Application Note, V1.0, September. 2008

AP16150

XC2000/XE166 family

Understanding the XC2000/XE166 ADC

Microcontrollers

Never stop thinking

Edition 2008-09-11

Published by
Infineon Technologies AG
81726 München, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP16150		
Revision History:	2008-09	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of Contents **Page**

1 Introduction

1.1 Overview

This application note provides some detailed information about the key functional units of the Analog-to-Digital converter module in the XC2000/XE166 family of 16-bit Microcontrollers. The functional description and configuration options of the ADC module are discussed together with examples that give a good insight on the usage of the module.

1.2 Background

In order to meet the real time needs of applications, an autonomous ADC module which offloads the CPU is required. The XC2000/XE166 family of microcontrollers offers a specialized ADC that best suits the needs of the automotive and industrial applications. This ADC module will work autonomously with its specialized hardware thereby reducing CPU load. Hence the CPU can be utilized for other major tasks.

The ADC module contains 2 independent kernels (ADC0, ADC1) that can operate autonomously or can be synchronized to each other. An ADC kernel is a unit used to convert an analog input signal into a digital value and provides means for triggering conversions, data handling and storage.

With this structure, parallel conversion of up to two analog input channels is supported.

1.3 Features of the ADC module

The XC2000/XE166 family of microcontrollers includes 10-bit Analog-to-Digital Converter (ADC) with sixteen multiplexed analog input channels. The ADC uses a successive approximation technique to convert the analog voltage levels from up to eight different sources.

Features of the ADC module include:

- 3 conversion request sources for external or timer-driven events, auto-scan, programmable sequences, SW-driven conversions, etc.
- Synchronization of the ADC kernels for concurrent conversion starts and parallel sampling and measuring of analog input signals, e.g. for phase current measurements in AC drives.
- Control capability for an external analog multiplexer, respecting the additional set up time.
- Adjustable sampling times to accommodate output impedance of different analog signal sources (sensors, etc.).
- Possibility to cancel running conversions on demand with automatic restart.
- Flexible interrupt generation (possibility of PEC support).
- Limit checking to reduce interrupt load (e.g. for temperature measurements or overload detection, only values exceeding a programmable level lead to an interrupt).
- Programmable data reduction filter, e.g. for digital anti-aliasing filtering, by adding a programmable number of conversion results.
- Independent result registers (8 independent registers).
- Support of conversion result FIFO mechanism to allow longer interrupt latency.
- Support of suspend and power saving modes.
- Individually programmable reference selection for each channel, e.g. to allow measurements of 3.3 V and 5 V signals in the full measurement range with the same ADC kernel.

2 Working principle of the ADC module

2.1 Functional representation of the ADC module

In XC2000/XE166 family of 16-bit microcontrollers, the ADC module consists of two ADC kernels (ADC0 & ADC1) that can operate autonomously or can be synchronized to each other. A block level representation of the ADC module is shown in Figure 1.

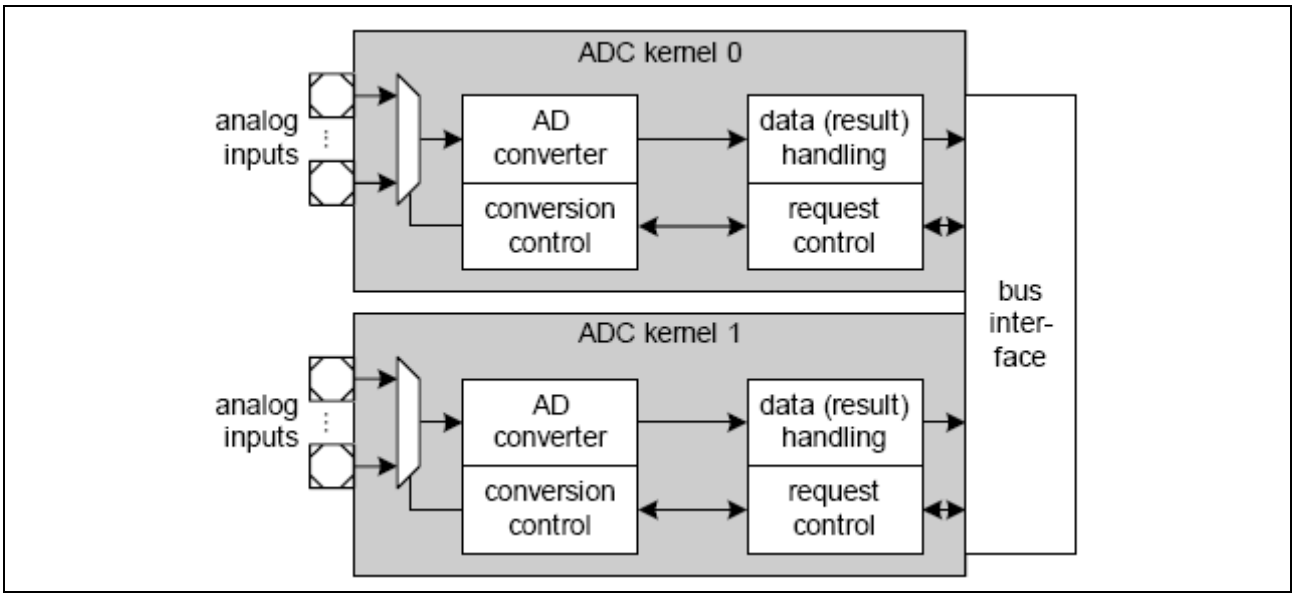

Figure 1 ADC kernel block diagram

Each ADC Kernel comprises:

- An **analog to digital converter** with a maximum of 16 analog inputs (CH0 - CH15). This block selects an input signal and translates the analog voltage into a digital value.
- A **conversion control** unit defining the conversion parameters like, the length of the sample phase, the resolution and the reference for each conversion. The length of the sample phase and the resolution values are similar for several channels and, therefore, are grouped together to form the so-called input classes. Each channel can be individually assigned to an input class to define these parameters. The conversion control also handles the start conditions for the conversions, such as the immediate start (cancel-inject-repeat), overwrite of former results (wait-for-read), or synchronization of the ADC kernels (parallel conversions).
- A **request control** unit defining which analog input channel has to be converted next. It contains 3 request sources that can trigger conversions depending on different events, such as edges of PWM or timer signals or events at port pins. Each request source can trigger either 1, up to 4, or up to 16 conversions in a sequence.
- A **result handling** unit providing 8 result registers for the conversion results. The result handling block also supports data reduction (e.g. for digital anti-aliasing filtering) by automatically adding up to 4 conversion results before informing the CPU that new data is available. Additionally, the results registers can be concatenated to FIFO structures to provide storage capability for more than one conversion result without overwriting previous data. This feature also helps to handle CPU latency effects.
- An **interrupt generation** unit issuing interrupt requests to the CPU depending on ADC events. The interrupt generation in the ADC kernels support different mechanisms, e.g. some interrupts can be coupled to a value range of the conversion result (limit checking), some interrupts can be used to transport conversion data to locations in memory for further treatment, and other interrupts are generated after a complete sequence of conversions.

2.2 ADC clocking scheme

The ADC Kernel modules are driven by clock signals that are based on the clock f_{ADC}. The clocking scheme of the ADC kernel is shown in Figure 2.

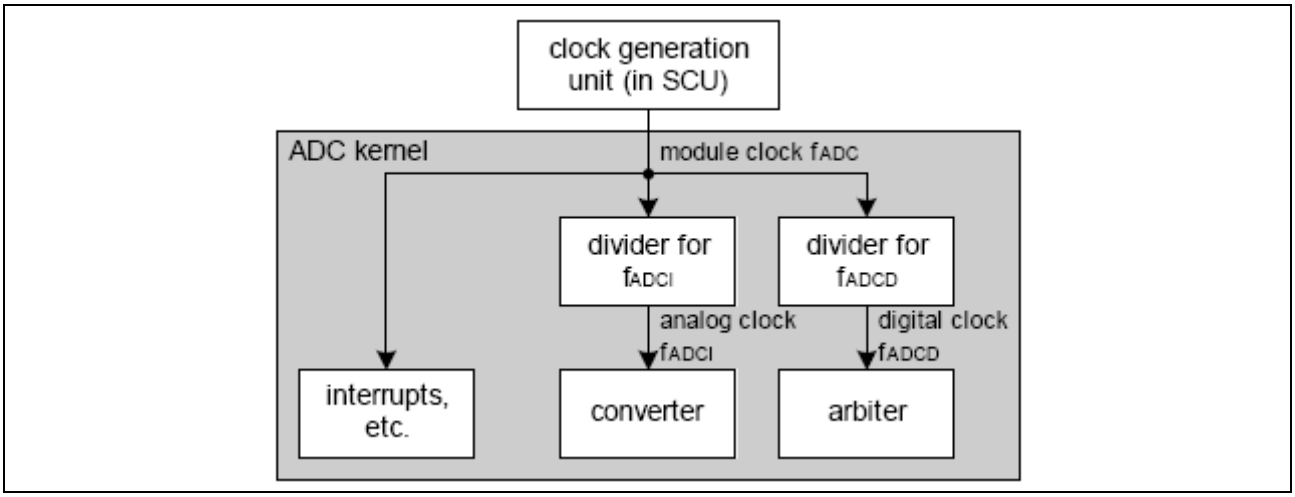

Figure 2 ADC clocking Scheme

Like other modules, ADC uses the system clock for its operation.

There is no divider for the ADC clock selection. Hence if the ADC module is selected, then $f_{ADC} = f_{SYS}$.

The ADC module uses two internal clock signals f_{ADCI} (used for converter) and f_{ADCD} (used for arbiter). All other digital structures (such as interrupts, etc) are directly driven by the module clock f_{ADC}.

2.2.1 Analog clock generation for converter

The analog-to-digital converter uses the internal analog clock f_{ADCI} which defines the conversion length and sample time. This can be adjusted by the bitfield DIVA in the GLOBCTR register. This is shown in Figure 3.

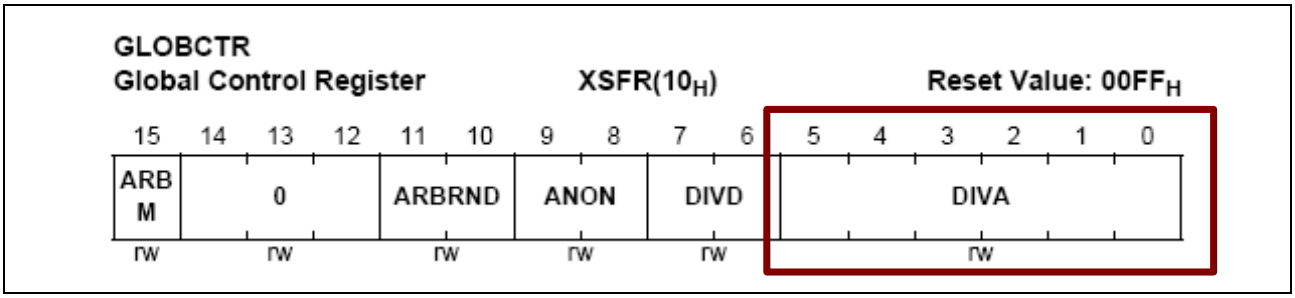

Figure 3 f_{ADCI} analog clock adjustment

The bitfield DIVA defines the number of f_{ADC} clock cycles to generate the f_{ADCI} clock for the converter. The configuration details of DIVA will result:

00_H $f_{ADCI} = f_{ADC}/$

01_H $f_{ADCI} = f_{ADC}/2$

02_H $f_{ADCI} = f_{ADC}/3$

…. ……

$3F_H$ $f_{ADCI} = f_{ADC}/64$

2.2.2 Digital clock generation for arbiter

The arbiter uses the digital clock f_{ADCD} which defines the duration of an arbiter round. This can be adjusted by the bitfield DIVD in the GLOBCTR register. This is shown in Figure 4.

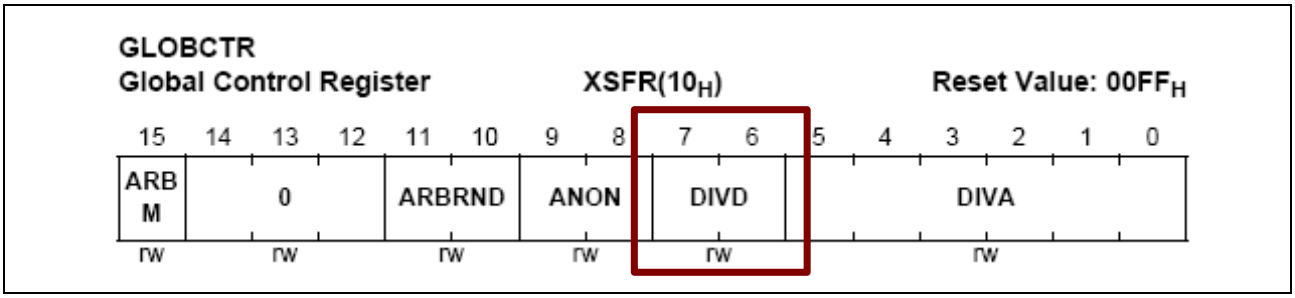

Figure 4 f_{ADCD} digital clock adjustment

The bitfield DIVD defines the number of f_{ADC} clock cycles within each arbitration slot.

