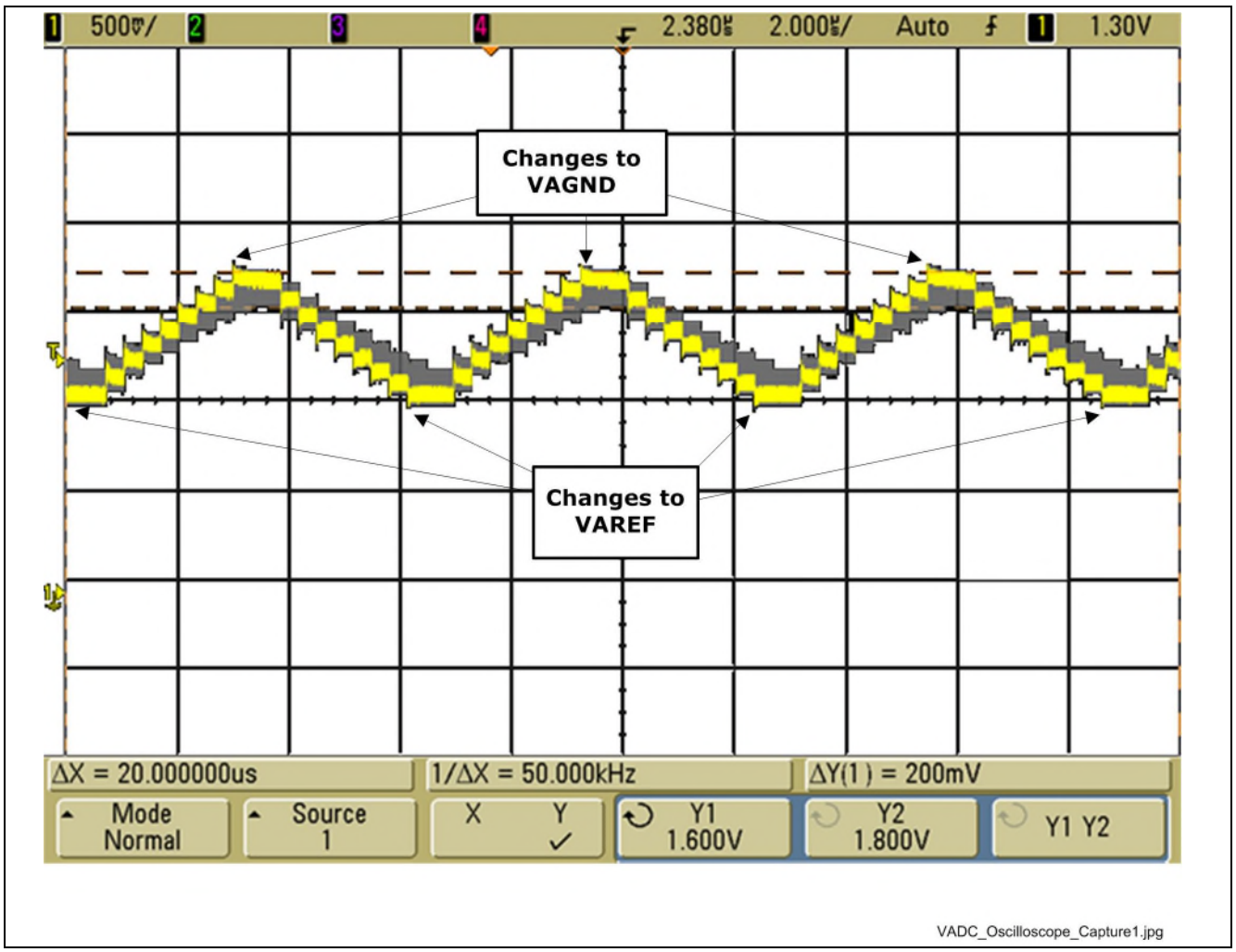

Figure 81 Broken Wire Detection oscilloscope capture

Every rising step is a conversion using Broken Wire Detection with VAREF, and every descending step represents a conversion using Broken Wire Detection with VAGND.

