Application Note, V1.0, August. 2008

AP08063

XC800 family

Programming the Analog-to-Digital Converter on XC800 family of Microcontrollers

Microcontrollers

Never stop thinking

Edition 2008-08-27

Published by
Infineon Technologies AG
81726 München, Germany

AP08063		
Revision History:	2008-08	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

Table of Contents **Page**

1 Introduction

1.1 Overview

This application note provides a quick reference to start work with the Analog-to-Digital converter module of the XC800 family of 8-bit Microcontrollers (XC866, XC88x and XC878). The functional description and configuration options of the ADC module are also discussed with examples in order to give a good insight on the usage of the module for the application needs.

1.2 Background

With Standard ADC's, the CPU is involved in most of its control operation. The tasks like Switching between channels, Start of conversion and Wait on ready and read out result etc., consume CPU time. Synchonizing a standard ADC conversion to events such has PWM for noise free sampling can be tricky and require fast PWM interrupts and possibly high CPU load.

In order to meet the real time needs of applications, an autonomous ADC module which offloads the CPU is required. The XC800 family of microcontrollers offers a specialized ADC that is designed to meet the needs of the motor control, power supply and other general control applications. This ADC module works autonomously thereby reducing the CPU load. Hence the CPU can be utilized for other major tasks.

For example, in Drive applications, the ADC is used intensively for the measurement of Phase voltages and currents. As these are fast changing signals that are corrupted by PWM, they have to be measured accurately and quickly at the specified time synchronously to the PWM. These measured values are used in control loops which run faster than 2-4 PWM periods, say 100-200µs. Also the ADC is used for the measurement of temperature, Speed reference setting etc., and their sampling rate differs from the Phase voltages and currents.

1.3 Features of the ADC module

The XC800 family of microcontrollers includes 10-bit Analog-to-Digital Converter (ADC) with up to eight 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:

- Successive approximation
- 8-bit or 10-bit resolution (TUE of ± 1 LSB and ± 2 LSB, respectively)
- Up to eight analog channels and four independent result registers
- Programmable Result data protection for slow CPU access (wait-for-read mode)
- Single or multiple conversion modes
- Autoscan functionality
- Limit checking for conversion results
- Data reduction filter (accumulation of up to 2 conversion results)
- Two independent conversion request sources with programmable priority
- Selectable conversion request trigger. Software or Hardware (e.g. Timer) Triggers.
- Flexible interrupt generation with configurable service nodes
- Cancel/restart feature for running conversions
- Low power modes

2 Detailed description of the ADC module

2.1 Functionality of the ADC module

The functionality of the ADC module includes:

- Two different conversion request sources (sequential and parallel) with independent registers. The request sources are used to trigger one or more conversions by software or by external events (synchronization to PWM signals), sequencing schemes, etc.
- An arbiter that regularly scans the request sources to find the channel with the highest priority for the next conversion. The priority of each source can be programmed individually to obtain the required flexibility to cover the desired range of applications.
- Control registers for each of the eight channels that define the behavior of each analog input (such as the interrupt behavior, a pointer to a result register, etc.).
- An input class register that delivers general channel control information (sample time) from a centralized location.
- Four result registers for storing the conversion results and controlling the data reduction.
- A decimation stage for conversion results, adding the incoming result to the value already stored in the targeted result register.

2.2 ADC Operational Process

For simplicity we say, The ADC process involves three phases namely Conversion Request Phase, Conversion Phase and Result Handling Phase. The simple representation of the operational process is shown in Figure 1. The functions of each phase were explained briefly in the following sections.

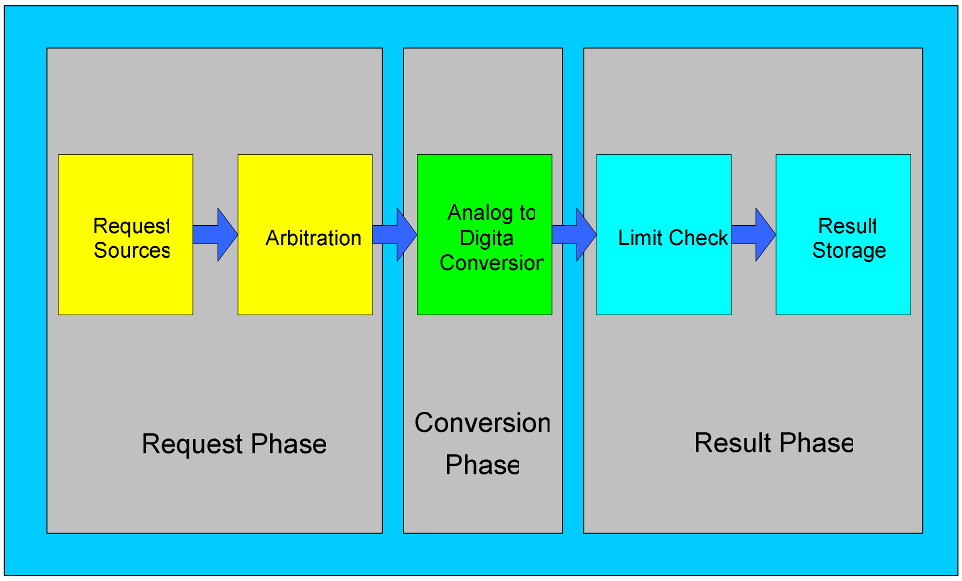


Figure 1 ADC Operational Process

2.2.1 Conversion Request Phase

Conversion request phase consists of handling the conversion request sources, arbiter and their attibutes. In XC800 family of microcontrollers there are two conversion request sources (Sequential and Parallel). The sequential and parallel sources will participate in the arbitration from slot 0 and slot 1 respectively.

Sequential requests are queued approach, where requests are processed in the order in which they are entered in the queue.

Parallel requests are grouped approach, where the user selects an arbitrary set of channels to be converted (one bit flag for each channel to be converted). The request flags are then processed starting at the highest numbered channel, working down to the lowest numbered channel.

The request source arbiter evaluates which analog input channel has to be converted. Therefore, it regularly polls the two request sources (source x at slot x, x = 0 - 1) one after the other for pending conversion requests. At the end of the round, the arbiter delivers the winning request on to the next step of process.

The request sources can be prioritized. Hence the arbiter issues the higher priority wins the arbitration if both the request sources request conversions at the same time. If both request sources are programmed with the same priority, the channel number specified by request source 0 will be converted first since it is connected to arbitration slot 0. The general arbitration process is shown in Figure 2.

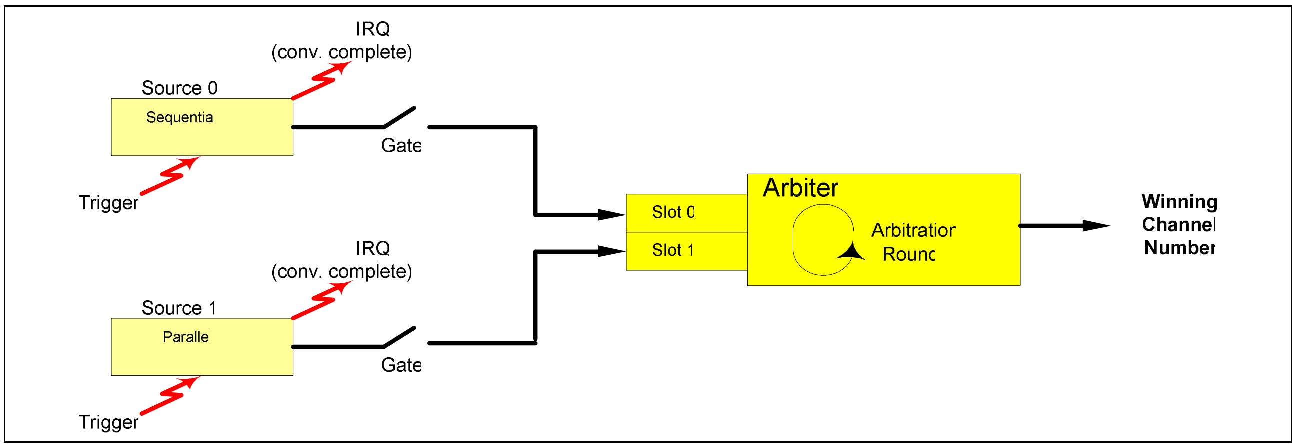


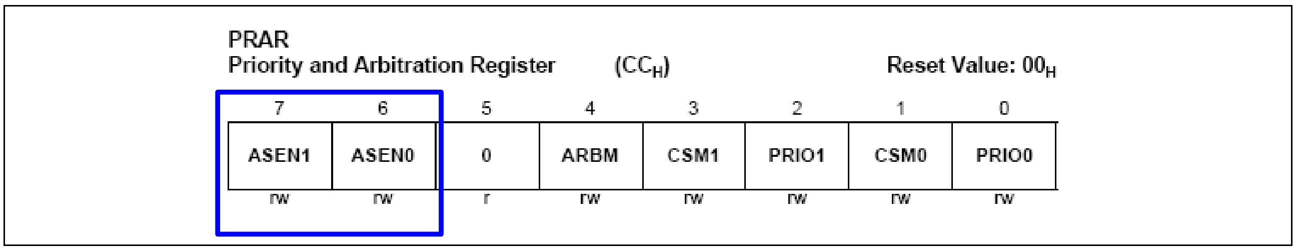
Figure 2 Arbitration for conversion requests

The general & specific attributes of the request sources, arbiter are explained in the following sections.

2.2.1.1 Attibutes of the Request source arbiter

2.2.1.1.1 Arbitration slot Enable/Disable

Register PRAR contains bits that enable/disable the conversion request treatment in the arbitration slots. Bits ASEN0 and ASEN1 enable/disable arbitration slot 0 and arbitration slot 1 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 and ASEN1 of PRAR register is shown in Figure 3.



PRAR
Priority and Arbitration Register (CC$_H$) Reset Value: 00$_H$

7	6	5	4	3	2	1	0
ASEN1	ASEN0	0	ARBM	CSM1	PRIO1	CSM0	PRIO0
rw	rw	r	rw	rw	rw	rw	rw

Figure 3 Arbitration slot enable/disable option

2.2.1.1.2 Arbitration mode

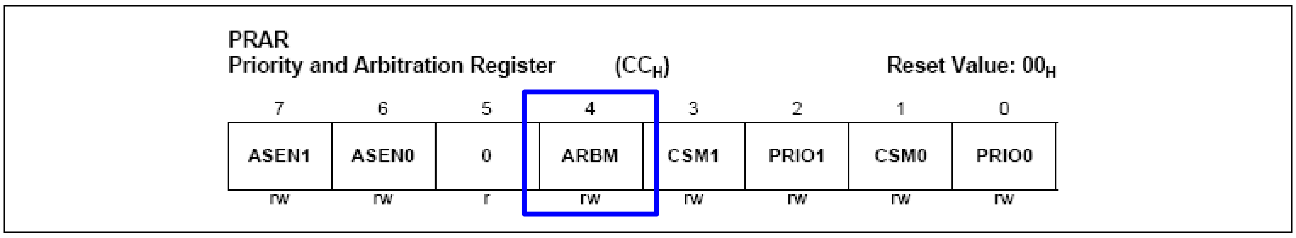
Register PRAR contains bit ARBM and is shown in Figure 4. Bit value ‘0’ (default) selects permanent arbitration whereas bit value ‘1’ selects the option of starting the arbitration on pending conversion request.

- **Permanent arbitration:**

In this mode, the arbiter will continuously poll the request sources even when there is no pending conversion request.

- **Arbitration started by pending conversion request:**

In this mode, the arbiter will start polling the request sources only if there is at least one conversion pending request.



PRAR
Priority and Arbitration Register (CC$_H$) Reset Value: 00$_H$

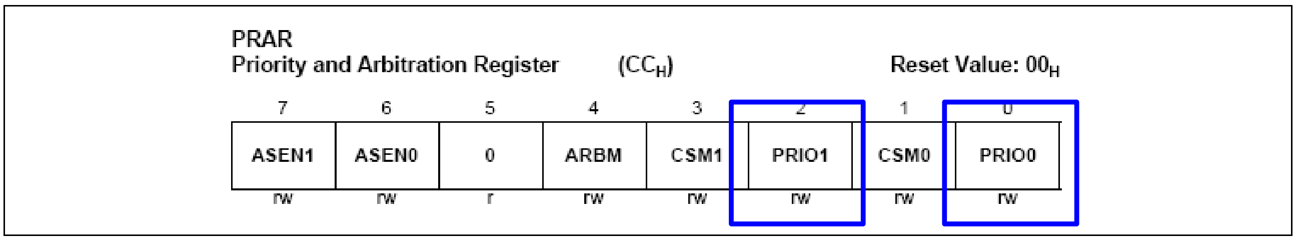
7	6	5	4	3	2	1	0
ASEN1	ASEN0	0	ARBM	CSM1	PRIO1	CSM0	PRIO0
rw	rw	r	rw	rw	rw	rw	rw

Figure 4 Arbitration Mode

2.2.1.2 General attributes of Request sources

2.2.1.2.1 Request Source priority

The request sources can be prioritized. Register PRAR contains bits that define the request source priority and conversion start mode (this is explained in next chapter). It also contains bits that enable/disable the conversion request treatment in the arbitration slots. In Figure1, PRIO0 bit defines the priority of the sequential request source 0 and PRIO1 bit defines the priority of the parallel reqest source 1. Low priority is defined by bit ‘0’ and High priority is defined by bit ‘1’.