The configuration details of DIVD will result:

00_B $f_{ADCD} = f_{ADC}/$

01_B $f_{ADCD} = f_{ADC}/2$

10_B $f_{ADCD} = f_{ADC}/3$

11_B $f_{ADCD} = f_{ADC}/4$

2.3 Simplified ADC conversion process

The simplified Analog to Digital Conversion process is shown in Figure 5.

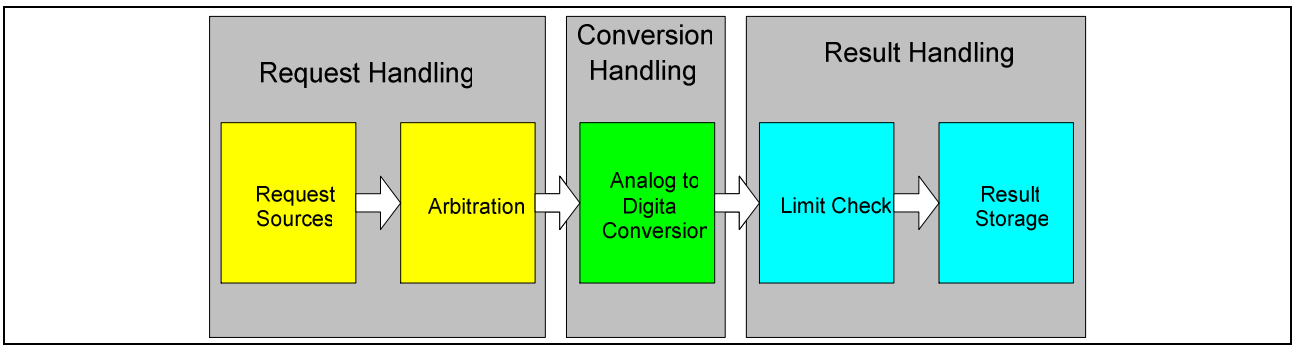

Figure 5 Simplified ADC conversion process

For simplicity we say, The ADC process involves three phases namely Conversion Request Phase, Conversion Phase and Result Handling Phase.

The functional units of the ADC kernel along with the conversion process are explained in the following sections.

3 Conversion request phase

The conversion request phase handles the processing of the conversion requests. Arbiter and the conversion request sources are mainly involved in this phase.

3.1 Conversion request sources

In **XC2000/XE166** family of microcontrollers, each ADC kernel has three request sources namely, **Request source 0**, **Request source 1** and **Request source 2**. The request source 0 and request source 2 are of sequential source type whereas request source 1 is of scan (parallel) source type. Channel conversions are requested through these sources and are shown in Figure 6.

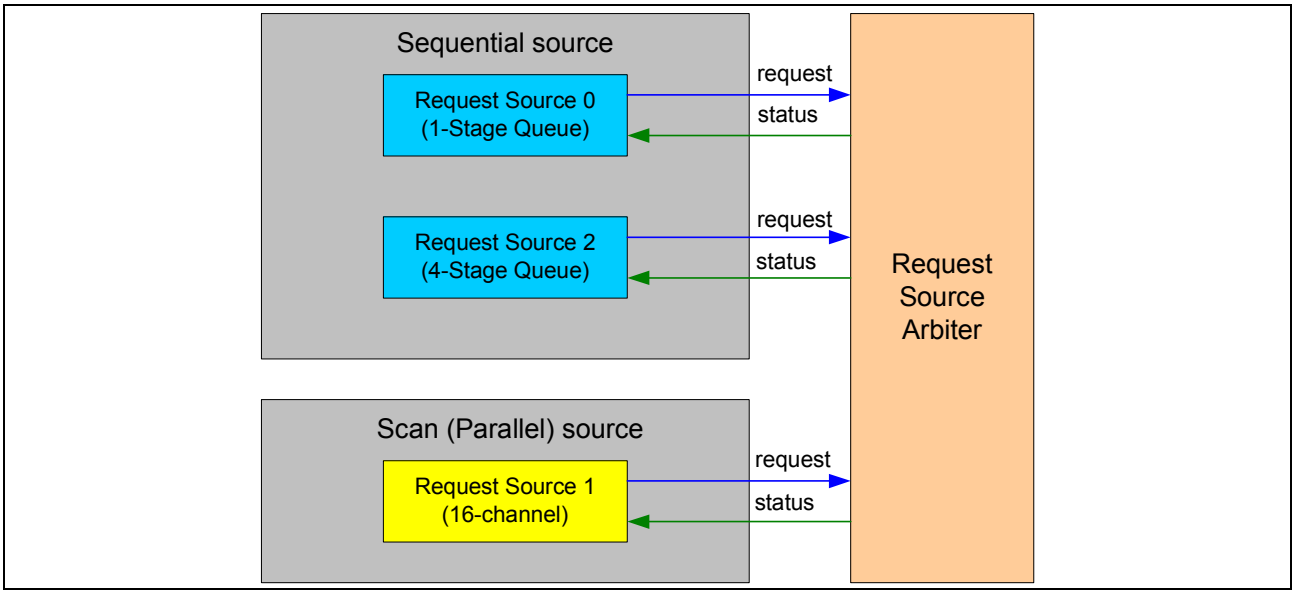

Figure 6 Conversion request sources

In **Sequential request source**, the requests are handled in a queued manner i.e., requests are processed the order in which they are entered in queue. In **Scan request source** the requests are handled as a batch i.e., requests are processed for the arbitrary set of channels selected by the user starting at the highest numbered channel to the lowest numbered channel.

These request sources can be prioritized, gated and triggered based on application needs. Each request source defines the analog input channel to be converted if a defined event occurs. For example, a trigger pulse from a timer unit generating a PWM signal can start the conversion of a single input channel or a programmed sequence of input channels.

Request source 0 (1-stage sequential source):

This source can issue a conversion request for a single input channel. The channel number can directly be programmed. This mechanism could be used for SW-controlled conversion requests or HW triggered conversions of a single input channel.

Request source 1 (16-channel scan source):

This source can issue conversion requests for a sequence of up to 16 input channels. It can be programmed which channel takes part in this sequence. The sequence always starts with the highest enabled channel number and continues towards lower channel numbers (order defined by the channel number, each channel can be converted only once per sequence).

Request source 2 (4-stage sequential source):

This source can issue a conversion request for a sequence of up to 4 input channels. The channel numbers can be freely programmed, especially multiple conversions of the same channel within the sequence are supported.

3.2 Request source arbiter

The request source arbiter decides which analog channel has to be converted. Therefore, it regularly polls the request sources one after the other for pending conversion requests. The polling sequence is based on time slots with programmable length, arbitration slots. If a request source is disabled or if no request source is available for an arbitration slot, the slot is considered being empty and has no influence on the evaluation of the arbitration winner.

The arbitration round is shown in Figure 7.

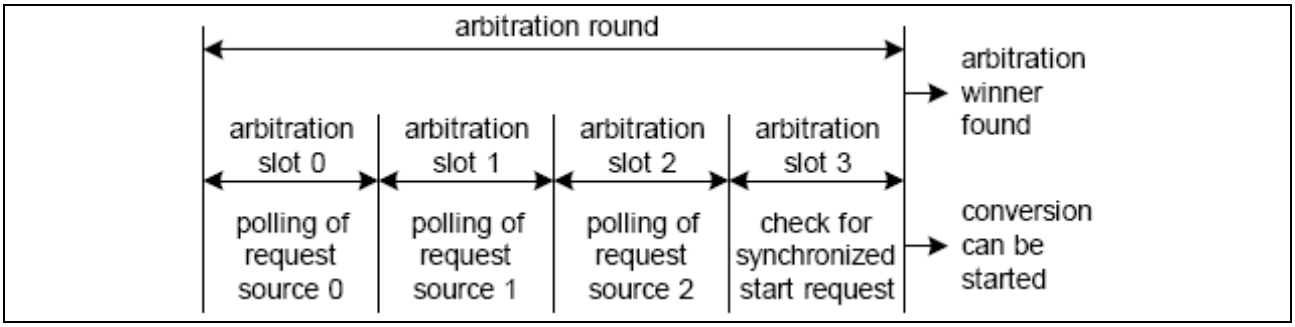

Figure 7 Arbitration round

The mapping of arbitration slots with the request sources are as follows:

Arbitration slot 0 Request source 0 (1-Stage queue)

Arbitration slot 1 Request source 1 (16-channel parallel)

Arbitration slot 2 Request source 2 (4-Stage queue)

Arbitration slot 3 Synchronization source

In arbitration slot 3, the arbiter checks for a synchronized request from another ADC kernel and does not evaluate any internal request source. A request for a synchronized conversion is always handled with the highest priority in a synchronization slave kernel (pending requests from other sources are not considered).

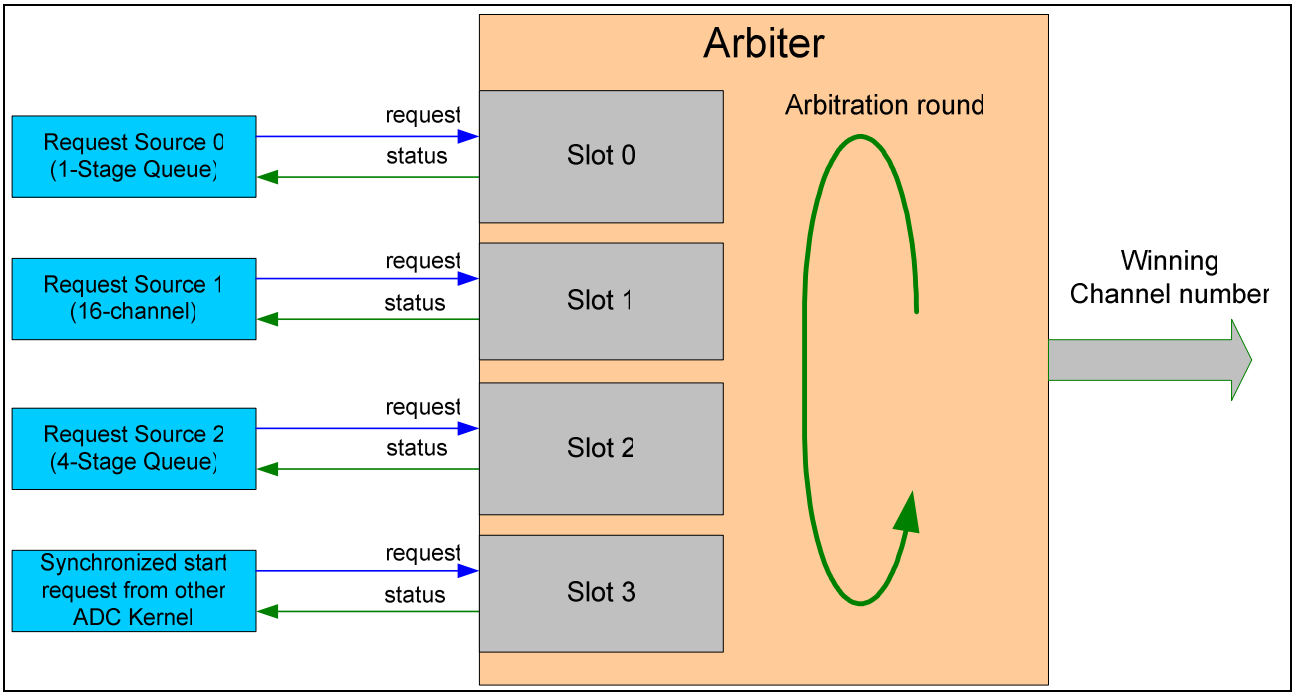

Figure 8 ADC request processing

3.3 Sequential request source handling

Sequential request sources are useful for short conversion sequences as the channels for conversion can be programmed in any order. Two versions of sequential request sources are available in each ADC kernel:

Request source in arbitration slot 2: (Request source 2)

This request source can handle a sequence of up to 4 input channels (4-stage queue for 4 entries). This mechanism could be used to support application-specific conversion sequences, especially for timing-critical sequences containing multiple conversions of the same channel.