The grey shadow depicts the voltage variation in the channel when the sensor is touched (the external capacitance becomes higher). As can be seen, the voltage difference measured by the oscilloscope's cursors is of approximately 200mV. Detecting and processing this variation of voltage will make it possible to efficiently implement touch-sensing applications.

XMC4000 microcontrollers can of course execute post-processing of this ADC measured data.

8.6 Power Supply Applications: Boost Converter, Peak Current Control/Zero Current Detection

Most electronic devices are powered by a DC power supply that provides regulated and filtered DC voltage.

These power supplies can be voltage or current mode controlled. Due to the many advantages of using this mode, including improved load line regulation, cycle-by-cycle current limiting and protection for example, current mode control is almost a standard.

Peak, valley, average, hysteretic, constant on/off-time, and emulated current-mode techniques are all possible. Nevertheless, the majority of the applications use PCC (Peak Current Control) with slope

compensation. Notable advantages of PCC include automatic input line feed-forward, inherent cycle-by-cycle overload protection, and current sharing capability in multi-phase converters. However, due to the nature of PCC, it has been difficult to implement in a digital architecture due to the requirement of slope compensation, which means additional support circuitry.

Thanks to the high performance of the XMC4000, it is now possible to control successfully and efficiently a complete power supply with a single microcontroller.

The following figure represents an application example with a boost converter topology controlled by a XMC4000 microcontroller whose duty cycle is determined by recourse to the principles of PCC.

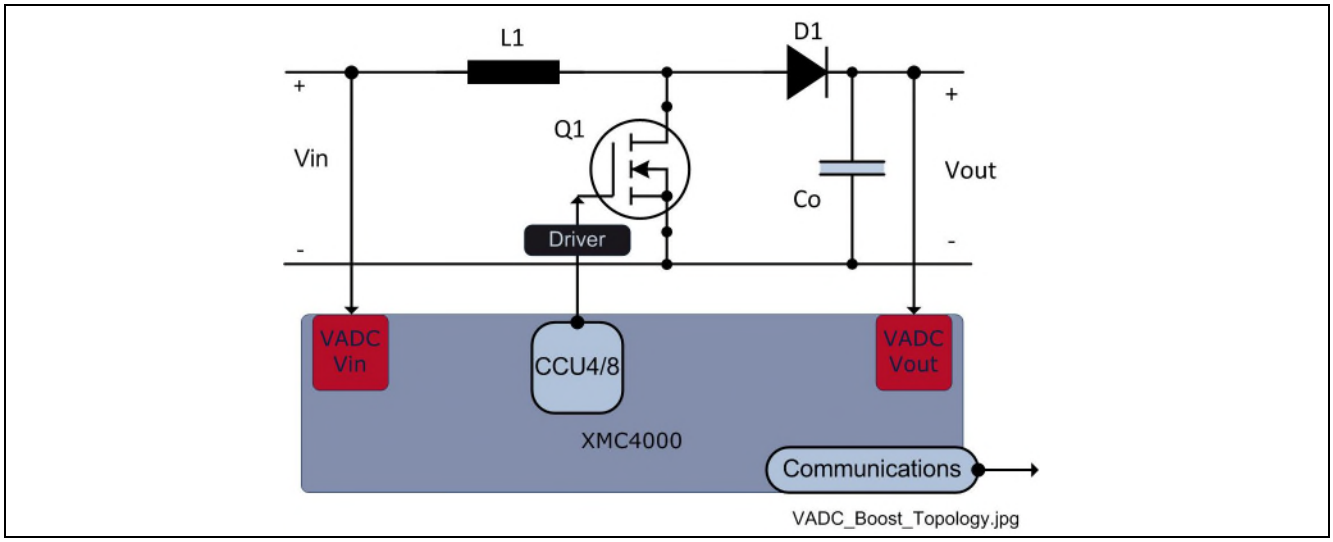

Figure 82 Boost converter

The operating waveforms of this example are depicted in the next figure.

