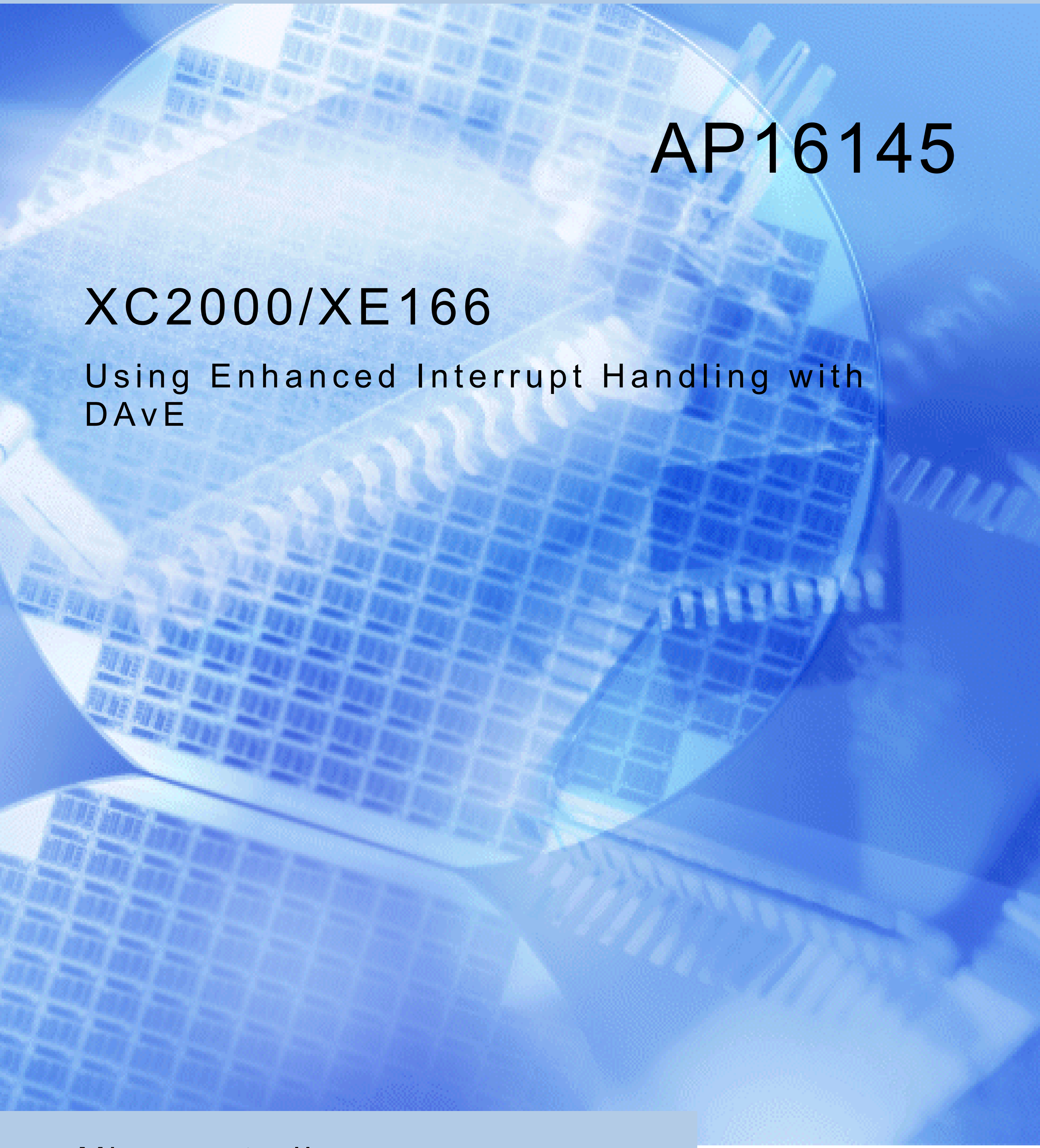
Application Note, V1.0, Jul. 2008



AP16145

XC2000/XE166

Using Enhanced Interrupt Handling with DAvE

Microcontrollers

Never stop thinking

Edition 2008-08-07

Published by
Infineon Technologies AG
81726 München, Germany

AP16145

Revision History:	2008-07	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

The architecture of the XC2000 supports several mechanisms for fast and flexible response to service requests from various sources internal or external to the microcontroller. Different kinds of exceptions are handled in a similar way:

- Interrupts generated by the interrupt controller (ITC).
- DMA transfers issued by the Peripheral Event Controller (PEC).
- Traps caused by the TRAP instruction or issued by faults or specific system states.

The XC2000 family fits perfectly in embedded applications. The application note ap1611110_interrupt_response_time.pdf gives detailed information about the real time capabilities of the interrupt architecture. This can be available at Interrupt Response Time of the XC2000/XE166 Family (ap1611111_interrupt_response_time.pdf).

This application note demonstrates the support of enhanced interrupt handling in DAvE (Digital Application virtual Engineer).

2 Interrupt System Structure

The XC2000 architecture provides 96 separate interrupt nodes assignable to 16 priority levels, with 8 sub-levels (group priority) on each level. These nodes can be activated by several source requests.

The XC2000 architecture provides a vectored interrupt system. Whenever a request occurs, the CPU branches to the location that is associated with the respective interrupt source. This allows direct identification of the source which caused the request.

All pending interrupt requests are arbitrated. The arbitration winner is indicated to the CPU together with its priority level and action request. The CPU triggers the corresponding action based on the required functionality (normal interrupt, PEC, jump table cache, etc.) of the arbitration winner.

An action request will be accepted by the CPU if the requesting source has a higher priority than the current CPU priority level and interrupts are globally enabled. If the requesting source has a lower (or equal) interrupt level priority than the current CPU task, it remains pending.

The block diagram in **Figure 1** shows the dedicated registers in the Interrupt Handler Control section, which are mainly used for enhanced interrupt handling.

The dedicated registers are available under Fast Bank Switching and Interrupt Jump Table Cache blocks.

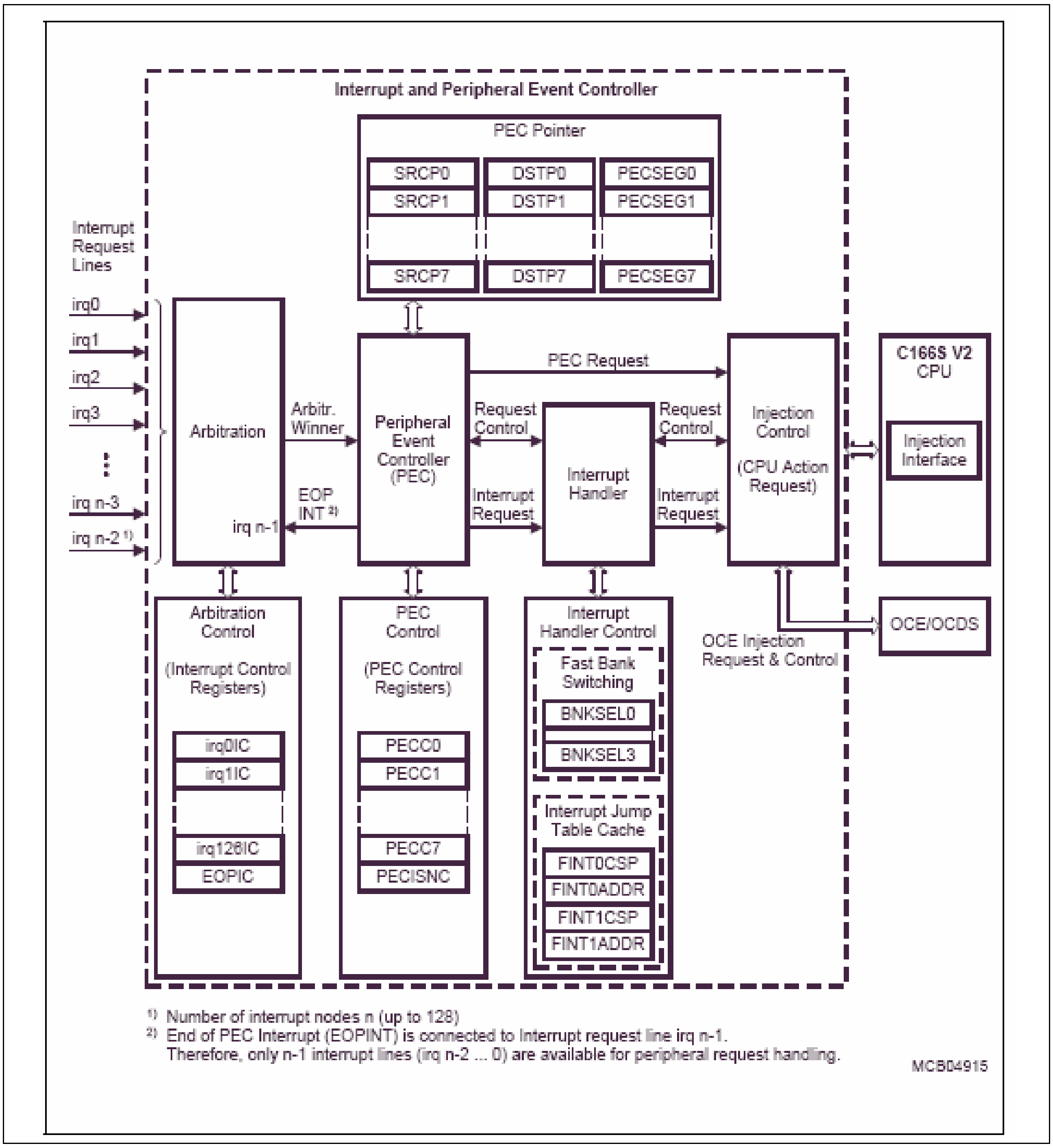

Interrupt and Peripheral Event Controller
PEC Pointer
SRCP0
SRCP1
SRCP7
DSTP0
DSTP1
DSTP7
PECSEG0
PECSEG1
PECSEG7
Interrupt Request Lines
irq0
irq1
irq2
irq3
irq n-3
irq n-2 1)
Arbitration
irq n-1
Arbitr. Winner
EOP INT 2)
Peripheral Event Controller (PEC)
PEC Request
Request Control
Interrupt Request
Interrupt Handler
Request Control
Interrupt Request
Injection Control (CPU Action Request)
C166S V2 CPU
Injection Interface
OCE/OCDS
OCE Injection Request & Control
Arbitration Control (Interrupt Control Registers)
irq0IC
irq1IC
irq126IC
EOPIC
PEC Control (PEC Control Registers)
PECC0
PECC1
PECC7
PECISNC
Interrupt Handler Control
Fast Bank Switching
BNKSEL0
BNKSEL3
Interrupt Jump Table Cache
FINT0CSP
FINT0ADDR
FINT1CSP
FINT1ADDR
1) Number of interrupt nodes n (up to 128)
2) End of PEC Interrupt (EOPINT) is connected to Interrupt request line irq n-1. Therefore, only n-1 interrupt lines (irq n-2 ... 0) are available for peripheral request handling.
MCB04915

Figure 1 Block diagram of the Interrupt and PEC controller

3 Enhanced Interrupt Handling

The XC2000 architecture offers a couple of dedicated registers to configure the interrupt handler. The configuration can be divided into two groups:

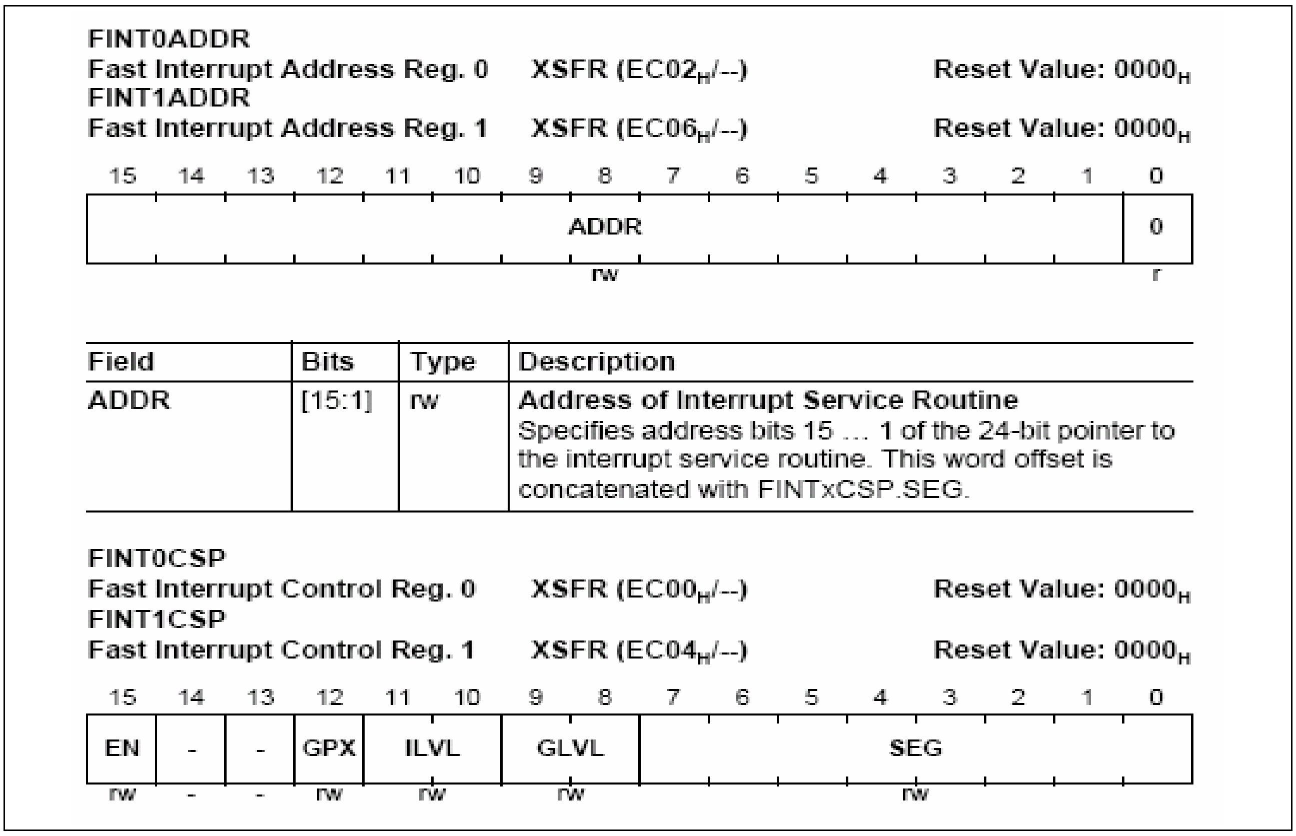
- Interrupt jump table cache.
- Fast bank switching.

3.1 Interrupt Jump Table Cache

Servicing an interrupt request via the vector table usually incurs two subsequent branches: an implicit branch to the vector location and an explicit branch to the actual service routine. The interrupt servicing time can be reduced by the Interrupt Jump Table Cache (ITC, also called “fast interrupt”). This eliminates the second explicit branch to the ISR by directly providing CPU with the service routine location.

The ITC provides two 24-bit pointers, so the CPU can directly branch to the respective routines. These fast interrupts can be selected for two interrupt sources on priority levels 15…12.

