XC800 Family

AP08107

XC82x/XC83x Analog-to-Digital Converter Basic and Advance Mode using DAvE

Application Note

V1.0 2010-05

Microcontrollers

Edition 2010-05

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

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

Information

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

Warnings

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

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

AP08107

Revision History: V1.0, 2010-05

Previous Version: None

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Is there any information in this document that you feel is wrong, unclear or missing?
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

1 Introduction 5
2 Features of the XC82x/XC83x ADC module 8
3 Basic and Advance Mode 9
4 Basic and Advance Mode in DAvE 10
5 Conclusion, Related Documents and Links 45

1 Introduction

This application note explains the support of Basic and Advance Mode of the Analog-to-Digital Converter module of XC82x/XC83x 8-bit Microcontrollers in DAvE (Digital Application virtual Engineer). The configuration options of the ADC module are explained in order to give insight on the usage of the module for the application needs.

The concept of Basic and Advance mode is introduced for easy use of the Analog-to-Digital Converter module for the user of DAvE.

1.1 Block Diagram of XC82x:

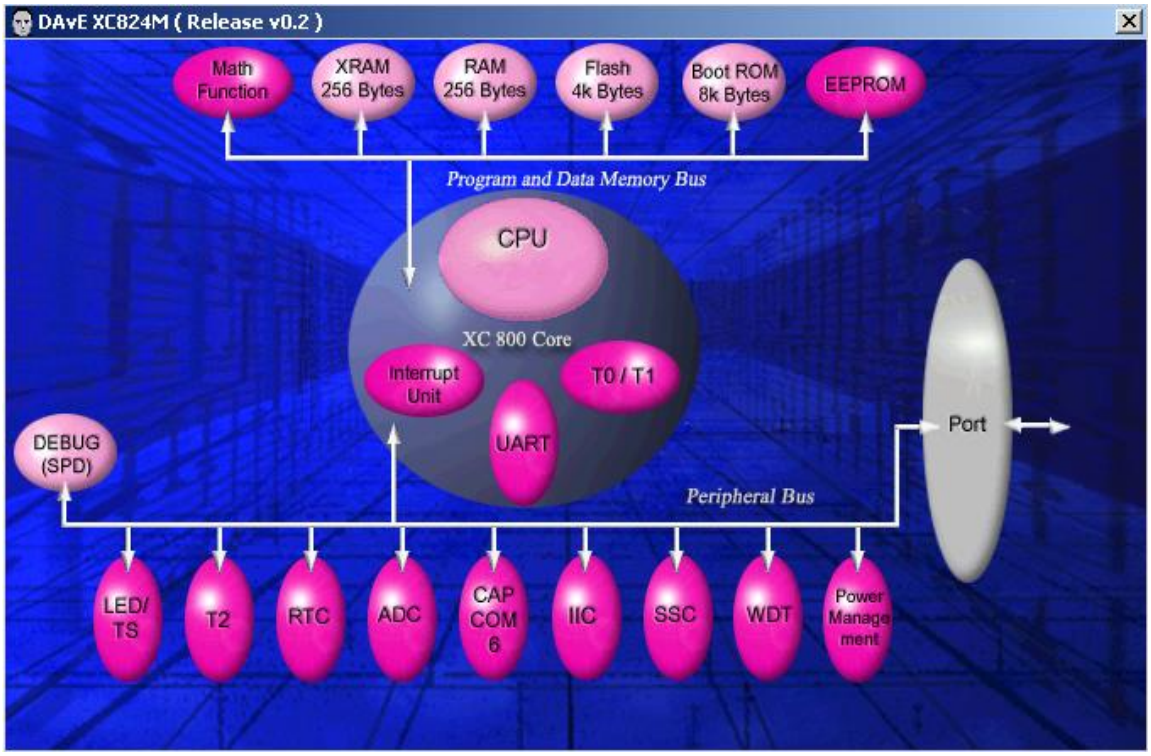


Figure 1 XC82x DAvE Block Diagram

XC82xM-1F Block Diagram

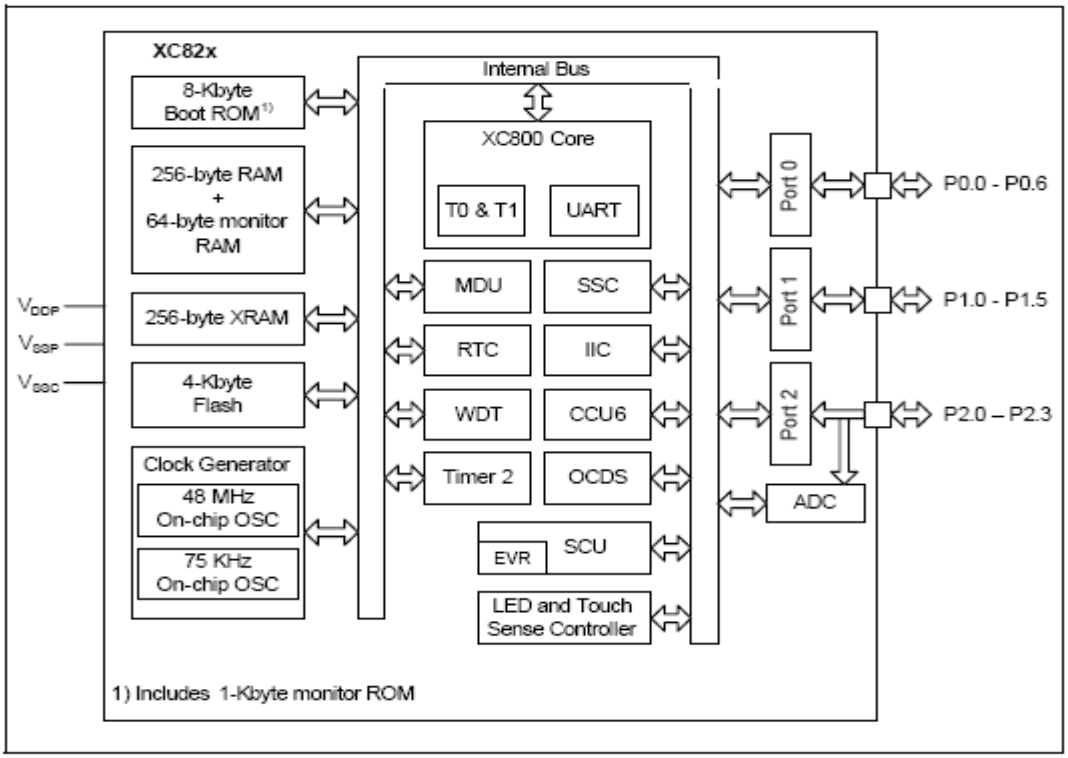


Figure 2 XC82xM-1F Block Diagram

1.2 Block Diagram of XC83x:

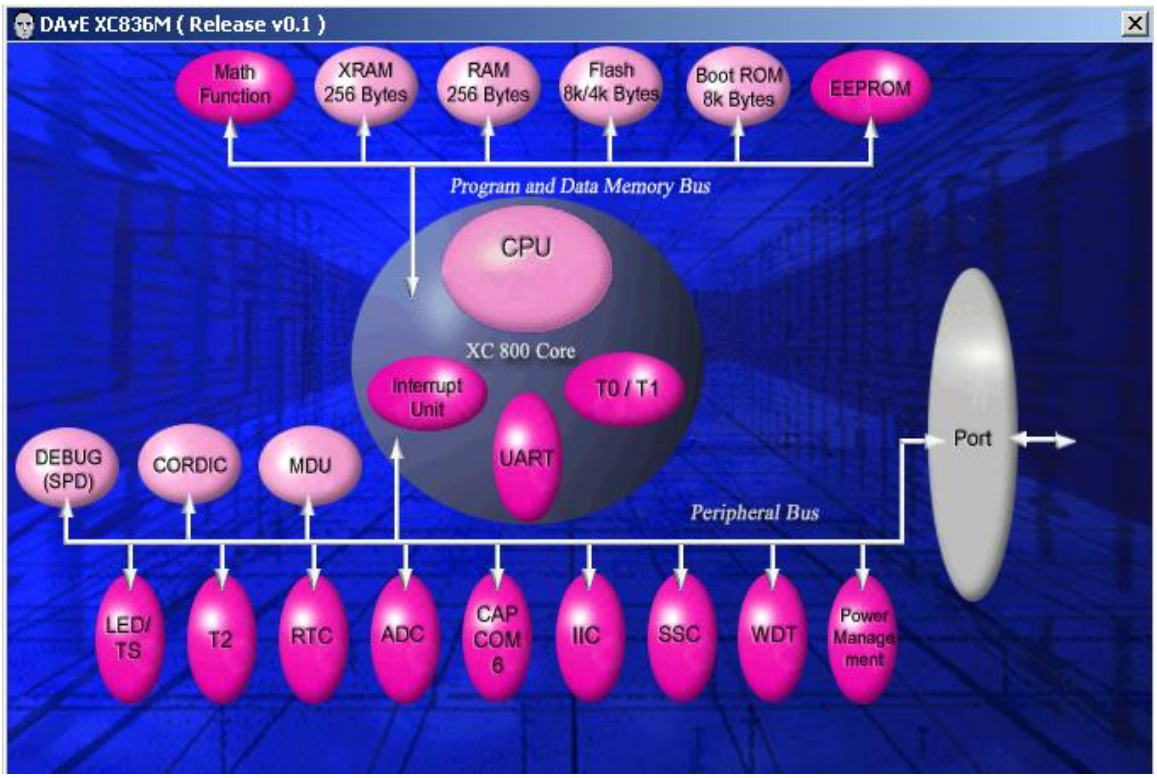


Figure 3 XC83x DAvE Block Diagram

XC83xM-2F Block Diagram

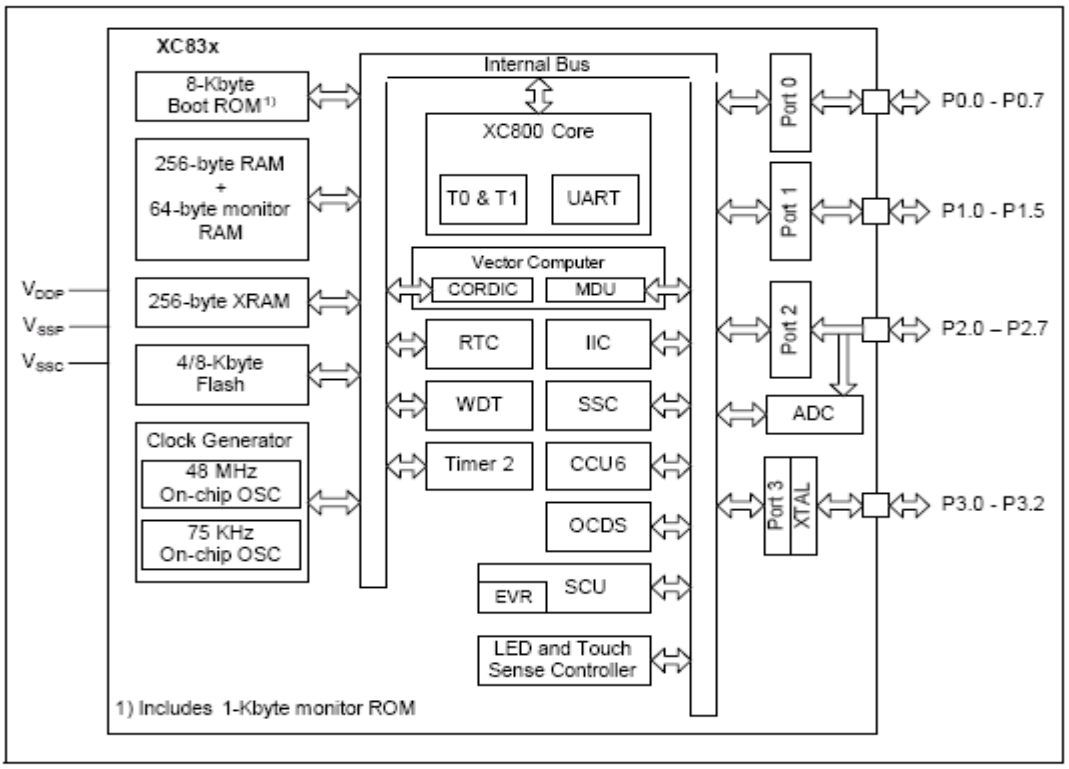


Figure 4 XC83xM-2F Block Diagram

2 Features of the XC82x/XC83x ADC module

The Analog-to-Digital Converter module (ADC) of the XC82x/XC83x uses the successive approximation method to convert analog input values (voltages) to discrete digital values.

Features of the ADC module include:

- Successive approximation
- 8-bit or 10-bit resolution
- Up to eight/four (XC83x/XC82x) analog channels and four independent result registers
- Programmable Result data protection in case of slow CPU access (wait-for-read mode)
- Single or multiple conversion modes
- Auto scan 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 Events, External Events) Triggers
- Flexible interrupt generation with fixed service nodes
- Cancel/restart feature for interrupted conversions
- Low power modes

Apart from the above features it also includes additional features:

- Out of range voltage comparator (ORC) detection for each input channel that is able to trigger other modules
- First order digital low pass filter of the conversion results
- Three internal reference voltage sources selectable for each channel
- Configurable limit checker boundary flags that can trigger other modules