Request source in arbitration slot 0: (Request source 0)

This request source can handle a single input channel (1-stage queue for 1 entry). This mechanism could be used for SW-controlled conversion requests or HW triggered conversions of a single input channel (to “inject” a single conversion into a running sequence).

The functional representation of the sequential request source is shown in Figure 9.

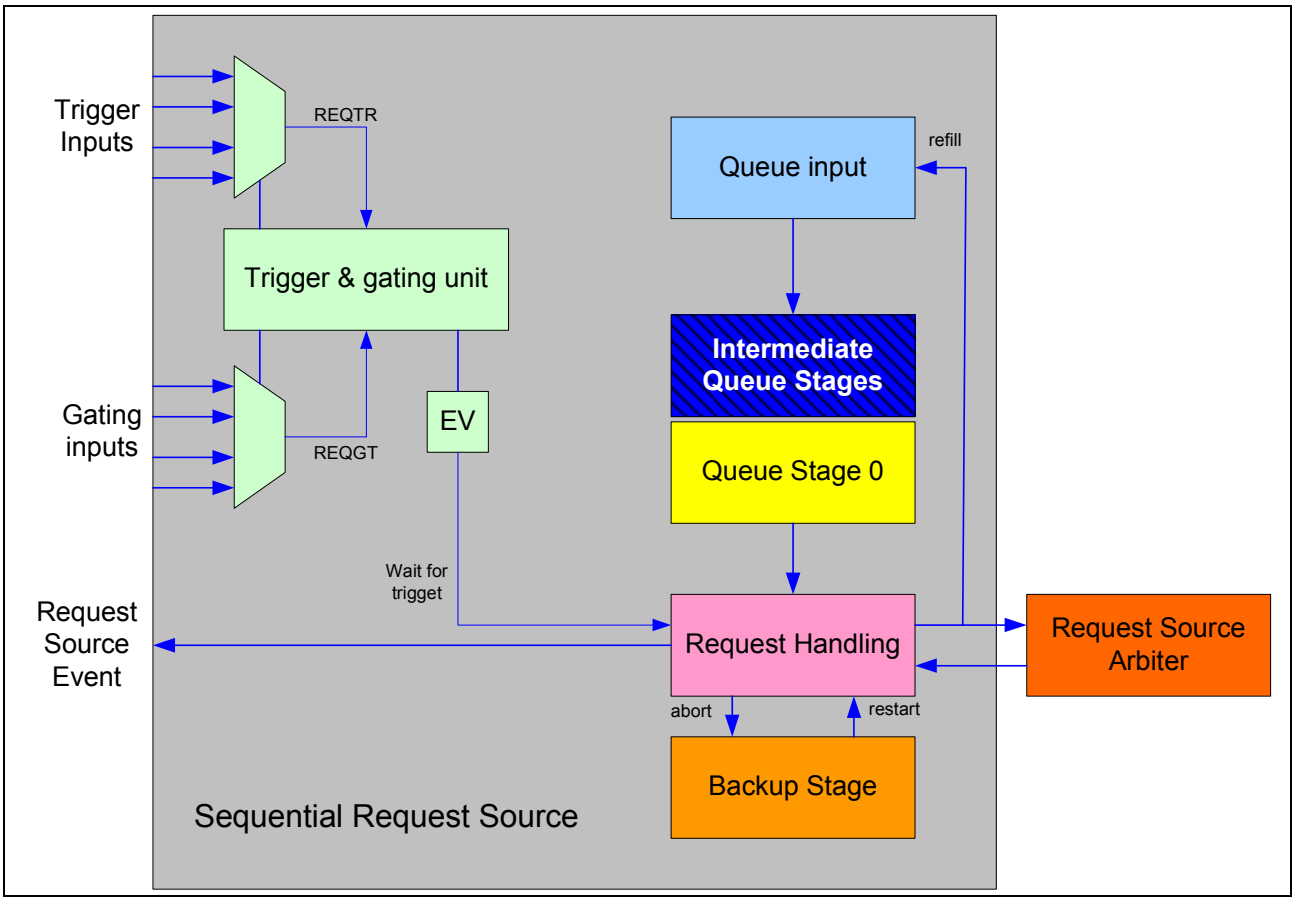

Figure 9 Sequential request source

The only difference between both versions is given by the number of intermediate queue stages for storing the sequence. The request source in arbitration slot 0 does not provide intermediate queue stages (1-stage queue with only queue stage 0), whereas the one in arbitration slot 2 provides 3 intermediate queue stages in additional to queue stage 0 (leading to a 4-stage queue). But the internal structure and the handling of sequential sources are similar for both versions.

The four queue stage of request source 2 is shown in Figure 10.

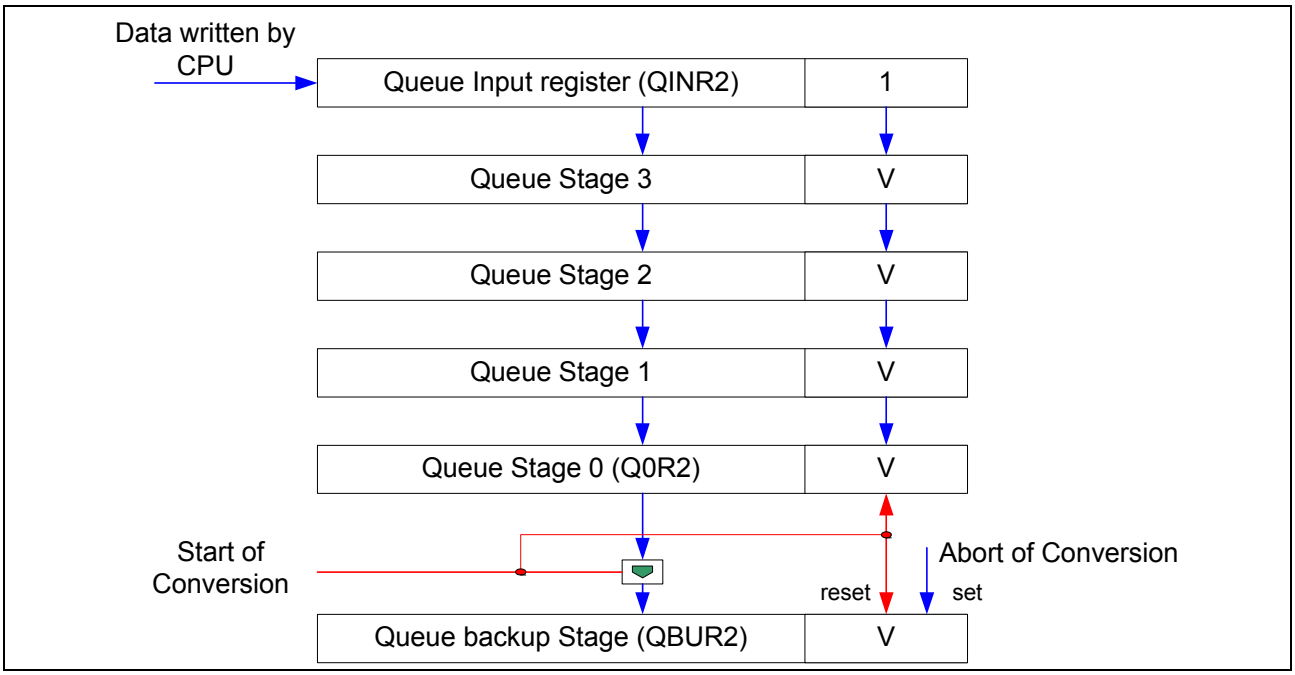

Figure 10 Queue stage of request source 2

The single stage queue of request source 0 is shown in Figure 11.

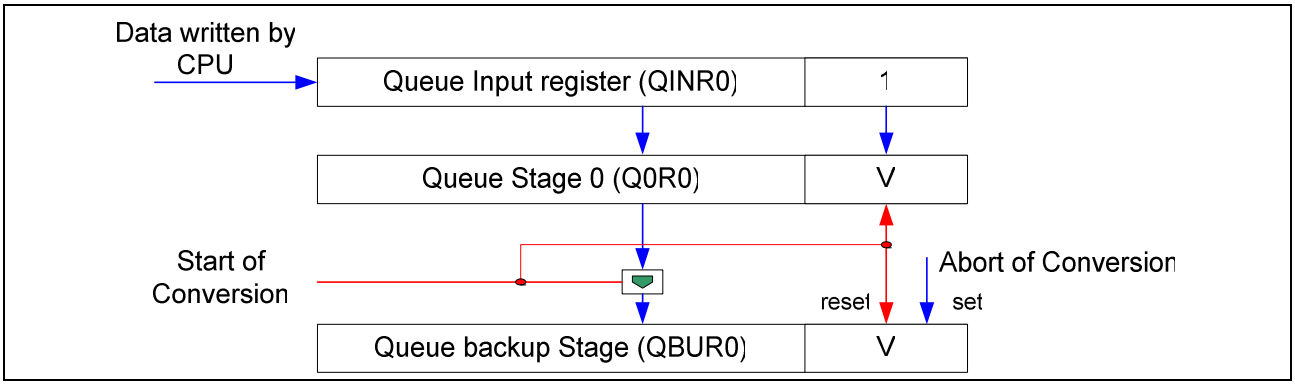

Figure 11 Queue stage of request source 0

The sequential request source can take part in the source arbitration if the backup stage (contains previously aborted conversion if any) or queue stage contains a valid request (V = 1). This is shown in Figure 12.

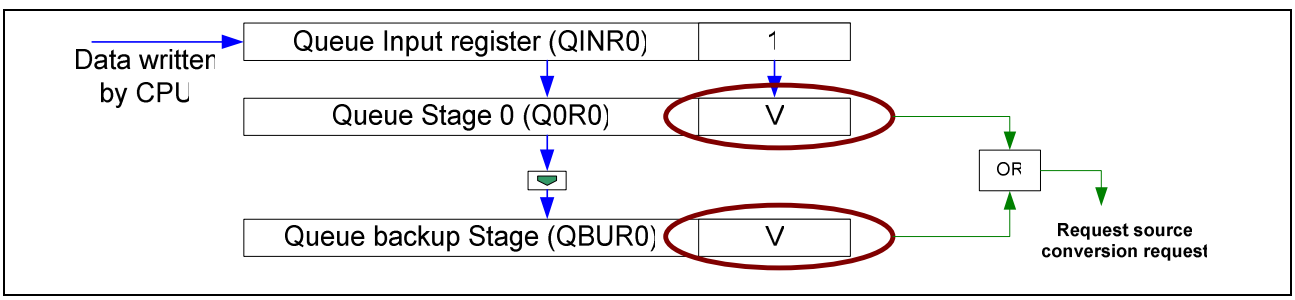

Figure 12 Request source conversion request

The functionality remains the same for both request source 0 and 2. Hence the following explanations are valid for both the sources.

The possible cases are described here:

Case 0: Both Queue stage and backup state contains V = 0.

In this case no conversion is requested and the request source will not take part in the arbitration.

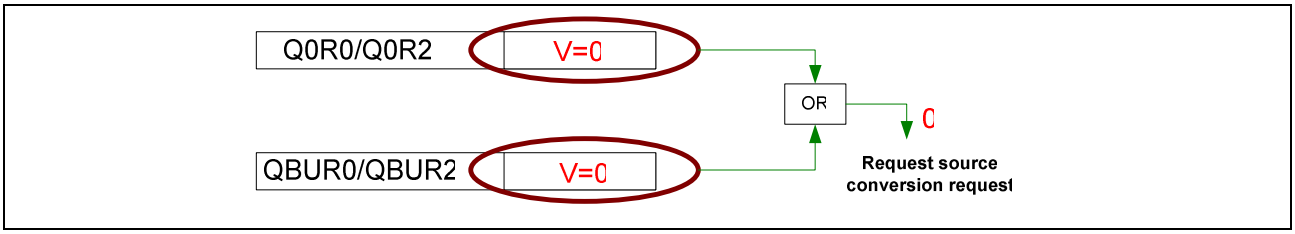

Figure 13 No conversion request as V=0 for Q0R0 and QBUR0

Case 1: Queue stage contains valid bit V = 1 and backup stage contains V = 0.

In this case conversion is requested for the queue stage 0 and the request source will take part in arbitration.