The two pointers are each stored in a pair of interrupt jump table cache registers (FINTxADDR, FINTxCSP), which store an 8 bit pointer segment and a 16 bit offset along with the priority level (priority level 12-15).



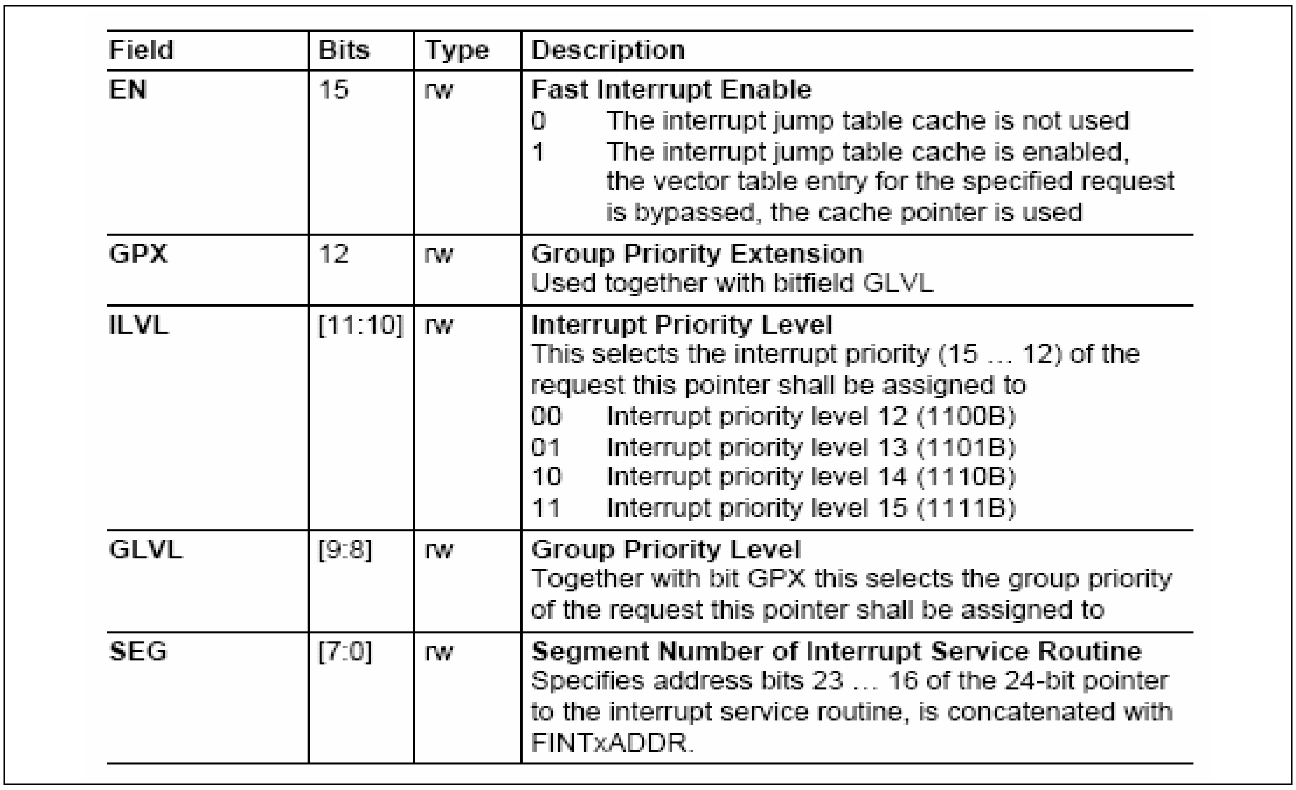
FINT0ADDR
Fast Interrupt Address Reg. 0 XSFR (EC02$_H$/--) Reset Value: 0000$_H$
FINT1ADDR
Fast Interrupt Address Reg. 1 XSFR (EC06$_H$/--) Reset Value: 0000$_H$

15–1	0
ADDR	0
rw	r

Field	Bits	Type	Description
ADDR	[15:1]	rw	**Address of Interrupt Service Routine** Specifies address bits 15 … 1 of the 24-bit pointer to the interrupt service routine. This word offset is concatenated with FINTxCSP.SEG.

FINT0CSP
Fast Interrupt Control Reg. 0 XSFR (EC00$_H$/--) Reset Value: 0000$_H$
FINT1CSP
Fast Interrupt Control Reg. 1 XSFR (EC04$_H$/--) Reset Value: 0000$_H$

15	14	13	12	11–10	9–8	7–0
EN	-	-	GPX	ILVL	GLVL	SEG
rw	-	-	rw	rw	rw	rw



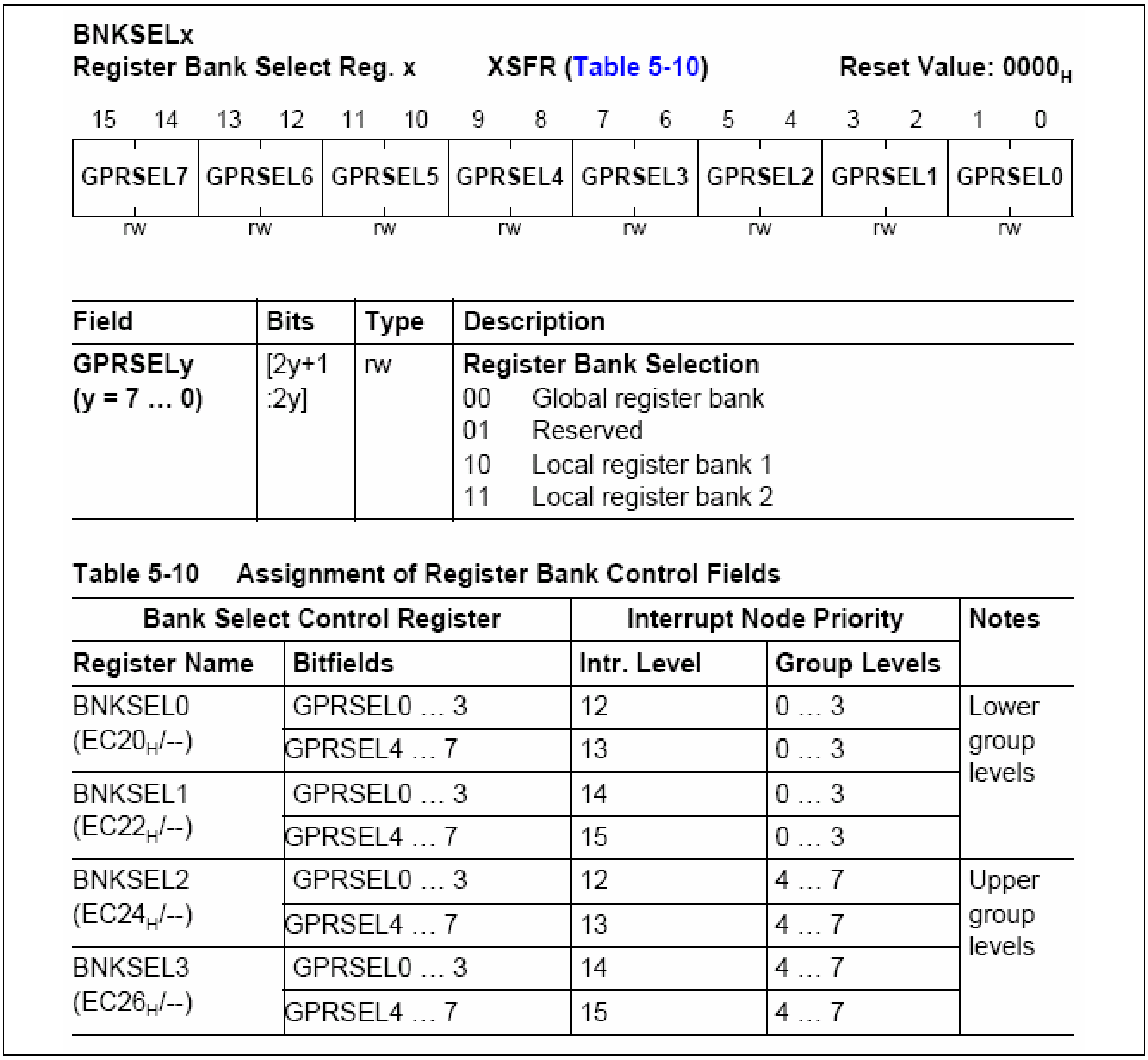
Field	Bits	Type	Description
EN	15	rw	**Fast Interrupt Enable** 0 The interrupt jump table cache is not used 1 The interrupt jump table cache is enabled, the vector table entry for the specified request is bypassed, the cache pointer is used
GPX	12	rw	**Group Priority Extension** Used together with bitfield GLVL
ILVL	[11:10]	rw	**Interrupt Priority Level** This selects the interrupt priority (15 … 12) of the request this pointer shall be assigned to 00 Interrupt priority level 12 (1100B) 01 Interrupt priority level 13 (1101B) 10 Interrupt priority level 14 (1110B) 11 Interrupt priority level 15 (1111B)
GLVL	[9:8]	rw	**Group Priority Level** Together with bit GPX this selects the group priority of the request this pointer shall be assigned to
SEG	[7:0]	rw	**Segment Number of Interrupt Service Routine** Specifies address bits 23 … 16 of the 24-bit pointer to the interrupt service routine, is concatenated with FINTxADDR.

Figure 2 Jump Table Cache Registers

3.2 Fast Bank Switching

The XC2000 architecture allows switching of the selected physical register bank. By updating the bitfield BANK in register PSW the active register bank is switched. In case of an interrupt service, the bank switch is automatically executed by updating the bitfield BANK from BNKSELx in the interrupt controller. For interrupt priority levels 12-15 the target register bank can be pre-selected.

The registers BNKSELx provides a 2-bit field for each possible arbitration priority level. The respective bitfield is then copied to bitfield BANK in register PSW to select the register bank, as soon as the respective interrupt request is accepted. After a switch to a local register bank, the new bank is immediately available.



BNKSELx
Register Bank Select Reg. x **XSFR (Table 5-10)** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPRSEL7		GPRSEL6		GPRSEL5		GPRSEL4		GPRSEL3		GPRSEL2		GPRSEL1		GPRSEL0	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
GPRSELy **(y = 7 … 0)**	[2y+1 :2y]	rw	**Register Bank Selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 Local register bank 2

Table 5-10 Assignment of Register Bank Control Fields

Bank Select Control Register		Interrupt Node Priority		Notes
Register Name	**Bitfields**	**Intr. Level**	**Group Levels**	
BNKSEL0 (EC20$_H$/--)	GPRSEL0 … 3	12	0 … 3	Lower group levels
	GPRSEL4 … 7	13	0 … 3	
BNKSEL1 (EC22$_H$/--)	GPRSEL0 … 3	14	0 … 3	
	GPRSEL4 … 7	15	0 … 3	
BNKSEL2 (EC24$_H$/--)	GPRSEL0 … 3	12	4 … 7	Upper group levels
	GPRSEL4 … 7	13	4 … 7	
BNKSEL3 (EC26$_H$/--)	GPRSEL0 … 3	14	4 … 7	
	GPRSEL4 … 7	15	4 … 7	

Figure 3 Registers BNKSEL0-3 for Interrupt Pirority Level 12-15

4 Enhanced Interrupt Handling in DAvE

The example shown below illustrates handling of interrupts for ADC0 module in DAvE using XC2287 starter kit with Keil v6.10 compiler. The DAvE generated code for different compilers (Keil, Tasking Classic and Tasking Viper) is also shown at the end to illustrate difference.

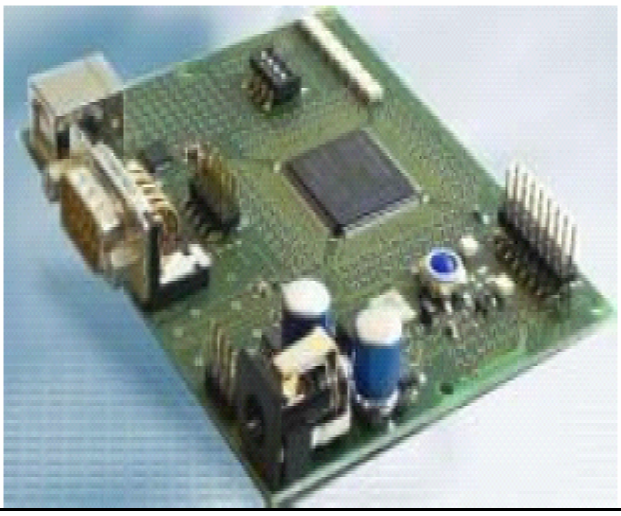


Figure 4 XC2287 Starter Kit

4.1 Description on Example

The example configures ADC0 request sources 0, 1 and 2. But request sources 0 and 1 are used for conversion.

The channels 0 and 1 are used for converting input analog signals. The channel 0 interrupt is configured, which uses interrupt node pointer 0 (SRN0).

This interrupt is placed at Group 0 and Level 12 in interrupts page, which is again placed under Local Reg Bank1 in Bank Select page.

The request source 0 (Sequential source 0) interrupt is enabled and this is configured for interrupt node pointer 1 (SRN1).

This interrupt is placed at Group 1 and Level 13 in interrupts page, which is again placed under Local Reg Bank2 in Bank Select page.

The request source 1 (Parallel source 1) interrupt is enabled and this is configured for interrupt node 3 (SRN3).

This interrupt is placed at Group 7 and Level 15 in Int. Extension page, which is again placed at Fast Interrupt 1 under Fast Interrupts section in Bank Select page.

The result register 0 interrupt is enabled and this is configured for interrupt node pointer 2 (SRN2).

This interrupt is placed at Group 6 and Level 14 in Int. Extension page, which is again placed at Fast Interrupt 0 under Fast Interrupts section in Bank Select page.

Here, both ADC0 sources Sequential source 0 and Parallel source 1 are triggered through software.The converted results of channel 0 and channel 1 are available in result registers 0 and 1 respectively.