The functionality of the XC82x and XC83x ADC module is same as the ADC module of other XC800 family members except for the additional features mentioned above. The application note ap0806310_Programming_ADC_XC800_Family_Microcontrollers.pdf gives detailed information about the functionality of the ADC module. This can be available at: ap0806310_Programming_ADC_XC800_Family_Microcontrollers.pdf.

3 Basic and Advance Mode

For easy use of the ADC module the concept of Basic and Advance mode has been introduced in XC82x and XC83x ADC module.

In Basic mode the basic settings have been configured automatically by DAvE for typical use similar to a standard ADC.

In Advance mode all of the settings need to be configured by the user. This is the normal mode in other 8 bit controller ADC modules.

The following are the basic settings configured in Basic mode by DAvE automatically,

General:

- Analog Clock Divider – fADC / 6
- Conversion result – 10 bit resolution
- Enable arbitration slot 0 (Sequential source)
- Arbitration started by pending conversion request

Channel Configuration:

- All channels are enabled and configured to result register 0
- Reference voltage is VDDP,VSSP

Request Sources:

- Source 0(Sequential)
  - Permanently 1 (On)
  - Priority – Low
  - Wait-For-Start

Result Register:

- Filter – None
- Wait-For-Read
- Valid-Flag-Reset

Functions:

- ADC_vInit
- ADC_Convert_8bit
- ADC_Convert_10bit
- The above two functions takes channel number as a parameter, will start conversion at the requested channel and wait till the conversion is completed and returns the conversion result.

  ADC_Convert_8bit returns upper 8 bit result of the 10 bit conversion.
  ADC_Convert_10bit returns the 10 bit conversion result.

With these basic settings the ADC module can be used for simple applications.

In Advance mode the functions ADC_Convert_8bit and ADC_Convert_10bit are not available.

4 Basic and Advance Mode in DAvE

The example shown below illustrates Basic and Advance mode for ADC module in DAvE using the XC822 easy kit board with Keil v8.18 compiler.

4.1 Description of Example for Basic Mode:

The example configures the ADC module for the Basic mode. The basic settings listed in chapter 3 have been configured by DAvE automatically as shown below.

Here, all the controls are disabled except mode selection control and Basic mode functions.

4.1.1. Select the ADC of the XC822M

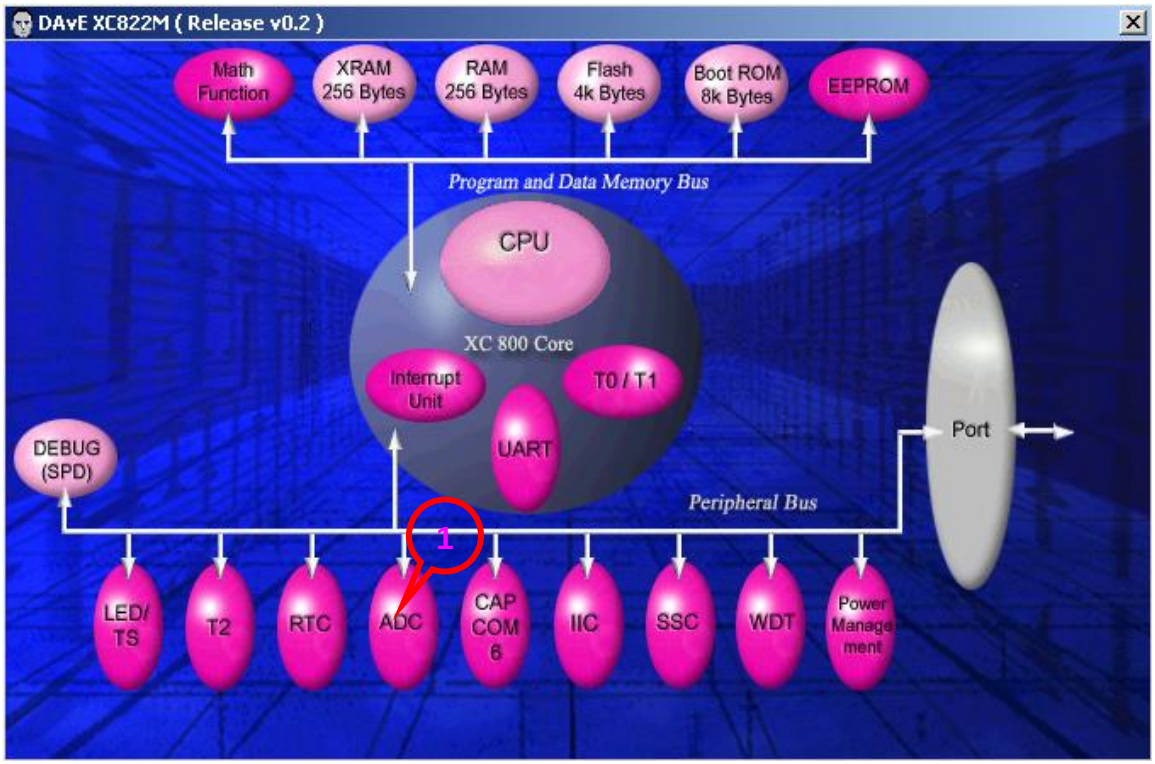


Figure 5 DAvE Bit Map showing the ADC module in Basic Mode

4.1.2. Configuration of the Module Clock Page of the ADC

1. Select “Enable Module; the peripheral is supplied with the clock signal”
The input clock for the ADC module (fADC) is 48.0 MHz

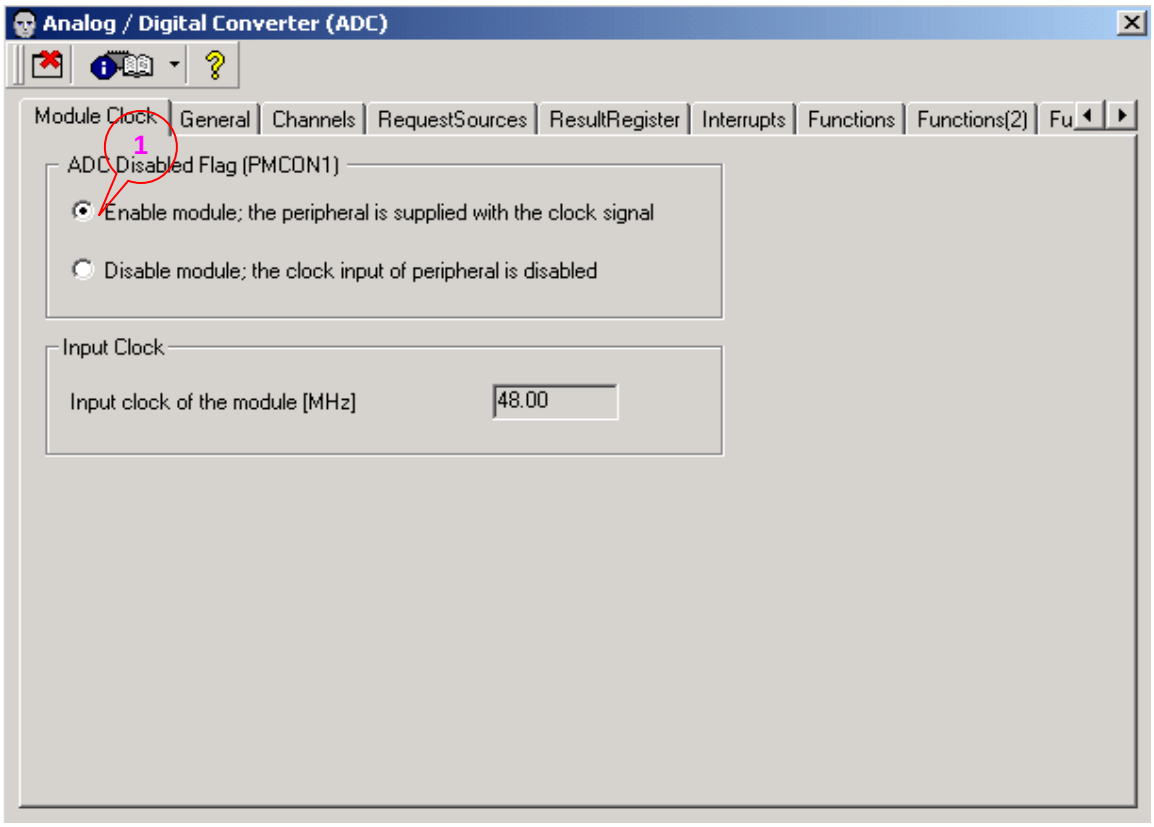


Figure 6 ADC Module Clock Page in Basic Mode

4.1.3. Configuration of the General Page of the ADC

1. Select “Basic mode; the basic settings have been configured”

When user selects Basic mode the following configurations are done automatically by DAvE under General Page,