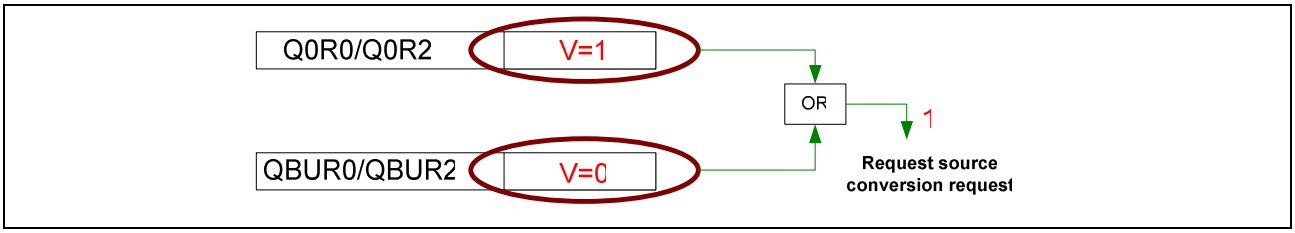

Figure 14 Sequential conversion request as V = 1 for queue stage

When the request conversion is started, the bit V is queue stage is reset.

Case 2: Backup stage contains valid bit V = 1 and queue stage contains V = 0.

In this case conversion is requested for the queue stage 0 and the request source will take part in arbitration.

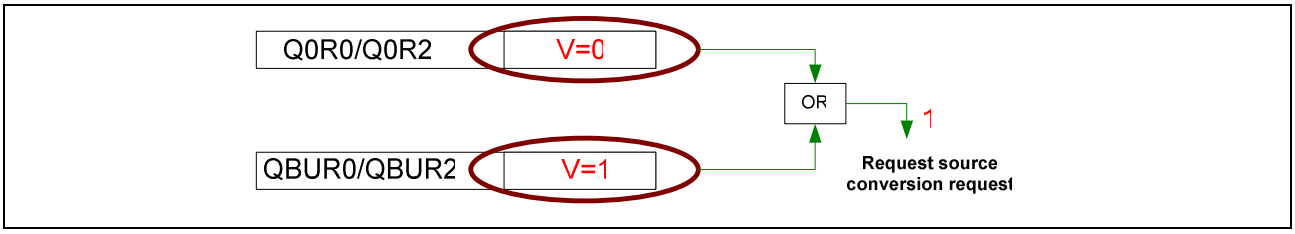

Figure 15 Sequential conversion request as V = 1 for backup stage

When the request conversion is started, the bit V is backup stage is reset.

Case 3: Both queue stage and backup stage contains valid bit V = 1.

In this case the aborted request in the backup stage is treated before the entries in the queue stage. Hence the request source will take part in arbitration for the aborted conversion.

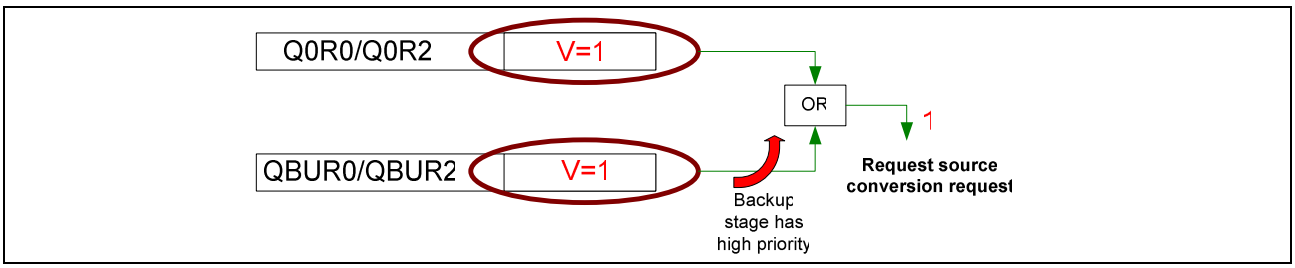

Figure 16 Sequential conversion request for as V=1 for both Q0R0 and QBUR0

Only the valid bit V=1 in the backup stage is cleared when conversion is started.

3.3.1 Refill mechanism

This is an automatic re-insertion of a started conversion from queue stage 0 (including the control parameters) as new queue input. This feature allows a single setup (by SW) of a conversion sequence and multiple repetitions of the same sequence without the need to re-program it each time. A conversion sequence is repeated automatically if all queue entries of the sequence are setup for refill mode.

To enable this feature, RF bit in the QINR0/QINR2 register must be set along with the channel number. This is shown in Figure 17.

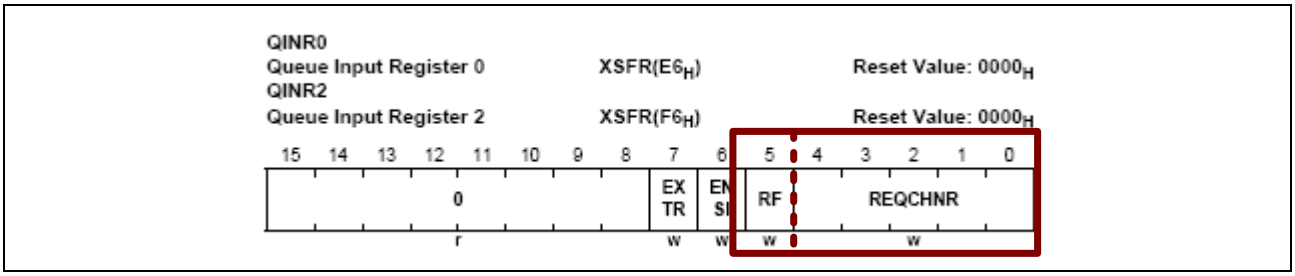

Figure 17 Refill mechanism

The bitfield RF value meant for:

0_B The content of this queue entry is not entered again in QINRx (x = 0 or 2) when the related conversion is started.

1_B The content of this queue entry is automatically entered again in QINRx (x = 0 or 2) when the related conversion is started

3.3.2 How to start sequential request source conversion sequence

Normal start (Write operation at QINR0 or QINR2):

A write operation to a queue input leads to a (new) valid queue entry. If the queue is empty (no valid entry), the written data arrives in queue stage 0 and starts a conversion request (if enabled by QMRx.ENGT and without waiting for an external trigger). If the refill mechanism is used, the queue inputs must not be written while the queue is running. Write operations to a completely filled queue are ignored. Here x = 0 or 2.

External trigger (Hardware Start):

An external trigger signal can be selected to start a scan sequence controlled by HW by an external module or signal, e.g. a timer unit or an input pin. The trigger feature is enabled by QMRx.ENTR = 1. The trigger event is generated if a rising edge is detected at the selected trigger input. Here x = 0 or 2.

External trigger (Software Start):

A trigger event is generated under SW control by writing QMRx.TREV = 1. This mechanism starts a request if queue stage 0 contains valid data (or the queue backup stage respectively). Here x = 0 or 2.

3.3.3 How to stop or abort an ongoing sequential request source conversion sequence

Using external gating:

An external gating signal can be selected to stop and to continue a sequence at any point in time controlled by an external module or signal, e.g. a timer unit or an input pin. The gating feature can be enabled and the polarity of the gating signal can be selected by QMRx.ENGT. The gating mechanism does not modify the queue entries, but only prevents the request handling block from issuing conversion requests to the arbiter. Here x = 0 or 2.

Using arbitration slot enable/disable:

The arbiter can be disabled by SW for this arbiter slot by clearing the corresponding bit **ASENR**.ASENx. This mechanism does not modify the queue entries, but only prevents the arbiter from accepting requests from the request handling block. Here x = 0 or 2.

Using clear flag of QMRx register (x = 0 or 2):

The next pending queue entry is cleared by writing bit QMRx.CLRV = 1. It is recommended to stop the sequence before clearing a queue entry (ENGT = 00B). If the queue backup stage contains a valid entry, this one is cleared; otherwise a valid entry in queue register 0 is cleared. Here x = 0 or 2.

Using FLUSH operation in QMRx Register (x = 0 0r 2):

All queue entries are cleared by writing bit QMRx.FLUSH = 1. It is recommended to stop the sequence before clearing queue entries.

3.4 Scan request source handling

This mechanism could be used to scan input channels permanently or on a regular time base. For example, if programmed with a low priority, some input channels can be scanned in a background task to update information that is not time-critical.

The functional representation of the scan request source is shown in Figure 18.

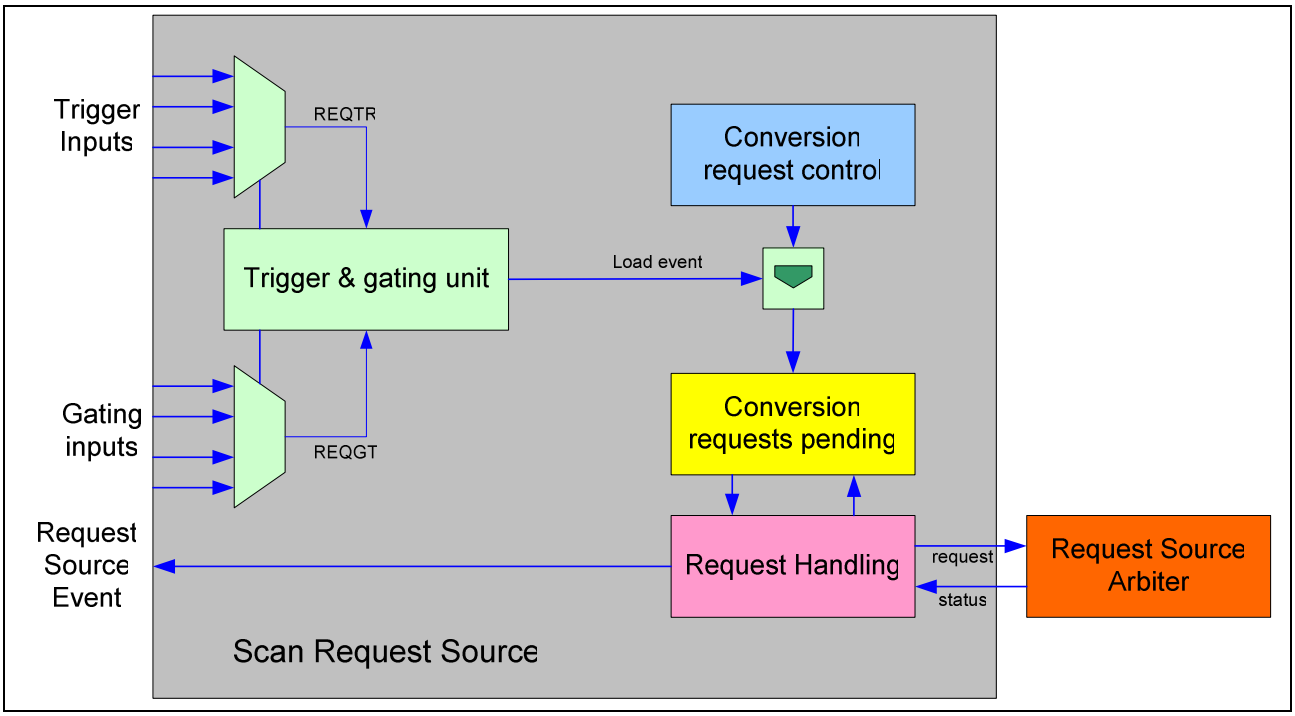

Figure 18 Scan request source

3.4.1 Conversion request control

Here the conversion request control defines if an analog input channel takes part in the scan sequence (see bits in register **CRCR1**). The programmed register value is kept unchanged by an ongoing scan sequence. An example scan sequence is configured in CRCR1 register and is shown in Figure 19.

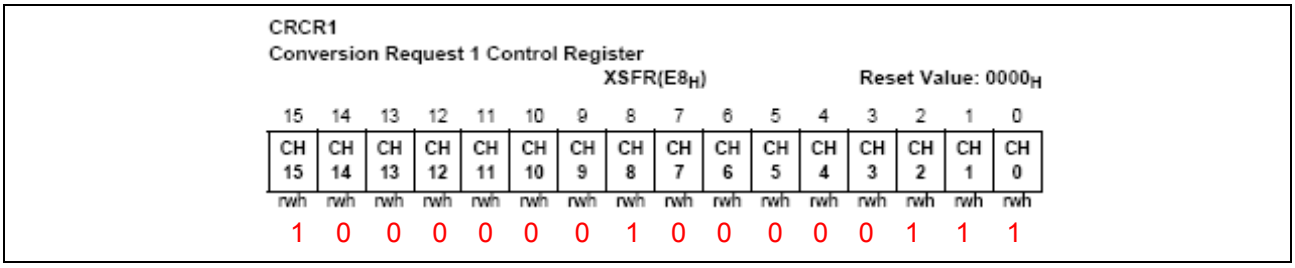

Figure 19 Conversion request control

The bifields (corresponding channels) holding values ‘1’ will be participating in the particular scan sequence. Also the conversion scan sequence starts from the highest channel number. In this case it starts with Channel 15 and then move towards Channel 0 (Ch15, Ch8, Ch2, Ch1 and Ch0).