PRAR
Priority and Arbitration Register (CC$_H$) Reset Value: 00$_H$

7	6	5	4	3	2	1	0
ASEN1	ASEN0	0	ARBM	CSM1	PRIO1	CSM0	PRIO0
rw	rw	r	rw	rw	rw	rw	rw

Figure 5 Priority and Arbitration Register

2.2.1.2.2 Conversion Start Modes

At the end of each arbitration round, the arbiter will have found the request source with the highest priority and a pending conversion request. It stores the arbitration result, namely the channel number, the sample time and the targeted result register for further actions. In case the new conversion request has a higher priority than the current conversion, two conversion start modes exist (selectable by bit CSMx, x = 0 - 1):

- **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.

These conversion start modes are shown in Figure 6.

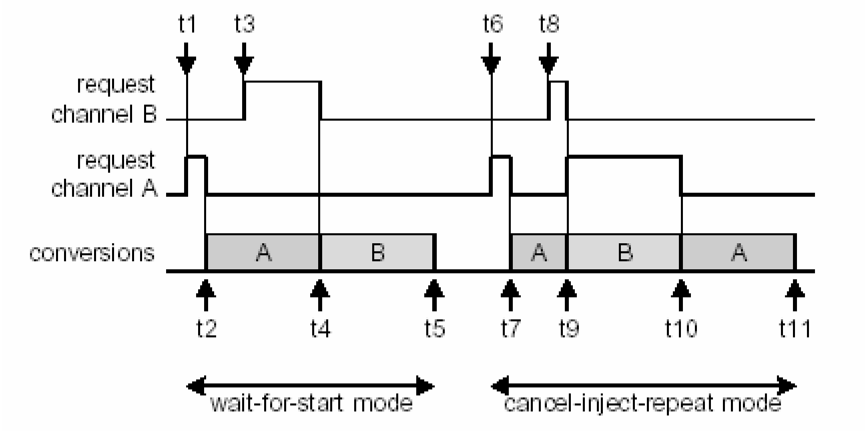
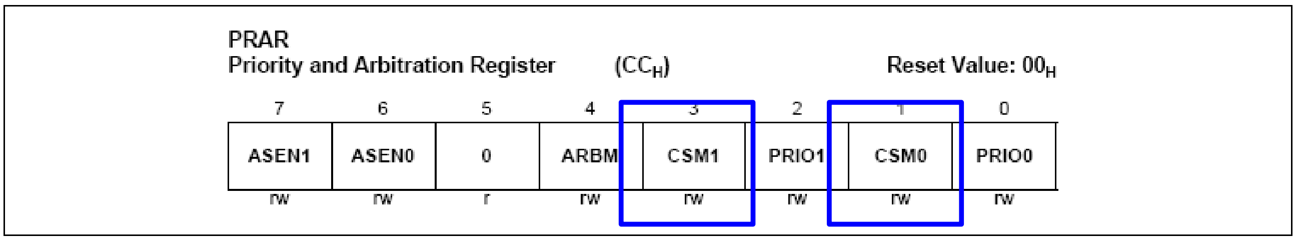


Figure 6 ADC Conversion Start Modes

In Figure 7, bit CSM0 defines the conversion start mode of the sequential request source 0 and bit CSM1 defines the conversion start mode of the parallel reqest source 1. Bit value ‘0’ (default) selects Wait for Start mode whereas bit value ‘1’ selects the option of Cancel-Inject-Repeat mode.



PRAR
Priority and Arbitration Register (CC$_H$) Reset Value: 00$_H$

7	6	5	4	3	2	1	0
ASEN1	ASEN0	0	ARBM	CSM1	PRIO1	CSM0	PRIO0
rw	rw	r	rw	rw	rw	rw	rw

Figure 7 Conversion Start modes

2.2.1.3 Attributes of sequential request source

The main attributes of sequential request source are mentioned here and are explained in detail in later sections with examples.

- Trigger control
- Gate control
- Refill mode
  - Discard request after conversion, or
  - Place back on top of queue
- Source interrupt control (explained in section 2.2.4)
- Queue control
  - Queue input
  - Each queue input allows individual control of trigger, refill mode, and source interrupt
  - Flush queue

2.2.1.4 Attributes of parallel request source

The main attributes of parallel request source are mentioned here and are explained in detail in later sections with examples.

- Trigger control
- Gate control
- Autoscan mode
  - Reload channel requests after last channel converted
- Source interrupt control (explained in section 2.2.4)
- Channel request control
  - Channel request (one bit per channel)
  - Pending status
  - Load event and pending controls

2.2.2 Conversion Phase

The Conversion Phase is associated with channel control register and input class register. The channel number is the key in determining the remaining steps of the conversion process.

Each channel has its own control information that defines the target result register for the conversion result. The only control information that is common to all channels is the sampling time defined by the input class register (in many of Infineons 16 and 32-bit MCUs the ADCs have multiple input class registers).

The channel control register contents are shown in Figure 8.

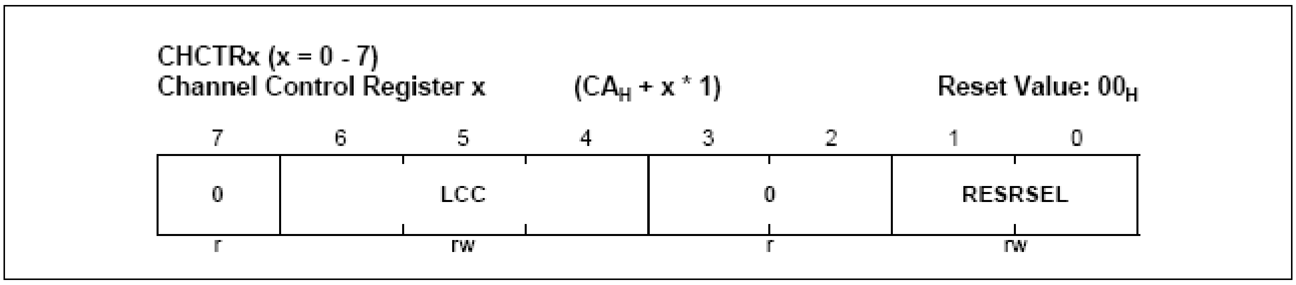


Figure 8 Channel control register

Register CHCTRx defines the settings for the input channel x (where, x = 0 – 7).

The bitfield LCC defines the Limit checking mechanism and a detailed explanation on this can be found in the User manuals of XC800 family of microcontrollers.

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

00	- Result Register 0 is selected.
01	- Result Register 1 is selected.
10	- Result Register 2 is selected.
11	- Result Register 3 is selected.

The input class register INPCR0 contains bits that control the sample time for the input channels. This is shown in Figure 9.

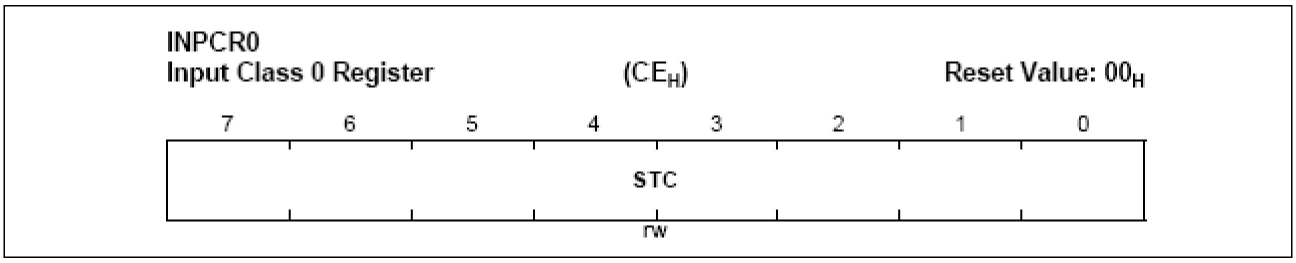


Figure 9 Input Class register

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

2.2.3 Result Phase

The result phase handles the storage of the conversion result, data decimation, limit checking and interrupt generation.

The result generation of the ADC module consists of several parts:

- Four result registers, for storing the conversion results
- A data reduction filter, for accumulating (adding) the conversion results
- A limit checking unit, comparing the conversion result to two selected boundary values (BOUND0 and BOUND1)
- All result registers can be viewed by either Normal read view (for 8/10bit) or Accumulated read view (for 9/11bit)
- Wait for Read Mode

A detailed explanation about the result phase can be found in the User manual of the XC800 family of microcontrollers. However, the Wait-for-Read mode is explained here.

The wait-for-read mode can be used for all request sources to allow the CPU to treat each conversion result independently without the risk of data loss. Data loss can occur if the CPU does not read a conversion result in a result register before a new result overwrites the previous one.

In wait-for-read mode, the conversion request generated by a request source for a specific channel will be disabled (and conversion not possible) if the targeted result register contains valid data (indicated by its valid flag being set). Conversion of the requested channel will not start unless the valid flag of the targeted result register is cleared (data is invalid).

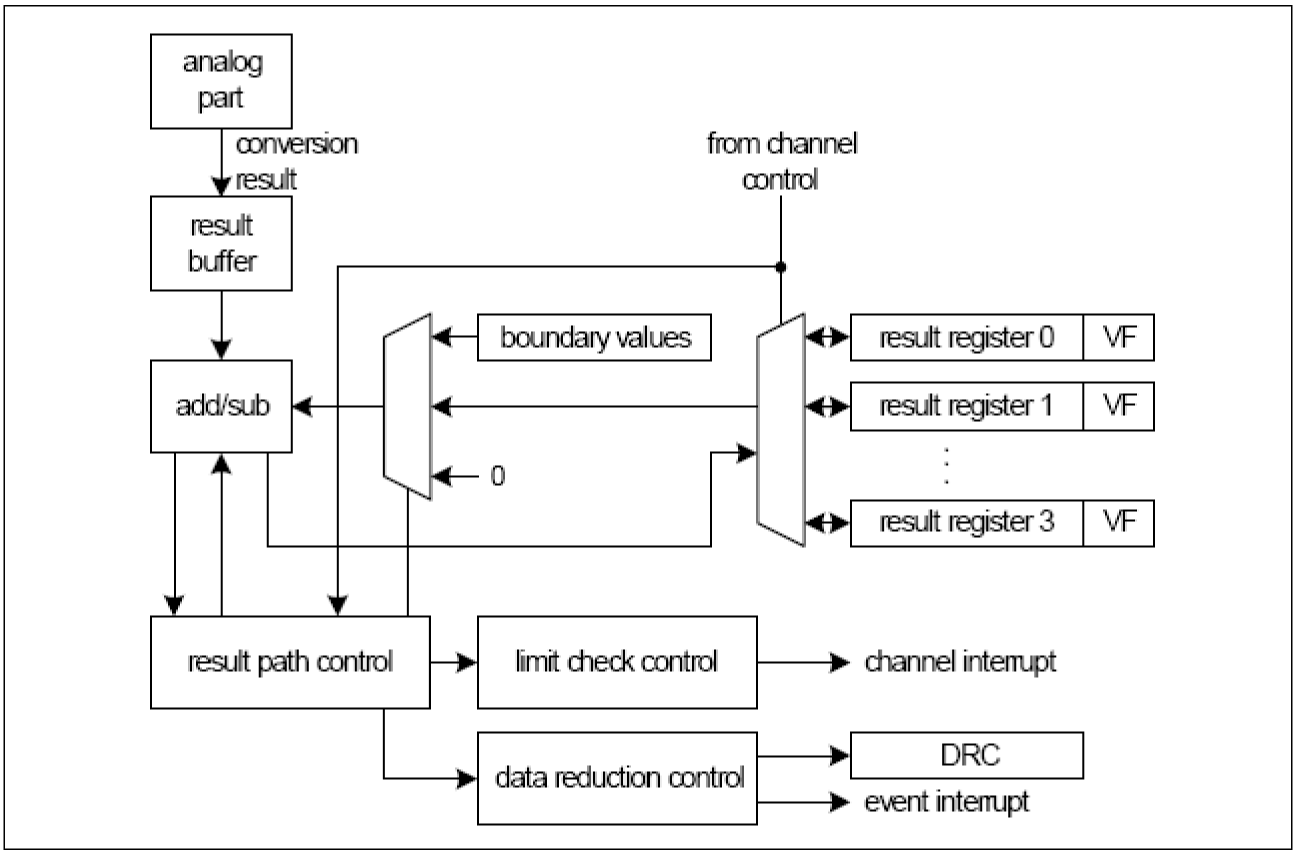


Figure 10 Result Generation

2.2.4 Interrupts

The XC800 family of devices each has 2 interrupt service nodes and interrupt vector table locations that are used by the ADC. These are referred to as service request outputs SR [1:0] that can be activated by different interrupt sources within the ADC.