2. Analog Clock Divider – fADC / 6
3. Conversion result – 10 bit resolution
4. Enable arbitration slot 0 (Sequential source)
5. Arbitration started by pending conversion request

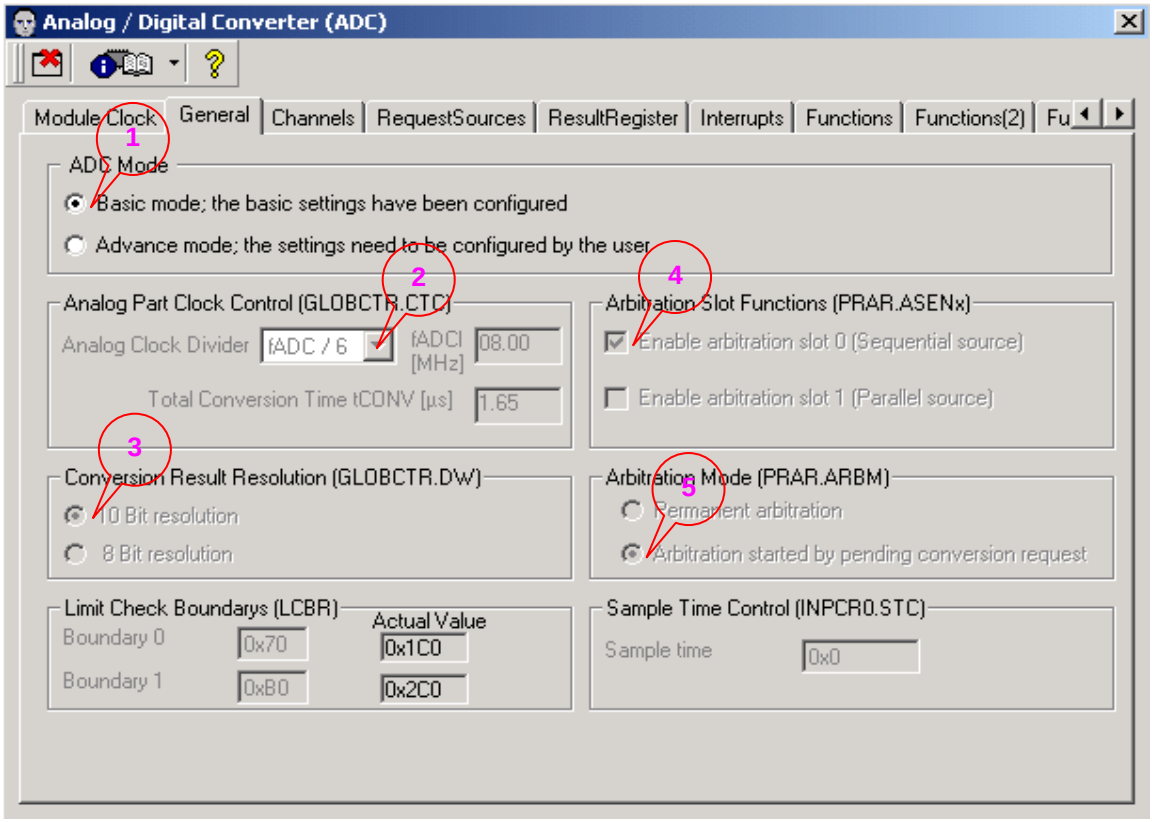


Figure 7 ADC General Page in Basic Mode

4.1.4. Configuration of the Channels Page of the ADC

All the available channels are selected automatically by DAvE.