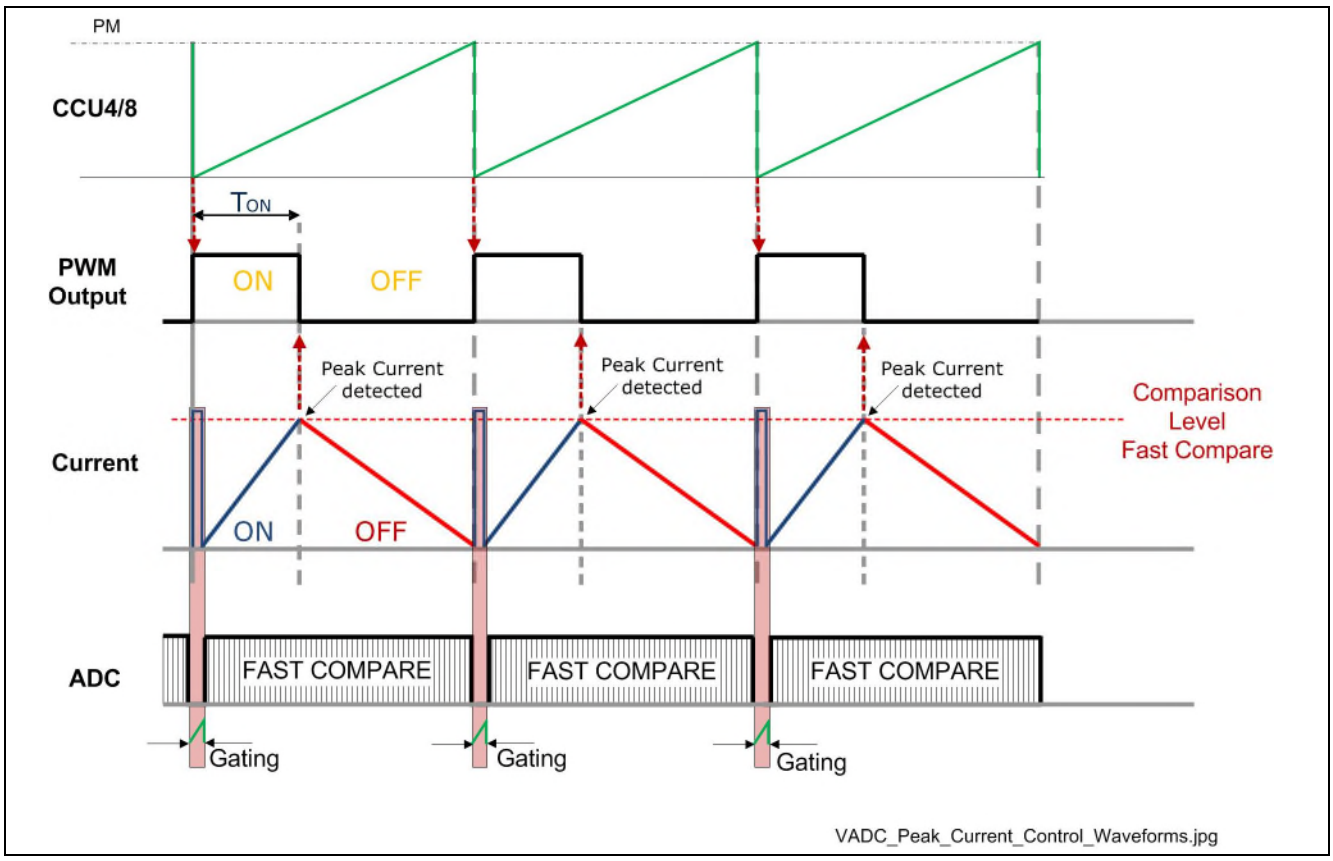

Figure 83 Peak-Current Control example using VADC Fast Compare Mode

The high level of the PWM is being triggered by a Capture and Compare Unit (CCU). In order to detect the peak current, Fast Compare Mode is continuously comparing the current level with the reference value. When the transistor switches on, typically a peak in the I_L appears. This makes a blanking mechanism necessary. Typically this blanking time is implemented by means of using a synchronized-to-the-PWM, CCU4/8 timer that acts as gating for the ADC Fast Compare Mode, at the start of T_{ON} (when the peak appears).