The interrupt structure of the ADC supports two different types of interrupt sources:

- Event Interrupts: Activated by events of the request sources (parallel and sequential source interrupts) or result registers (result interrupts).
- Channel Interrupts: 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.

Each of the individual interrupts that make up the Event and Channel interrupts can individually be mapped to SR0 or SR1. For example a parallel source interrupt and a channel 4 interrupt could both be mapped to SR0 while channel 1 and result register 3 interrupts could be mapped SR1.

The request source event interrupt, the interrupt which is generated after the completion of convertion from a request source is shown in Figure 11.

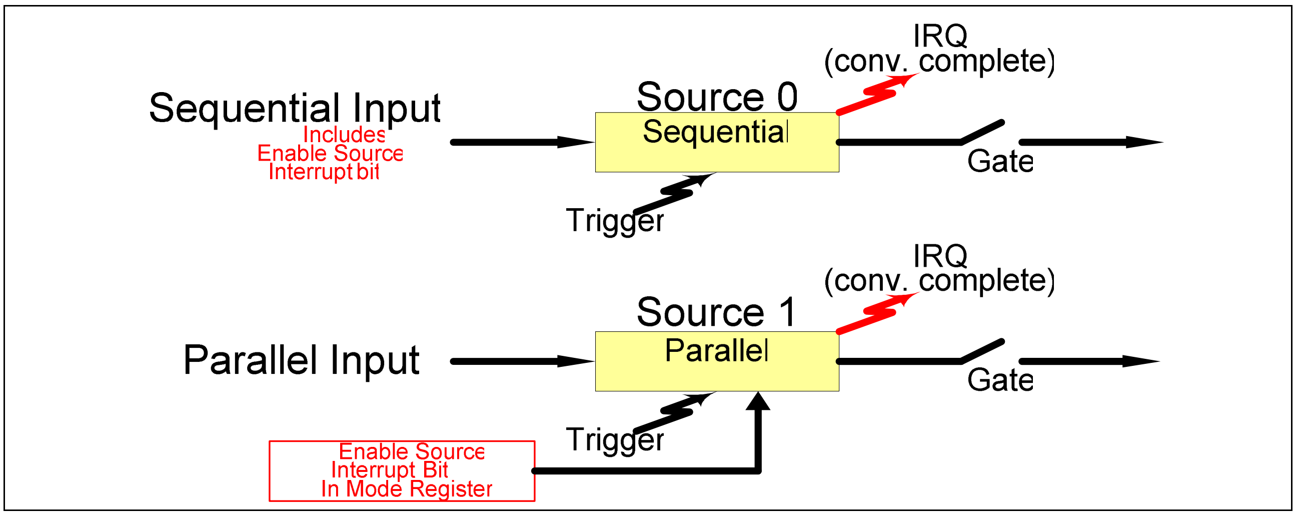


Figure 11 Source Event Interrupt

The result register interrupt generation is shown in Figure 12. This interrupt occurs when the data reducation counter reaches (or already is) zero. Refer the user manual for more detailed explanation.

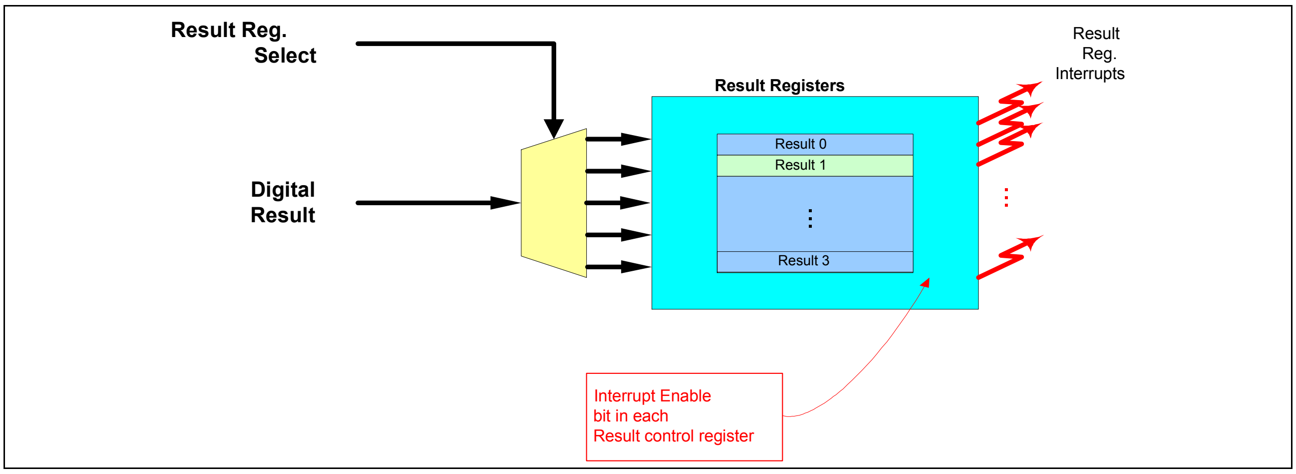


Figure 12 Result register Event interrupt

The channel interrupt that occur through limit checking is shown in Figure 13. More detailed explanation on the limit checking and boundaries can be found in the user manual.

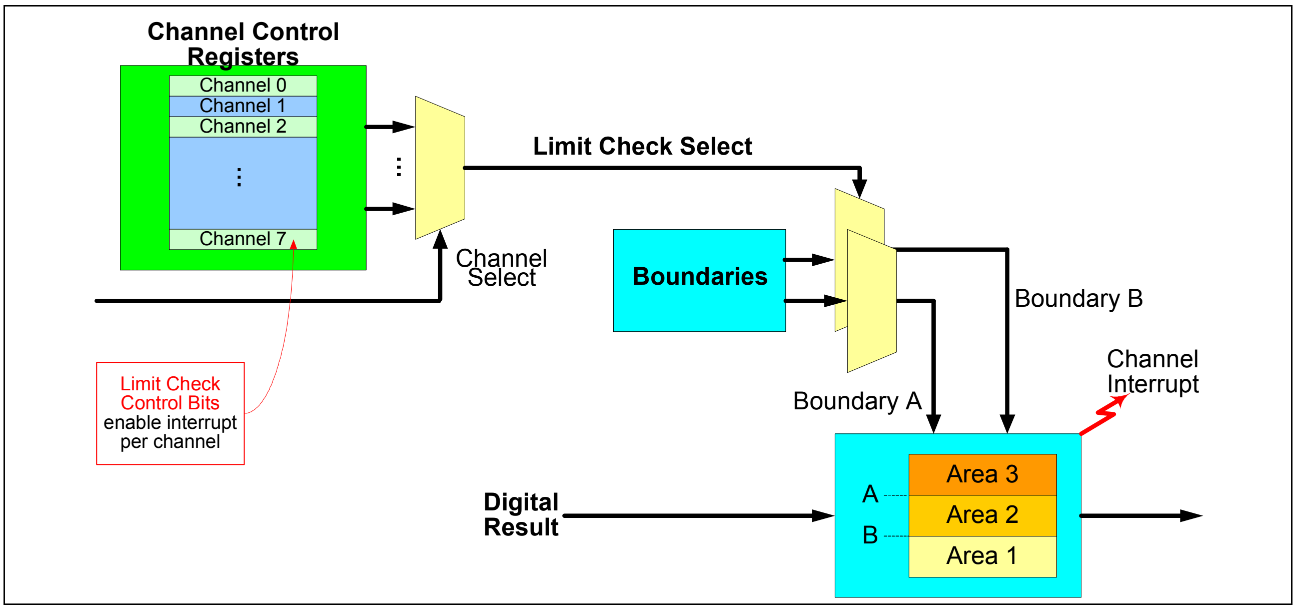


Figure 13 Channel interrupt

2.2.5 External Trigger Inputs

The sequential and parallel request sources each have one external request trigger input REQTRx (x = 0 - 1), through which a hardware triggered conversion request can be started. The input to REQTRx is selected from eight external trigger inputs (ETRx0 to ETRx7) via a multiplexer depending on bit field ETRSELx. It is possible to bypass the synchronization stages for external trigger requests that come synchronous to ADC. This selection is done via bit SYNENx.

One fADCI period is required for synchronization between the conversion start trigger (from the digital part) and the beginning of the sample phase (in the analog part). The BUSY and SAMPLE bits will be set with the conversion start trigger.

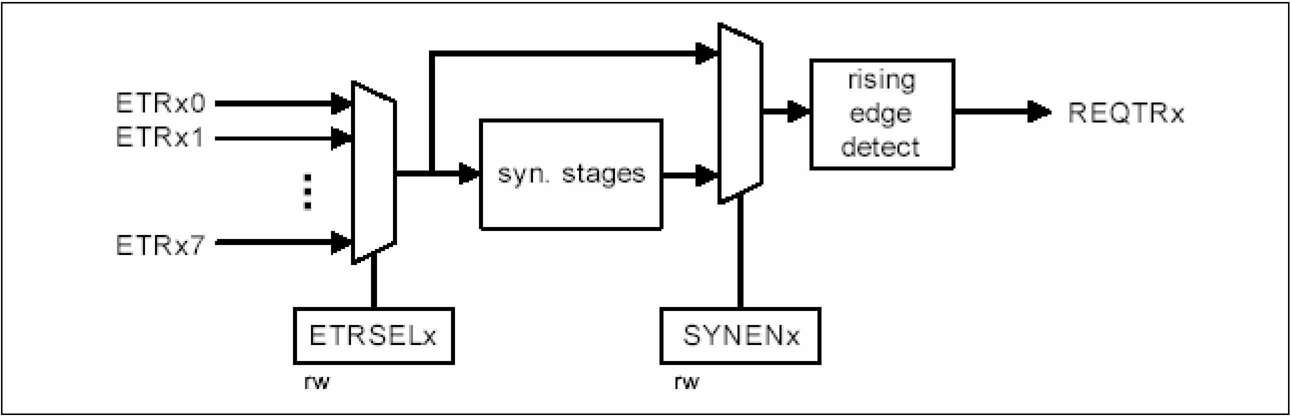


Figure 14 External Trigger Input

The external trigger inputs to the ADC module are driven by events occuring in the CCU6 module (and in T2CCU module in case XC878). This allows ADC conversions to be triggered synchronously to PWM events.

Table 1 External Trigger Input Source

External Trigger Input	CCU6 Event or T2CCU Event
ETRx0	T13 Period Match or CCT Overflow
ETRx1	T13 Compare Match or T2CC5 compare-match
ETRx2	T12 Period Match
ETRx3	T12 Compare match for channel 0
ETRx4	T12 Compare match for channel 1
ETRx5	T12 Compare match for channel 2
ETRx6	Shadow transfer event for multi-channel mode
ETRx7	Correct hall event for multi-channel mode

2.3 How to start an ADC conversion request

The user has the freedom of selecting the channels for conversion and their request sources. Channels 0-7 can be requested for conversion through the sequential request source at arbitration slot 0 whereas only Channels 4-7 can be requested for conversion through the parallel request source at arbitration slot 1.

Also the delay comparison on the ADC conversion start by hardware and software is shown in Figure 15.

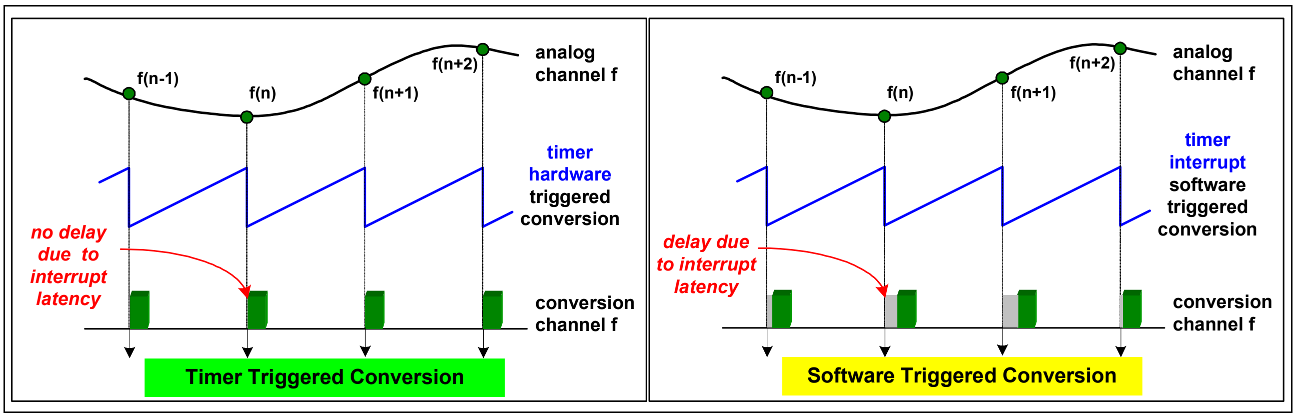

f(n-1)
f(n)
f(n+1)
f(n+2)
analog channel f
timer hardware triggered conversion
no delay due to interrupt latency
conversion channel f
Timer Triggered Conversion
f(n-1)
f(n)
f(n+1)
f(n+2)
analog channel f
timer interrupt software triggered conversion
delay due to interrupt latency
conversion channel f
Software Triggered Conversion

Figure 15 Delay comparison on ADC conversion start

2.3.1 Parallel Request Source:

In XC800 family of microcontrollers, the parallel source can issue conversion requests for a sequence of upto 4 input channels (Channel 4-7). Each channel can be individually enabled to take part in the scan sequence. The scan 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). For example, with the parallel source a single channel (ch 4-7) conversion can be triggered via software or hardware. Or up to 4 channels can be triggered for converision (one after the other with no delay) via software or hardware. E.g. Ch7, Ch6, Ch5, Ch4 or Ch7, Ch5, Ch4, or Ch6, Ch4, etc.