1. Channel 0 is selected.
Similarly remaining available channels are also selected.

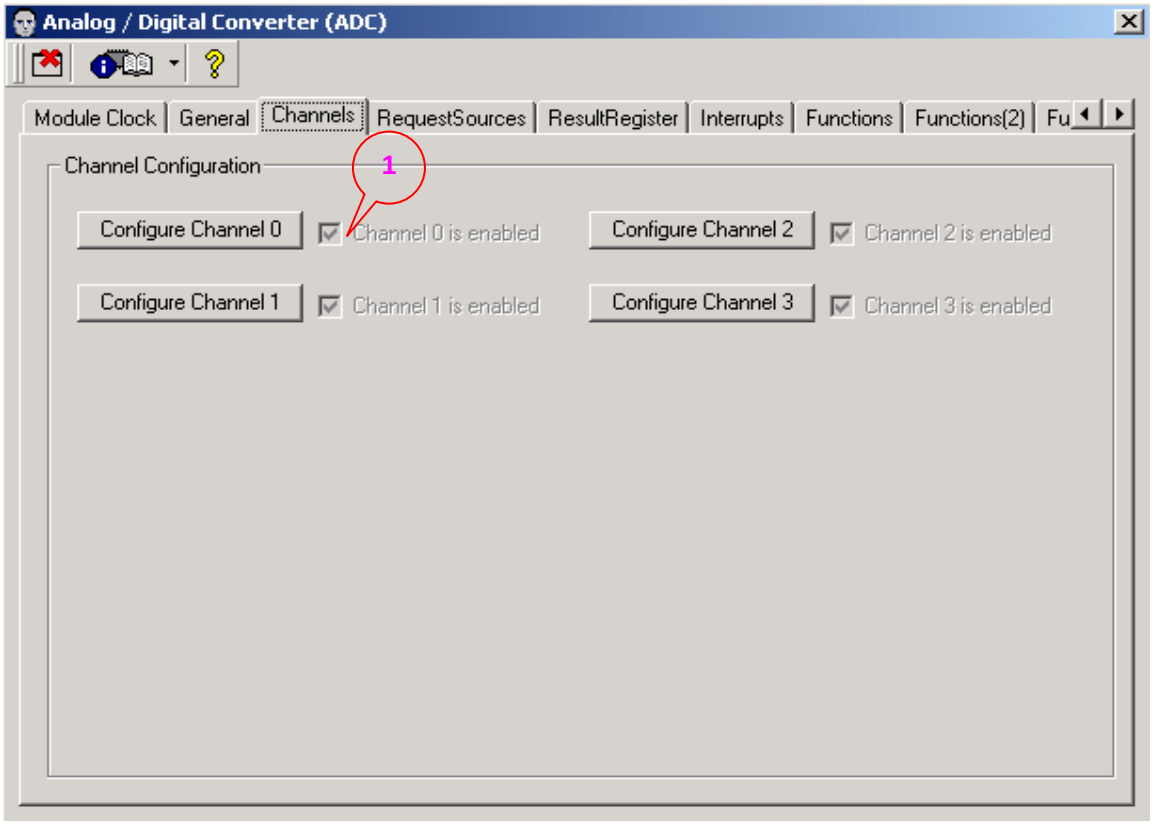


Figure 8 ADC Channel Selection Page in Basic Mode

4.1.5. Configuration for the Channel 0 General Settings Page of the ADC

1. Channel 0 is selected
2. Result register 0 is selected
3. Vddp, Vssp is selected as reference voltage

The same configuration applies for the remaining selected channels

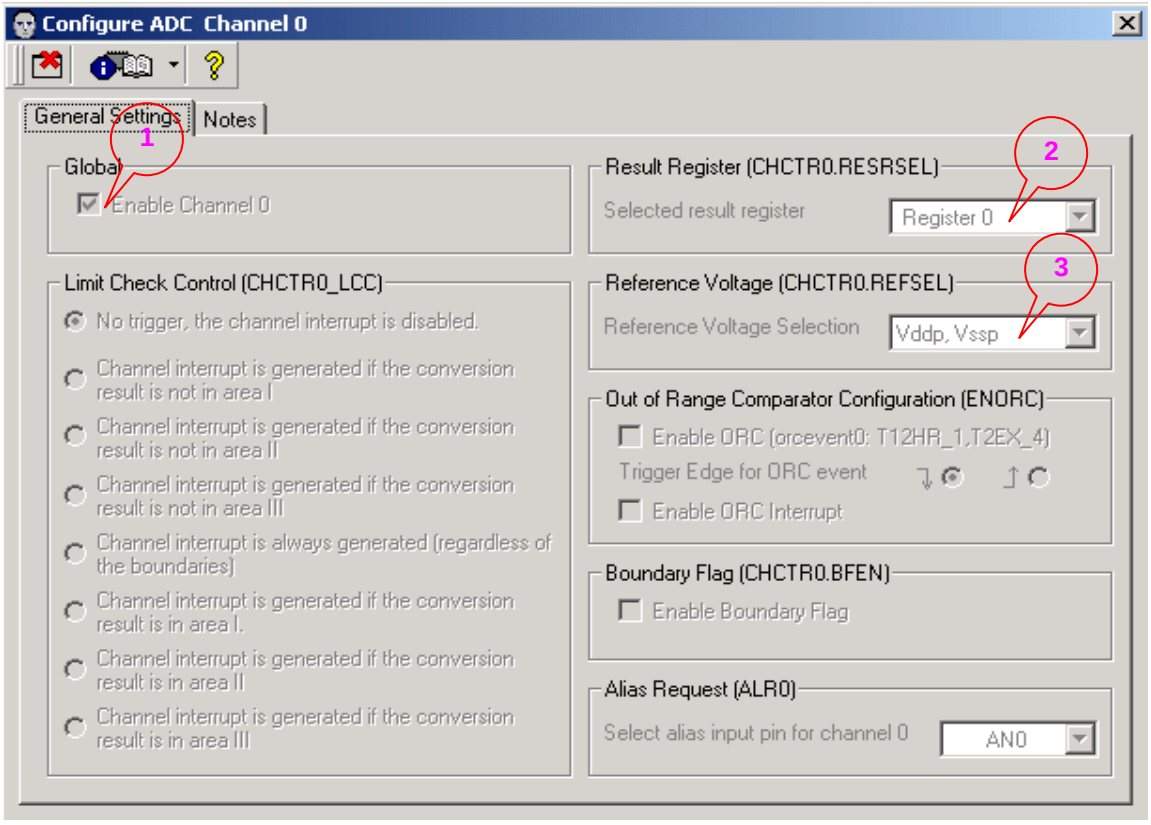


Figure 9 ADC Channel 0 General Settings Page in Basic Mode

4.1.6. Configuration for the Request Sources Page of the ADC

1. Sequential source 0 is enabled
2. Priority is set to low
3. Wait-for-start mode is selected

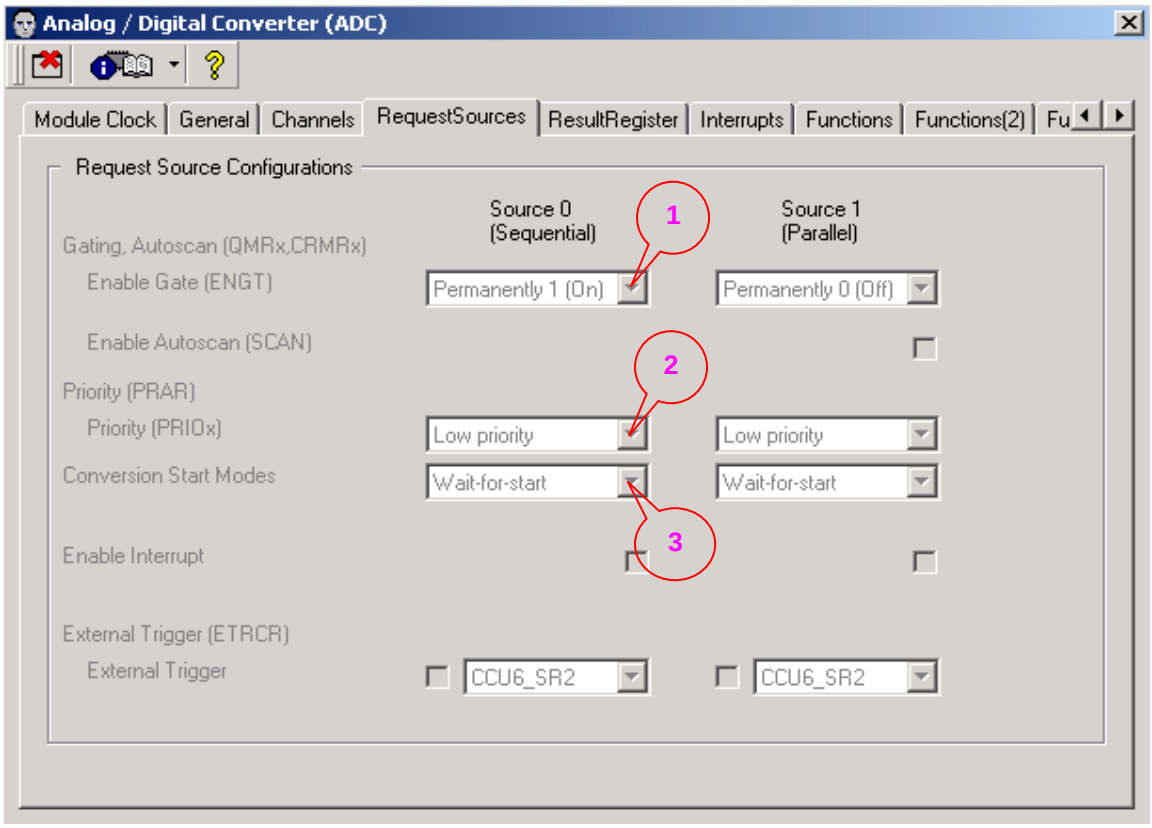


Figure 10 ADC Request Sources Page in Basic Mode

4.1.7. Configuration for the Result Register Page of the ADC in Basic Mode

1. No Filter is selected
2. Wait-For-Read mode is selected
3. Valid-Flag-Reset is selected

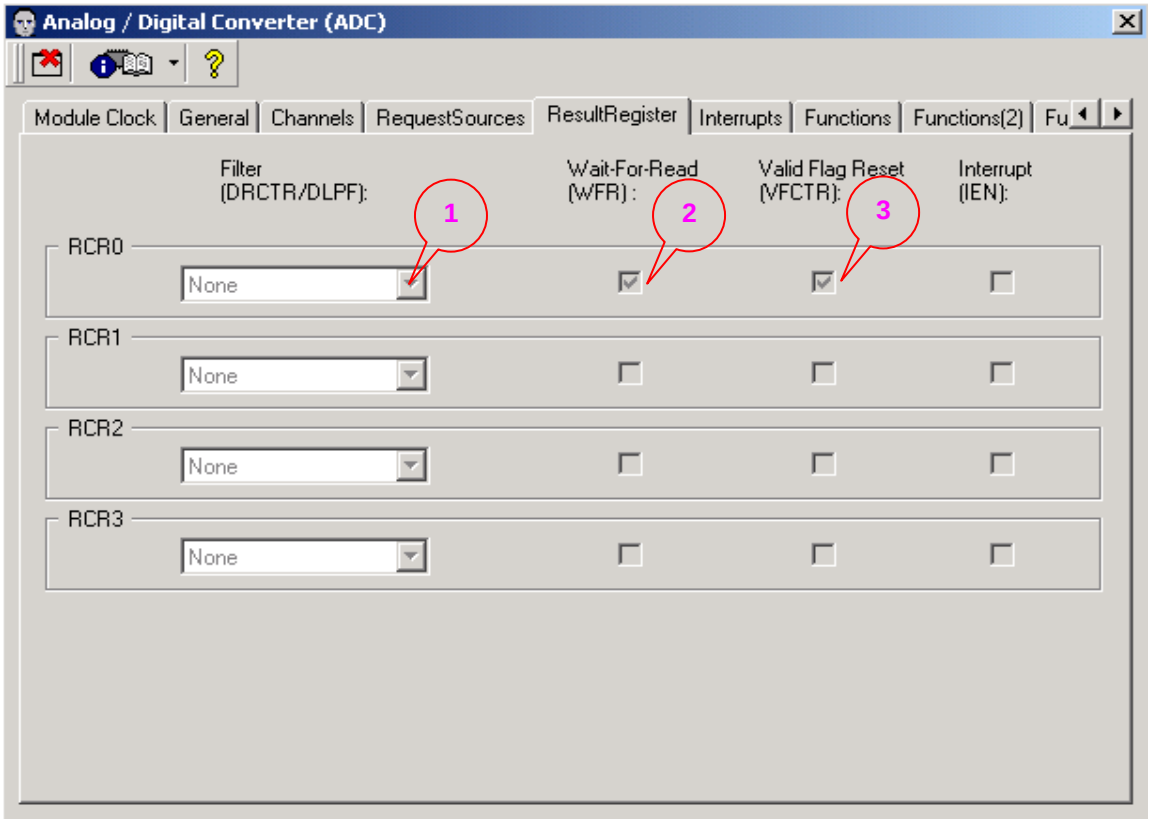
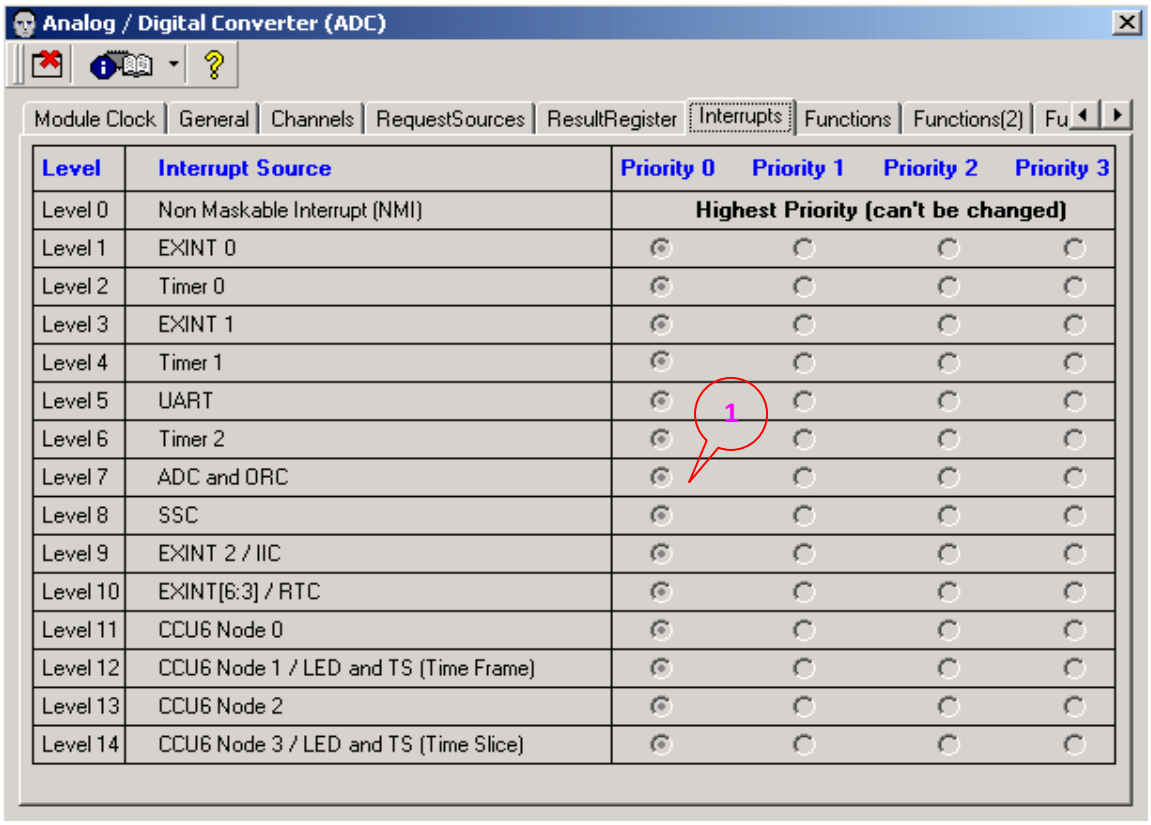