Peak Current detection generates a Boundary Flag that triggers the clear of the PWM output.

The following scheme represents a possible solution for the connections needed to implement this application example.

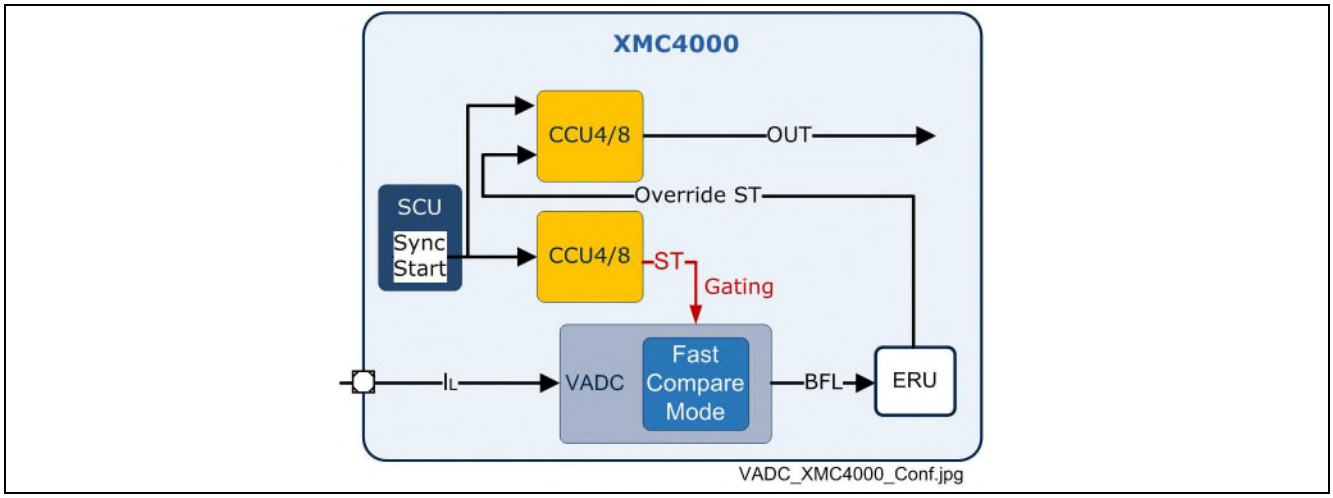

Figure 84 XMC4000 microcontroller configuration scheme for Peak-Current Control (PCC)

The trigger that sets the PWM output can also be generated by detecting the zero current. This is known as Zero Current Detection (ZCD). ZCD will generate a trigger for setting the PWM output instead of using CCU4 or CCU8.

In XMC4100, 4200 and 4400 devices, the zero current can be easily detected through one of the available Analog Comparators.

The Analog Comparator will trigger the ERU after detecting a zero level in the current. Then the ERU will trigger the PWM Output. Of course the VADC can also be used to detect the ZCD, for example, the comparison level can be modified from “Peak-Current” to near to zero after the Peak-Current detection. The comparison level must therefore be modified from near zero to “Peak-Current” when ZCD is sensed in a kind of hysteretic comparison mode. An out of range comparator can also be used for this purpose. See also the 'Use Cases Comparators' chapter.

Slope compensation can be generated in XMC4100, 4200 and 4400 devices thanks to the High Resolution PWM, or with extend components controlled by CCU timers.