1. Select ADC Clock from XC2287

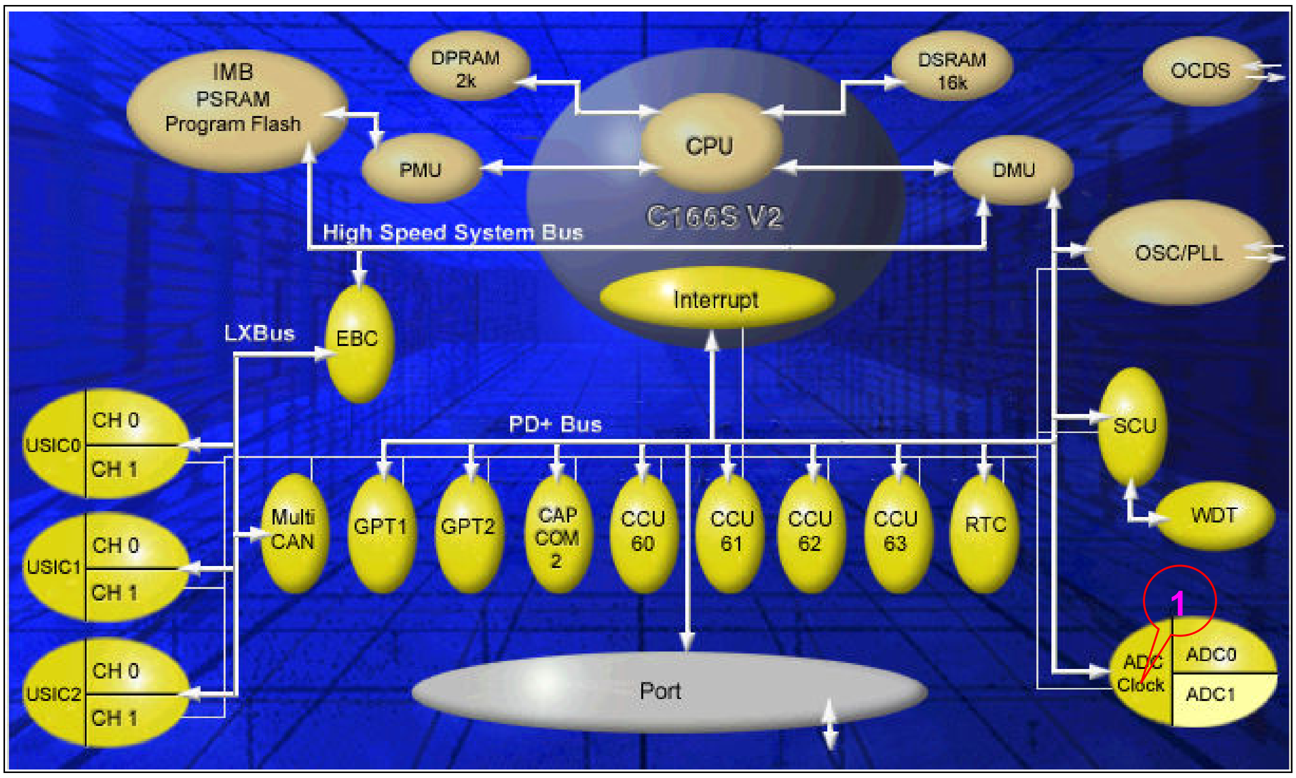
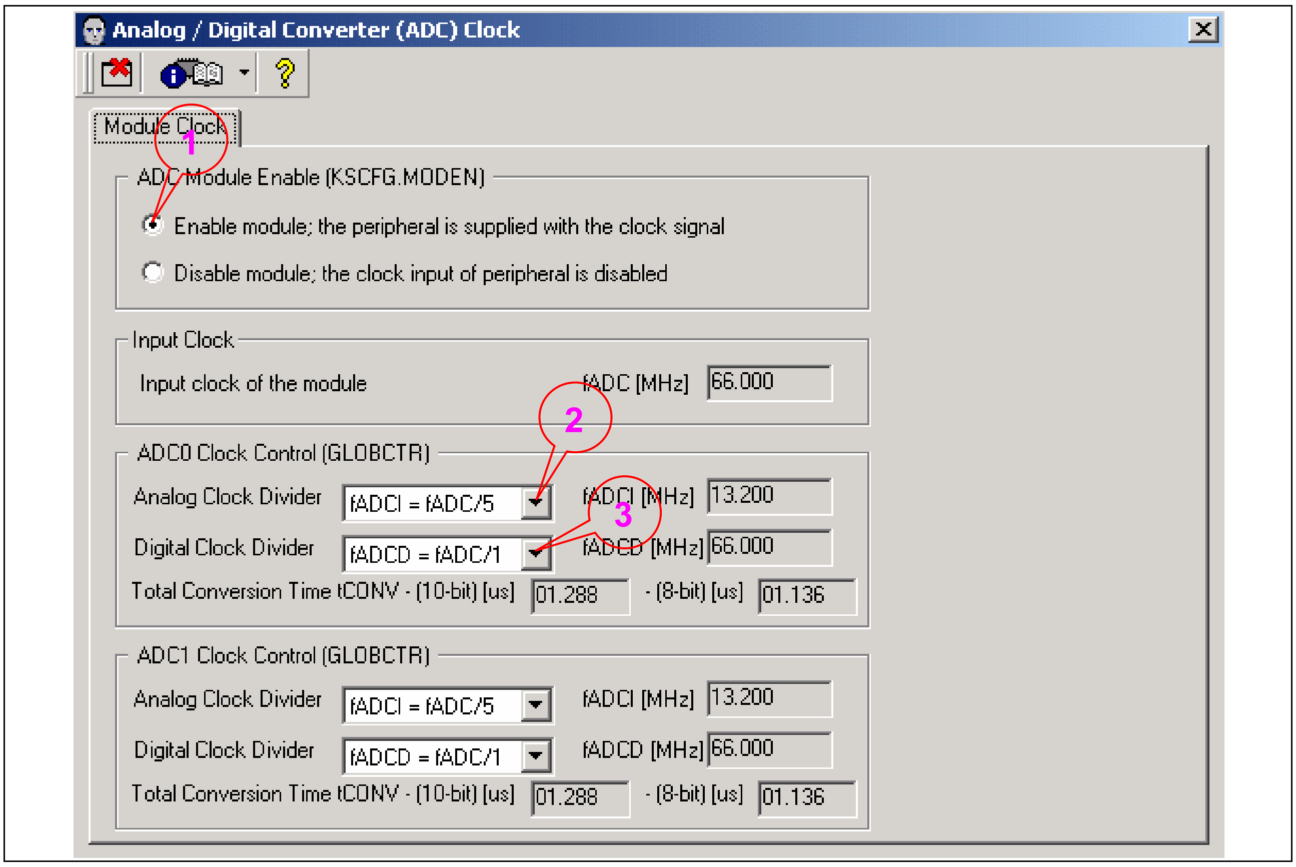


Figure 5 DAvE Bit Map showing ADC module

2. Configuration for Module Clock Page of ADC

1. Select Enable Module; the peripheral is supplied with the clock signal
2. Select Analog Clock Divider value
3. Select Digital Clock Divider value

For steps 2 and 3 default settings are used, user can select any value



ADC Module Enable (KSCFG.MODEN)

[✓ Enable module; the peripheral is supplied with the clock signal]
Disable module; the clock input of peripheral is disabled

Input Clock

Input clock of the module fADC [MHz] 66.000

ADC0 Clock Control (GLOBCTR)

Analog Clock Divider	fADCI = fADC/5	fADCI [MHz]	13.200
Digital Clock Divider	fADCD = fADC/1	fADCD [MHz]	66.000
Total Conversion Time tCONV - (10-bit) [us]	01.288	- (8-bit) [us]	01.136

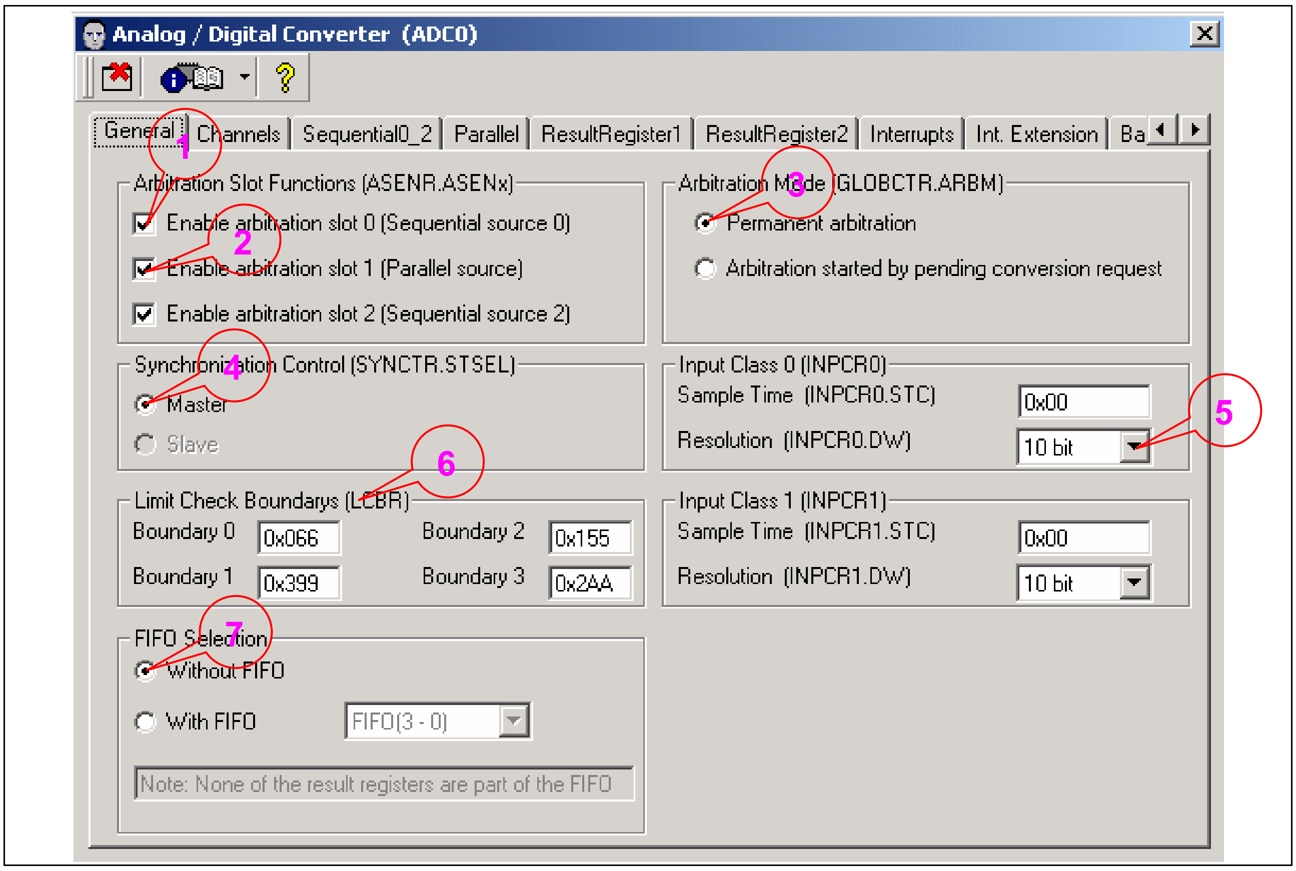
ADC1 Clock Control (GLOBCTR)

Analog Clock Divider	fADCI = fADC/5	fADCI [MHz]	13.200
Digital Clock Divider	fADCD = fADC/1	fADCD [MHz]	66.000
Total Conversion Time tCONV - (10-bit) [us]	01.288	- (8-bit) [us]	01.136

Figure 6 ADC Module Clock Page

3. Configuration for General Page of ADC

1. Select Enable arbitration slot 0 (Sequential source 0)
2. Select Enable arbitration slot 1 (Parallel source)
3. Select Permanent arbitration
4. Select Master mode
5. Select 10 bit for resolution
6. Enter Boundary 0 – 3 values as required
7. Select Without FIFO option



Arbitration Slot Functions (ASENR.ASENx)
- [✓] Enable arbitration slot 0 (Sequential source 0)
- [✓] Enable arbitration slot 1 (Parallel source)
- [✓] Enable arbitration slot 2 (Sequential source 2)

Arbitration Mode (GLOBCTR.ARBM)
- [✓ Permanent arbitration]
- Arbitration started by pending conversion request

Synchronization Control (SYNCTR.STSEL)
- [✓ Master]
- Slave

Input Class 0 (INPCR0)

Sample Time (INPCR0.STC)	0x00
Resolution (INPCR0.DW)	10 bit

Limit Check Boundarys (LCBR)

Boundary 0	0x066	Boundary 2	0x155
Boundary 1	0x399	Boundary 3	0x2AA

Input Class 1 (INPCR1)

Sample Time (INPCR1.STC)	0x00
Resolution (INPCR1.DW)	10 bit

FIFO Selection
- [✓ Without FIFO]
- With FIFO FIFO(3 - 0)

Note: None of the result registers are part of the FIFO

Figure 7 ADC General Page

4. Configuration for Channel Page of ADC

1. Select Channel 0 - which is configured for sequential source 0
2. Select Channel 1 - which is configured for parallel source