Figure 11 ADC Result Register Page in Basic Mode

4.1.8. Configuration for the Interrupt Page of the ADC in Basic Mode

1. No interrupts are selected



Level	Interrupt Source	Priority 0	Priority 1	Priority 2	Priority 3
Level 0	Non Maskable Interrupt (NMI)	Highest Priority (can't be changed)			
Level 1	EXINT 0				
Level 2	Timer 0				
Level 3	EXINT 1				
Level 4	Timer 1				
Level 5	UART				
Level 6	Timer 2				
Level 7	ADC and ORC				
Level 8	SSC				
Level 9	EXINT 2 / IIC				
Level 10	EXINT[6:3] / RTC				
Level 11	CCU6 Node 0				
Level 12	CCU6 Node 1 / LED and TS (Time Frame)				
Level 13	CCU6 Node 2				
Level 14	CCU6 Node 3 / LED and TS (Time Slice)				

Figure 12 ADC Interrupt Page in Basic Mode

4.1.9. Configuration for the Functions Page of the ADC in Basic Mode

The following functions are selected by DAvE automatically in Basic mode

1. ADC_vInit
2. ADC_Convert_8bit
3. ADC_Convert_10bit

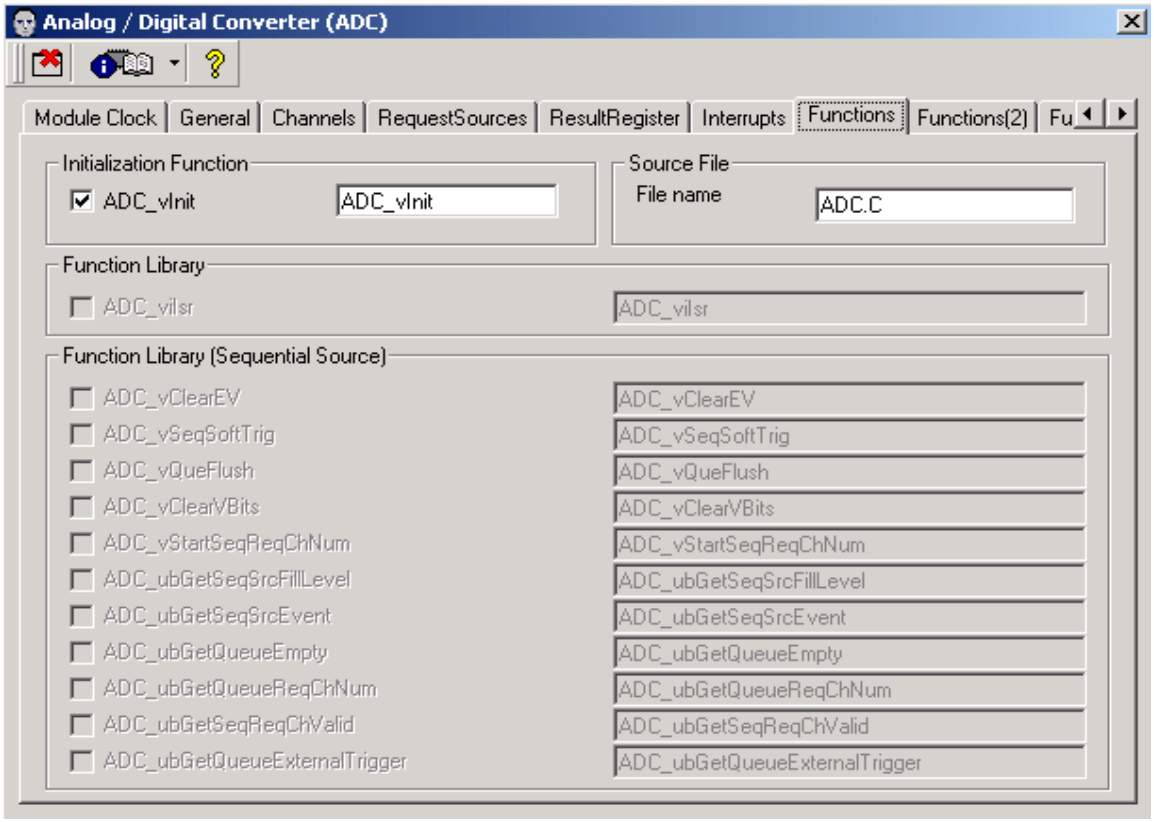


Figure 13 ADC Functions Page in Basic Mode

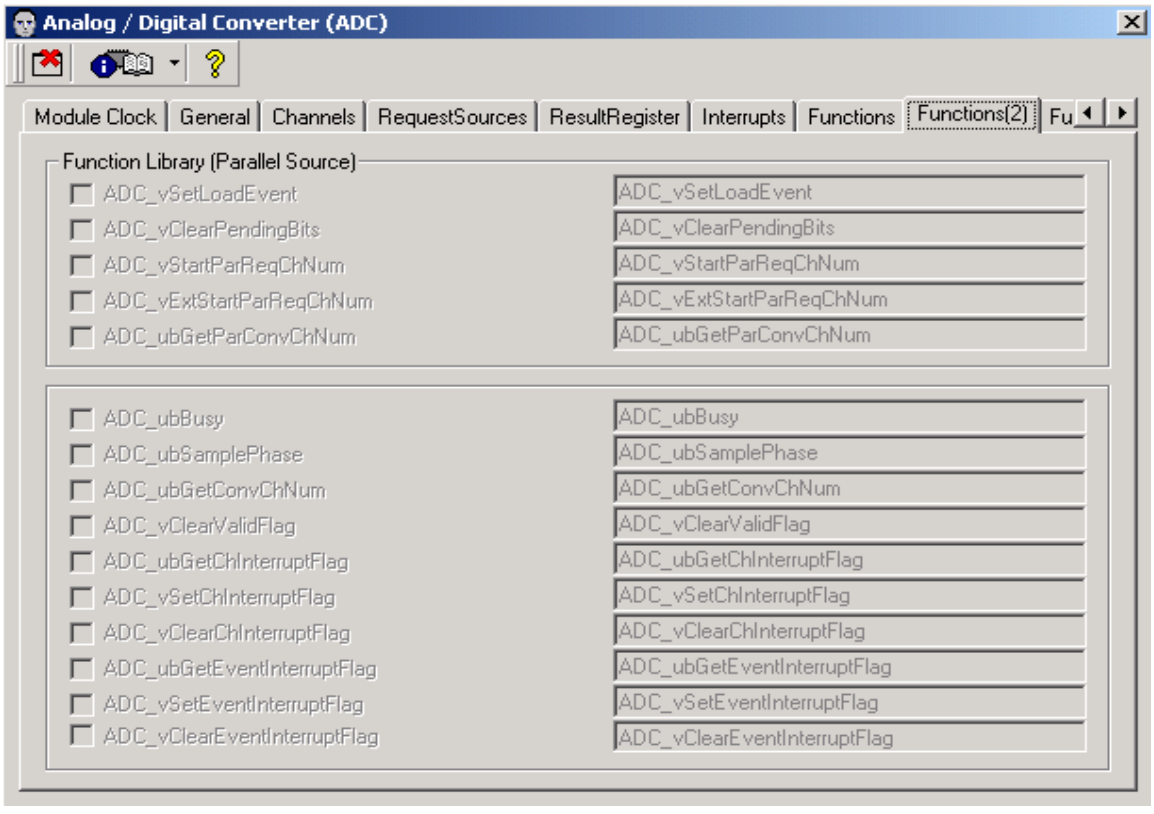


Figure 14 ADC Functions(2) Page in Basic Mode

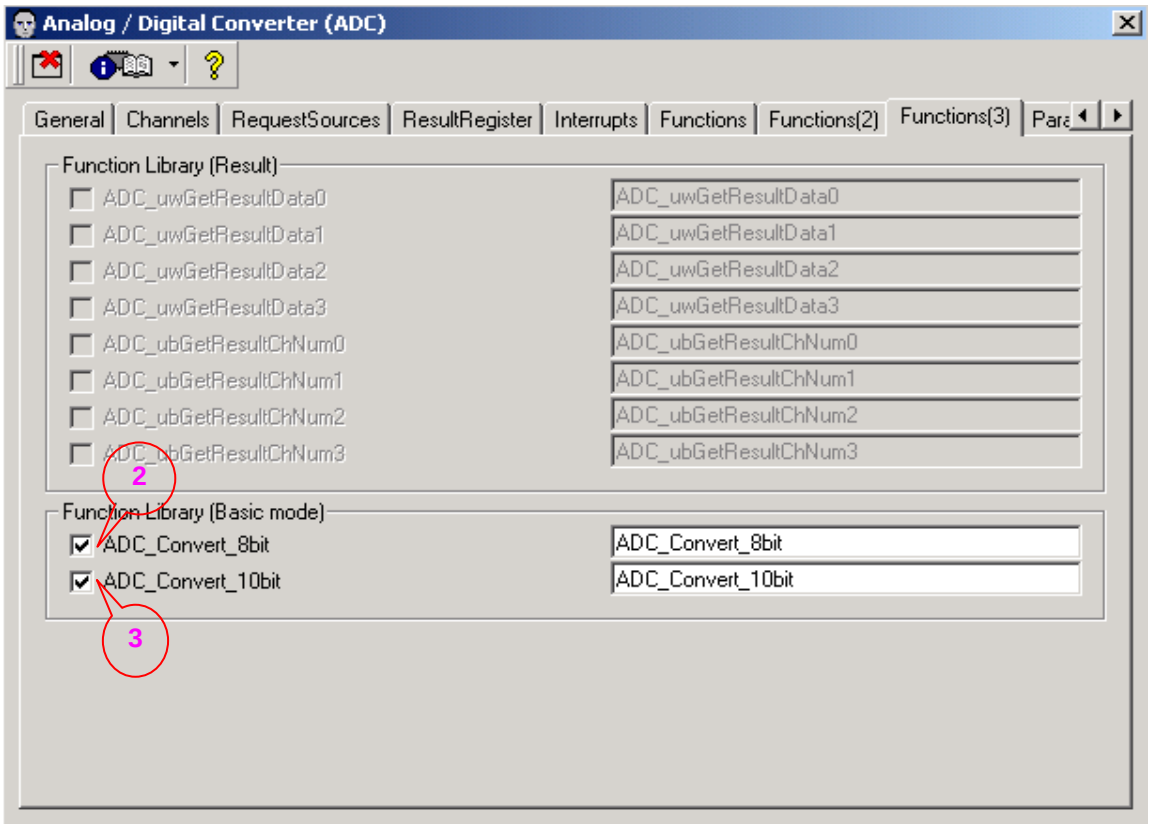


Figure 15 ADC Functions(3) Page in Basic Mode

4.2 Example Code for ADC Basic Mode:

The following C-code shows the handling of Basic mode for the ADC module.