3.4.2 Conversion request pending

The pending conversion requests indicate if an input channel has to be converted in an ongoing scan sequence (see bits in register **CRPR1**). A conversion request can only be issued to the request source arbiter if at least one pending bit is set. When load event bit is set, the contents of the CRCR1 register will be copied to the CRPR1 register. If we take content of CRCR1 from previous example, then the CRPR1 will have bitfield values as shown in Figure 20.

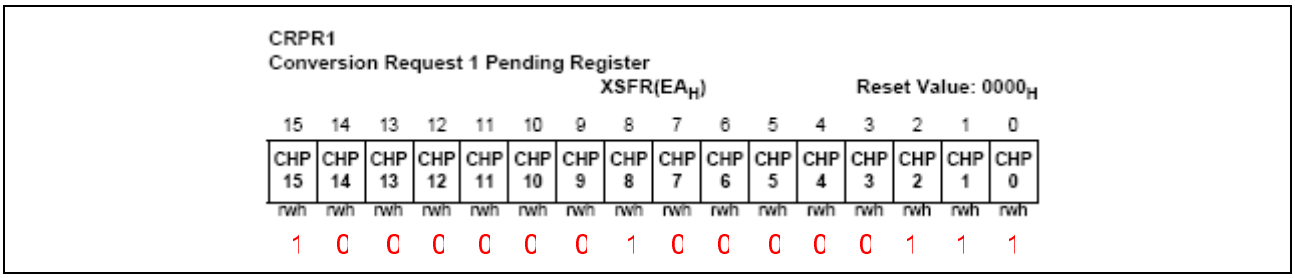

Figure 20 Conversion request pending

With each conversion start that has been triggered by the scan request source, the corresponding pending bit is automatically cleared. The scan sequence is considered finished and a request source event is generated if the last conversion triggered by the scan source is finished and all pending bits have been cleared. For example, if the conversion of Ch15 and Ch8 is completed, then the content of CRPR1 becomes as shown in Figure 21. Here the conversion pending bits of Ch15 and Ch8 are cleared.

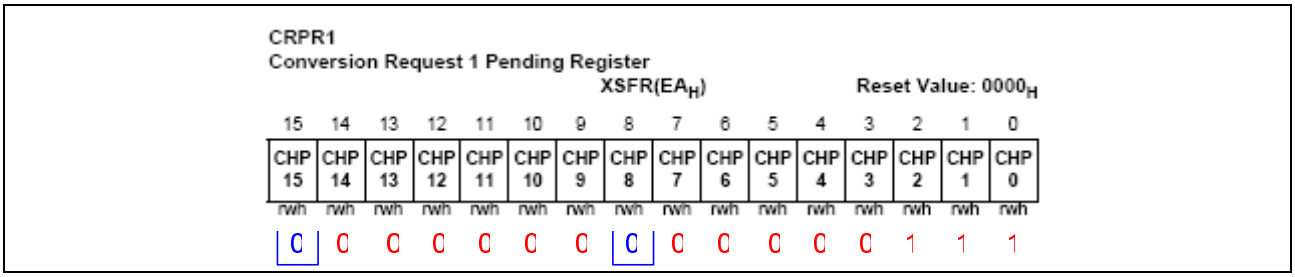

Figure 21 Clearing pending bits on conversion completion

When all the channel conversions of a scan sequence are completed, all the bitfields of CRPR1 are cleared. This is shown in Figure 22.

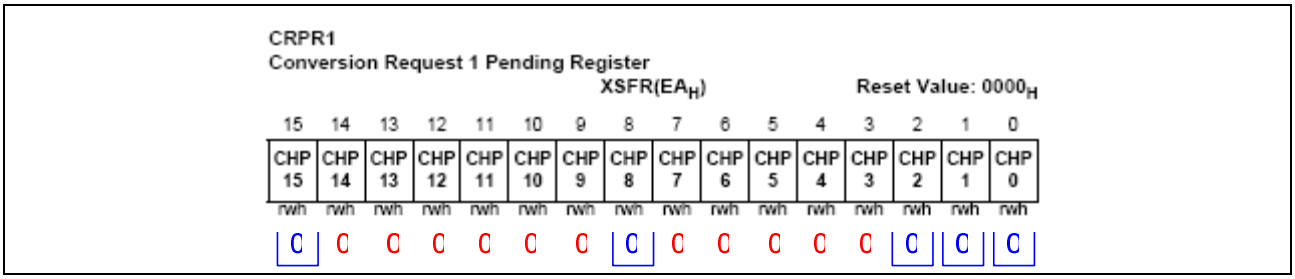

Figure 22 Completion of a scan sequence

3.4.3 Request handling

The request handling blocks interfaces with the request source arbiter. It requests conversion due to pending bits in the scan sequence and handles the conversion status information. If a conversion triggered by the scan request source is aborted due to a conversion request from another request source with a higher priority, the corresponding pending bit is automatically set. This mechanism ensures that an aborted conversion takes part in the next arbitration round and does not get lost. The control of the scan sequence is done based on bits in register **CRMR1**.

3.4.4 Trigger and gating signal handling

The trigger and gating unit interfaces with signals and modules outside the ADC module that can request conversions. For example, a timer unit can issue a request signal to synchronize conversions to PWM events. A load event starts a scan sequence by modifying the request pending bits according to the request control bits.

3.4.5 Autoscan mechanism

The autoscan is a functionality of the scan request source. If autoscan mode is enabled, the load event takes place when the conversion is completed while PND = 0 (PND is an internal signal to deliver the bitwise ORed result of the conversion pending bits in the CRPR register), provided the scan request source has triggered the conversion. This automatic reload feature allows channels 15 to 0 to be constantly scanned for pending conversion requests without the need for external trigger or software action.

The autoscan functionality can be enabled in **CRMR1** register. This is shown in Figure 23.

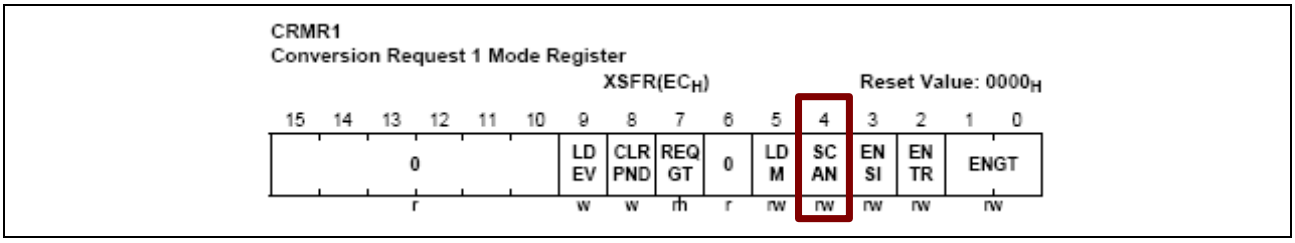

Figure 23 Autoscan functionality enable

The bitfield **SCAN** value meant for:

0_B The permanent scan functionality is disabled.

1_B The permanent scan functionality is enabled.

3.5 Attributes of arbiter

3.5.1 Arbitration slot enable/disable

Register ASENR contains bits that enable/disable the conversion request treatment in the arbitration slots. Bits ASEN0, ASEN1 and ASEN2 enable/disable arbitration slot 0, arbitration slot 1 and arbitration slot 2 respectively. Bit value '0' disables the corresponding arbitration slot whereas bit value '1' enables the corresponding arbitration slot. If an arbitration slot is disabled, a pending conversion request of a request source connected to this slot is not taken into account for arbitration. Bits ASEN0, ASEN1 and ASEN2 of ASENR register is shown in Figure 24.

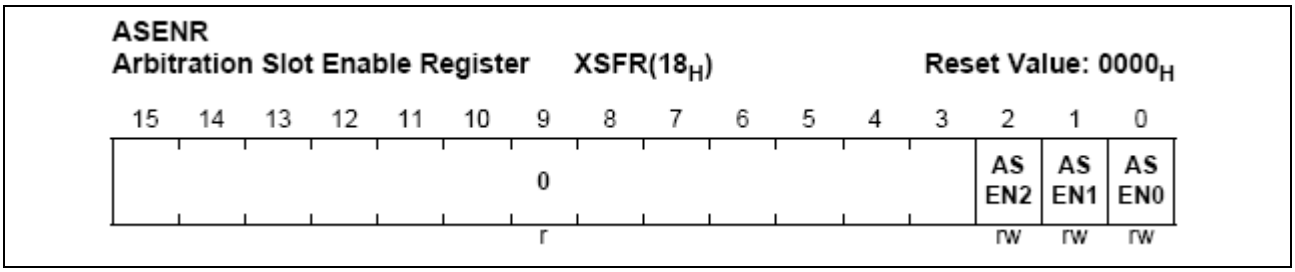

Figure 24 Arbitration slot enable/disable option

3.5.2 Arbitration mode

Register GLOBCTR contains bit ARBM and is shown in Figure 25. Bit value '0' (default) selects permanent arbitration whereas bit value '1' selects the option of starting the arbitration on pending conversion request. Permanent arbitration setting has to be chosen in synchronization slave. Synchronized conversion for parallel sampling is explained in section 7.

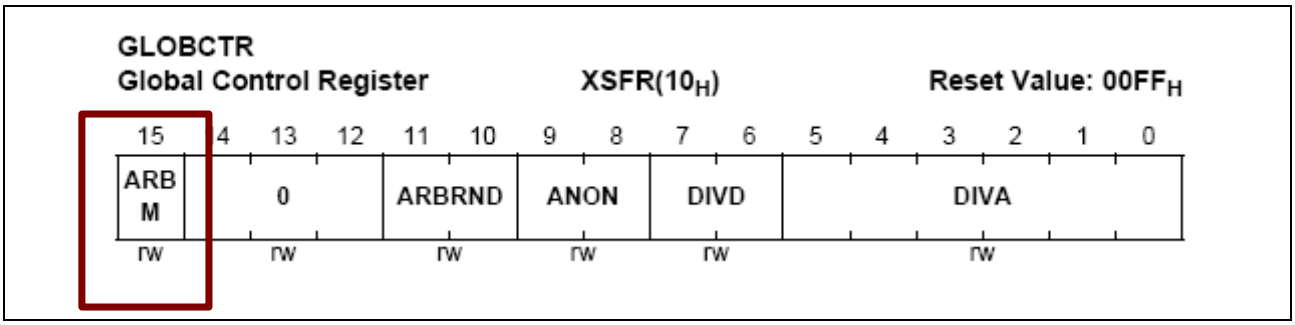

Figure 25 Arbitration mode selection

3.5.3 Arbitration round length

The number of arbitration slots forming an arbitration round can be programmed.

The period tARB of an arbitration round is given by:

$$t_{ARB} = N \times (GLOBCTR.DIVD + 1) / f_{ADC}$$

with N being 4, 8, 16, or 20 as defined by **GLOBCTR**.ARBRND

The configuration of bitfield ARBRND, arbitration round length in GLOBCTR register is shown in Figure 26.

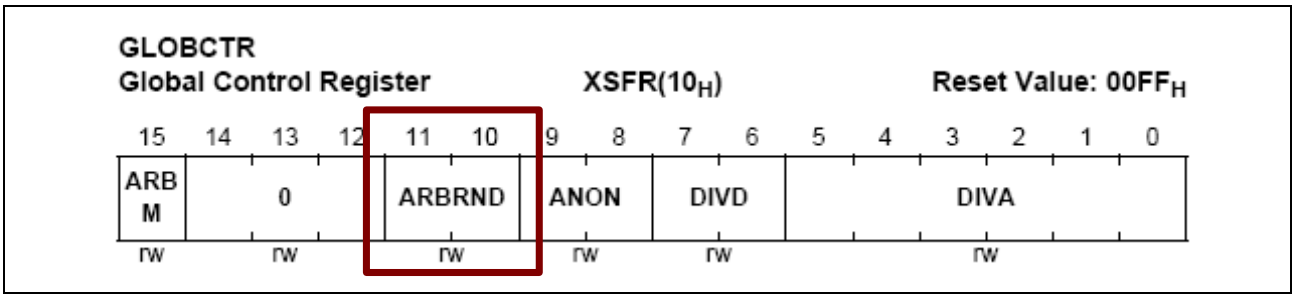

Figure 26 Arbitration round configuration

The bitfield ARBRND values meant for

00_B An arbitration round contains 4 arbitration slots (t_{ARB} = 4 / f_{ADC})

01_B An arbitration round contains 8 arbitration slots (t_{ARB} = 8 / f_{ADC})

10_B An arbitration round contains 16 arbitration slots (t_{ARB} = 16 / f_{ADC})

11_B An arbitration round contains 20 arbitration slots (t_{ARB} = 20 / f_{ADC})

3.6 General attributes of the request sources

3.6.1 Request source priority

Each request source has an individually programmable priority to be able to adapt to different applications. The arbiter selects the winning request based on this priority when two or more request sources indicate pending conversion requests at the same time. In RSPR0 (Request Source Priority Register 0) register, bitfield PRIOx (x=0, 1, 2) defines the priority of the conversion request source x. This is shown in Figure 27.