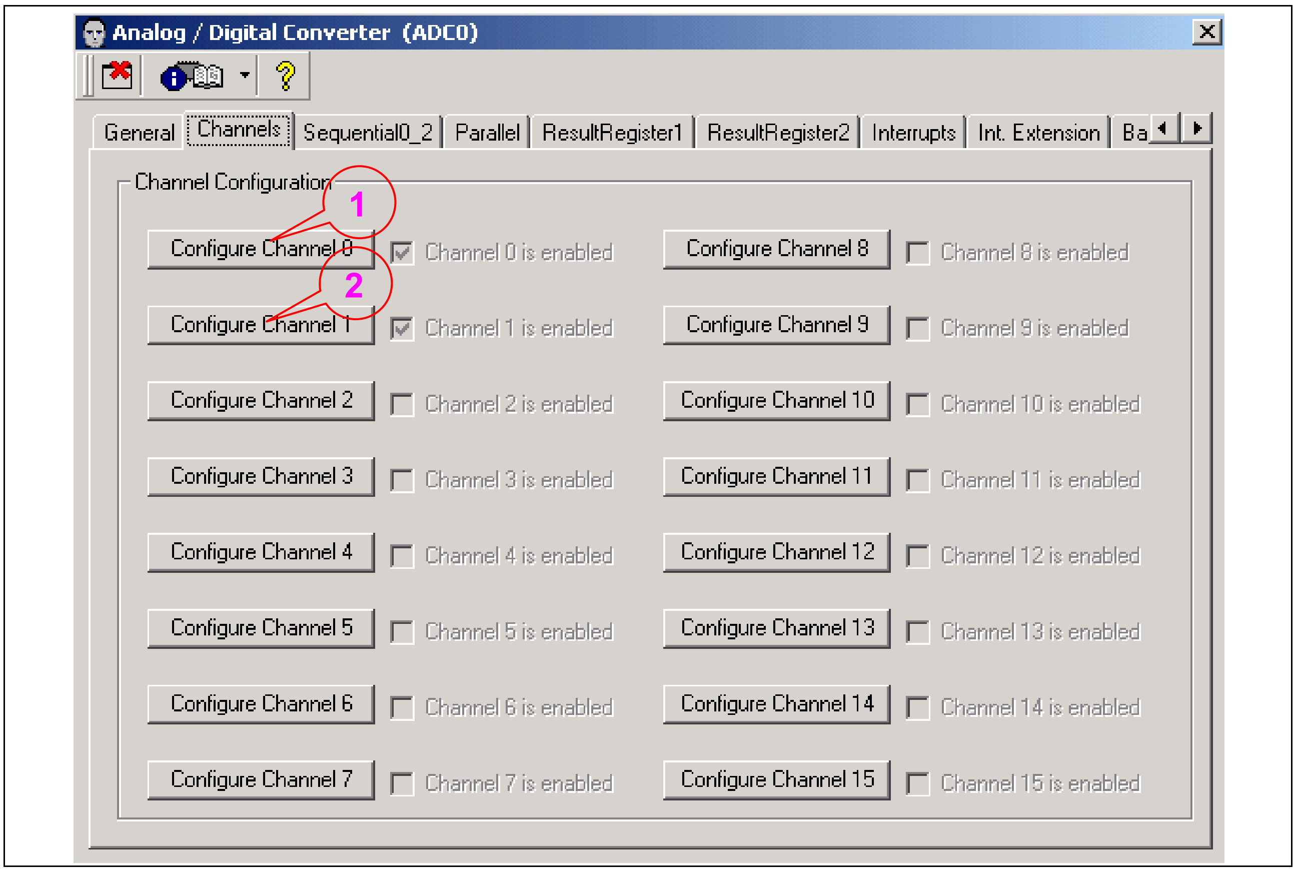
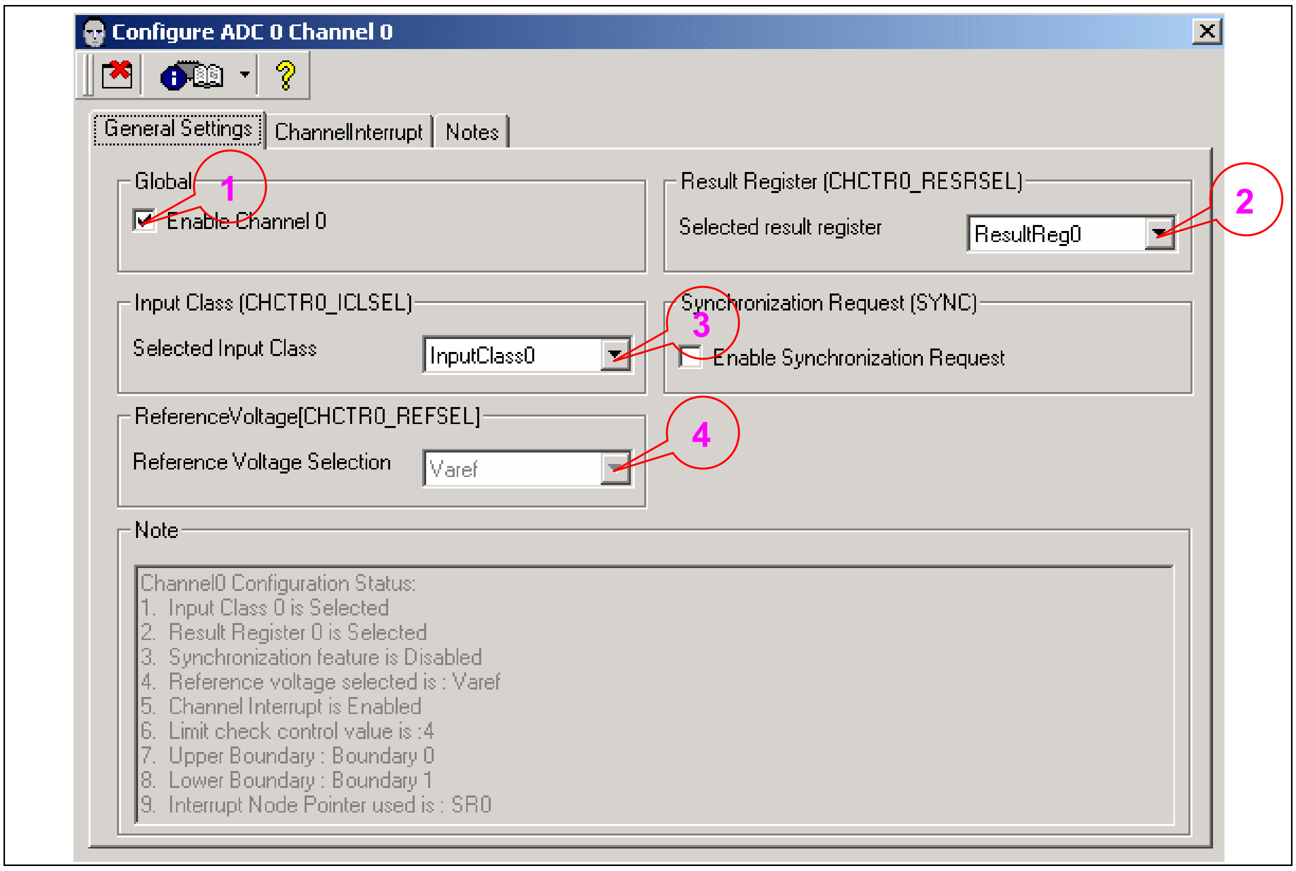


Figure 8 ADC Channel Selection Page

5. Configuration for Channel 0 General Settings Page of ADC

1. Select Enable Channel 0
2. Select Result Register 0
3. Select Input Class 0 – 10 bit resolution and 0 sample time
   10 bit resolution and sample time control are selected via INPCRx register. It must be configured under section “3. Configuration for General Page of ADC”
   STC = 0x00 means the minimum sample time of ADC.
4. For channel 0 Varef is only available as reference voltage



Enable Channel 0 [✓]

Result Register (CHCTR0_RESRSEL)
Selected result register: ResultReg0

Input Class (CHCTR0_ICLSEL)
Selected Input Class: InputClass0

Synchronization Request (SYNC)
Enable Synchronization Request [ ]

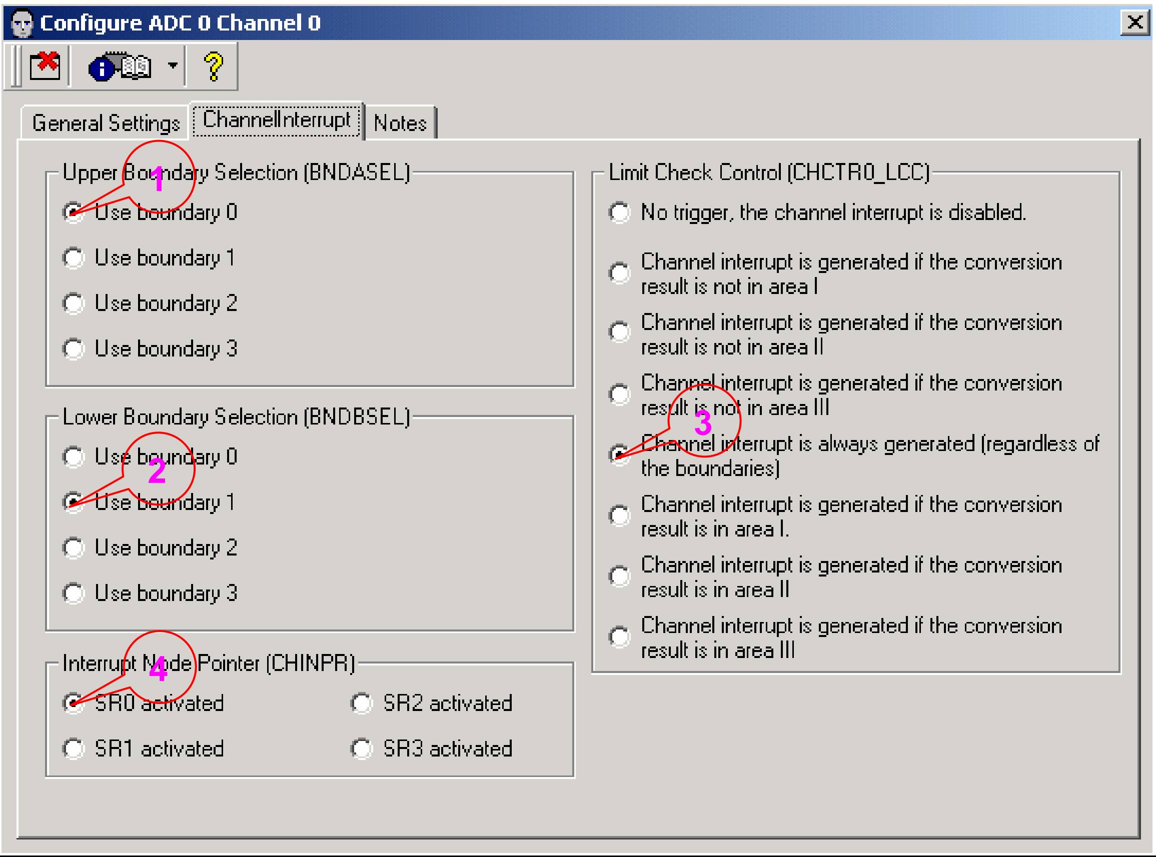
ReferenceVoltage[CHCTR0_REFSEL]
Reference Voltage Selection: Varef

Note
Channel0 Configuration Status:
1. Input Class 0 is Selected
2. Result Register 0 is Selected
3. Synchronization feature is Disabled
4. Reference voltage selected is : Varef
5. Channel Interrupt is Enabled
6. Limit check control value is :4
7. Upper Boundary : Boundary 0
8. Lower Boundary : Boundary 1
9. Interrupt Node Pointer used is : SR0

Figure 9 ADC Channel 0 General Settings Page

6. Configuration for Channel 0 Channel Interrupt Page of ADC

1. Select Boundary 0 as upper boundary
2. Select Boundary 1 as lower boundary
3. Select Channel interrupt always generated (regardless of boundary)
4. Select SR0 as interrupt node pointer



Upper Boundary Selection (BNDASEL)
- [✓ Use boundary 0]
- Use boundary 1
- Use boundary 2
- Use boundary 3

Lower Boundary Selection (BNDBSEL)
- Use boundary 0
- [✓ Use boundary 1]
- Use boundary 2
- Use boundary 3

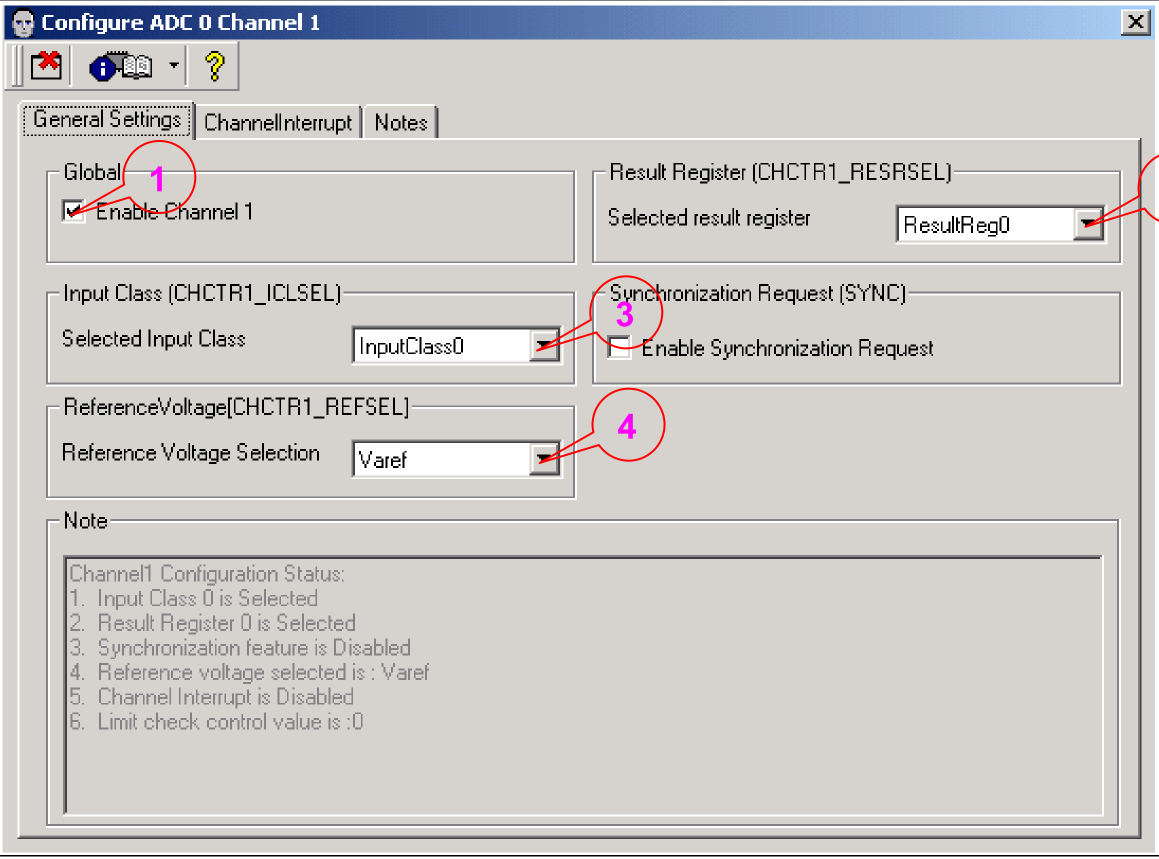
Interrupt Node Pointer (CHINPR)
- [✓ SR0 activated]
- SR1 activated
- SR2 activated
- SR3 activated

Limit Check Control (CHCTR0_LCC)
- No trigger, the channel interrupt is disabled.
- Channel interrupt is generated if the conversion result is not in area I
- Channel interrupt is generated if the conversion result is not in area II
- Channel interrupt is generated if the conversion result is not in area III
- [✓ Channel interrupt is always generated (regardless of the boundaries)]
- Channel interrupt is generated if the conversion result is in area I.
- Channel interrupt is generated if the conversion result is in area II
- Channel interrupt is generated if the conversion result is in area III

Figure 10 ADC Channel 0 Interrupt Page

7. Configuration for Channel 1 General Settings Page of ADC

1. Select Enable Channel 1
2. Select Result Register 0
3. Select Input Class 0 – 10 bit resolution and 0 sample time
   10 bit resolution and sample time control are selected via INPCRx register. It must be configured under section “3. Configuration for General Page of ADC”
   STC = 0x00 means the minimum sample time of ADC.
4. Select Varef as reference voltage



Global
[✓] Enable Channel 1

Result Register (CHCTR1_RESRSEL)
Selected result register: ResultReg0

Input Class (CHCTR1_ICLSEL)
Selected Input Class: InputClass0

Synchronization Request (SYNC)
[ ] Enable Synchronization Request

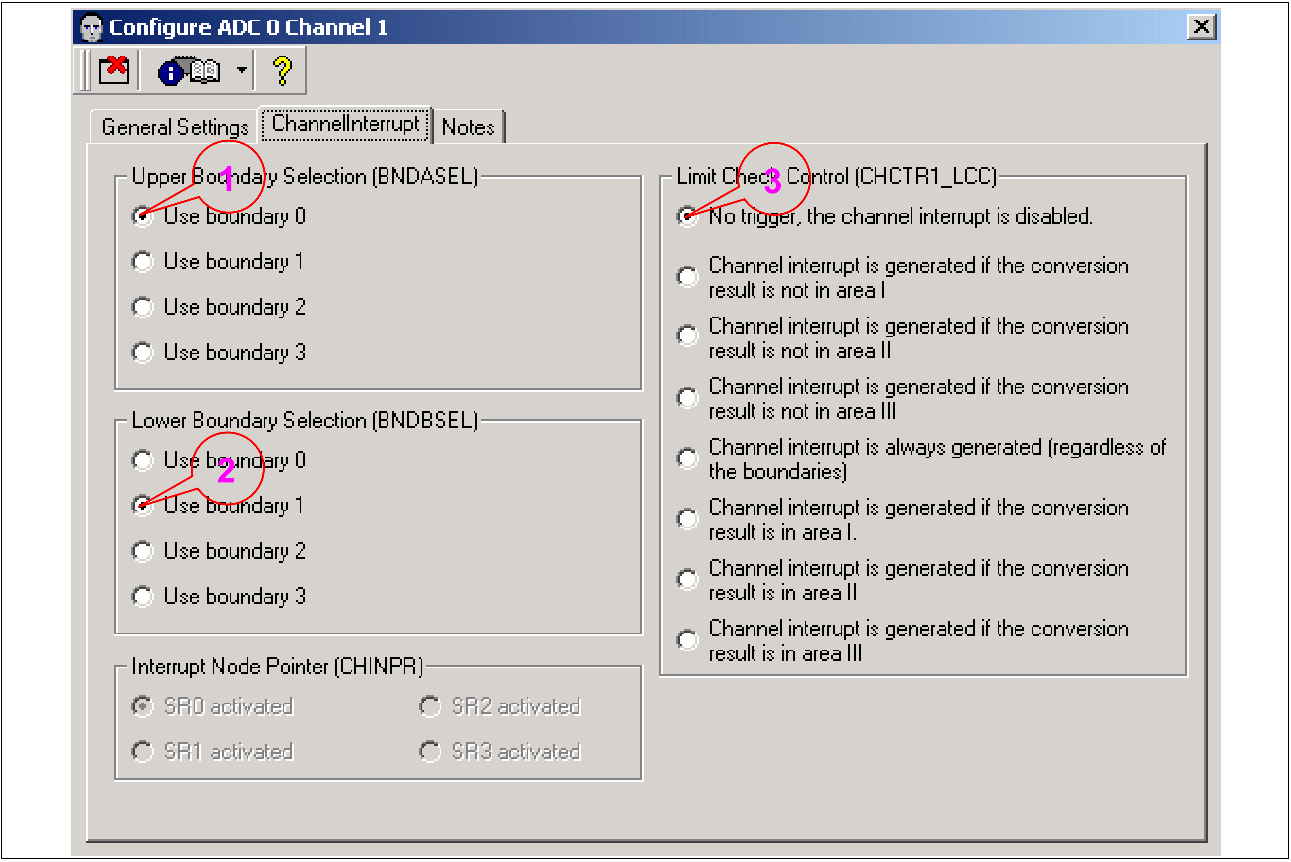
ReferenceVoltage[CHCTR1_REFSEL]
Reference Voltage Selection: Varef