The code is for a XC822M device with Keil v8.18 compiler.

Tools Used:

DAvE v2.1r22
XC82xM_Series_v0_2.dip
XC822 Easy Kit Board
XC800_FLOAD_v50d12
Keil Compiler v8.18

The following code is the main function. The ADC conversion is done and the result is read using Basic Mode functions. This main function is available in the MAIN.C file of DAvE generated code.

```
void main(void)
{
  // USER CODE BEGIN (MAIN_Main,2)
            uword uwResult;
  // USER CODE END

  MAIN_vInit();

  // USER CODE BEGIN (MAIN_Main,3)
    TI = 1;
     Printf("-----------------------------------------------------\n");
     printf("\r\n");
     printf("\r\n");
     printf("-----------------------------------------------------\r\n");
     printf("Microcontroller :Infineon XC82X \r\n");
     printf("Module              :ADC \r\n");
     printf("Example             :ADC_BasicMode_Example\r\n");
     printf("-----------------------------------------------------\r\n");
     printf("\r\n");
     printf("Demonstration of ADC Basic Mode..\r\n");


  // USER CODE END

  while(1)
  {

      // USER CODE BEGIN (MAIN_Main,4)

      // Start Channel 0 conversion and read 10 bit converted result from result
      // register 0
        uwResult = ADC_Convert_10bit(0);
        printf("\r\nConverted 10 bit result for channel 0 = 0x%x\r\n",uwResult);

      // Start Channel 0 conversion and read 8 bit converted result from result
      // register 0
        uwResult = ADC_Convert_8bit(0);
        printf("\r\nConverted 8 bit result for channel 0 = 0x%x\r\n" ,uwResult);
```

```
    // Start Channel 1 conversion and read 10 bit converted result from result
    // register 0
      uwResult = ADC_Convert_10bit(1);
      printf("\r\nConverted 10 bit result for channel 1 = 0x%x\r\n",uwResult);

    // Start Channel 1 conversion and read 8 bit converted result from result
    // register 0
      uwResult = ADC_Convert_8bit(1);
      printf("\r\nConverted 8 bit result for channel 1 = 0x%x\r\n" ,uwResult);


    // Start Channel 2 conversion and read 10 bit converted result from result
    // register 0
      uwResult = ADC_Convert_10bit(2);
      printf("\r\nConverted 10 bit result for channel 2 = 0x%x\r\n",uwResult);

    // Start Channel 2 conversion and read 8 bit converted result from result
    // register 0
      uwResult = ADC_Convert_8bit(2);
      printf("\r\nConverted 8 bit result for channel 2 = 0x%x\r\n" ,uwResult);


    //Start Channel 3 conversion and read 10 bit converted result from result
    // register 0
      uwResult = ADC_Convert_10bit(3);
      printf("\r\nConverted 10 bit result for channel 3 = 0x%x\r\n",uwResult);

    // Start Channel 3 conversion and read 8 bit converted result from result
    // register 0
      uwResult = ADC_Convert_8bit(3);
      printf("\r\nConverted 8 bit result for channel 3 = 0x%x\r\n" ,uwResult);

    // USER CODE END
  }

} //  End of function main
```

The following code shows the ADC Basic mode configuration.This code is available in the ADC.C file.

```
void ADC_vInit(void)
{
  // USER CODE BEGIN (ADC_Init,2)

  // USER CODE END

  ///  ----------------------------------------------------------------------
  ///  Configuration of Global Control:
  ///  ----------------------------------------------------------------------
  ///  - the ADC module clock is enabled
  ///  - the ADC module clock = 48.00 MHz
  ///

  ///  - the ADC Basic mode is selected
  ///

  ///  - the result is 10 bits wide
  ///  --- Conversion Timing -----------------
  ///  - conversion time (CTC)    = 01.65 us

  ///  - Configure global control functions

  SFR_PAGE(_ad0, noSST);         // switch to page 0

  ADC_GLOBCTR   =  0x30;         // load global control register

  ///  ----------------------------------------------------------------------
  ///  Configuration of Priority and Arbitration:
  ///  ----------------------------------------------------------------------
  ///  - 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 arbitration started by pending conversion request is selected
  ///  - Arbitration Slot 0 is enabled
  ///  - Arbitration Slot 1 is disabled

  ADC_PRAR      =  0x50;          // load Priority and Arbitration register

  SFR_PAGE(_ad1, noSST);      // switch to page 1

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

  ADC_CHCTR0    =  0x00;         // load channel control register

  ///  Configuration of Channel 1
  ///  - the result register0 is selected
  ///  - the limit check 0 is selected

  ADC_CHCTR1    =  0x00;         // load channel control register
```

```
///  Configuration of Channel 2
///  - the result register0 is selected
///  - the limit check 0 is selected

ADC_CHCTR2    =  0x00;        // load channel control register

///  Configuration of Channel 3
///  - the result register0 is selected
///  - the limit check 0 is selected

ADC_CHCTR3    =  0x00;        // load channel control register

SFR_PAGE(_ad0, noSST);      // switch to page 0

///  ---------------------------------------------------------------------
///  Configuration of Sample Time Control:
///  ---------------------------------------------------------------------

ADC_INPCR0    =  0x00;        // load input class register

///  ---------------------------------------------------------------------
///  Configuration of Out of range comparator:
///  ---------------------------------------------------------------------

ADC_ENORC     =  0x00;        // load out of range comparator register

SFR_PAGE(_ad4, noSST);      // switch to page 4

///  ---------------------------------------------------------------------
///  Configuration of Out of range comparator edge trigger:
///  ---------------------------------------------------------------------

ADC_CNF       =  0x00;          // load out of range comparator trigger edge
                                // select register


///  ---------------------------------------------------------------------
///  Configuration of alias register:
///  ---------------------------------------------------------------------

ADC_ALR0      =  0x00;        // load alias register 0

///  ---------------------------------------------------------------------
///  Configuration of Result Control Registers:
///  ---------------------------------------------------------------------
///  Configuration of Result Control Register 0
///  - the data reduction filter is disabled
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is enabled
///  - the VF reset by read access to RESRxH

ADC_RCR0      =  0xC0;        // load result control register 0

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

```
///  - the VF unchanged by read access to RESRxH

ADC_RCR1     =  0x00;        // load result control register 1

///  Configuration of Result Control Register 2
///  - the data reduction filter is disabled
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled
///  - the VF unchanged by read access to RESRxH

ADC_RCR2     =  0x00;        // load result control register 2

///  Configuration of Result Control Register 3
///  - the data reduction filter is disabled
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled
///  - the VF unchanged by read access to RESRxH

ADC_RCR3     =  0x00;        // load result control register 3


///  ---------------------------------------------------------------------
///  Channel Interrupt Node Pointer:
///  ---------------------------------------------------------------------
///  - the SR 0 line become activated if any channel  interrupt is generated

///  ---------------------------------------------------------------------
///  Out of range comparator Interrupt Node Pointer:
///  ---------------------------------------------------------------------
///  - the SR 1 line become activated if out of range comparator  interrupt
///  for any channel is generated

///  ---------------------------------------------------------------------
///  Event Interrupt Node Pointer:
///  ---------------------------------------------------------------------
///  - the SR 0 line become activated if the event 0-1  interrupt is
///  generated
///  - the SR 0 line become activated if the event 4-7  interrupt is
///  generated

SFR_PAGE(_ad0, noSST);   // switch to page 0

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

ADC_LCBR0    =  0x70;       // load limit check boundary register 0
ADC_LCBR1    =  0xB0;       // load limit check boundary register 1

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

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

```
ADC_QMR0     = 0x01;      // load queue mode register
///  ----------------------------------------------------------------------
///  Configuration of Conversion Request Mode Registers:
///  ----------------------------------------------------------------------
///  - the gating line is permanently 0
///  - the external trigger is disabled
///  - the source interrupt is disabled
///  - the autoscan functionality is disabled

ADC_CRMR1    = 0x00;    // load conversion request mode register 1

SFR_PAGE(_ad0, noSST);  // switch to page 0

ADC_GLOBCTR  |= 0x80;  // turn on Analog part

///  - Out of range comparator -Interrupt (ORCIEN) remains disabled

///  - Channel limit checking -Interrupt (CLCIEN) remains disabled

///  - ADC -Interrupt (EADC) remains disabled


// USER CODE BEGIN (ADC_Init,3)

// USER CODE END

} //  End of function ADC_vInit
```

The following code shows the ADC Basic mode function definitions which are used to start ADC conversion for the requested channel. Wait until the conversion is complete and read the 8 bit and 10 bit results from result register 0. These functions are also available in the ADC.C file. These functions take the channel number as the input parameter.

```
uword ADC_Convert_8bit(ubyte ubChannelNum)
{

  ubyte ubVal = 0;
  uword uwResult = 0xFFF;
  ubVal = ubVal + (ubChannelNum & 0x07);

  SFR_PAGE(ADC_QINR0_PAGE, SST1); // switch to page 6
  ADC_QINR0 = ubVal;            // requested channel
  SFR_PAGE(_ad0, RST1);         // restore the old ADC page

  SFR_PAGE(_ad0, SST1);         // switch to page 0
  //   wait until conversion is complete
  while ((ADC_GLOBSTR & 0x01));
  SFR_PAGE(_ad0, RST1);         // restore the old ADC page

  SFR_PAGE(ADC_RESR0L_PAGE, SST1); // switch to page 2
  if ( ADC_RESR0L & 0x10 )      //  if Result Register 0 contains valid data
  {
     // 8-bit conversion (without accumulation))
     uwResult = ((ADC_RESR0L >> 7) & 0x01); // Result Register 0 Low
     uwResult = (((uword)(ADC_RESR0H << 1)) + uwResult); // Result Register 0
                                                         // High
  }
  SFR_PAGE(_ad0, RST1);         // restore the old ADC page

  return(uwResult);

} //  End of function ADC_Convert_8bit
```

```
uword ADC_Convert_10bit(ubyte ubChannelNum)
{

  ubyte ubVal = 0;
  uword uwResult = 0xFFF;
  ubVal = ubVal + (ubChannelNum & 0x07);

  SFR_PAGE(ADC_QINR0_PAGE, SST1); // switch to page 6
  ADC_QINR0 = ubVal;            // requested channel
  SFR_PAGE(_ad0, RST1);         // restore the old ADC page

  SFR_PAGE(_ad0, SST1);         // switch to page 0
  //   wait until conversion is complete
  while ((ADC_GLOBSTR & 0x01));
  SFR_PAGE(_ad0, RST1);         // restore the old ADC page

  SFR_PAGE(ADC_RESR0L_PAGE, SST1); // switch to page 2
  if ( ADC_RESR0L & 0x10 )      //  if Result Register0 contains valid data
  {
     // 10-bit conversion (without accumulation))
     uwResult = ((ADC_RESR0L >> 5) & 0x07); // Result Register 0 Low
     uwResult = (((uword)(ADC_RESR0H << 3)) + uwResult); // Result Register 0
                                                         // High
  }
  SFR_PAGE(_ad0, RST1);         // restore the old ADC page

  return(uwResult);

} //  End of function ADC_Convert_10bit
```

This is how the ADC Basic mode functionality can be used in applications without bothering with the advanced configurations.