The parallel request source consists of a conversion request control register (CRCR1), a conversion request pending register (CRPR1) and a conversion request mode register (CRMR1). The contents of the conversion request control register are copied (overwrite) to the conversion request pending register when a selected load event (LDE) occurs. The type of the event defines the behavior and the trigger of the request source.

The parallel Request Source Control is depicted in Figure 16.

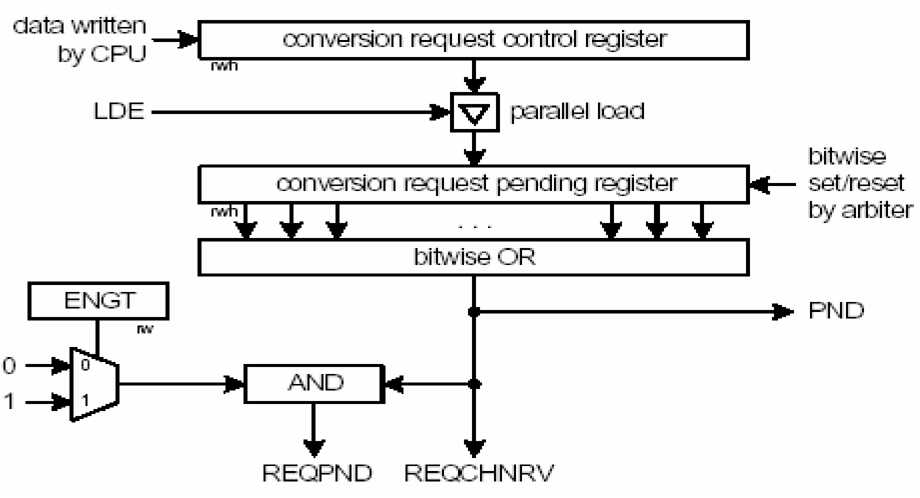

data written by CPU
conversion request control register
rwh
LDE
parallel load
conversion request pending register
rwh
bitwise set/reset by arbiter
. . .
bitwise OR
ENGT
rw
PND
0
1
0
1
AND
REQPND
REQCHNRV

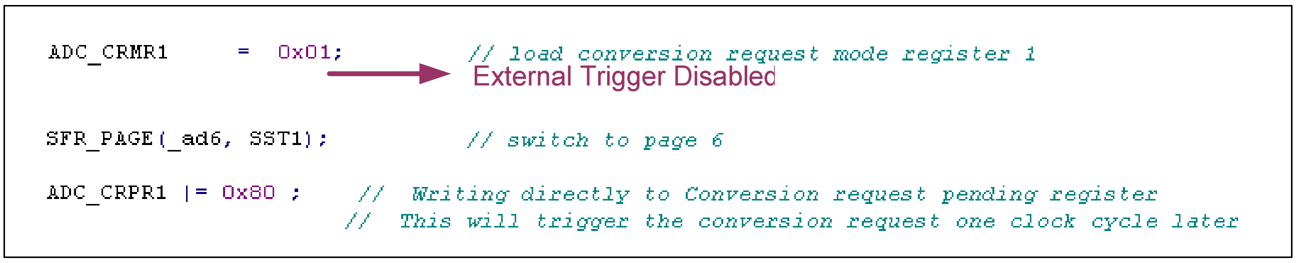
Figure 16 Parallel Request Source Control

2.3.1.1 Start Parallel Request

The channel information for conversion should be written into the CRCR1 or CRPR1. The load event for a parallel load can be started by one of the following ways:

2.3.1.1.1 Conversion Channel write operation to CRPR1

The conversion request control register can be written at two different addresses (CRCR1 and CRPR1). Accessed at CRCR1, the write action changes only the bits in this register. Accessed at CRPR1, a load event will take place one clock cycle after the write access. This automatic load event can be used to start conversions with a single move operation. In this case, the information about the channels to be converted is given as an argument in the move instruction. The following example triggers a single conversion on Channel 7 via software.

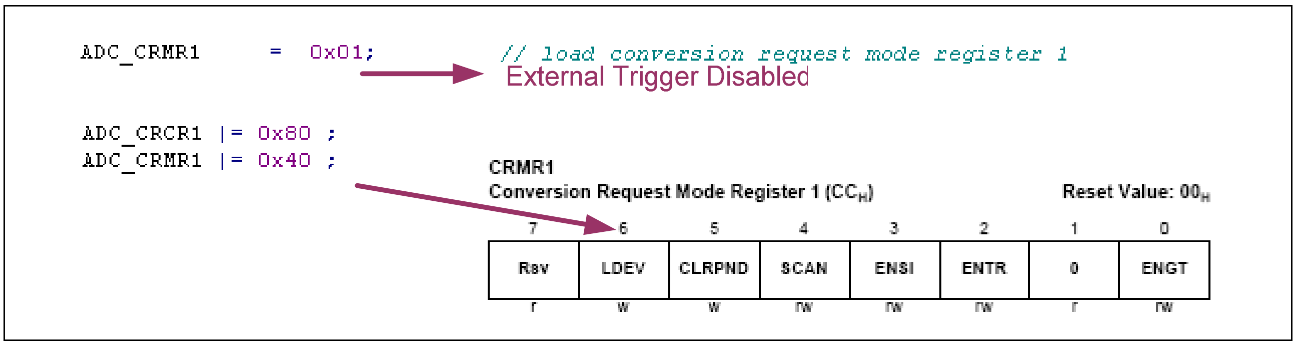


```
ADC_CRMR1     =  0x01;           // load conversion request mode register 1


SFR_PAGE(_ad6, SST1);            // switch to page 6

ADC_CRPR1  |= 0x80 ;     //  Writing directly to Conversion request pending register
                        //  This will trigger the conversion request one clock cycle later
```

Figure 17 Conversion channel write operation at CRPR1

2.3.1.1.2 Software Trigger by setting Load Event bit

Bit LDEV can be written with 1 by software to trigger the load event. In this case, the load event does not contain any information about the channels to be converted, but always takes the contents of the conversion request control register. This allows the conversion request control register to be written at a second address without triggering the load event. The following example also triggers a single conversion on channel 7 via software.



```
ADC_CRMR1     =  0x01;           // load conversion request mode register 1


ADC_CRCR1  |= 0x80 ;
ADC_CRMR1  |= 0x40 ;
```

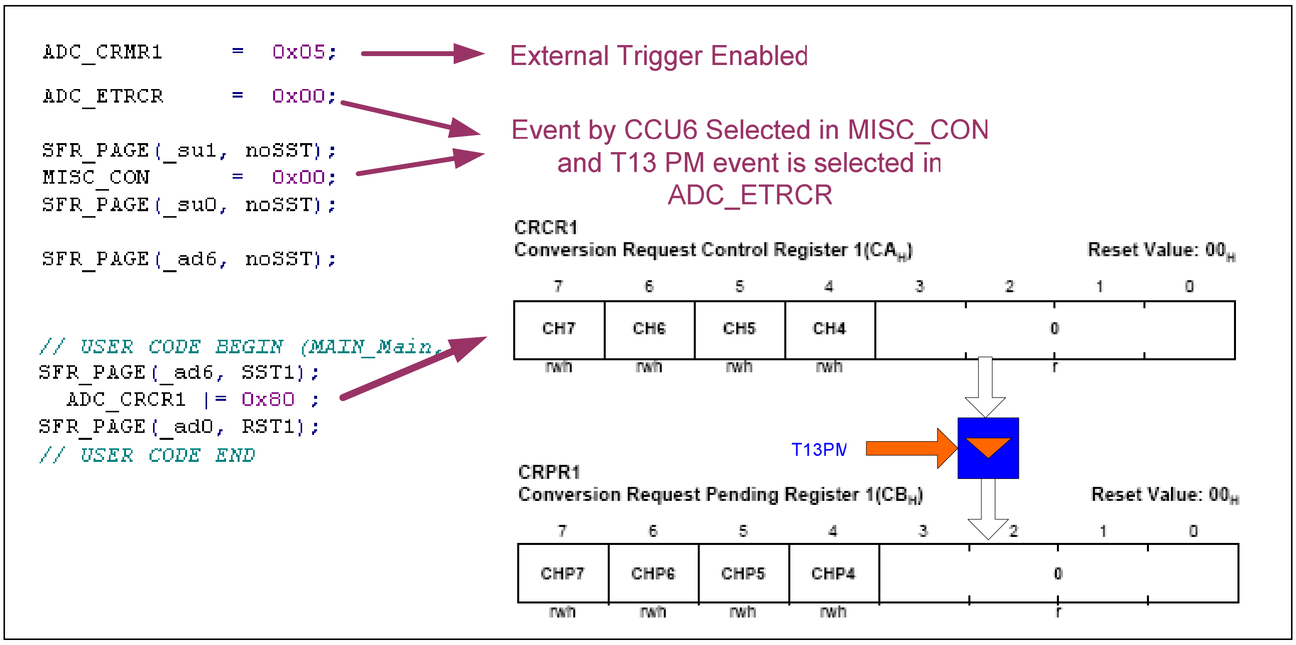
External Trigger Disabled
CRMR1
Conversion Request Mode Register 1 (CC$_H$)
Reset Value: 00$_H$

7	6	5	4	3	2	1	0
Rsv	LDEV	CLRPND	SCAN	ENSI	ENTR	0	ENGT
r	w	w	rw	rw	rw	r	rw

Figure 18 Software Trigget by setting Load Event bit (LDEV = 1)

2.3.1.1.3 External trigger at the input line REQTR. (By Hardware)

A hardware triggered parallel conversion is done by coupling of the reload event to a request trigger input, REQTR. A software example is shown in Figure 6 for conversion request on CCU6 T13 PM Event (Timer 13 Period Match). On a CCU6 T13 PM event, the contents of CRCR1 register will be copied to the contents of CRPR1 register and a parallel conversion (in this example channel 7 only) will be triggered.



```
ADC_CRMR1      =  0x05;
ADC_ETRCR      =  0x00;

SFR_PAGE(_su1, noSST);
MISC_CON       =  0x00;
SFR_PAGE(_su0, noSST);

SFR_PAGE(_ad6, noSST);


// USER CODE BEGIN (MAIN_Main,
SFR_PAGE(_ad6, SST1);
   ADC_CRCR1  |= 0x80 ;
SFR_PAGE(_ad0, RST1);
// USER CODE END
```

Figure 19 Parallel source conversion request by external CCU6 T13 PM event

2.3.1.1.4 Conversion completed and PND = 0 for Autoscan mode

After the parallel conversion(s) are completed the parallel source can be automatically triggered again using the autoscan mode. In this case software only needs to trigger the first conversion. Successive convertions are triggered automatically. The following example enables the autoscan mode so channel 7 is converted continuously.

```
    ///  - the gating line is permanently 1
    ///  - the external trigger is disabled
    ///  - the source interrupt is disabled
    ///  - the autoscan functionality is enabled

    ADC_CRMR1     =  0x11;         // load conversion

    SFR_PAGE(_ad6, SST1);

    ADC_CRPR1  |= 0x80 ;
```

External Trigger Disabled and Autoscan mode is enabled

First time trigger from hardware or software is required for autoscan mode and mere write operation to CRCR1 will not trigger any conversion and in this example write operation to CRPR1 is done so as to initiate the conversion at the beginning

Figure 20 Conversion completed and PND = 0 for Autoscan mode

2.3.2 Sequential Request Source

The sequential request source at arbitration slot 0 requests one conversion after another for channel numbers between 0 and 7. The queue stage stores the requested channel number and some additional control information. As a result, the order in which the channels are to be converted is freely programmable without restrictions in the sequence. The only restriction is that the length of the sequence is limited by the number of queue stages in the sequential request source. The XC886 and XC864 have only one queue stage, however the XC886, XC888 and XC878 each have a 4 stage queue. The additional control information is used to enable the request source interrupt (when the requested channel conversion is completed) and to enable the automatic refill process.

On a device with 4 queue stages (e.g. the XC878) the sequential request source can trigger up to 4 conversions in a row (triggered by hardware or software) in any order. For example a sequence like Ch 2, Ch1, Ch7, and Ch1 (again) can be programmed into the queue.

A sequential source consists of one or more queue stages, a backup stage (QBUR0) and a mode control register (QMR0).

The input register for the queue is the write only register QINR0. If there is still an empty queue stage (V = 0), the value written to QINR0 will be added to the queue (bit V becomes set if the queue is full). If the queue is full the write action is ignored.

The Sequential Request Source Control is depicted in the following Figure 21.