Note
Channel1 Configuration Status:
1. Input Class 0 is Selected
2. Result Register 0 is Selected
3. Synchronization feature is Disabled
4. Reference voltage selected is : Varef
5. Channel Interrupt is Disabled
6. Limit check control value is :0

Figure 11 ADC Channel 1 General Settings Page

8. Configuration for Channel 1 Channel Interrupt Page of ADC

1. Select Boundary 0 as upper boundary
2. Select Boundary 1 as lower boundary
3. Select No trigger, the channel interrupt is disabled


Configure ADC 0 Channel 1

General Settings | ChannelInterrupt | Notes

Upper Boundary Selection (BNDASEL)
- [✓ Use boundary 0]
- Use boundary 1
- Use boundary 2
- Use boundary 3

Lower Boundary Selection (BNDBSEL)
- Use boundary 0
- [✓ Use boundary 1]
- Use boundary 2
- Use boundary 3

Interrupt Node Pointer (CHINPR)
- [✓ SR0 activated]
- SR1 activated
- SR2 activated
- SR3 activated

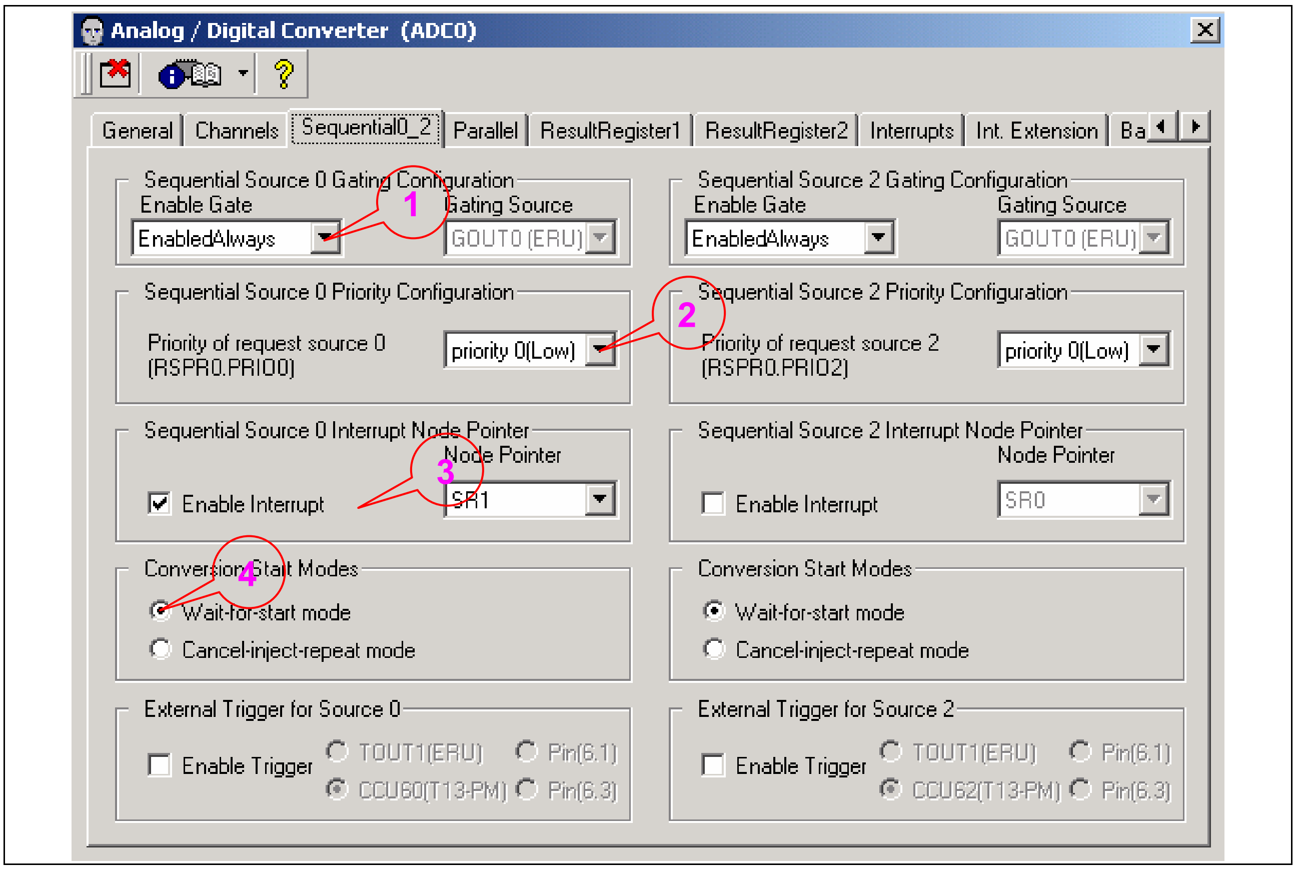
Limit Check Control (CHCTR1_LCC)
- [✓ No trigger, the channel interrupt is disabled.]
- Channel interrupt is generated if the conversion result is not in area I
- Channel interrupt is generated if the conversion result is not in area II
- Channel interrupt is generated if the conversion result is not in area III
- Channel interrupt is always generated (regardless of the boundaries)
- Channel interrupt is generated if the conversion result is in area I.
- Channel interrupt is generated if the conversion result is in area II
- Channel interrupt is generated if the conversion result is in area III

Figure 12 ADC Channel 1 Interrupt Page

9. Configuration for Sequential0_2 Page of ADC

Here only Sequential source 0 is used.

1. Select Enable Gate with Enabled Always option
2. Select Priority as low
3. Select Enable Interrupt with node pointer SR1
4. Select Wait-for-start mode as conversion start mode


Analog / Digital Converter (ADC0)

General | Channels | Sequential0_2 | Parallel | ResultRegister1 | ResultRegister2 | Interrupts | Int. Extension | Ba

Sequential Source 0 Gating Configuration
Enable Gate: EnabledAlways
Gating Source: GOUT0 (ERU)

Sequential Source 2 Gating Configuration
Enable Gate: EnabledAlways
Gating Source: GOUT0 (ERU)

Sequential Source 0 Priority Configuration
Priority of request source 0 (RSPR0.PRIO0): priority 0(Low)

Sequential Source 2 Priority Configuration
Priority of request source 2 (RSPR0.PRIO2): priority 0(Low)

Sequential Source 0 Interrupt Node Pointer
[✓] Enable Interrupt
Node Pointer: SR1

Sequential Source 2 Interrupt Node Pointer
[ ] Enable Interrupt
Node Pointer: SR0

Conversion Start Modes
(●) Wait-for-start mode
( ) Cancel-inject-repeat mode

Conversion Start Modes
(●) Wait-for-start mode
( ) Cancel-inject-repeat mode

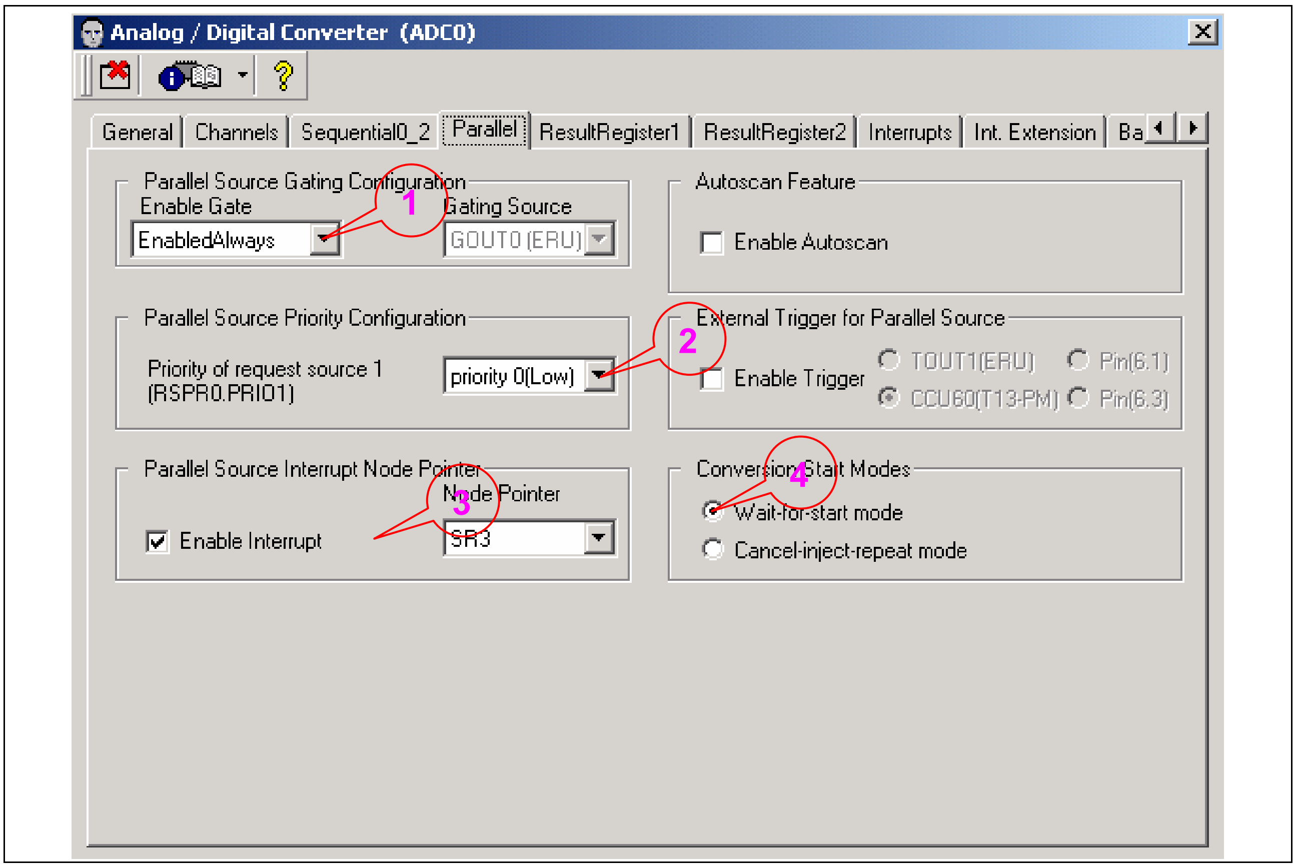
External Trigger for Source 0
[ ] Enable Trigger
( ) TOUT1(ERU) ( ) Pin(6.1)
(●) CCU60(T13-PM) ( ) Pin(6.3)

External Trigger for Source 2
[ ] Enable Trigger
( ) TOUT1(ERU) ( ) Pin(6.1)
(●) CCU62(T13-PM) ( ) Pin(6.3)

Figure 13 ADC Sequential0_2 Page

10. Configuration for Parallel Page of ADC

1. Select Enable Gate with Enabled Always option
2. Select Priority as low
3. Select Enable Interrupt with node pointer SR3
4. Select Wait-for-start mode as conversion start mode


Analog / Digital Converter (ADC0)

General | Channels | Sequential0_2 | Parallel | ResultRegister1 | ResultRegister2 | Interrupts | Int. Extension | Ba

Parallel Source Gating Configuration
Enable Gate: EnabledAlways
Gating Source: GOUT0 (ERU)

Autoscan Feature
☐ Enable Autoscan

Parallel Source Priority Configuration
Priority of request source 1 (RSPR0.PRIO1): priority 0(Low)

External Trigger for Parallel Source
☐ Enable Trigger
TOUT1(ERU) | Pin(6.1) | [✓ CCU60(T13-PM)] | Pin(6.3)

Parallel Source Interrupt Node Pointer
☑ Enable Interrupt
Node Pointer: SR3

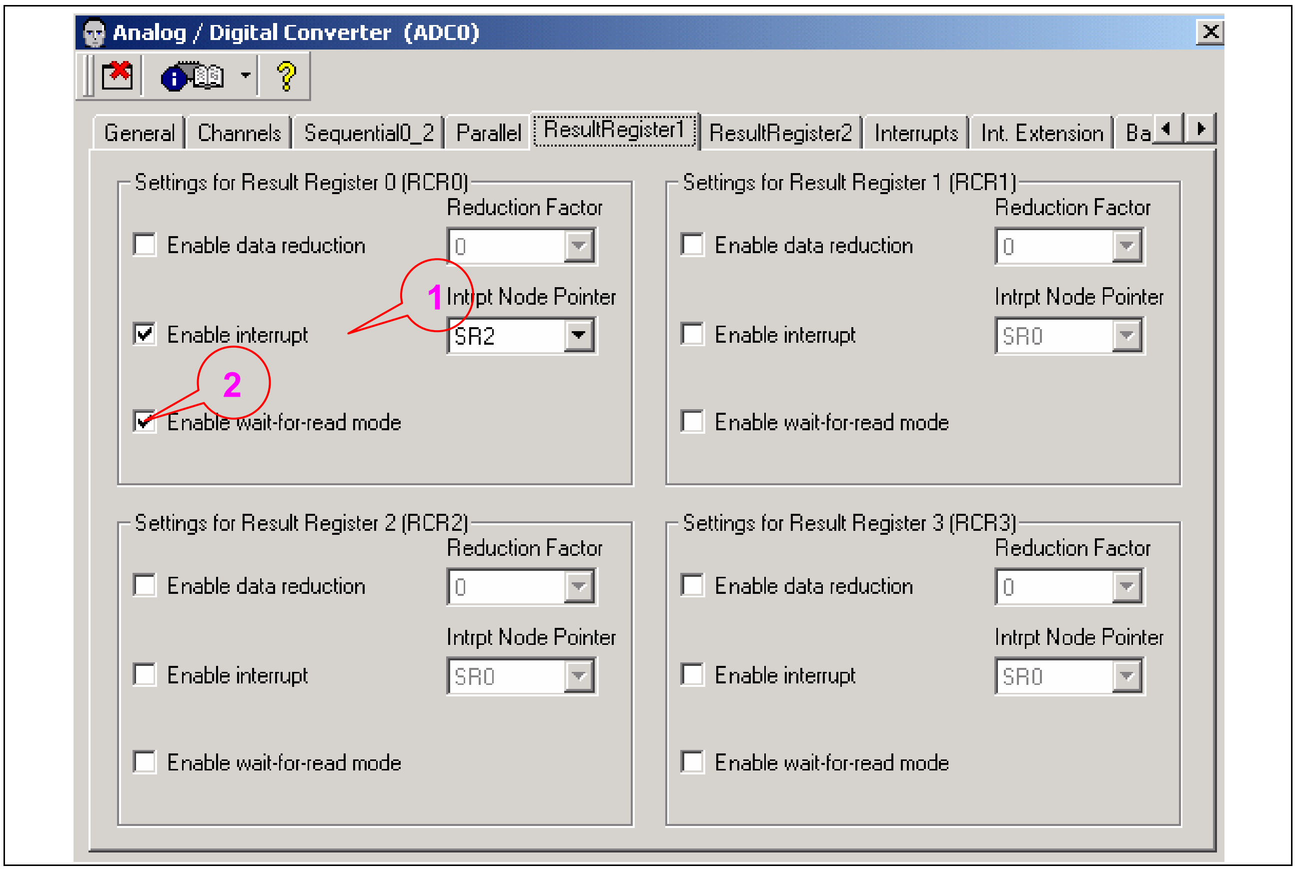
Conversion Start Modes
[✓ Wait-for-start mode]
Cancel-inject-repeat mode

Figure 14 ADC Parallel Page

11. Configuration for Result Register 1 Page of ADC

Since, we used result register 0 for both the channels 0 and 1 hence only result register 0 attributes are configured.