4.3 Description of Example for Advance Mode:

The example configures ADC module for the Advance mode. The configurations for the different DAvE screens are shown below.

Here, all the controls are enabled for the user to configure the ADC and the Basic mode functions are disabled.

4.3.1 Select the ADC of the XC822M

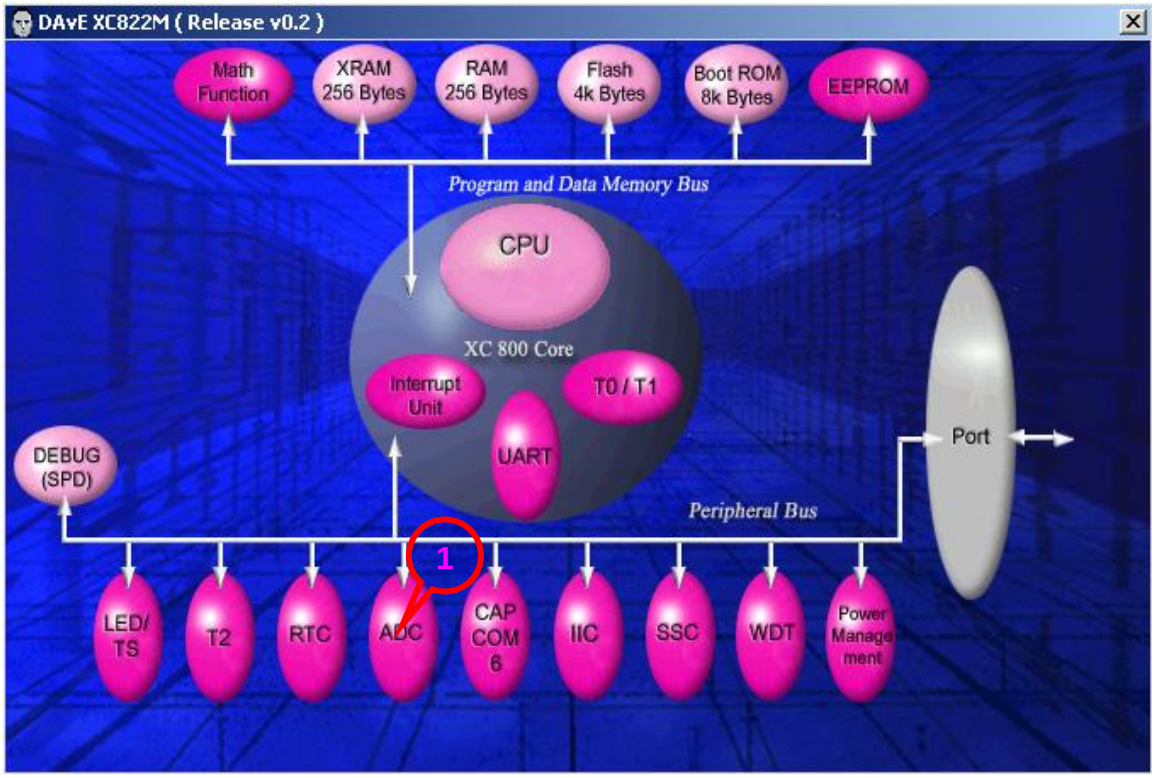


Figure 16 DAvE Bit Map showing the ADC module in Advance Mode

4.3.2 Configuration of the Module Clock Page of the ADC

1. Select “Enable Module; the peripheral is supplied with the clock signal”
The input clock for the ADC module (fADC) is 48.0 MHz

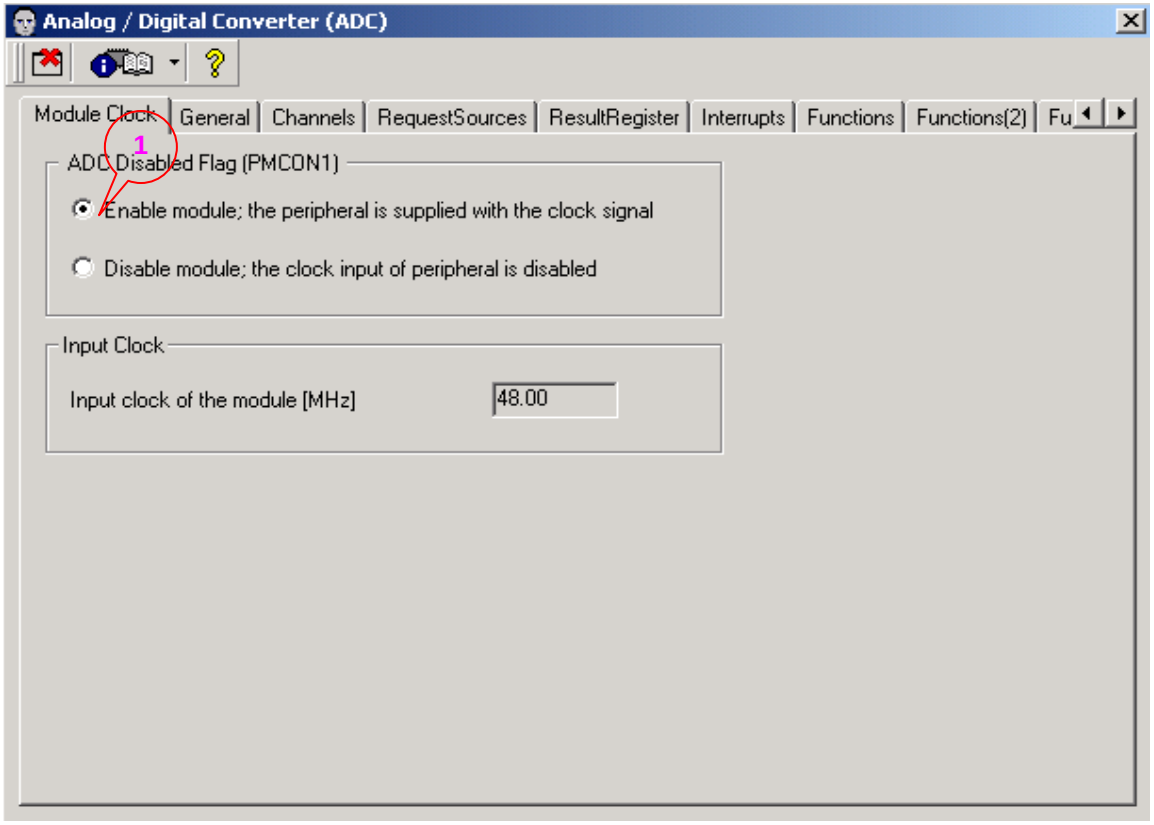


Figure 17 ADC Module Clock Page in Basic Mode

4.3.3 Configuration of the General Page of the ADC in Advance Mode

1. Select “Advance mode; the settings need to be configured by the user”

The following configurations are done in the General Page,

2. Analog Clock Divider – fADC / 4
3. Conversion result – 8 bit resolution
4. Enable arbitration slot 1 (Parallel source)
5. Permanent arbitration