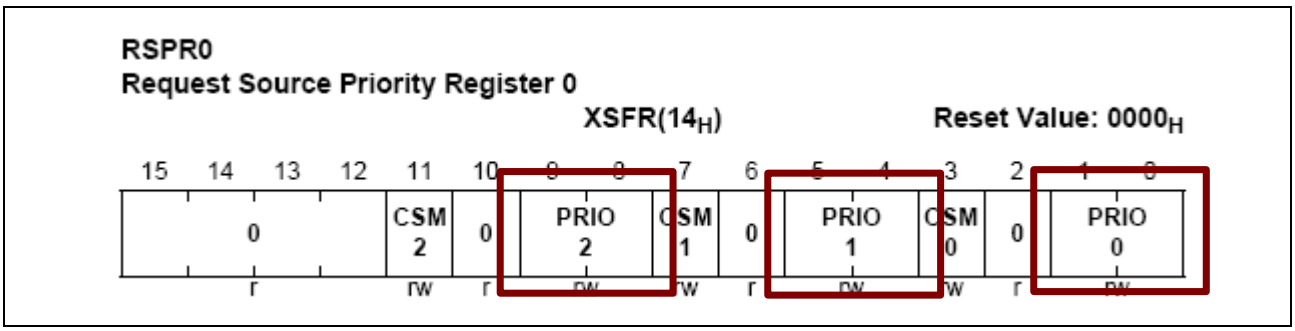

Figure 27 Priority configuration in request source priority register 0

Where the bitfield value meant for,

00_B Lowest Priority is selected

……

11_B Highest Priority is selected

3.6.2 Conversion start modes

To start the conversion request of the arbitration winner, any of the following two modes can be programmed by the user

- **Wait-for-start:**

In this mode, the current conversion is completed normally. The pending conversion request will be treated immediately after the conversion is completed. The conversion start takes place as soon as possible.

- **Cancel-inject-repeat:**

In this mode, the current conversion is aborted immediately if a new request with a higher priority has been found. The new conversion is started as soon as possible after the abort action. The aborted conversion request is restored in the request source that has requested the aborted conversion. As a result, it takes part in the next arbitration round.

The conversion start modes are explained in Figure 28. In this example, channel A is issued by a request source with a lower priority than the request source requesting the conversion of channel B. The diagram is self explanatory and more importantly, the setting and clearing of the channel conversion requests during start, end and abort of a conversion should be noticed.

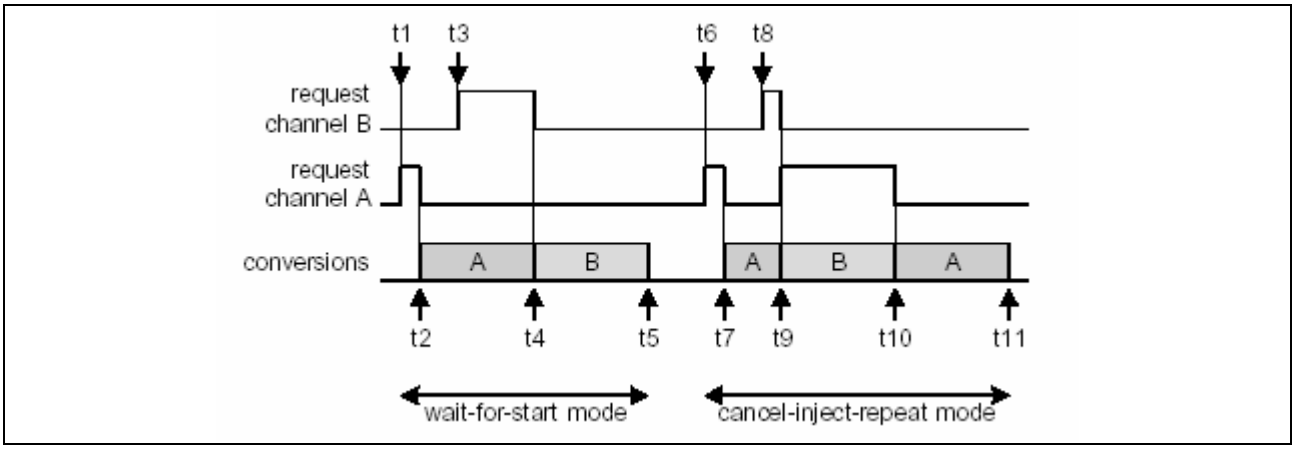

Figure 28 Conversion start modes

The conversion start mode of request sources can be individually programmed in RSPR0 register. Bit CSMx (x=0, 1, 2) defines the conversion start mode of request source x in arbitration slot x.

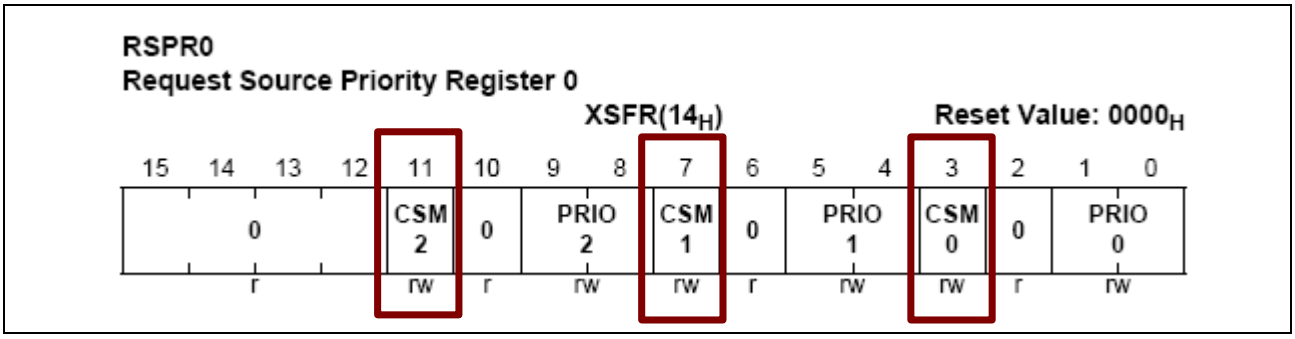

Figure 29 Conversion start mode configuration in RSPR0 register

Where the bit value meant for,

0_B Wait for Start mode is selected

1_B Cancel-inject-repeat mode is selected

4 Conversion phase

The conversion phase is associated with channel control registers, input class registers and AD converter. The channel number is the key in determining the steps in the next process.

Each channel has its own control information that defines the target result register for the conversion result, limit check control, reference voltage & input class selection and synchronized conversion request enabling. Two input classes (class 0 and class 1) are supported. The content of the channel control register x (x = 0-15) for the conversion channel x is shown in Figure 30.

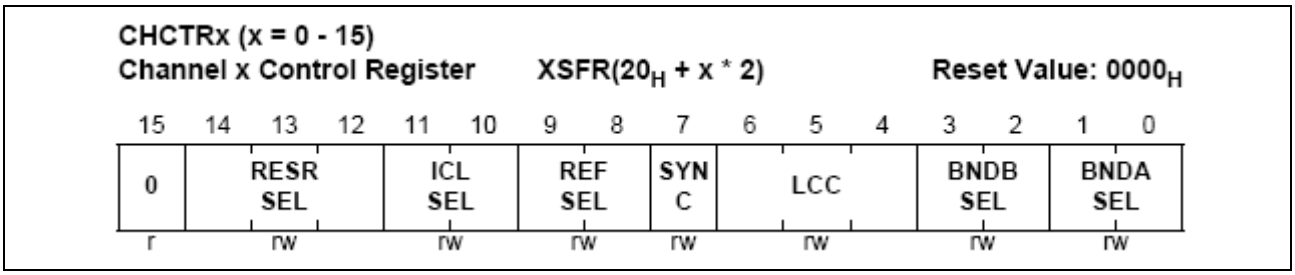

Figure 30 Channel control register x of conversion channel x

RESRSEL (Result Register Selection) bitfield defines which result register will be the target of a conversion of this channel.

000_b - Result Register 0 is selected.

001_b - Result Register 1 is selected.

…..

111_b - Result Register 7 is selected.

ICLSEL bitfield specifies the input class. Only values 00_b and 01_b are allowed. 00_b means Input class 0 is selected and 01_b means Input class 1 is selected.

LCC bitfield defines the limit checking mechanism and a detailed explanation on this explained in section 5.1

REFSEL bitfield defines the reference source for that particular channel. Only values 00_b and 01_b are allowed. 00_b selects the standard reference input VAREF for reference and 01_b selects the alternative reference input CH0.

The input class defines the sampling time and resolution (data width) of the conversion result. The content of the input class register x (x=0,1) is shown in Figure 31.

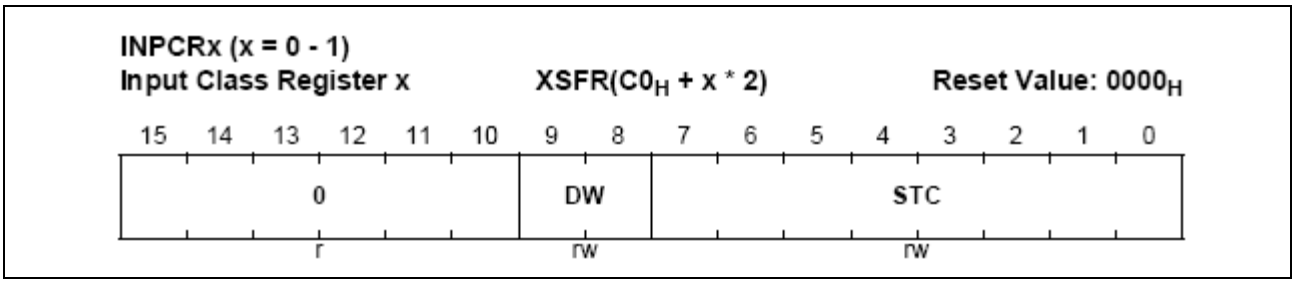

Figure 31 Input class register

STC bitfield defines the additional length of the sample time, given in terms of f_{ADCI} clock cycles. A sample time of 2 analog clock cycles is extended by the programmed value.

DW bitfield defines how many bits are converted for the result. Only values 00_b and 10_b are allowed. Value 00_b means the result is 10bit wide and value 10_b means the result is 8bit wide.

5 Result handling phase

The result handling phase involves the processing of the converted results. It includes the Limit control checking, data reduction, interrupt generation on meeting the limit checking criteria, data loss control (wait-for-read mode) and handling FIFO functionality on result registers.

5.1 Storage of conversion results

For each analog input channel, the associated channel control register **CHCTRx (x = 0 - 15)** contains a pointer bit field (RESRSEL) defining the result register to store the conversion result of this channel. This structure allows the user to direct conversion results of different channels to one or more result registers. Depending on the application needs (data reduction, auto-scan, alias feature, result FIFO, etc.), the user can distribute the conversion results to minimize CPU load or to be more tolerant against interrupt latency.

An individual data valid flag **VFR**.VFx (x=0 – 7) for each result register indicates that “new” valid data has been stored in the corresponding result register and can be read out. The different result handling mechanisms are explained here.

5.1.1 Data reduction mechanism

A data reduction filter implements the data reduction mechanism. This data reduction filter can be used to as digital filter for anti-aliasing or decimation purposes. It can accumulate a maximum of 4 conversion results to generate a final result. An example for the result data accumulation of 4 conversion results is shown in Figure 32.