1. Select Enable interrupt with node pointer SR2
2. Select Enable wait-for-read mode



Settings for Result Register 0 (RCR0)
- [ ] Enable data reduction — Reduction Factor: 0
- [✓] Enable interrupt — Intrpt Node Pointer: SR2
- [✓] Enable wait-for-read mode

Settings for Result Register 1 (RCR1)
- [ ] Enable data reduction — Reduction Factor: 0
- [ ] Enable interrupt — Intrpt Node Pointer: SR0
- [ ] Enable wait-for-read mode

Settings for Result Register 2 (RCR2)
- [ ] Enable data reduction — Reduction Factor: 0
- [ ] Enable interrupt — Intrpt Node Pointer: SR0
- [ ] Enable wait-for-read mode

Settings for Result Register 3 (RCR3)
- [ ] Enable data reduction — Reduction Factor: 0
- [ ] Enable interrupt — Intrpt Node Pointer: SR0
- [ ] Enable wait-for-read mode

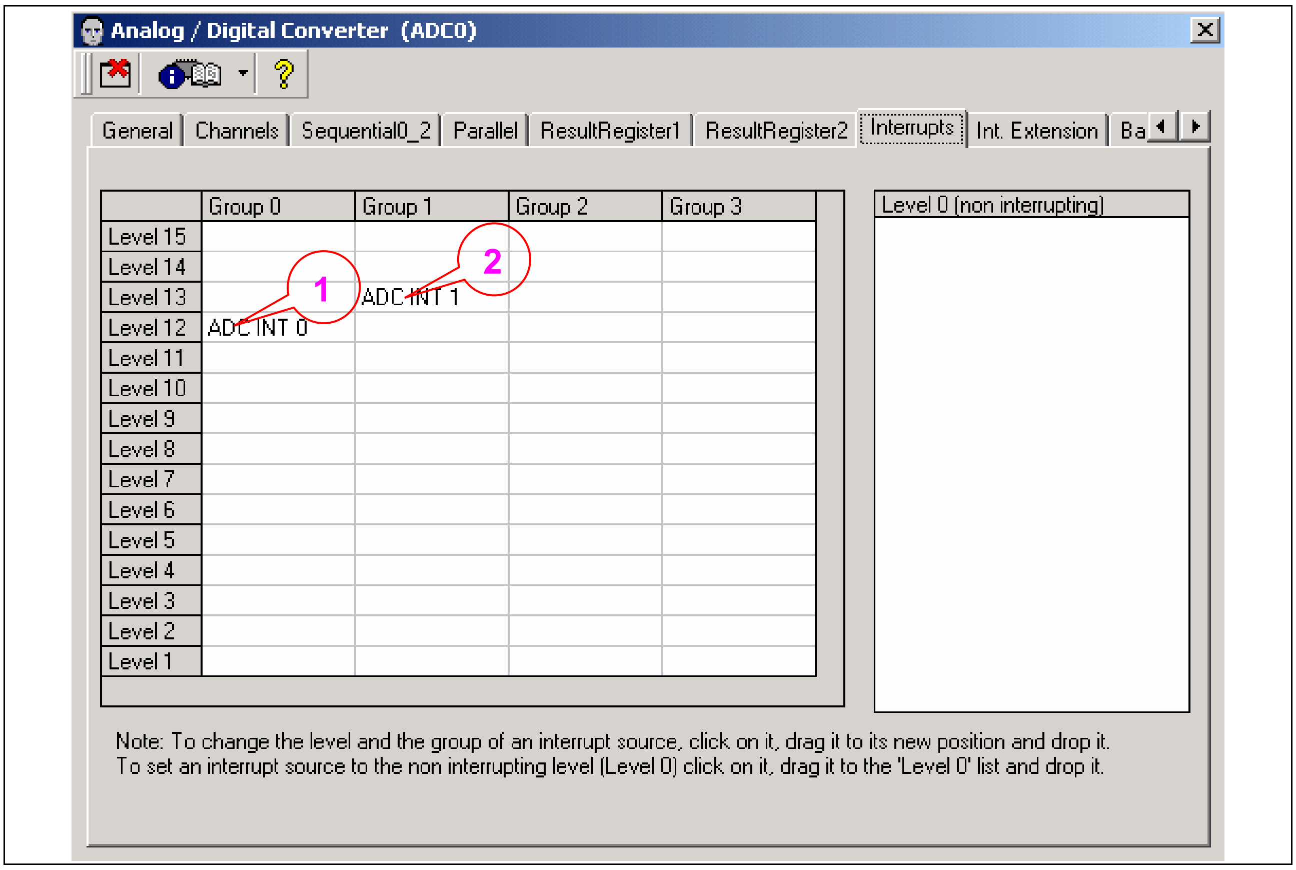
Figure 15 ADC Result Register 1 Page

12. Configuration for Interrupts Page

Here actual enhanced interrupt handling is done.

All the selected interrupts are available in the list box Level 0 (non interrupting), from here these interrupts should be drag and dropped on different Group and Levels. For enhanced interrupt handling interrupts should be placed from levels 12 – 15.

1. Place ADC INT 0 under Group 0 and Level 12
   ADC INT 0 is channel 0 interrupt
2. Place ADC INT 1 under Group 1 and Level 13
   ADC INT 1 is Sequential source 0 interrupt



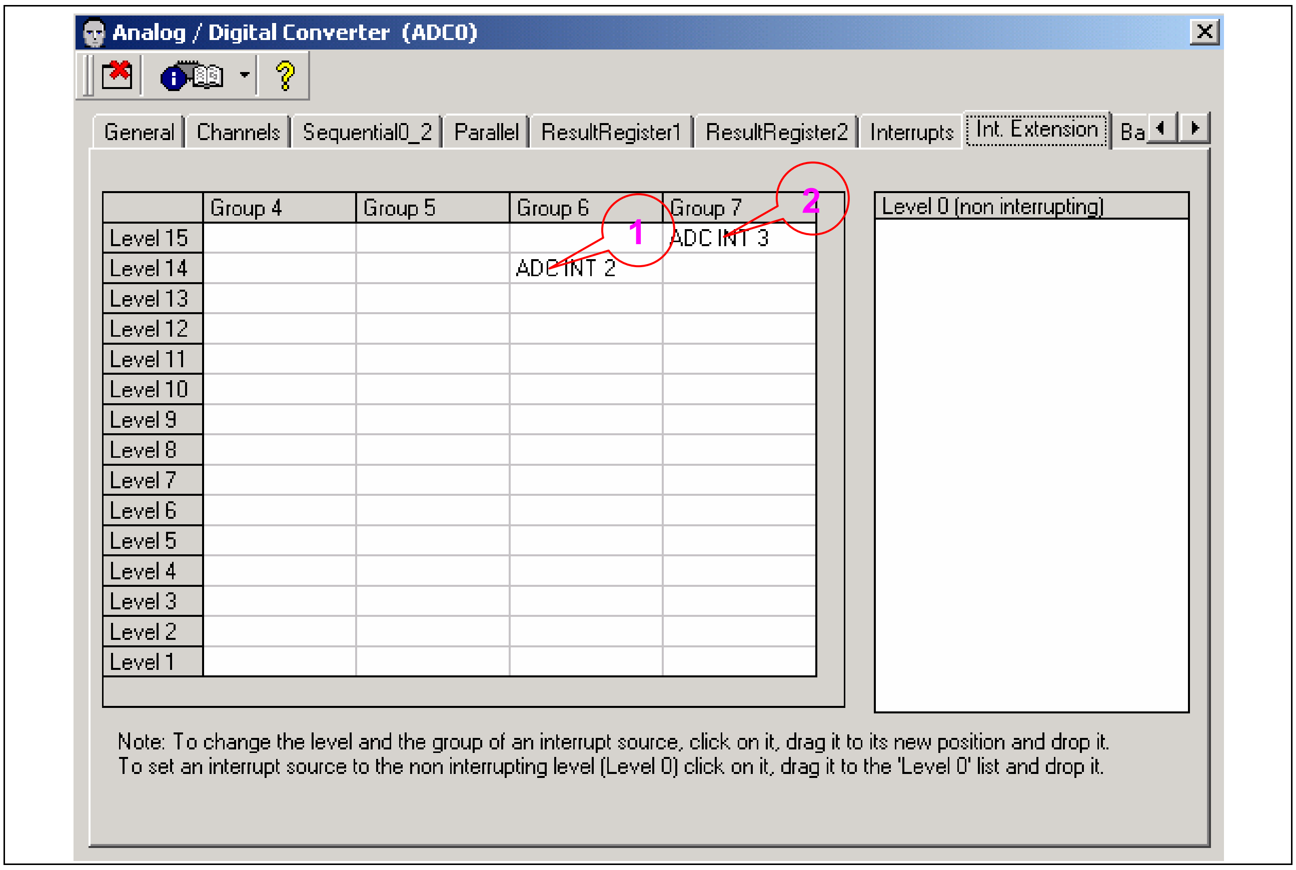
	Group 0	Group 1	Group 2	Group 3
Level 15				
Level 14				
Level 13		ADC INT 1		
Level 12	ADC INT 0			
Level 11				
Level 10				
Level 9				
Level 8				
Level 7				
Level 6				
Level 5				
Level 4				
Level 3				
Level 2				
Level 1				

Note: To change the level and the group of an interrupt source, click on it, drag it to its new position and drop it.
To set an interrupt source to the non interrupting level (Level 0) click on it, drag it to the 'Level 0' list and drop it.

Figure 16 Interrupts Page

13. Configuration for Interrupts Extension Page

1. Place ADC INT 2 under Group 6 and Level 14
   ADC INT 2 is result register 0 interrupt
2. Place ADC INT 3 under Group 7 and Level 15
   ADC INT 3 is Parallel source interrupt



	Group 4	Group 5	Group 6	Group 7
Level 15				ADC INT 3
Level 14			ADC INT 2	
Level 13				
Level 12				
Level 11				
Level 10				
Level 9				
Level 8				
Level 7				
Level 6				
Level 5				
Level 4				
Level 3				
Level 2				
Level 1				

Level 0 (non interrupting)

Note: To change the level and the group of an interrupt source, click on it, drag it to its new position and drop it.
To set an interrupt source to the non interrupting level (Level 0) click on it, drag it to the 'Level 0' list and drop it.

Figure 17 Interrupts Extension Page

14. Configuration for Bank Select Page

Here the interrupts which are placed under levels 12 – 15 can be configured for Jump Table Cache and Fast Bank Switching.

Here also all the interrupts which are placed under levels 12 – 15 are available in the Global Register Bank list box.

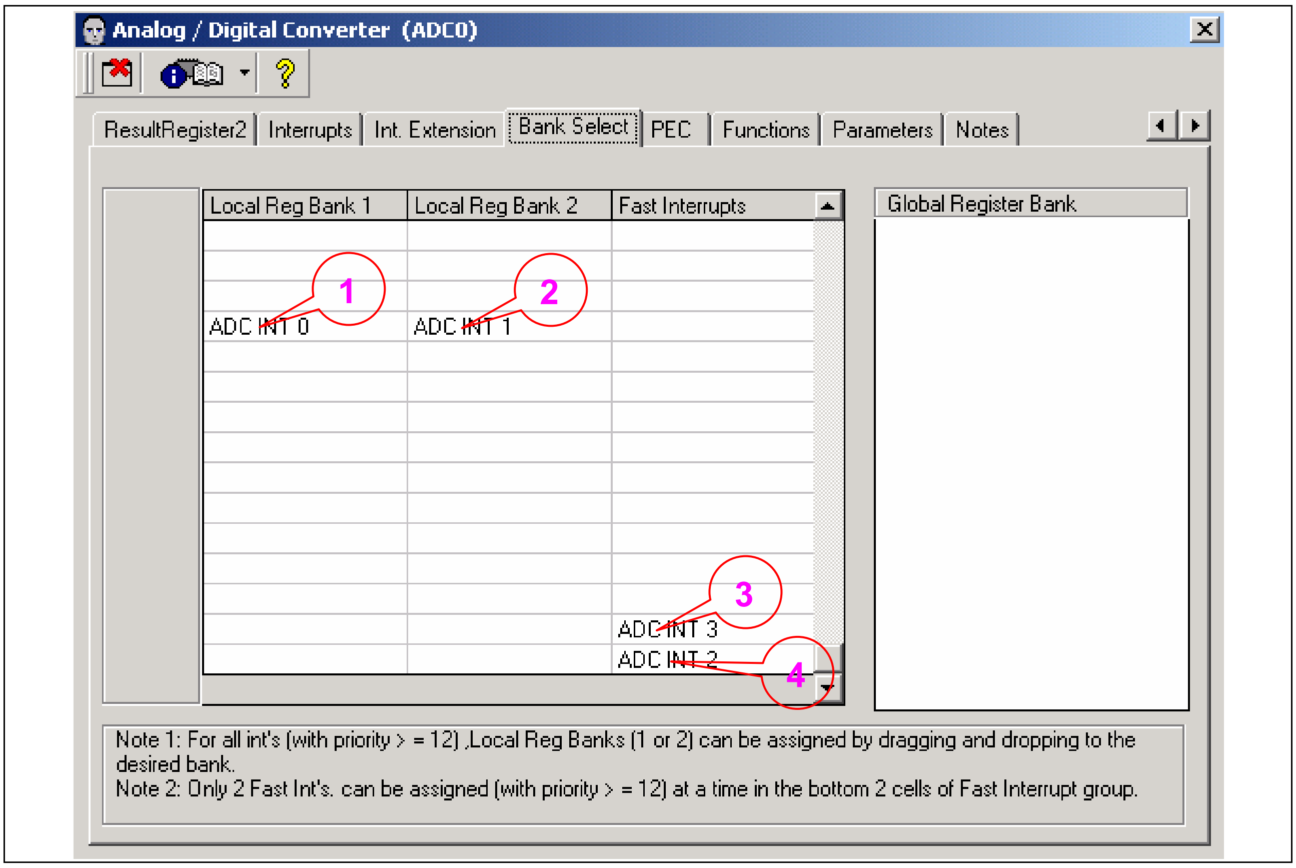
1. Place ADC INT 0 under Local Register Bank 1
2. Place ADC INT 1 under Local Register Bank 2

The above two interrupts are configured for Fast Bank Switching.
Here, the interrupts which are from the same group can't be placed under the same Local Register Bank. User will be notified if he tries to place two interrupts from the same group under any one local register bank.