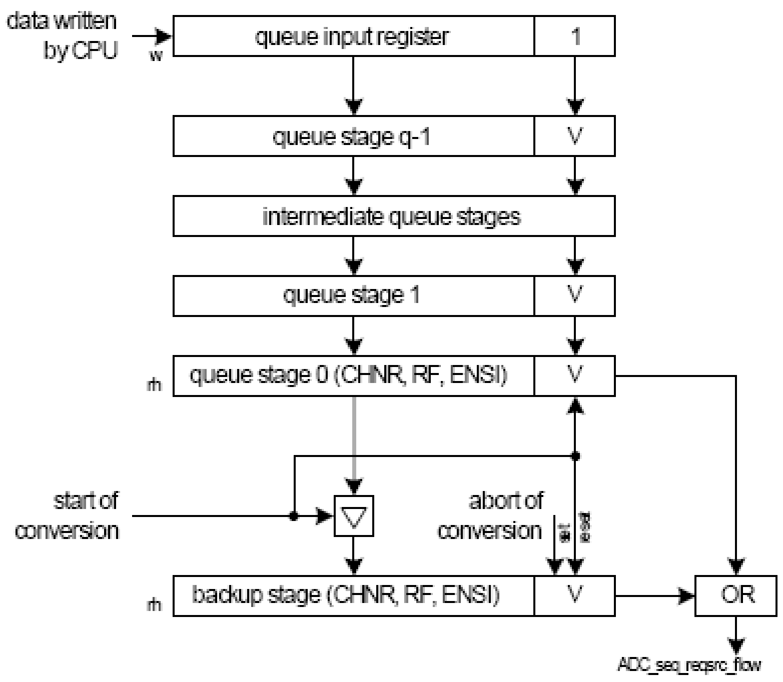


Figure 21 Structure of Sequential Request Source

2.3.2.1 Start Sequential Request

The sequential source conversion can be started by software or hardware:

2.3.2.1.1 Source Conversion Request not related to External trigger event (software triggered conversions)

External triggers are disabled when bit EXTR=0. In this case the valid bit V = 1 directly requests the conversion by setting the signals REQPND and REQCHNRV to 1. No conversion will be requested if V = 0. A gating mechanism allows the user to enable/disable the conversion requests according to bit ENGT. The V bit gets set by writing to the queue input register QINR0.

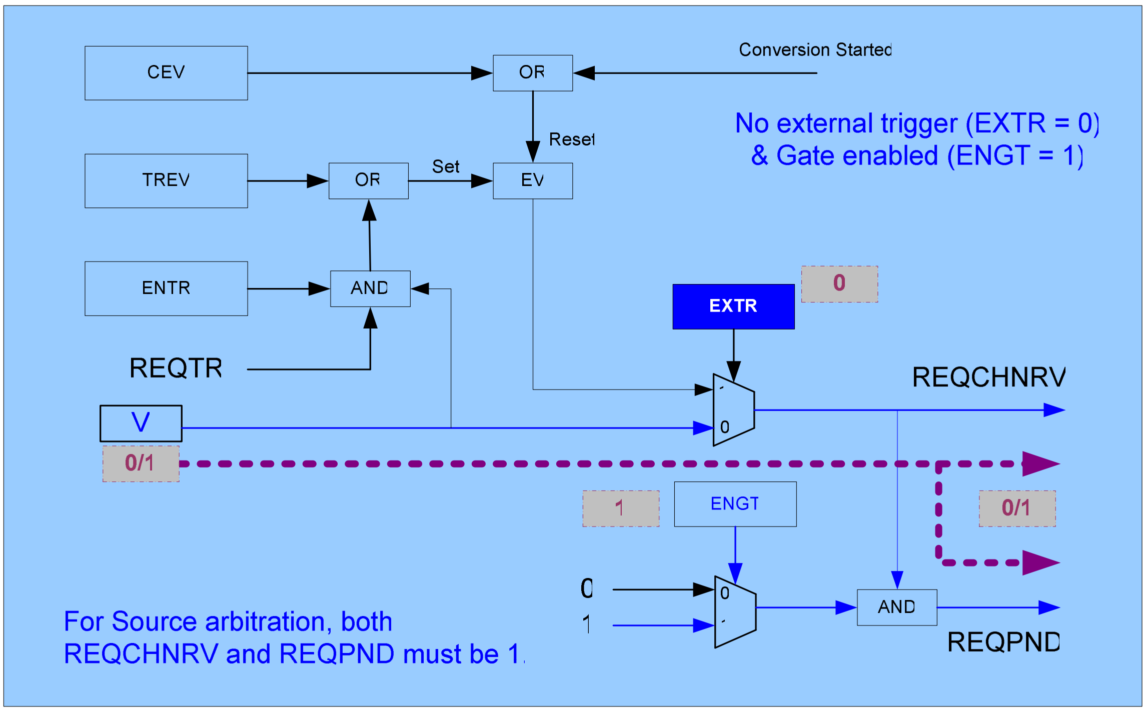
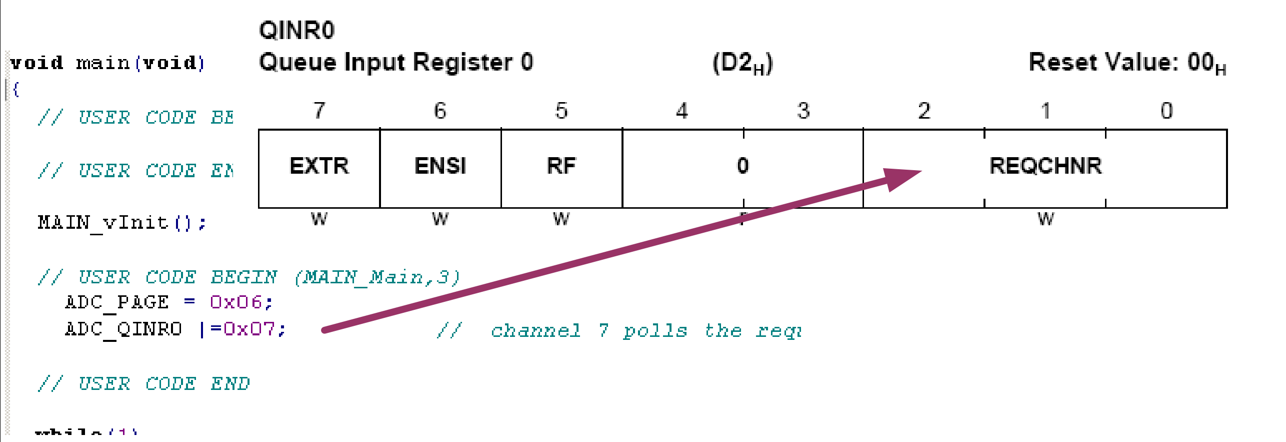

CEV
OR
Conversion Started
Reset
No external trigger (EXTR = 0)
& Gate enabled (ENGT = 1)
TREV
OR
Set
EV
ENTR
AND
0
EXTR
REQTR
REQCHNRV
V
0
0/1
1
ENGT
0/1
0
1
0
AND
REQPND
For Source arbitration, both
REQCHNRV and REQPND must be 1.

Figure 22 Source Conversion Request not related to External event

An example code for a software triggered start of sequential source conversion request for channel 7 is shown in Figure 23.


```
void main(void)
{
  // USER CODE BE

  // USER CODE EN

  MAIN_vInit();

  // USER CODE BEGIN (MAIN_Main,3)
    ADC_PAGE = 0x06;
    ADC_QINR0 |=0x07;              //  channel 7 polls the req

  // USER CODE END
```

QINR0
Queue Input Register 0 (D2$_H$) Reset Value: 00$_H$

7	6	5	4	3	2	1	0
EXTR	ENSI	RF	0		REQCHNR		
w	w	w	r		w		

Figure 23 Example code for Simple Sequential Source conversion start request

2.3.2.1.2 Source conversion request related to External trigger event

Event triggered by Software:

An alternate way to trigger a sequential conversion request by software is to setup the channel number and set ENGT when writing to queue input register QINRO. The conversion can then be triggered by setting bit TREV in register QMR0.

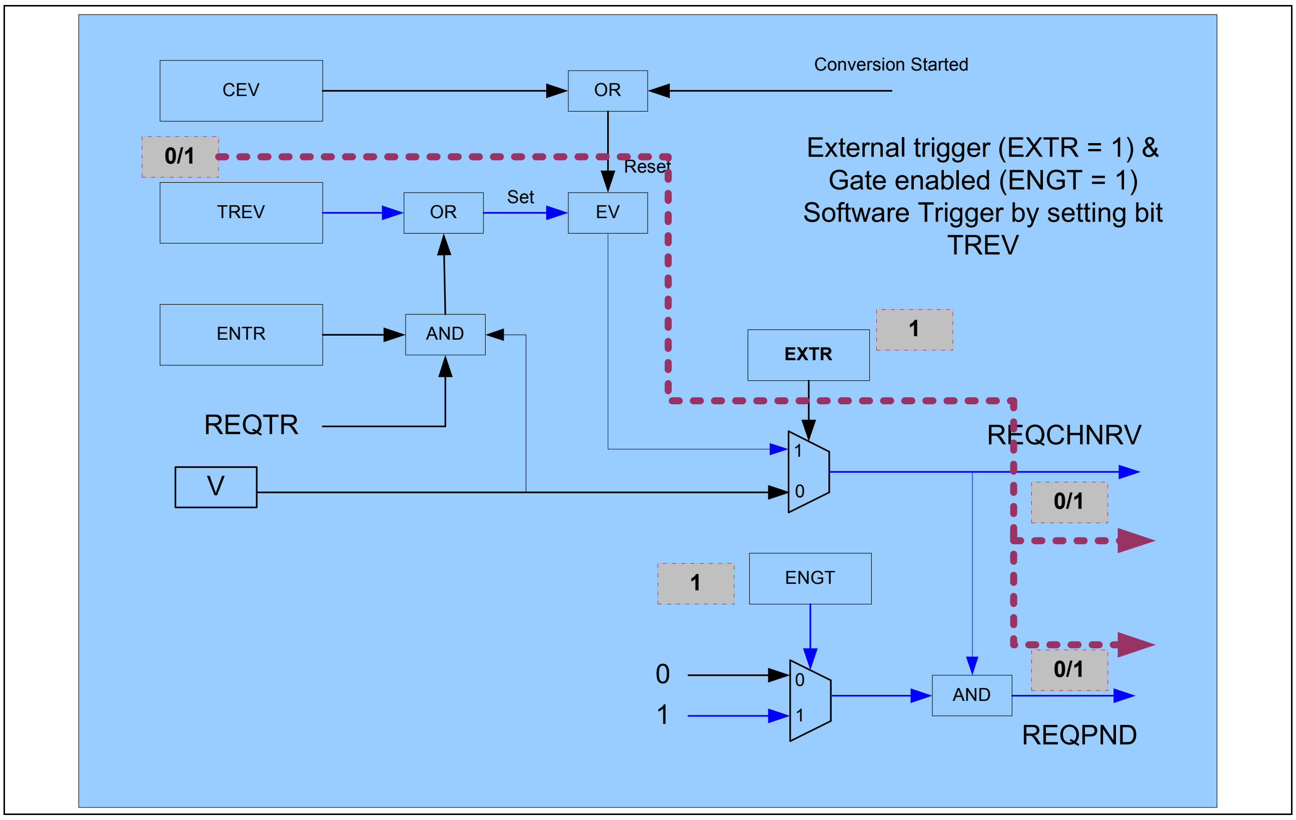
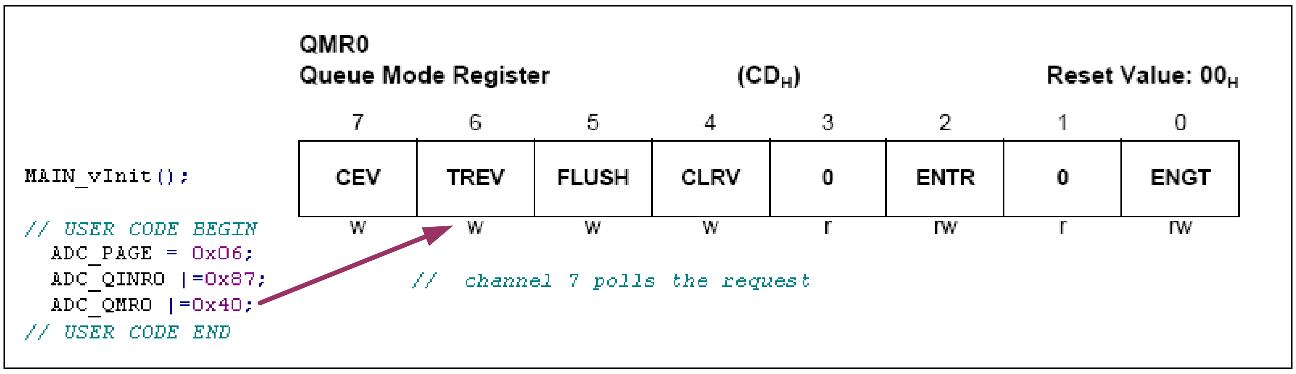


Figure 24 Sequential source conversion request by external trigger software event

An example code for the External Trigger by software is shown in Figure 25.