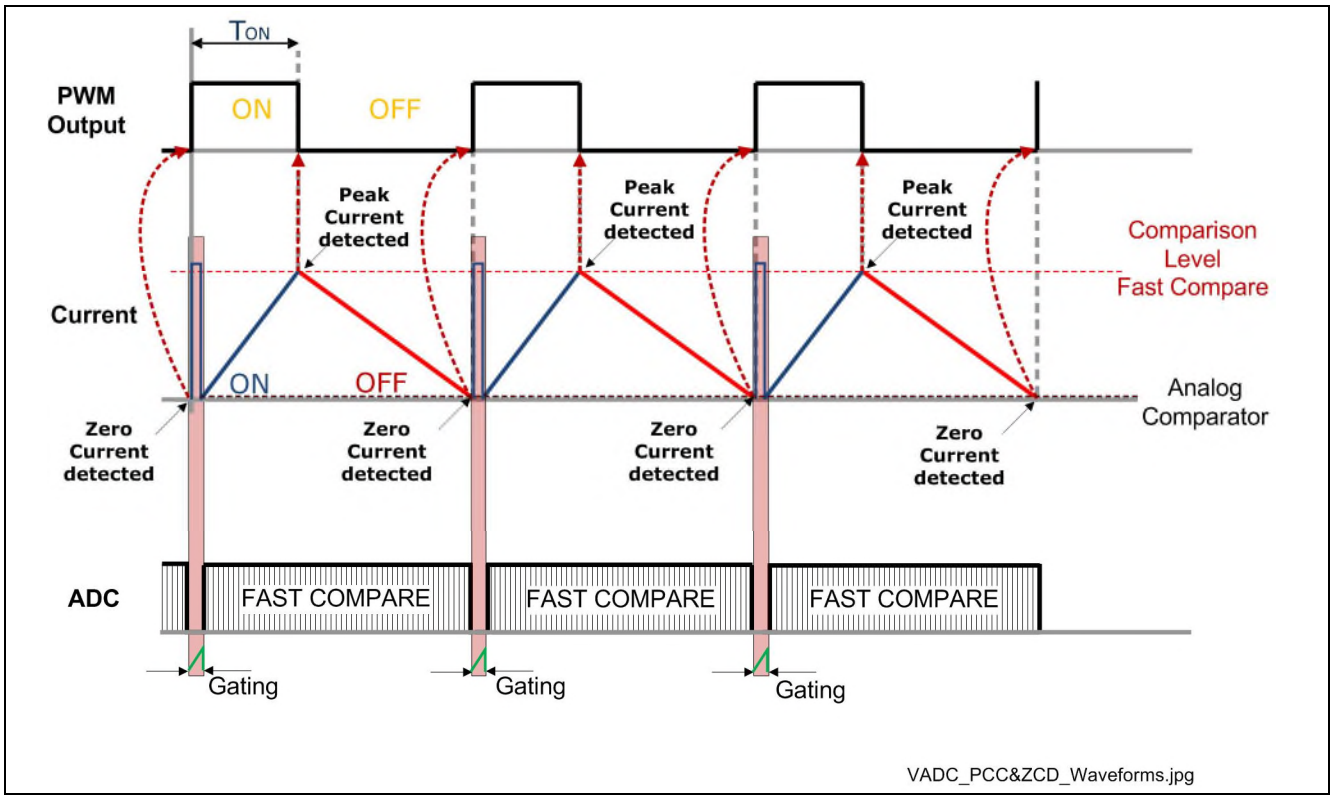

Figure 85 Peak-Current Control (PCC) and Zero-Current Detection (ZCD)

9 Revision History

Current Version is V1.2, 2016-07

Page or Reference	Description of change
V1.0 2015-07	
	Initial Version
V1.1 2015-11	
	Synchronous Conversion example updated.
V1.2 2016-07	
	Update in “Result Register FIFO” and “Synchronous Conversion”

www.infineon.com

Edition 2016-07

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

AP32305