3. Place ADC INT 2 under Fast Interrupts, which is in turn configured for fast interrupt 0
4. Place ADC INT 3 under Fast Interrupts, which is in turn configured for fast interrupt 1

The above two interrupts are configured for Jump Table Cache.

Here, only two interrupts can be placed at a time, because XC2000 architecture supports only two fast interrupts.



Local Reg Bank 1	Local Reg Bank 2	Fast Interrupts
ADC INT 0	ADC INT 1	
		ADC INT 3
		ADC INT 2

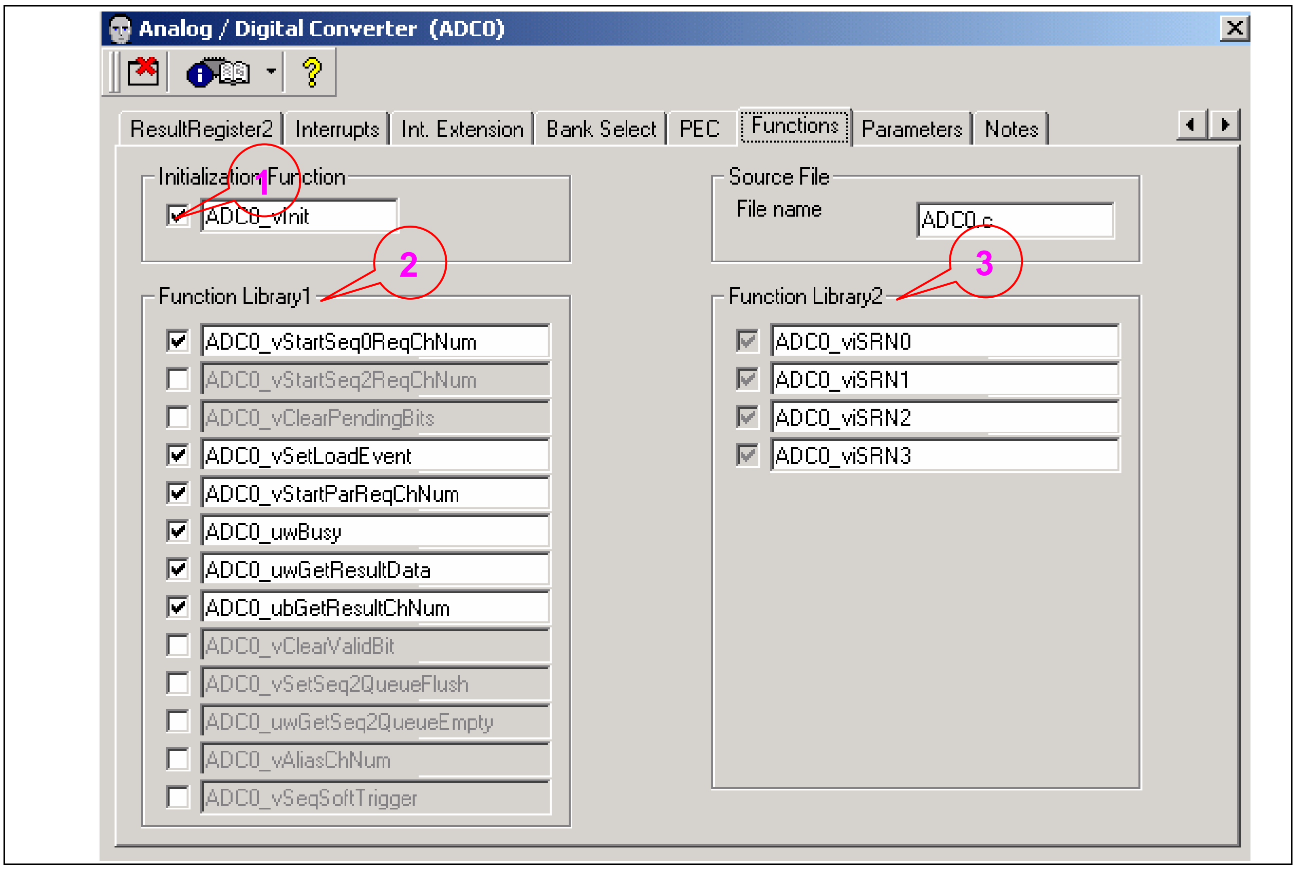
Global Register Bank

Note 1: For all int's (with priority > = 12) ,Local Reg Banks (1 or 2) can be assigned by dragging and dropping to the desired bank.
Note 2: Only 2 Fast Int's. can be assigned (with priority > = 12) at a time in the bottom 2 cells of Fast Interrupt group.

Figure 18 Bank Select Page

15. Configuration for Functions Page

1. Select Initialization function
2. Select all the necessary functions required for the example
3. All the Interrupt Subroutines are selected



Initialization Function
[✓] ADC0_vInit

Source File
File name: ADC0.c

Function Library1
[✓] ADC0_vStartSeq0ReqChNum
[ ] ADC0_vStartSeq2ReqChNum
[ ] ADC0_vClearPendingBits
[✓] ADC0_vSetLoadEvent
[✓] ADC0_vStartParReqChNum
[✓] ADC0_uwBusy
[✓] ADC0_uwGetResultData
[✓] ADC0_ubGetResultChNum
[ ] ADC0_vClearValidBit
[ ] ADC0_vSetSeq2QueueFlush
[ ] ADC0_uwGetSeq2QueueEmpty
[ ] ADC0_vAliasChNum
[ ] ADC0_vSeqSoftTrigger

Function Library2
[✓] ADC0_viSRN0
[✓] ADC0_viSRN1
[✓] ADC0_viSRN2
[✓] ADC0_viSRN3

Figure 19 Functions Page

5 Example Code for XC2287

The following C- code shows enhanced interrupt handling for ADC module interrupts.

These interrupts are

1. Channel 0 interrupt
2. Sequential source 0 interrupt
3. Parallel source interrupt
4. Result register 0 interrupt

The code is for a XC2287 device with Keil compiler.

Tools Used:

DAvE v2.1r22

XC22xx_Series_v2.0.dip

Easy Kit XC2287.100A

Keil Compiler v6.10

Universal Debug Engine, Release 2.00.11 (Universal Access Device 2)

The following code is a main function where in which the ADC conversion sequence is done and jump table cache, fast bank switching interrupts status flags are checked. This main function is available in MAIN.C file of DAvE generated code.

```
void main(void)
{
 // USER CODE BEGIN (Main,2)
    ubyte ubChannel;
    uword uwResult;
    char s[100];
 // USER CODE END
 MAIN_vInit();
 // USER CODE BEGIN (Main,3)
    sprintf(s, "\r\nTesting Fast and Bank Selected Interrupts Using Keil.\r\n\r\n");
    Doprintf(s);
    ADC0_vStartSeq0ReqChNum(0x00, 0x01, 0x00, 0x00);
    // Converting Channel 0 configured for Sequential source 0
    while(ADC0_uwBusy());
    while (!Flag[1]);
    // Wait till sequential source 0 conversion completes (Local Bank 2 interrupt flag set)
    ubChannel = ADC0_ubGetResultChNum(RESULT_REG_0);
    // Reading converted channel number from result register 0
    sprintf(s, "Converted channel number                              :  %0x     \r\n",ubChannel);
    Doprintf(s);
    uwResult = ADC0_uwGetResultData(RESULT_REG_0);
    // Reading converted result from result register 0
    sprintf(s, "Channel 0 conversion result                            :  %0x     \r\n",uwResult);
    Doprintf(s);
    sprintf(s, "Channel 0 event interrupt flag (Fast Bank Swith Local Bank 1 Interrupt)   :  %0x\r\n",Flag[0]);
    // Local Bank 1 Interrupt
    Doprintf(s);
    sprintf(s, "Source 0 event interrupt flag  (Fast Bank Swith Local Bank 2 Interrupt)   :  %0x \r\n",Flag[1]);
    // Local Bank 2 Interrupt
    Doprintf(s);
    ADC0_vStartParReqChNum(0x0002);
    // Converting Channel 1 configured for Parallel source
    ADC0_vSetLoadEvent();
    // Triggering through software
    while(ADC0_uwBusy());
    while (!Flag[3]);
    // Wait till parallel source conversion completes (Fast interrupt 1 flag set)
   ubChannel = ADC0_ubGetResultChNum(RESULT_REG_0);
   // Reading converted channel number from result register 0
   sprintf(s, "Converted channel number                              :  %0x     \r\n",ubChannel);
   Doprintf(s);
   uwResult = ADC0_uwGetResultData(RESULT_REG_0);
   // Reading converted result from result register 0
   sprintf(s, "Channel 1 conversion result                            :  %0x     \r\n",uwResult);
   Doprintf(s);
   sprintf(s, "Source 1 event interrupt flag (Jump Table Cache Fast Interrupt 1)         :  %0x     \r\n",Flag[3]);
   // Fast Interrupt 1
   Doprintf(s);
   sprintf(s, "Result register 0 event interrupt flag (Jump Table Cache Fast Interrupt 0):  %0x\r\n",Flag[2]);
   // Fast Interrupt 0
   Doprintf(s);
// USER CODE END
 while(1)
 {
    // USER CODE BEGIN (Main,4)

   // USER CODE END
 }
} //  End of function main
```

The following code shows the enhanced interrupt handling for keil compiler. This code is available in ADC0.C file.
Since SR2 and SR3 are configured for fast interrupt 0 and 1 (Jump Table Cache Interrupts) hence their declaration is available as follows

```
//****************************************************************************
// @External Prototypes
//****************************************************************************
// USER CODE BEGIN (ADC0_General,8)

// USER CODE END
  extern void ADC0_viSRN2(void);
  extern void ADC0_viSRN3(void);
```