QMR0
Queue Mode Register **(CD_H)** **Reset Value: 00_H**

7	6	5	4	3	2	1	0
CEV	TREV	FLUSH	CLRV	0	ENTR	0	ENGT
w	w	w	w	r	rw	r	rw

```
MAIN_vInit();

// USER CODE BEGIN
  ADC_PAGE = 0x06;
  ADC_QINRO |=0x87;
  ADC_QMRO |=0x40;
// USER CODE END
```

```
// channel 7 polls the request
```

Figure 25 Example code for sequential source conversion request by software event

Event triggered by CCU6 event or T2CCU event (Hardware Triggered Conversions):

The external trigger event signal REQTR, which can be set by CCU6 or T2CCU (XC878 only) events, can also trigger a sequential conversion (bit ENTR = 1 and V=1 must also be set). The conversion will not be requested if V=0. V is set when the queue is not empty. The exact source of the trigger is specified by the ADC_ETRCR register and the MIS_CON register.

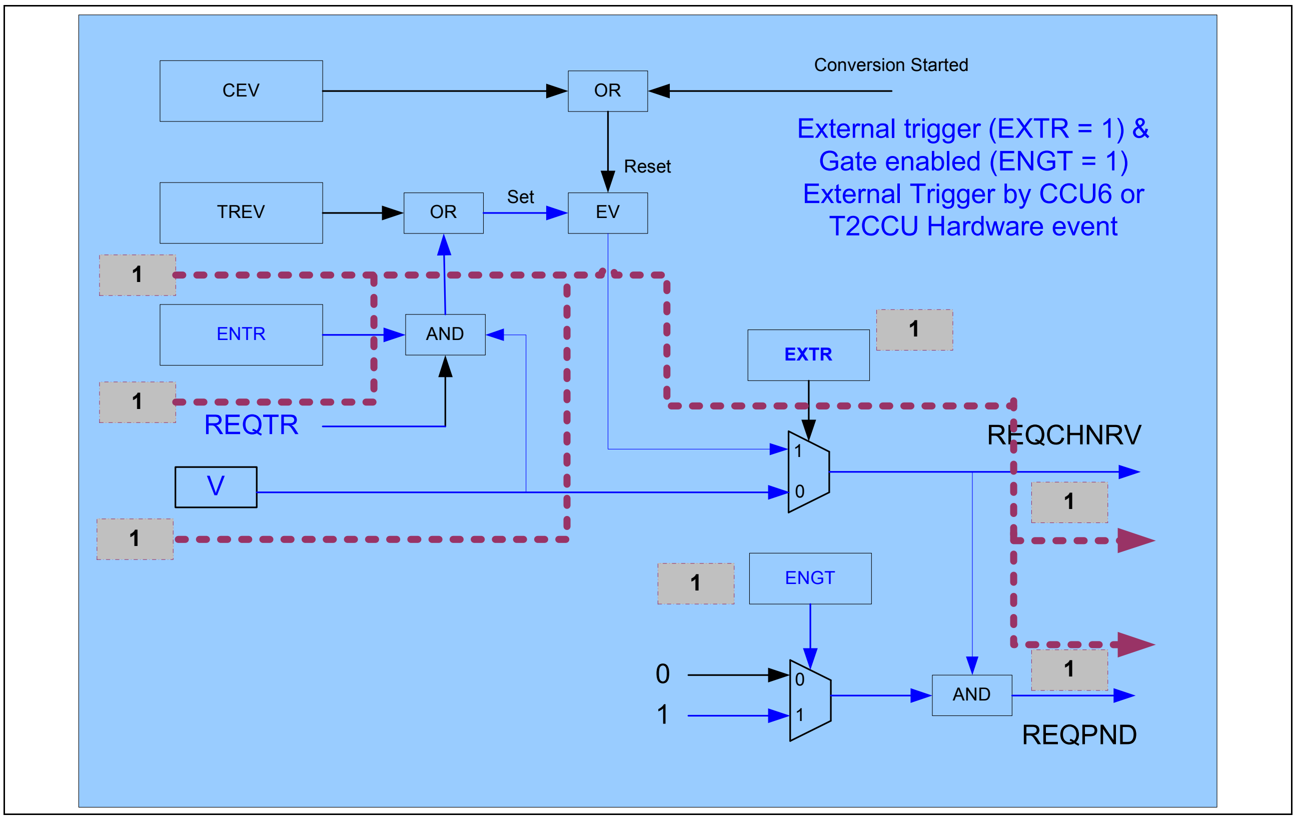
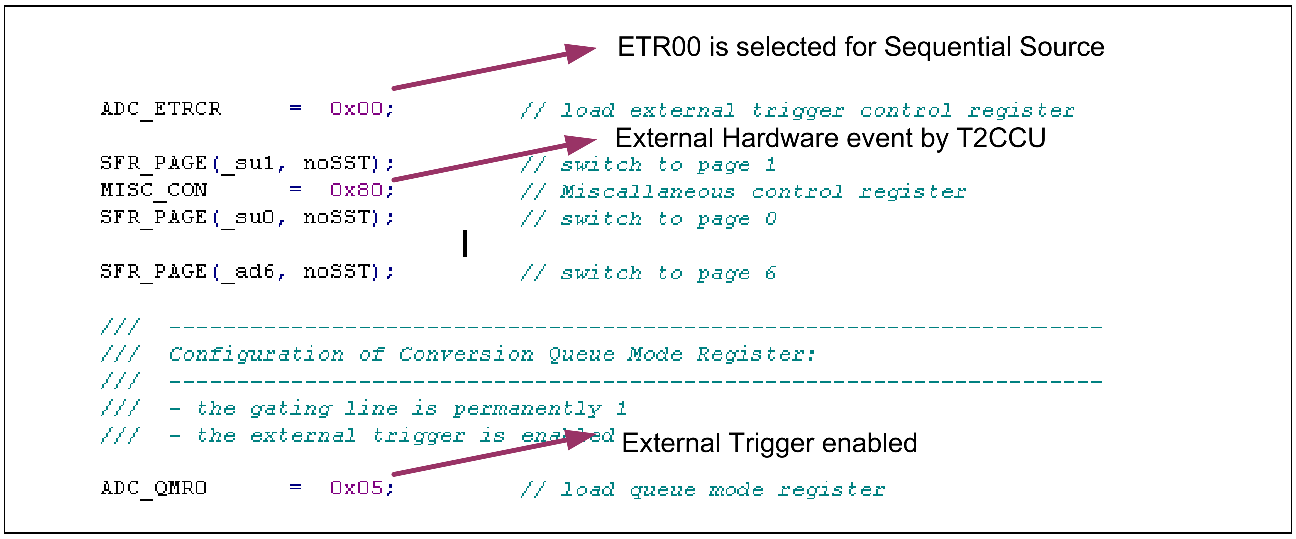

CEV
OR
Conversion Started
Reset
External trigger (EXTR = 1) & Gate enabled (ENGT = 1) External Trigger by CCU6 or T2CCU Hardware event
TREV
OR
Set
EV
1
ENTR
AND
EXTR
1
1
REQTR
REQCHNRV
V
1
1
1
ENGT
1
0
1
AND
1
REQPND

Figure 26 Sequential source conversion request by external trigger hardware event

An example code for Sequential source conversion request of external trigger event by T2CCU hardware event is shown in Figure 27.


ETR00 is selected for Sequential Source
External Hardware event by T2CCU
External Trigger enabled

```
ADC_ETRCR      =  0x00;          // load external trigger control register

SFR_PAGE(_su1, noSST);           // switch to page 1
MISC_CON       =  0x80;          // Miscallaneous control register
SFR_PAGE(_su0, noSST);           // switch to page 0

SFR_PAGE(_ad6, noSST);           // switch to page 6

///  -------------------------------------------------------------------
///  Configuration of Conversion Queue Mode Register:
///  -------------------------------------------------------------------
///  - the gating line is permanently 1
///  - the external trigger is enabled

ADC_QMR0       =  0x05;          // load queue mode register
```

Figure 27 Sequential source conversion request by T2CCU hardware event

3 Application development using ADC module

3.1 Parallel Request Source Control (Examples):

3.1.1 Single Channel and Multichannel – Normal / AutoScan conversions

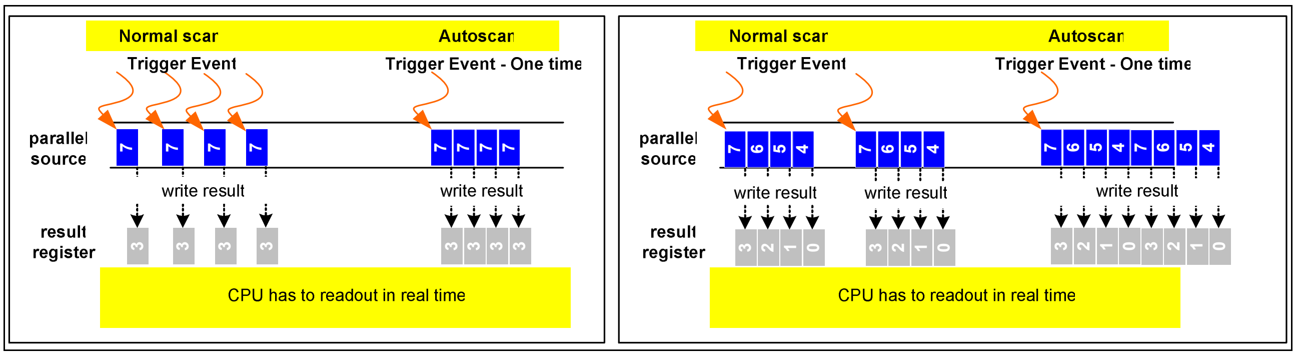


Figure 28 Single and Multichannel Normal/Autoscan conversions

The single (left diagram) and multichannel (right diagram) ADC conversions in normal and autoscan mode are shown in Figure 28.

For a single channel or multichannel conversion the channel number(s) for conversion is configured in CRCR1 register. There is one bit in this register for each of the channels that are allowed for parallel conversions (ch 7 – ch 4). A software or hardware event must trigger the conversion(s). This can be done either by setting the bit LDEV in CRMR1 by software or by using a CCU6 external hardware event [In case of XC878 a T2CCU external hardware event also possible]. Alternatively software can write the channel number for conversion to CRPR1 instead of the CRCR1 register and this will automatically start a conversion request in the next clock cycle.

In normal scan mode, for every new conversion request a trigger event should occur. But in autoscan mode the conversion request trigger operation is needed only to trigger the first conversion. And this can be done by any of the above mentioned procedures. And in case of CCU6, the user can opt for the single shot mode of Timer T13.

In this example wait for read mode is not enabled so the newly converted data will overwrite old data in the result register irrespective of whether the data was previously read or not.

3.1.2 Single Channel and Multichannel – Normal Scan – Wait for Read

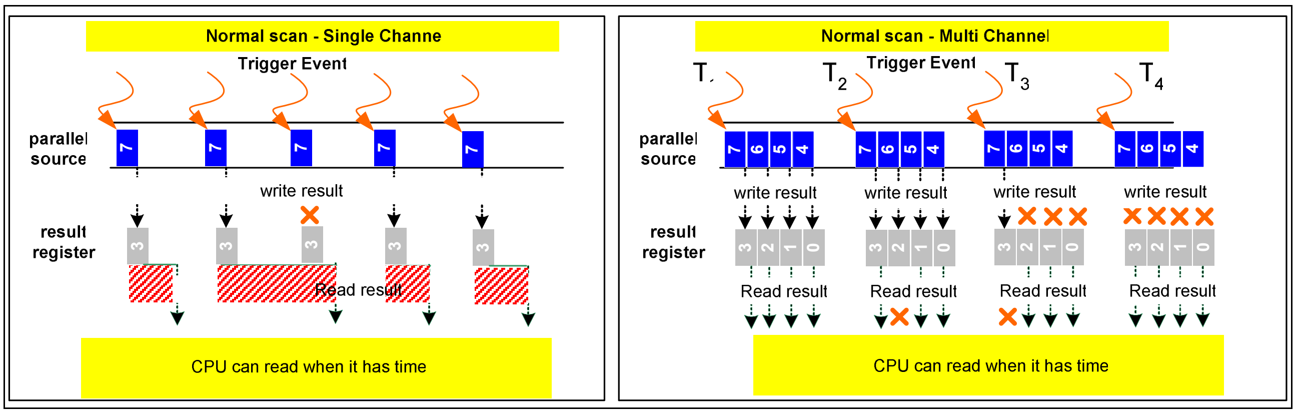


Figure 29 Single and Multi Channel Normal Scan – Wait for Read Mode

In Single channel Wait for read mode, if the result is not read from the result register, then the next conversion is not possible even if the conversion request is triggered. The conversion is possible only after the result was read.

In Multichannel conversion, if the result register of a channel is not read then that channel and the following channel conversions are not possible in the next conversion cycle. However the conversions of higher numbered channels are possible. In the above example the result register 2 of channel 6 is not read and therefore in the next scan sequence the analog to digital conversion of channel 6, 5 and 4 are not possible whereas the channel 7 conversion is possible. Similarly if the result register 3 of ADC channel 7 is not read then the conversion of all other channels is not possible.