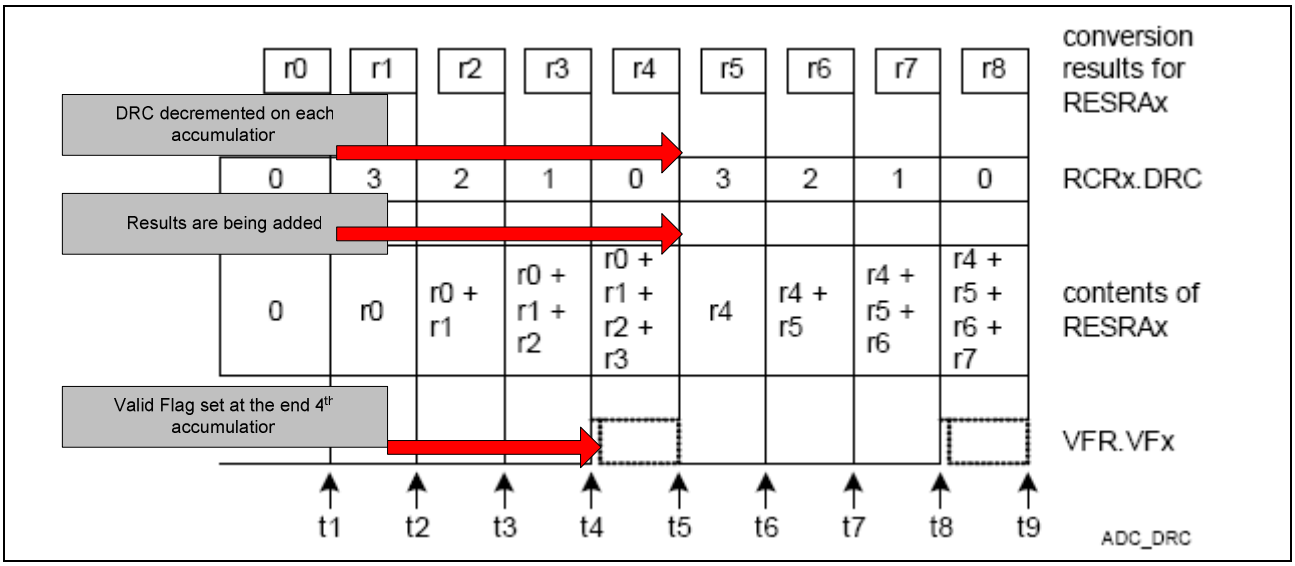

Figure 32 Data reduction control

Each result register can be individually enabled for data reduction. The feature is controlled by bit field DRCTR in registers **RCRx (x = 0 - 7)**. The actual status is given by bit field DRC (data reduction counter) in the same register. This is shown in Figure 33.

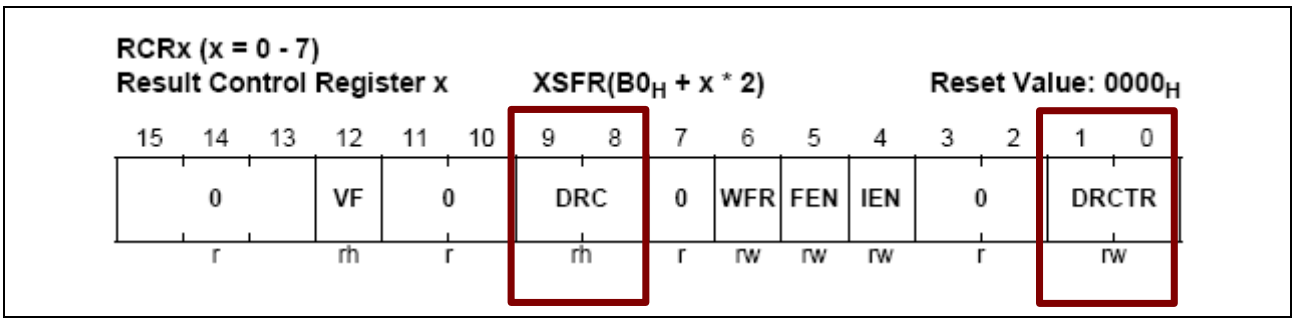

Figure 33 Data reduction control settings in result control register

DRCTR bitfield defines how many conversion results are accumulated for data reduction.

00_B The data reduction filter is disabled. The reload value for DRC is 0, so no accumulation is done.

01_B The data reduction filter is enabled. The reload value for DRC is 1, so accumulation is done over 2 conversions.

10_B The data reduction filter is enabled. The reload value for DRC is 2, so accumulation is done over 3 conversions.

11_B The data reduction filter is enabled. The reload value for DRC is 3, so accumulation is done over 4 conversions.

DRC bitfield indicates how many conversion results have still to be accumulated to generate the final result for data reduction.

00_B The final result is available in the result register.

01_B 1 more conversion result has to be added to obtain the final result in the result register.

10_B 2 more conversion result has to be added to obtain the final result in the result register.

11_B 3 more conversion result has to be added to obtain the final result in the result register.

Conversion delivering results to other result registers do not influence the data reduction filter of result register x. As a consequence, other channels can be converted between two conversions targeting result register x.

5.1.1.1 Data reduction filter disabled

The conversion result is maximum 10 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0. The data valid flag is set and a result event occurs each time a new conversion result is stored in the result register.

It is possible to share a result register among several analog input channels. The result register contents are shown in Figure 34.

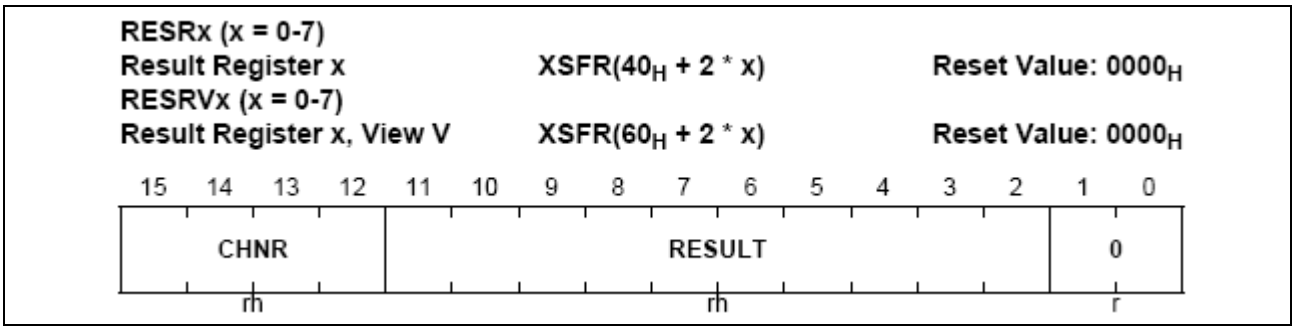

Figure 34 Result register

CHNR bitfield contains the channel number of the latest register update.

RESULT bitfield contains the conversion result.

5.1.1.2 Data reduction filter enabled

The conversion result is maximum 10 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0. The additional bits [13:12] show the MSBs of the data accumulation. The data valid flag is set and a result event occurs each time a data reduction sequence is finished and the final result is available in the result register.

The channel number is not included in the result register read view. This is shown in Figure 35.

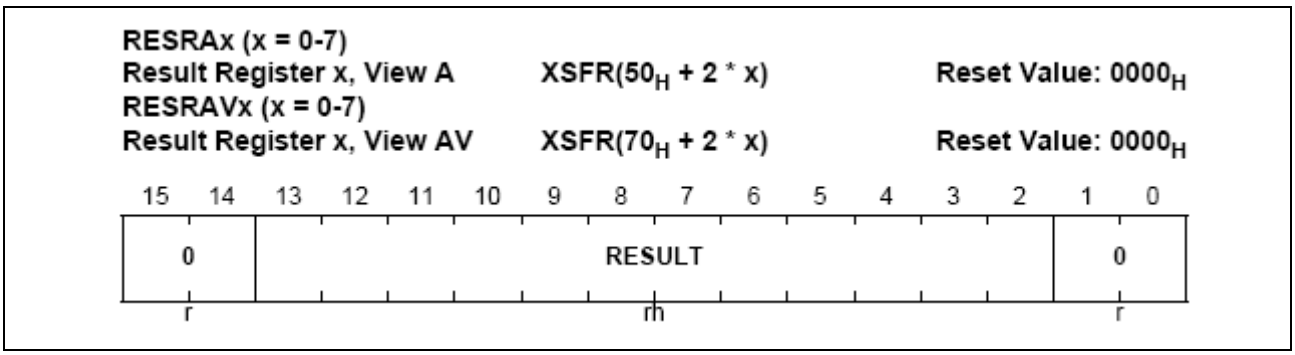

Figure 35 Data reduction result read register

5.1.2 Result FIFO buffer

If a result register is not used as direct target for a conversion result, it can be concatenated with other result registers of the same ADC kernel to form a result FIFO buffer (first-in-first-out buffer mechanism). This allows to store measurement results and to read them out later with a “relaxed” CPU access timing. It is possible to set up more than one FIFO buffer structure with the available result registers.

A FIFO structure can be built by at least two “neighbor” result registers with the indices x and z = x+1, where result register z represents the input and result register x represents the output of the FIFO buffer. The conversion result has to be delivered by the converter stage to the FIFO input, whereas the buffered data has to be read out from the FIFO output.

The FIFO buffer function can be enabled by setting bit FEN in registers **RCRx (x = 0 - 7)**.

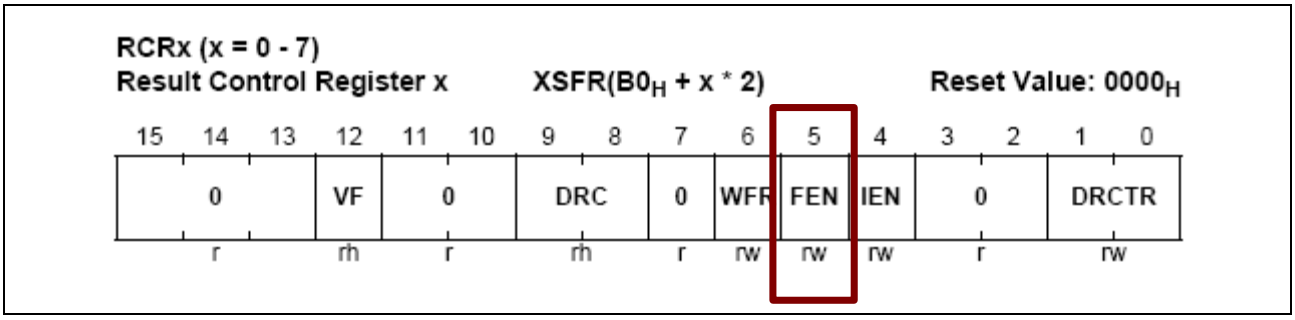

Figure 36 FIFO buffer enable

In the example shown in Figure 37, the result registers have been configured to form two FIFO buffers with two buffer stages (result registers 0/1 and 6/7, respectively), one FIFO buffer with three buffer stages (result registers 2/3/4), whereas result register 5 is used as “normal” result register without additional FIFO buffer functionality.

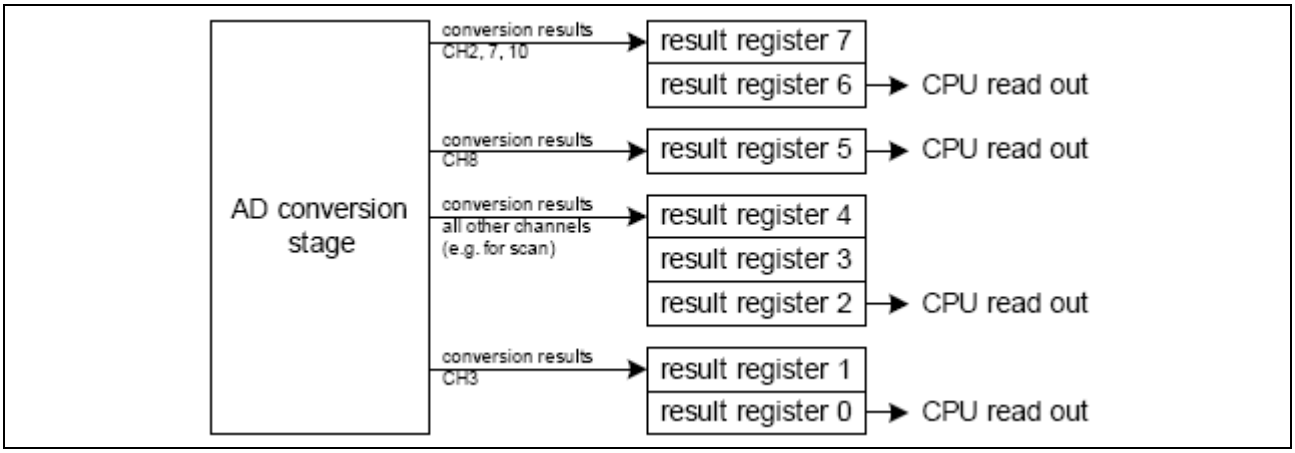

Figure 37 Result FIFO buffers

If more than two result neighbor registers are concatenated to a FIFO buffer (from result register z to result register x, with z > x), the one with the highest index (z) is always the input and the one with the lowest index (x) is always the output. All intermediate result registers y (x < y < z) are used as intermediate FIFO stages without data input or data output functionality.

Data Reduction and Wait-for-Read mode can be enabled only for the FIFO buffer input (z) result register. For more details on the FIFO usage conditions, refer the user manual.

If more than eight channel conversion results have to stored, then use one result register with the PEC functionality.

5.2 Data loss control (wait-for-read operation)

The wait-for-read mode is a feature of a result register allowing the CPU (or PEC) to treat each conversion result independently without the risk of data loss. Data loss could occur if the CPU does not read a conversion result from a result register before a new result overwrites the previous one.