The assignment of Jump Table Cache registers is done in ADC0.C ADC0_vInit() function as shown below.

```
void ADC0_vInit(void)
{
        // USER CODE BEGIN (ADC0_Init,2)

        // USER CODE END

        ///  -----------------------------------------------------------------------
        ///  Configuration of ADC0 kernel configuration register:
        ///  -----------------------------------------------------------------------
        ADC0_KSCFG     =  0x0003;
        /// load ADC0 kernel configuration register
        ///  - the ADC module clock is enabled
        ///  - the ADC module clock = 66.00 MHz
        ///
        _nop_();  // one cycle delay
        _nop_();  // one cycle delay

        ///  -----------------------------------------------------------------------
        ///  Configure global control register:
        ///  -----------------------------------------------------------------------
        ///  --- Conversion Timing -----------------
        ///  - conversion time (CTC)     = 01.29 us
        ///  _Analog clock is 1/5th of module clock and digital clock is 1/1 times
        ///  of module clock
        ///  - the permanent arbitration mode is selected
        ADC0_GLOBCTR   =  0x0004;
        /// load global control register
        ///  -----------------------------------------------------------------------
        ///  Configuration of Arbitration Slot enable register and also the Source
        ///  Priority register:
        ///  -----------------------------------------------------------------------
        ///  - Arbitration Slot 0 is enabled

        ///  - Arbitration Slot 1 is enabled

        ///  - Arbitration Slot 2 is enabled

        ///  - the priority of request source 0 is low
        ///  - the wait-for-start mode is selected for source 0
        ///  - the priority of request source 1 is low
        ///  - the wait-for-start mode is selected for source 1
        ///  - the priority of request source 2 is low
        ///  - the wait-for-start mode is selected for source 2
```

```
ADC0_ASENR     =  0x0007;
// load Arbitration Slot enable register

ADC0_RSPR0     =  0x0000;
// load Priority and Arbitration register

///  -------------------------------------------------- ------------------
///  Configuration of Channel Control Registers:
///  -------------------------------------------------------------------
///  Configuration of Channel 0
///  - the result register0 is selected
///  - the limit check 4 is selected

///  - the reference voltage selected is Standard Voltage (Varef)

///  - the input class selected is Input Class 0

///  - LCBR0 is selected as upper boundary

///  - LCBR1 is selected as lower boundary

ADC0_CHCTR0    =  0x0044;
// load channel control register

///  Configuration of Channel 1
///  - the result register0 is selected
///  - the limit check 0 is selected
///  - the reference voltage selected is Standard Voltage (Varef)

///  - the input class selected is Input Class 0

///  - LCBR0 is selected as upper boundary

///  - LCBR1 is selected as lower boundary

ADC0_CHCTR1    =  0x0004;
// load channel control register

///  -------------------------------------------------- ------------------
///  Configuration of Sample Time and Resolution:
///  -------------------------------------------------------------------

///  10 bit resolution selected

ADC0_INPCR0    =  0x0000;
// load input class0 register

///  10 bit resolution selected

ADC0_INPCR1    =  0x0000;
// load input class1 register

///  -------------------------------------------------- ------------------
///  Configuration of Result Control Registers:
///  -------------------------------------------------- ------------------
///  Configuration of Result Control Register 0
///  - the data reduction filter is disabled
///  - the event interrupt is enabled
///  - the wait-for-read mode is enabled
///  - the FIFO functionality is disabled

ADC0_RCR0      =  0x0050;
// load result control register 0

///  Configuration of Result Control Register 1
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR1      =  0x0000;
// load result control register 1

///  Configuration of Result Control Register 2
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR2      =  0x0000;
// load result control register 2

///  Configuration of Result Control Register 3
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR3      =  0x0000;
// load result control register 3

///  Configuration of Result Control Register 4
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR4      =  0x0000;
// load result control register 4

///  Configuration of Result Control Register 5
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR5      =  0x0000;
// load result control register 5

///  Configuration of Result Control Register 6
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled
///  - the FIFO functionality is disabled

ADC0_RCR6      =  0x0000;
// load result control register 6

///  Configuration of Result Control Register 7
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR7      =  0x0000;
// load result control register 7

///  -------------------------------------------------- ------------------
///  Configuration of Channel Interrupt Node Pointer
       Register:
///  --------------------------------------------------------------------
///  - the SR0 line become activated if channel 0 interrupt is generated

///  - the SR0 line become activated if channel 1 interrupt is generated

ADC0_CHINPR0   =  0x0000;
// load channel interrupt node pointer
// register

ADC0_CHINPR4   =  0x0000;
// load channel interrupt node pointer
// register

ADC0_CHINPR8   =  0x0000;
// load channel interrupt node pointer
// register

ADC0_CHINPR12  =  0x0000;
// load channel interrupt node pointer
// register

///  --------------------------------------------------------------------
///  Configuration of Event Interrupt Node Pointer Register for Source
///  Interrupts:
///  --------------------------------------------------------------------
///  - the SR 1 line become activated if the event 0 interrupt is generated

///  - the SR 3 line become activated if the event 1 interrupt is generated
///  - the SR 0 line become activated if the event 2 interrupt is generated

ADC0_EVINPR0   =  0x0031;
// load event interrupt set flag register

///  --------------------------------------------------------------------
///  Configuration of Event Interrupt Node Pointer Register for Result
///  Interrupts:
///  -------------------------------------------------- ------------------
///  - the SR 2 line become activated if the event 8 interrupt is generated

ADC0_EVINPR8   =  0x0002;
// load event interrupt set flag register
```

```
ADC0_EVINPR12  =  0x0000;
// load event interrupt set flag register

///  ---------------------------------------------------------------------
///  Configuration of Service Request Nodes 0 - 3 :
///  ---------------------------------------------------------------------
///  SRN0 service request node configuration:
///  - SRN0 interrupt priority level (ILVL) = 12
///  - SRN0 interrupt group level (GLVL) = 0
///  - SRN0 group priority extension (GPX) = 0

ADC_0IC        =  0x0070;

///  SRN1 service request node configuration:
///  - SRN1 interrupt priority level (ILVL) = 13
///  - SRN1 interrupt group level (GLVL) = 1
///  - SRN1 group priority extension (GPX) = 0

ADC_1IC        =  0x0075;

///  SRN2 service request node configuration:
///  - SRN2 interrupt priority level (ILVL) = 14
///  - SRN2 interrupt group level (GLVL) = 2
///  - SRN2 group priority extension (GPX) = 1

ADC_2IC        =  0x017A;

///  SRN3 service request node configuration:
///  - SRN3 interrupt priority level (ILVL) = 15
///  - SRN3 interrupt group level (GLVL) = 3
///  - SRN3 group priority extension (GPX) = 1

ADC_3IC        =  0x017F;


///  ------------------------------------------------------ -------------------
///  Configuration of Limit Check Boundary:
///  ---------------------------------------------------------------------

ADC0_LCBR0     =  0x0198;
// load limit check boundary register 0

ADC0_LCBR1     =  0x0E64;
// load limit check boundary register 1

ADC0_LCBR2     =  0x0554;
// load limit check boundary register 2

ADC0_LCBR3     =  0x0AA8;
// load limit check boundary register 3

///  ------------------------------------------------------ -------------------
///  Configuration of Gating source and External Trigger Control:
///  ---------------------------------------------------------------------
///  - No Gating source selected for Arbitration Source 0

///  - the trigger input ETR00 is selected for Source 0

///  - No Gating source selected for Arbitration Source 1
///  - the trigger input ETR00 is selected for Source 1

///  - No Gating source selected for Arbitration Source 2

///  - the trigger input ETR00 is selected for Source 1

ADC0_PISEL     =  0x0444;
// load external trigger control register

///  -----------------------------------------------------------------------
///  Configuration of Conversion Queue Mode Register:Sequential Source 0
///  -----------------------------------------------------------------------
///  - the gating line is permanently Enabled
///  - the external trigger is disabled

ADC0_QMR0      =  0x0001;
// load queue mode register

///  -----------------------------------------------------------------------
///  Configuration of Conversion Queue Mode Register:Sequential Source 2
///  -----------------------------------------------------------------------
///  - the gating line is permanently Enabled
///  - the external trigger is disabled

ADC0_QMR2      =  0x0001;
// load queue mode register

///  -----------------------------------------------------------------------
///  Configuration of Conversion Request Mode Registers:Parallel Source
///  -----------------------------------------------------------------------
///  - the gating line is permanently Enabled
///  - the external trigger is disabled
///  - the source interrupt is enabled
///  - the autoscan functionality is disabled

ADC0_CRMR1     =  0x0009;
// load conversion request mode register 1

///  -----------------------------------------------------------------------
///  Configuration of Synchronisation Registers:
///  ----------------------------------------------------- --------------------
///  - ADC0 is master
ADC0_SYNCTR   |=  0x0010;
// Synchronisation register

P5_DIDIS       =  0x0003;
// Port 5 Digital input disable register

ADC0_GLOBCTR  |=  0x0300;
// turn on Analog part

FINT0CSP = SEG(ADC0_viSRN2) | 0x9A00;
//Fast Interrupt Control Reg. 0
FINT0ADDR = SOF(ADC0_viSRN2);
//Fast Interrupt Address Reg. 0

FINT1CSP = SEG(ADC0_viSRN3) | 0x9F00;
//Fast Interrupt Control Reg. 1
FINT1ADDR = SOF(ADC0_viSRN3);
```

```
        //Fast Interrupt Address Reg. 1

        // USER CODE BEGIN (ADC0_Init,3)

        // USER CODE END

} //  End of function ADC0_vInit
```

The Global interrupt enabled in Main_vInit() function after all peripheral initialization functions.

The macros SOF() and SEG() are defined in MAIN.H file as follows

```
//****************************************************************************
// @Defines
//****************************************************************************
#define KEIL

#define SEG(func) (unsigned int)(((unsigned long)((void (far*)(void))func) >> 16))
#define SOF(func) (unsigned int)(((void (far *) (void))func))

// USER CODE BEGIN (MAIN_Header,4)

// USER CODE END
```

The DAvE generated ISR with Keil compiler is as shown below

```
//****************************************************************************
// @Function      void ADC0_viSRN0(void)
//
//---------------------------------------------------------- -------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 0 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN0,0)

// USER CODE END

void ADC0_viSRN0(void) interrupt ADC0_SRN0INT using _FAST_ABANK1_
{

    if((ADC0_CHINFR & 0x0001) ==  0x0001)
    //Channel0 interrupt
    {
        ADC0_CHINCR = 0x0001;
        // Clear Channel0 interrupt
        // USER CODE BEGIN (ADC0_viSRN0,4)
          Flag[0]++;
        // USER CODE END

    }


} //  End of function ADC0_viSRN0



//****************************************************************************
// @Function      void ADC0_viSRN1(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 1 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN1,0)

// USER CODE END

void ADC0_viSRN1(void) interrupt ADC0_SRN1INT using _FAST_ABANK2_
{

    if((ADC0_EVINFR & 0x0001) == 0x0001)
    //Source0 event interrupt
    {
        ADC0_EVINCR = 0x0001;
        // Clear source0 event interrupt

        // USER CODE BEGIN (ADC0_viSRN1,1)
          Flag[1]++;
        // USER CODE END

    }

} //  End of function ADC0_viSRN1
```

```
//****************************************************************************
// @Function      void ADC0_viSRN2(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 2 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN2,0)

// USER CODE END

void ADC0_viSRN2(void) interrupt ADC0_SRN2INTFAST = CACHED
{

    if((ADC0_EVINFR & 0x0100) == 0x0100)
    //Result0 event interrupt
    {
      ADC0_EVINCR = 0x0100;
       // Clear Result0 event interrupt

       // USER CODE BEGIN (ADC0_viSRN2,20)
         Flag[2]++;
       // USER CODE END

    }


} //  End of function ADC0_viSRN2
```

```
//****************************************************************************
// @Function      void ADC0_viSRN3(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 3 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN3,0)

// USER CODE END

void ADC0_viSRN3(void) interrupt ADC0_SRN3INTFAST = CACHED
{

    if((ADC0_EVINFR & 0x0002) == 0x0002)
    //Source1 event interrupt
    {
        ADC0_EVINCR = 0x0002;
        // Clear source1 event interrupt

        // USER CODE BEGIN (ADC0_viSRN3,2)
          Flag[3]++;
        // USER CODE END

    }


} //  End of function ADC0_viSRN3
```

The DAvE generated ISR with Tasking Classic compiler is as shown below

Since SR2 and SR3 are configured for fast interrupt 0 and 1 (Jump Table Cache Interrupts) hence their declaration is available as follows,

```
//****************************************************************************
// @External Prototypes
//****************************************************************************

// USER CODE BEGIN (ADC0_General,8)

// USER CODE END

extern _interrupt(ADC0_SRN2INT) _cached void ADC0_viSRN2(void);
extern _interrupt(ADC0_SRN3INT) _cached void ADC0_viSRN3(void);
```

The assignment of Jump Table Cache registers is done in ADC0.C ADC0_vInit() function

```
FINT0CSP = 0x9A00 | (((unsigned long)&(ADC0_viSRN2))>>16);    //Fast Interrupt Control Reg. 0
FINT0ADDR = (unsigned int) & (ADC0_viSRN2);                  //Fast Interrupt Address Reg. 0
FINT1CSP = 0x9F00 | (((unsigned long) & (ADC0_viSRN3))>>16);  //Fast Interrupt Control Reg. 1
FINT1ADDR = (unsigned int) & (ADC0_viSRN3);                  //Fast Interrupt Address Reg. 1

//****************************************************************************
// @Function      void ADC0_viSRN0(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 0 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN0,0)

// USER CODE END

_interrupt(ADC0_SRN0INT) _localbank(-1) void ADC0_viSRN0(void)
{

        if((ADC0_CHINFR & 0x0001) ==  0x0001)
        //Channel0 interrupt
        {
            ADC0_CHINCR = 0x0001;
            // Clear Channel0 interrupt

            // USER CODE BEGIN (ADC0_viSRN0,4)
               Flag[0]++;
            // USER CODE END

        }


} //  End of function ADC0_viSRN0
```

```
//****************************************************************************
// @Function      void ADC0_viSRN1(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 1 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN1,0)

// USER CODE END

_interrupt(ADC0_SRN1INT) _localbank(-2) void ADC0_viSRN1(void)
{

    if((ADC0_EVINFR & 0x0001) == 0x0001)
    //Source0 event interrupt
    {
        ADC0_EVINCR = 0x0001;
        // Clear source0 event interrupt

        // USER CODE BEGIN (ADC0_viSRN1,1)
          Flag[1]++;
        // USER CODE END

    }


} //  End of function ADC0_viSRN1
```

```
//****************************************************************************
// @Function      void ADC0_viSRN2(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 2 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN2,0)

// USER CODE END

_interrupt(ADC0_SRN2INT) _cached void ADC0_viSRN2(void)
{

        if((ADC0_EVINFR & 0x0100) == 0x0100)
        //Result0 event interrupt
        {
            ADC0_EVINCR = 0x0100;
            // Clear Result0 event interrupt

            // USER CODE BEGIN (ADC0_viSRN2,20)
               Flag[2]++;
            // USER CODE END

        }


 } //  End of function ADC0_viSRN2
```

```
//****************************************************************************
// @Function      void ADC0_viSRN3(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 3 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************
// USER CODE BEGIN (ADC0_viSRN3,0)

// USER CODE END

_interrupt(ADC0_SRN3INT) _cached void ADC0_viSRN3(void)
{

    if((ADC0_EVINFR & 0x0002) == 0x0002)
    //Source1 event interrupt
    {
      ADC0_EVINCR = 0x0002;
      // Clear source1 event interrupt

      // USER CODE BEGIN (ADC0_viSRN3,2)
        Flag[3]++;
      // USER CODE END

    }


} //  End of function ADC0_viSRN3
```

The DAvE generated ISR with Tasking Viper compiler is as shown below

Since SR2 and SR3 are configured for fast interrupt 0 and 1 (Jump Table Cache Interrupts) they are generated as normal interrupts, since DAvE doesn't support Fast Interrupts for Tasking Viper compiler.

The key word _interrupt is defined in MAIN.H file as shown below for Tasking Viper compiler

```
//****************************************************************************
// @Defines
//****************************************************************************
#define TASKING_VIPER

#define far  __far
#define _inline  inline
#define _nop  __nop
#define _interrupt  __interrupt



//****************************************************************************
// @Function      void ADC0_viSRN0(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 0 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//-------------------------------------------------------- -------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN0, 0)

// USER CODE END

_interrupt(ADC0_SRN0INT) __registerbank(-1) void ADC0_viSRN0(void)
{

        if((ADC0_CHINFR & 0x0001) ==  0x0001)
        //Channel0 interrupt
        {
            ADC0_CHINCR = 0x0001;
            // Clear Channel0 interrupt

            // USER CODE BEGIN (ADC0_viSRN0,4)
               Flag[0]++;
            // USER CODE END

        }


} //  End of function ADC0_viSRN0
```

```
//****************************************************************************
// @Function      void ADC0_viSRN1(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 1 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN1,0)

// USER CODE END

_interrupt(ADC0_SRN1INT) __registerbank(-2) void ADC0_viSRN1(void)
{

    if((ADC0_EVINFR & 0x0001) == 0x0001)
    //Source0 event interrupt
    {
        ADC0_EVINCR = 0x0001;
        // Clear source0 event interrupt

        // USER CODE BEGIN (ADC0_viSRN1,1)
          Flag[1]++;
        // USER CODE END

    }

} //  End of function ADC0_viSRN1
```

```
//****************************************************************************
// @Function      void ADC0_viSRN2(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 2 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN2,0)

// USER CODE END

_interrupt(ADC0_SRN2INT) void ADC0_viSRN2(void)
{

    if((ADC0_EVINFR & 0x0100) == 0x0100)
    //Result0 event interrupt
    {
        ADC0_EVINCR = 0x0100;
        // Clear Result0 event interrupt

        // USER CODE BEGIN (ADC0_viSRN2,20)
          Flag[2]++;
        // USER CODE END

    }
} //  End of function ADC0_viSRN2
```

```
//****************************************************************************
// @Function      void ADC0_viSRN3(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 3 of the ADC0 module.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          6/11/2008
//
//****************************************************************************

// USER CODE BEGIN (ADC0_viSRN3,0)

// USER CODE END

_interrupt(ADC0_SRN3INT) void ADC0_viSRN3(void)
{

    if((ADC0_EVINFR & 0x0002) == 0x0002)
    //Source1 event interrupt
     {
          ADC0_EVINCR = 0x0002;
          // Clear source1 event interrupt

          // USER CODE BEGIN (ADC0_viSRN3,2)
             Flag[3]++;
          // USER CODE END

     }


} //  End of function ADC0_viSRN3
```

6 Conclusion, Related Documents and Links

The enhanced interrupt handling of the XC2000 architecture is supported by DAvE with the different tool chains. But, the jump table cache interrupts (fast interrupts) are not supported in DAvE for Tasking Viper Compiler, this can be taken as future enhancement.

RELATED DOCUMENTS AND LINKS:

User's Manual, V1.0, June 2007, Vol 1, System Unit

User's Manual, V1.0, June 2007, Vol 2, Peripheral Unit

XC2000 Product Information:

http://www.infineon.com/cms/en/product/channel.html?channel=ff80808112ab681d0112ab6b687b0811

Starter Kit (Board Manual) SK-EB XC2287 Starter Kit:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304312dc768d0112e740471d0bcb

XC2000 Development Tools and Software:

http://www.infineon.com/cms/en/product/channel.html?channel=ff80808112ab681d0112ab6b4b7407b7

DAvE for the Infineon XC2000 microcontroller Family:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304316f66ee80117630459b369df

Interrupt Response Time of the XC2000/XE166 Family:

Interrupt Response Time of the XC2000/XE166 Family (ap1611111_interrupt_response_time.pdf)