3.1.3 Single Channel and Multi Channel – Autoscan – Wait for Read

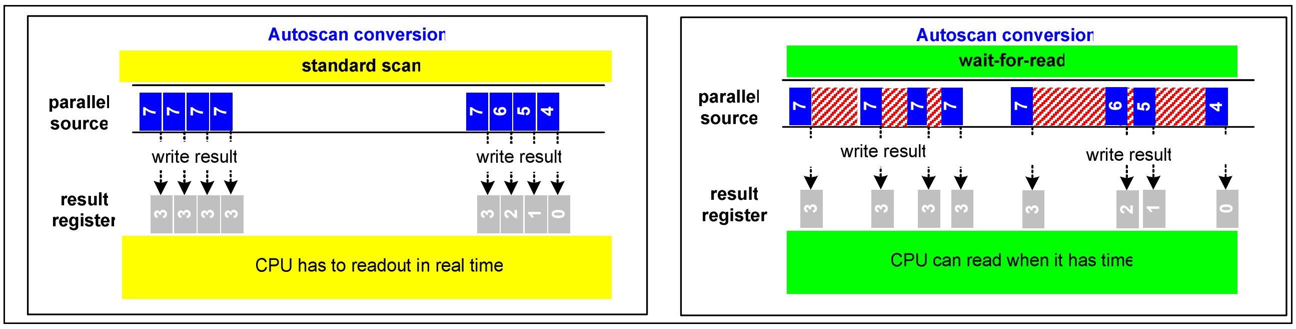


Figure 30 Autoscan conversion modes

For Autoscan conversion, the very first time the channel conversion information has to be written at the CRPR1 (start conversion) or CRCR1 (with setting the LDEV bit in CRMR1). Once this is done, the Parallel source will continuously scan the specified channels. The result of each conversion will be written to the result registers and there is the possibility of data loss if the CPU cannot read each result. To avoid data loss, the user can select the wait-for-read option which will halt the next ADC channel conversion till the data is read by the CPU from the result register.

3.1.4 Timer interrupt H/W triggered Multichannel conversion

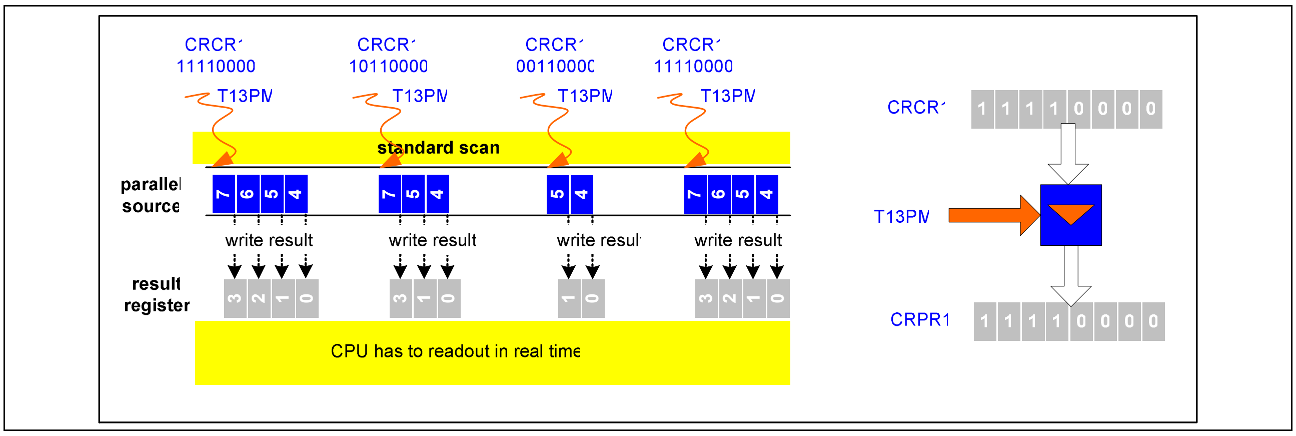


Figure 31 Timer triggered Multichannel conversion (CCU6 Timer 13 Period Match event)

In this example every CCU6 Timer 13 Period match the contents of the CRCR1 register will be copied to the CRPR1. A conversion will occur on the requeseted channels in order of channel number (highest channel number first).

3.1.5 Timer Interrupt S/W triggered Multichannel conversion

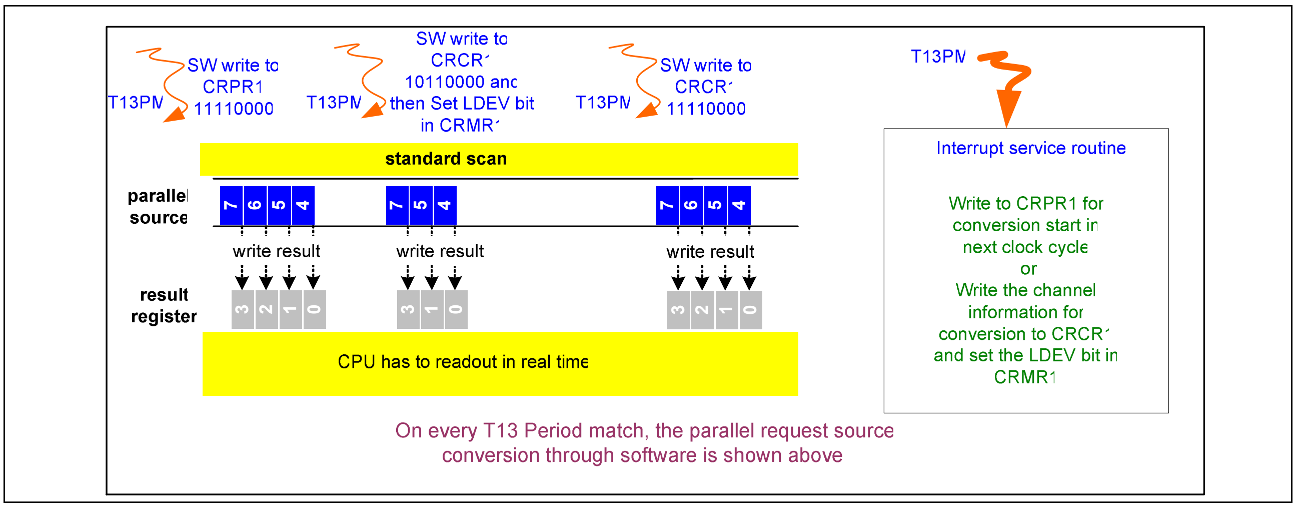

T13PM
SW write to CRPR1 11110000
T13PM
SW write to CRCR1 10110000 and then Set LDEV bit in CRMR1
T13PM
SW write to CRCR1 11110000
T13PM
standard scan
parallel source
7 6 5 4
7 5 4
7 6 5 4
write result
write result
write result
result register
3 2 1 0
3 1 0
3 2 1 0
CPU has to readout in real time
Interrupt service routine
Write to CRPR1 for conversion start in next clock cycle or Write the channel information for conversion to CRCR1 and set the LDEV bit in CRMR1
On every T13 Period match, the parallel request source conversion through software is shown above

Figure 32 Software Triggered Multichannel conversion

On CCU6 Timer 13 Period match interrupt, the software trigger is used for the purpose of the conversion request start. Writing the channel conversion information to the register CRPR1 will lead to a write operation at register CRCR1 and data will be copied to the CRPR1 in the next immediate clock cycle with the automatic trigger event. On the otherhand writing the channel conversion information to the register CRCR1 needs the manual trigger by software and this can be done by setting the LDEV bit in the CRMR1 register.

3.2 Sequential Request Source Control (Examples)

3.2.1 Single Channel and Multichannel Conversion – Normal Scan

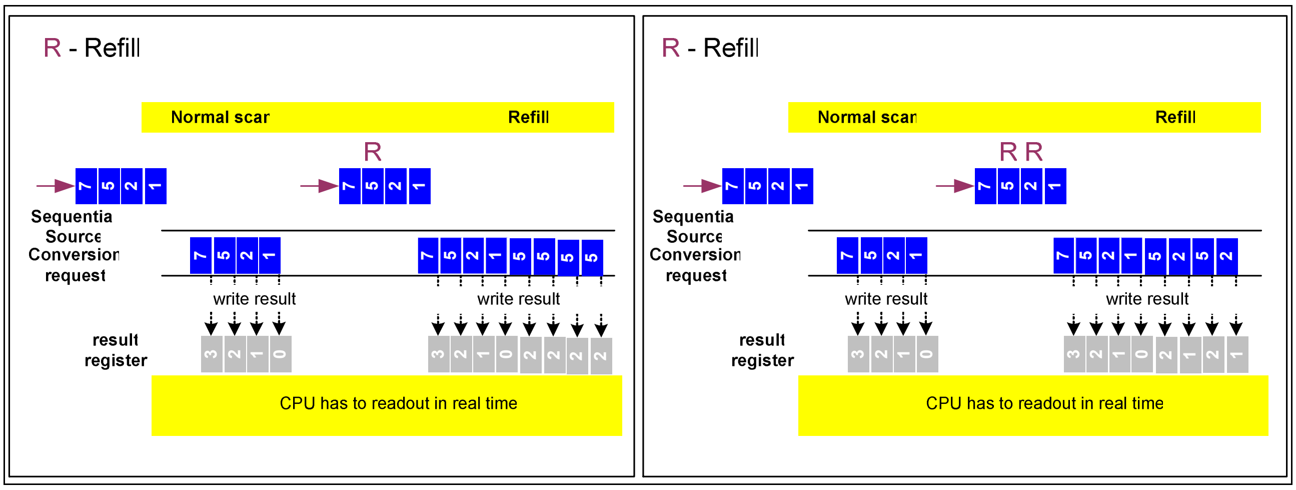

R - Refill
Normal scan
Refill
R
Sequential Source Conversion request
write result
result register
CPU has to readout in real time
R - Refill
Normal scan
Refill
R R
Sequential Source Conversion request
write result
result register
CPU has to readout in real time

Figure 33 Sequential source single channel and multichannel conversion – Normal scan

The single channel and multichannel conversions of sequential source is shown in Figure 33.

In example 1, the sequential source conversion is requested for channels 7, 5, 2 and 1 in sequence. The queue stage stores the channel conversion information. The conversion will take place in the order in which the conversion is requested.

In the same example, the sequential source conversion is requested for channels 7, 5, 2 and 1 in sequence with refill process enabled for channel 5. After the completion of conversions for 7, 5, 2 and 1, the channel conversion for channel 5 will be initiated continuously thereafter as the refill process is enabled.

In example 2, the sequential source conversion is requested for channels 7, 5, 2 and 1 in sequence with refill process enabled for channels 5 and 2. After the completion of conversions for 7, 5, 2 and 1, the channel conversion for channels 5 and 2 will be initiated continuously thereafter as the refill process is enabled.

3.2.2 Single Channel and Multichannel Conversions – Wait for Read

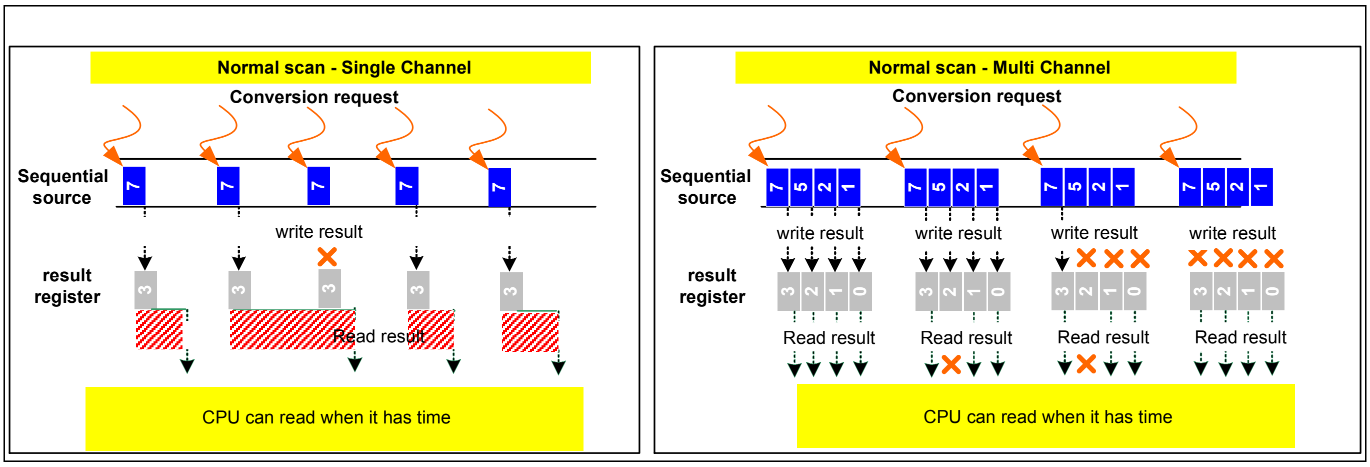


Figure 34 Sequential source single and multi channel conversion – Wait for Read mode

In Single channel Wait for read mode, if the result is not read from the result register, then the next conversion is not possible even if the conversion request is triggered. The conversion is possible only after the result was read.