If wait-for-read mode is enabled for a result register by setting bit WFR in register **RCRx (x = 0 - 7)**, a request source does not generate a conversion request while the targeted result register contains valid data (indicated by the valid flag VFx = 1) or if a currently running conversion targets the same result register.

A new conversion request is generated only after the targeted result register has been read out. If two request sources target the same result register with wait-for-read selected, a lower priority request started before the higher priority source has requested its conversion can not be interrupted by the higher priority request. If a higher priority request targets a different result register, the lower priority conversion can be cancelled and repeated afterwards.

The contents of the result control register x (x=0-7) are shown in Figure 38.

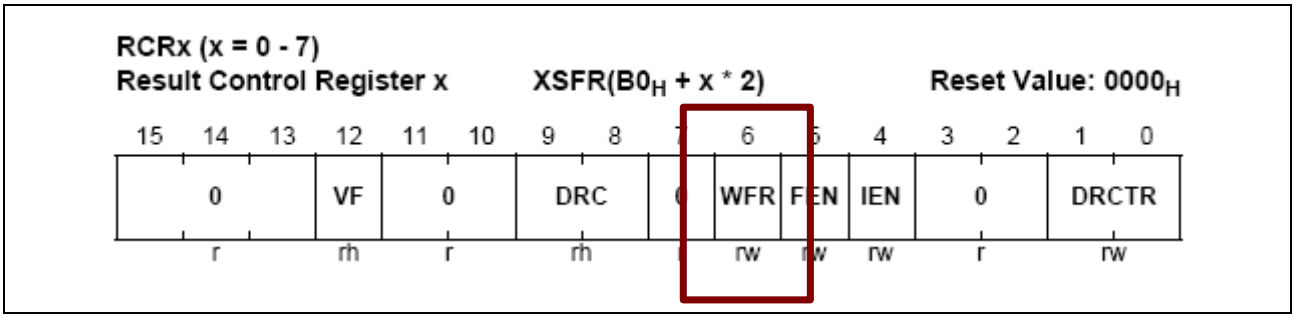

Figure 38 Wait for read mode setting in result control register

5.3 Limit checking

The limit checking mechanism automatically compares each conversion result to two boundary values (Boundary A and Boundary B). For each channel, the user can select these boundaries from the set of 4 programmable values (LCBR0 to LCBR3). The channel interrupt generation based on the limit checking functionality is shown in Figure 39.

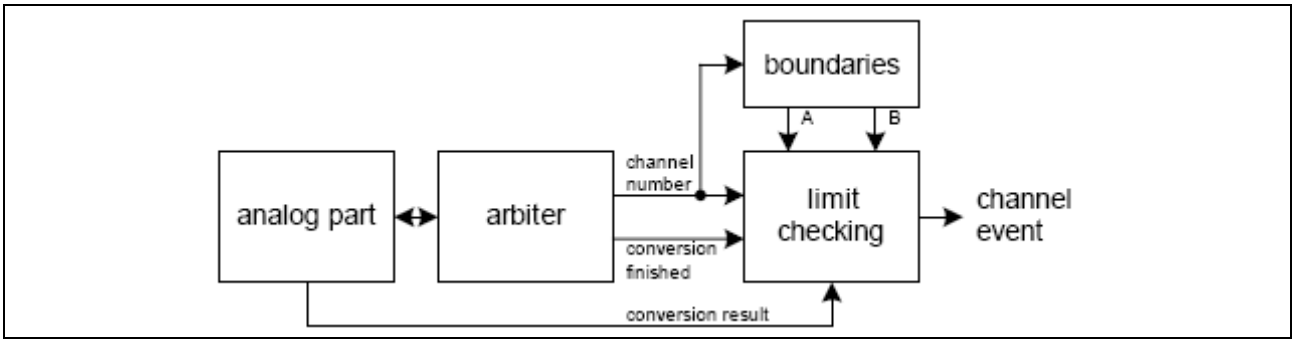

Figure 39 Channel interrupt generation with limit checking mechanism

With this structure, the conversion result range is split into three areas:

Area I: The conversion result is below or equal to both the boundaries.

Area II: The conversion result is above one boundary and below or equal to the other boundary.

Area III: The conversion result is above both the boundaries.

Bit field LCC in the channel control register defines the condition to generate a channel event, leading to a channel event interrupt:

LCC = 000_B: No trigger, the channel event generation is disabled.

LCC = 001_B: A channel event is generated if the conversion result is not in area I.

LCC = 010_B: A channel event is generated if the conversion result is not in area II.

LCC = 011_B: A channel event is generated if the conversion result is not in area III.

LCC = 100_B: A channel event is always generated (regardless of the boundaries).

LCC = 101_B: A channel event is generated if the conversion result is in area I.

LCC = 110_B: A channel event is generated if the conversion result is in area II.

LCC = 111_B: A channel event is generated if the conversion result is in area III.

Hence the interrupts can be generated, when the result is

- Within Area I, Area II and Area III
- Not within Area I, Area II and Area III
- Always (regardless of boundaries) or never

The limit checking mechanism is shown in Figure 40

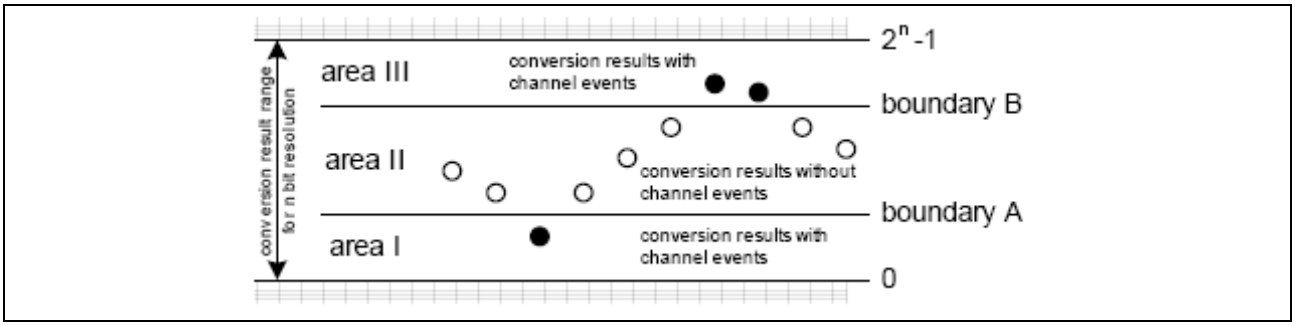

Figure 40 Limit checking

6 Interrupts

Each ADC kernel provides 4 independent interrupt handling service nodes. The interrupt generation inside the ADC kernel is based on three different types of events.

- Channel Events: Activated by the completion of any input channel conversion. They are enabled according to the control bits for the limit checking. The settings are defined individually for each input channel. The interrupt generation is explained with Limit checking mechanism in section 5.3. This allows the user to interrupt the CPU only if the specified conversion result range is met (or not met) instead of comparing each result by SW.
- Result Event: A result event is detected if a new result is available in a result register and can be read out, e.g. to store the data in memory for further treatment by SW.

  This is very useful with data reduction, FIFO usage and PEC transfer.
- Request source events: A request source event is detected if a scan source has completely finished the requested conversion sequence. For a sequential source, the user can define where inside a conversion sequence a request source event is generated, activated by events of the request sources (parallel and scan source interrupts) or result registers (result interrupts).

A node pointer mechanism allows the user to group interrupt events by selecting which service request output signals SRx (x=0-3) becomes activated by which event. Each ADC event can be individually directed to one of the service request output signals to adapt easily to application needs.

7 Synchronized conversion for parallel sampling

In order to measure two analog channels simultaneously, the two ADC kernels (ADC0 and ADC1) can be synchronized. In this case one ADC kernel will act as a master and the other as a slave. As both ADC0 and ADC1 are similar, each kernel can be setup to be a synchronization master or a synchronization slave. The master ADC kernel will issue a synchronization conversion request to the slave and the slave reacts to the request from the master.

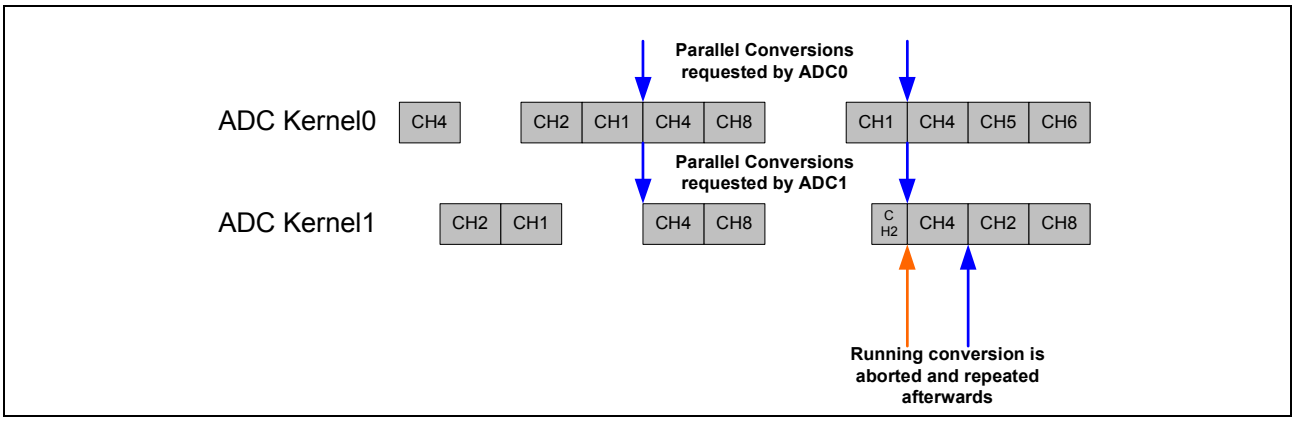

Figure 41 Synchronized conversion for parallel sampling

In the above example shown in Figure 41, the ADC0 kernel acts as a master and ADC1 acts as a slave. The Channel 4 of each ADC kernel is configured for parallel conversion. Please note that a parallel conversion request is always handled with highest priority and cancel-inject-repeat mode in a synchronization slave. In the above example, the ongoing CH2 conversion is aborted and resumed after the CH4 conversion is completed.

More details on handling the synchronized parallel sampling is found in the user manual.

8 Software Example

A software example is given for conversions through ADC0 request source 0 and 1 along with this application note. Request source 0 conversion is triggered by hardware (CCU60 T13 PM) whereas the request source 1 conversion is triggered by software (using Load event in CCU60 T13 PM interrupt service routine).

Channels 0 and 8 are used for converting analog input signals and conversions requested through request source 1 and 0 respectively. The conversion results are stored in the result register 0 and 1 respectively. Also the wait for read mode is selected for the both the result registers 0 and 1.

Here the example is given for a XC2267 device and the device configuration is done using DAvE (Digital Application virtual Engineer).

After the MAIN_vInit() function (MAIN.C), the following operations are performed.

```
ADC0_CRCR1 |= 0x0001;     //  Here the Channel 0 conversion is requested.  On every T13 PM ISR
                          //  the software trigger for conversion is enabled by setting the Load Event
                          //  (LDEV) bit.
ADC0_QINR0  = 0x00A8;     //  Here the Channel 8 conversion is requested.  On every T13 PM the
                          //  hardware trigger for conversion is enabled and refill and external trigger
                          //  bits are set to reload conversion request.
```

In the CCU60 T13 PM ISR (CCU60.C), the following operations are performed.

```
        ADC0_CRMR1 |= 0x0200;              // Set the Load Event bit (trigger Channel 2 conversion)
        while (ADC0_uwBusy( ));            //  Wait for ADC to complete the conversion
//      result0 = ((ADC0_RESR0 >> 4) & 0x00FF);  // Read the result register 0 result in result0 variable.
//      result1 = ((ADC0_RESR1 >> 4) & 0x00FF);  // Read the result register 1 result in result1 variable.
//      P10_OUT = result0;                  // Display the result0 variable in port 10
        P10_OUT = result1;                  // Display the result1 variable in port 10
```

As the wait for read mode is enabled, the user can see how the conversions are happening by commenting and uncommenting the lines (whether conversion is happening or not if the result is not read). Also the result is displayed at Port 10 (Result is 8 bits wide).

As the conversion result is eight bit wide, the valid result data is stored in bitfield 11:4. Hence the result register bits are right shifted four times and bitwise ANDed with 0x00FF.

Also refer the example given in the Application note AP16145 titled “Using Enhanced Interrupt Handling with DAvE”. Here the ADC conversion and result handling with PEC transfer are discussed in detail.

http://www.infineon.com

Published by Infineon Technologies AG