In Multichannel conversion, if the result register of a channel is not read then the following conversion requests will be ignored. In the above example the result register 2 of channel 5 is not read and therefore in the next scan sequence the conversion of channel 7 will happen whereas for 5, 2 and 1 conversion are not possible. This is shown in Figure 34.

.

Appendix A ADC Register map

The ADC SFRs are located in the standard memory area (RAMP=0) and are organized into 7 pages. The ADC_PAGE register is located at address $D1_H$. It contains the page value and page control information. The addresses of the ADC SFRs are listed below.

Table 2 SFR Address List for Pages 0-3

Address	Page 0	Page 1	Page 2	Page 3
CA_H	GLOBCTR	CHCTR0	RESR0L	RESRA0L
CB_H	GLOBSTR	CHCTR1	RESR0H	RESRA0H
CC_H	PRAR	CHCTR2	RESR1L	RESRA1L
CD_H	LCBR	CHCTR3	RESR1H	RESRA1H
CE_H	INPCR0	CHCTR4	RESR2L	RESRA2L
CF_H	ETRCR	CHCTR5	RESR2H	RESRA2H
$D2_H$	-	CHCTR6	RESR3L	RESRA3L
$D3_H$	-	CHCTR7	RESR3H	RESRA3H

Table 3 SFR Address List for Pages 4-7

Address	Page 4	Page 5	Page 6	Page 7
CA_H	RCR0	CHINFR	CRCR1	-
CB_H	RCR1	CHINCR	CRPR1	-
CC_H	RCR2	CHINSR	CRMR1	-
CD_H	RCR3	CHINPR	QMR0	-
CE_H	VFCR	EVINFR	QSR0	-
CF_H	-	EVINCR	Q0R0	-
$D2_H$	-	EVINSR	QBUR0/QINR0	-
$D3_H$	-	EVINPR	-	-

Appendix B ADC Initialization sequence

The following steps are meant to provide a general guideline on how to initialize the ADC module. Some steps may be varied or omitted depending on the application requirements:

Step 1) Configure global control functions:

a. Select conversion width (GLOBCTR.DW)

b. Select analog clock *f*ADCI divider ratio (GLOBCTR.CTC)

Step 2) Configure arbitration control functions:

a. Select priority level for request source x (PRAR.PRIOx)

b. Select conversion start mode for request source x (PRAR.CSMx)

c. Enable arbitration slot x (PRAR.ASENx)

d. Select arbitration mode (PRAR.ARBM)

Step 3) Configure channel control information:

a. Select limit check control for channel x (CHCTRx.LCC)

b. Select target result register for channel x (CHCTRx.RESRSEL)

c. Select sample time for all channels (INPCR0.STC)

Step 4) Configure result control information:

a. Enable/disable data reduction for result register x (RCRx.DRCTR)

b. Enable/disable event interrupt for result register x (RCRx.IEN)

c. Enable/disable wait-for-read mode for result register x (RCRx.WFR)

d. Enable/disable valid flag reset by read access for result register x (RCRx.VFCTR)

Step 5) Configure interrupt control functions:

a. Select channel x interrupt node pointer (CHINPR.CHINPx)

b. Select event x interrupt node pointer (EVINPR.EVINPx)

Step 6) Configure limit check boundaries:

a. Select limit check boundaries for all channels (LCBR.BOUND0, LCBR.BOUND1)

Step 7) Configure external trigger control functions:

a. Select source x external trigger input (ETRCR.ETRSELx)

b. Enable/disable source x external trigger input synchronization (ETRCR.SYNENx)

Step 8) Setup sequential source:

a. Enable conversion request (QMR0.ENGT)

b. Enable/disable external trigger (QMR0.ENTR)

Step 9) Setup parallel source:

a. Enable conversion request (CRMR1.ENGT)

b. Enable/disable external trigger (CRMR1.ENTR)

c. Enable/disable source interrupt (CRMR1.ENSI)

d. Enable/disable autoscan (CRMR1.SCAN)

Step 10) Turn on analog part:

a. Set GLOBCTR.ANON (wait for 100 ns)

Step 11) Start sequential request:

a. Write to QINR0 (with information such as REQCHNR, RF, ENSI and EXTR)

Step 12) Start parallel request:

a. Write to CRCR1 (no load event) or CRPR1 (automatic load event) the channels to be converted

b. Generate a load event (if not already available) to trigger a pending conversion request

Step 13) Wait for ADC conversion to be completed:

a. The source interrupt indicates that the conversion requested by the source is completed.

b. The channel interrupt indicates that the corresponding channel conversion is completed (with limit check performed).

c. The result interrupt indicates that the result (with/without accumulation) in the corresponding result register is ready and can be read.

Step 14) Read ADC result

Appendix C ADC for Motor Control

1 Field Oriented Control of PMSM Motor

In order to estimate the rotor position by a single shunt measurement, the PWM pattern generation and the triggering of the ADC for current measurement must be very fast and accurate. Any jitter in the triggerpoint will influence the actual rotor's angle estimation. As a result, the total harmonic distortion of the sinusoidal current signals will increase (see ApNote ap0805910_Sensorless_FOC.pdf for details). The ADC provides in total four result registers, from which two are used to hold the appropriate DC-link current values. The 10 bit ADC is configured in a way that the measurement result of channel 3 and 4 is stored in result register 2 and 3, respectively. This requires the DC-link current to be available at both channels. As a result, the ADC conversion result is stored in separate result registers and can be used for calculations at any time independent of the modulation angle.

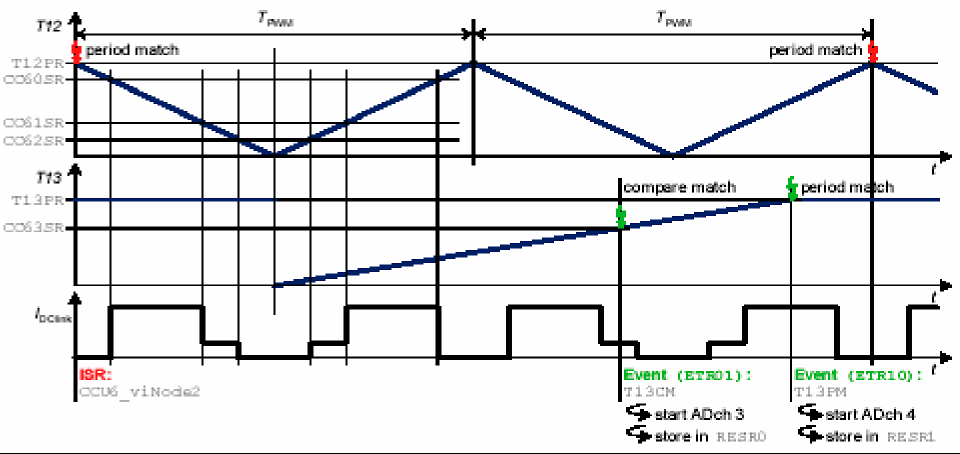


Figure 35 Field Oriented Control – ADC Channel usage

The ADC conversions are consecutively started every second period of the modulation. The first measurement is triggered by the compare match of CC63 and sampled by ADC channel 3. The second measurement by ADC channel 4 is triggered by the period match of T13. ADC channel 3 is used to measure the positive currents (segment b of in **Figure** 36), channel 4 measures the negative ones (segment e). Sampling is triggered in the center of each segment. Please refer to **Figure 35** for detailed timing of the current measurement. This method requires very fast switching and sampling times in order to ensure a proper current measurement. The time slot for the phase current measurement is limited by the sum of sampling time Tsample and deadtime Tdeadtime. Due to this restriction, the current can not be measured at the crossing of two sectors of the space vector modulation. Therefore the minimum time for the voltage space vector is limited to the minimum time for current measurement. Here, the measurements of ADC channel 3 and 4 are evaluated by reading the result register 0 and 1.

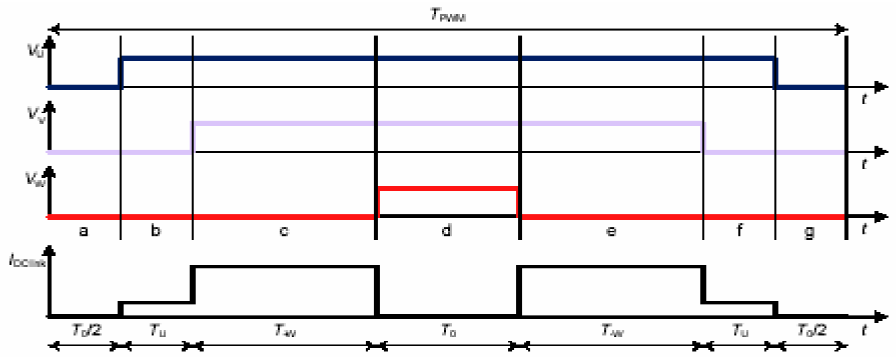


Figure 36 Voltage Space Vector – Sector A

2 Sensorless BLDC motor control

The main task in the case of Sensorless BLDC operation is the zero crossing detection using the ADC. This requires that the A/D converter samplings are synchronized with the PWM. This can be achieved with the aid of the external trigger feature of the A/D converter. Several events of CCU6 can be used to start the sampling and hold of A/D conversion, such as Timer 12/13 compare match and period match. In this example Timer T12 is used to measure the commutation time and T13 is used for PWM generation. Upon every Timer 13 period match, the corresponding AD channel is triggered to measure back emf voltage and generate a interrupt. In this ADC interrupt service routine, the captured voltage is compared with half the dc rail voltage. If the voltage value falls into the range of half the dc rail voltage considering some margin, then the calculation for one half of the time between two zero crossings will be conducted.

T12PM_ISR: Timer 12 period match interrupt service routine, in which commutation is conducted.

ADC_ISR: ADC interrupt service routine, which is triggered by Timer 13 period match. In this ISR, if correct zero-crossing is detected, Timer 12 period will be modified accordingly, to predict next commutation point.

cz: period from commutation to zero-crossing.

zc: period from zero-crossing to commutation.

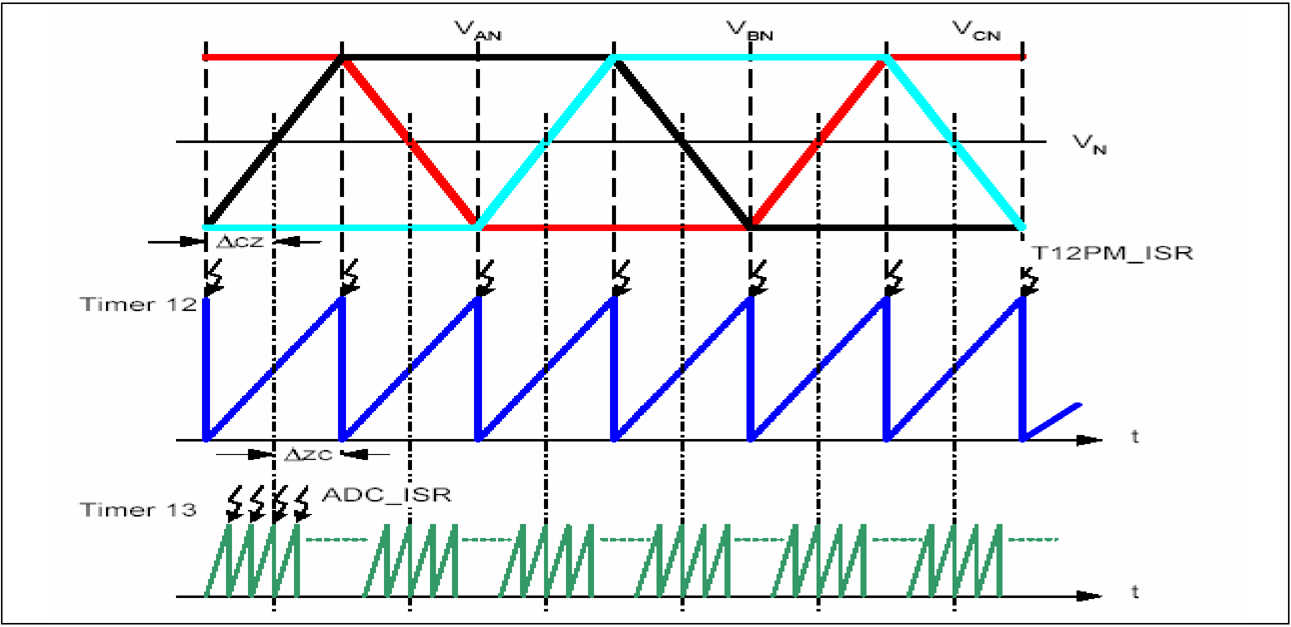


Figure 37 Sensorless BLDC operation – ADC triggering

Appendix C Software Examples

Refer the additional examples available with the Application note:

The examples are provided for XC878 microcontroller and grouped as follows:

1. Sequential request source
2. Parallel request source
3. Combined sequential & parallel request source