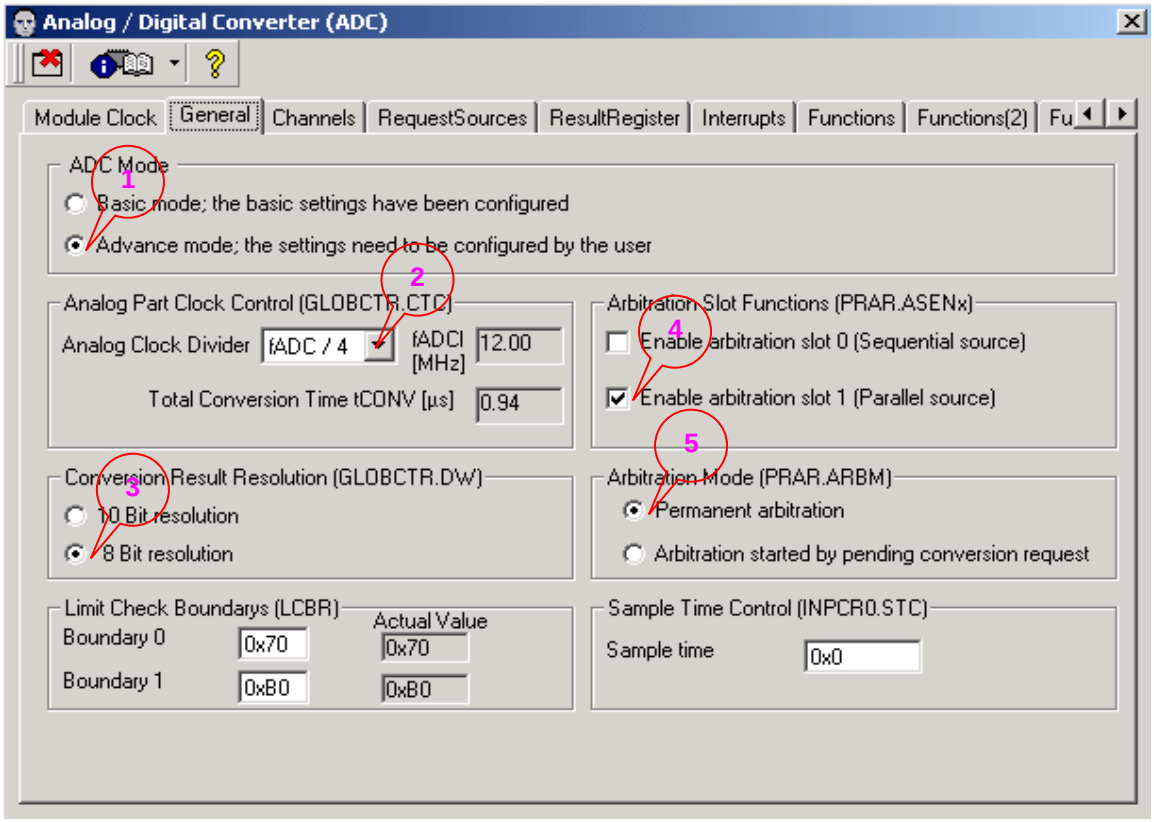


Figure 18 ADC General Page in Advance Mode

4.3.4 Configuration of the Channels Page of the ADC in Advance Mode

1. Channel 2 is selected.
2. Channel 3 is selected.

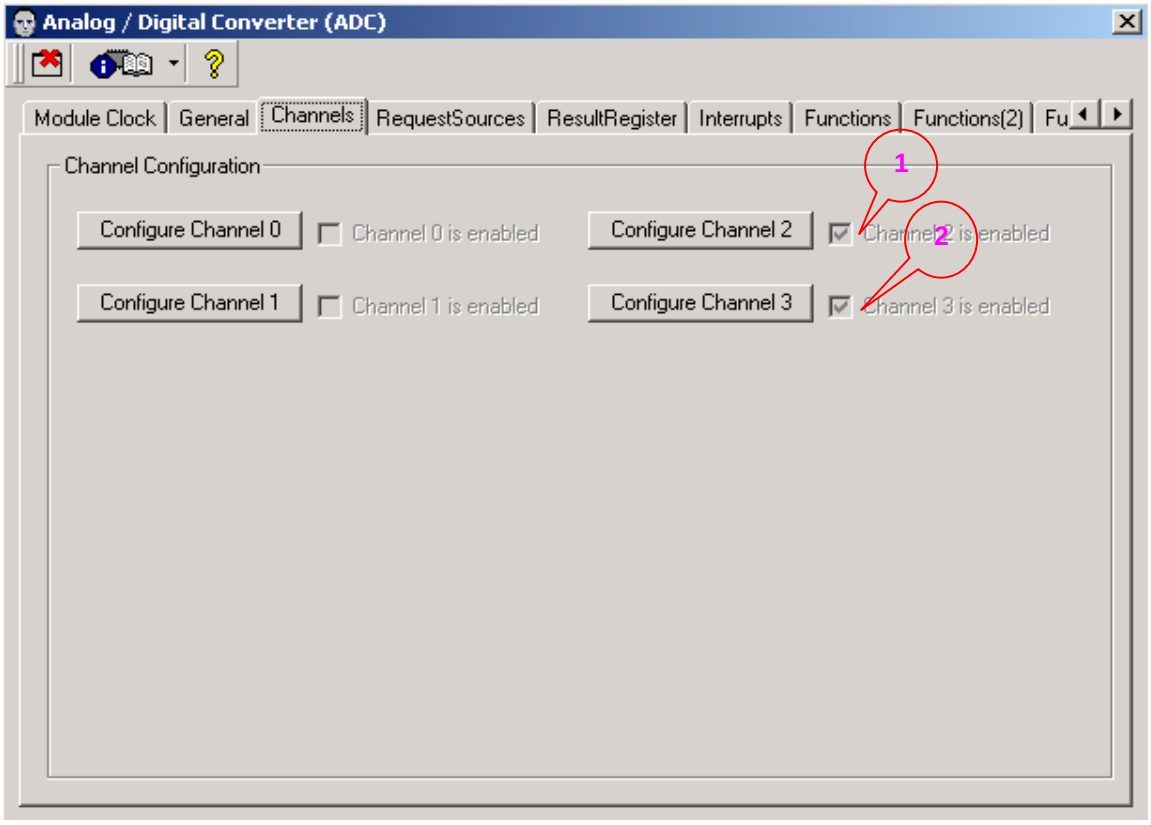


Figure 19 ADC Channel Selection Page in Advance Mode

4.3.5 Configuration for the Channel 2 General Settings Page of the ADC in Advance Mode

1. Channel 2 is selected
2. Result register 2 is selected
3. Vddp, Vssp is selected as reference voltage

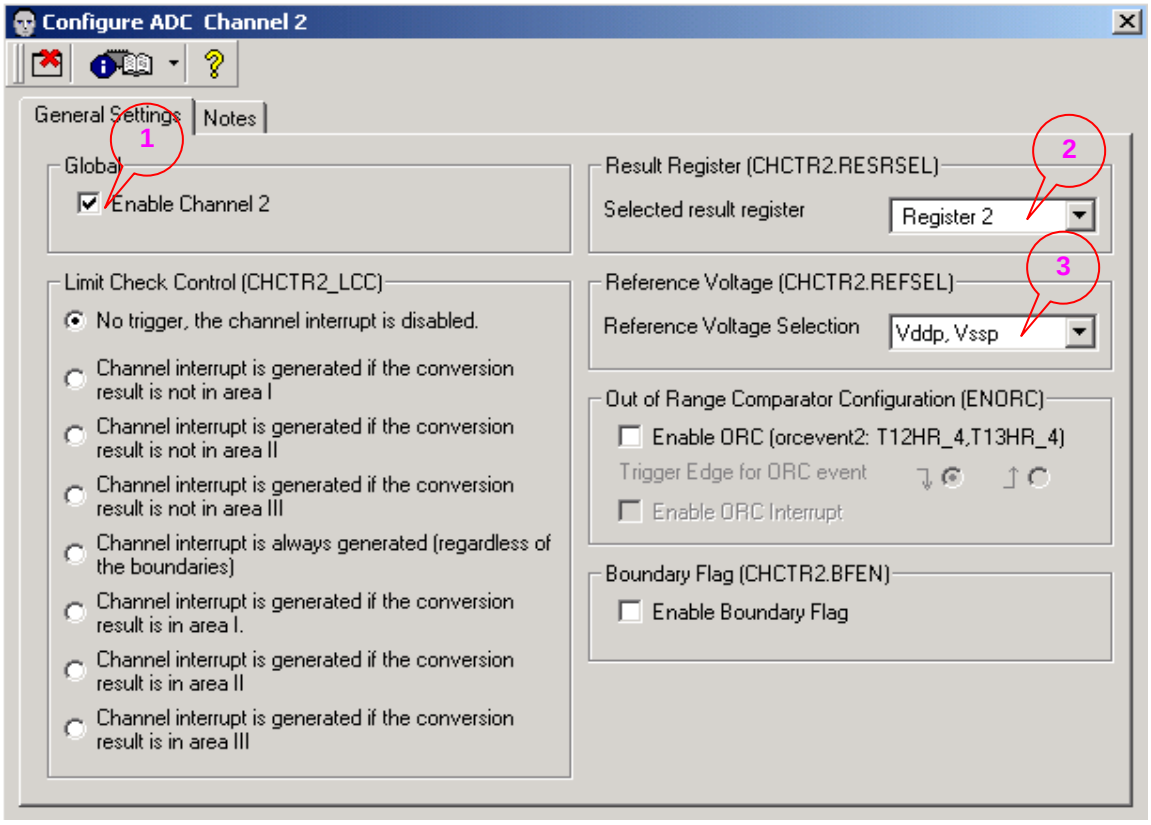


Figure 20 ADC Channel 2 General Settings Page in Advance Mode

4.3.6 Configuration for the Channel 3 General Settings Page of the ADC in Advance Mode

1. Channel 3 is selected
2. Result register 3 is selected
3. Vddp, Vssp is selected as reference voltage

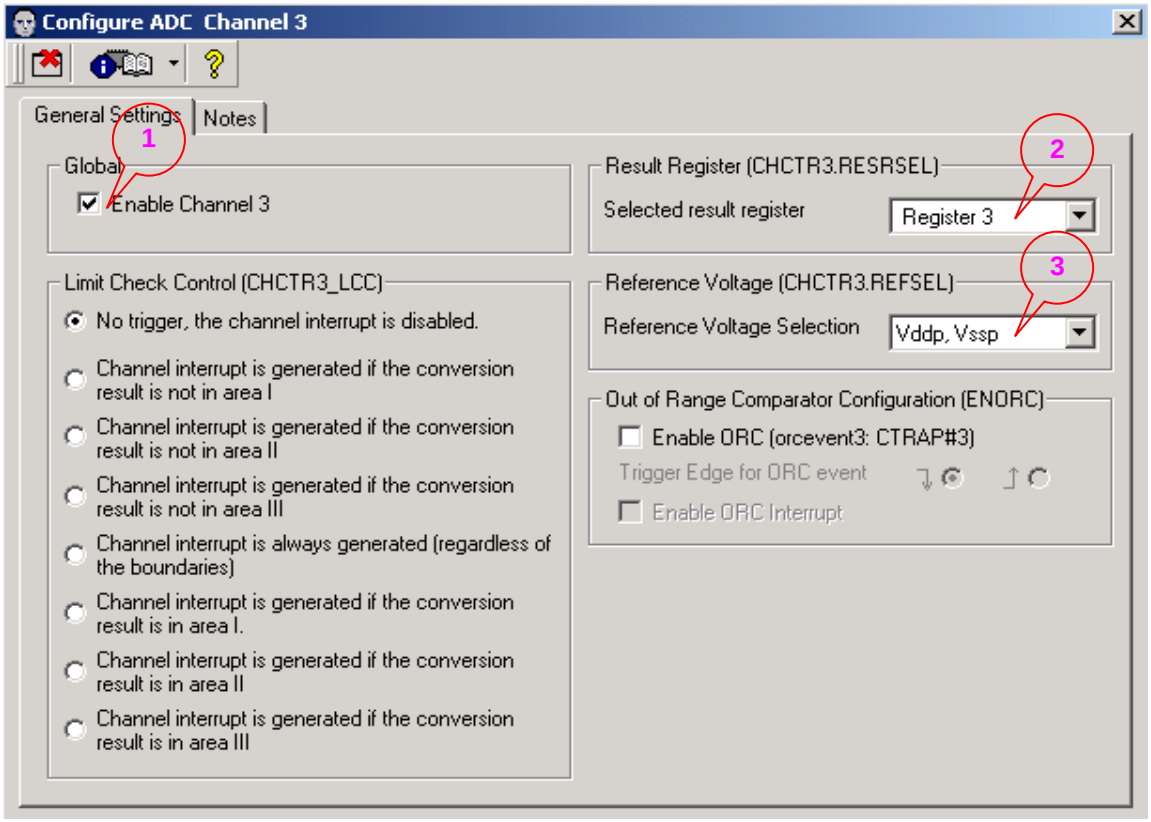


Figure 21 ADC Channel 3 General Settings Page in Advance Mode

4.3.7 Configuration for the Request Sources Page of the ADC in Advance Mode

1. Parallel source 1 is enabled
2. Auto scan is enabled
3. Priority is set to low
4. Wait-for-start mode is selected
5. Source interrupt is enabled

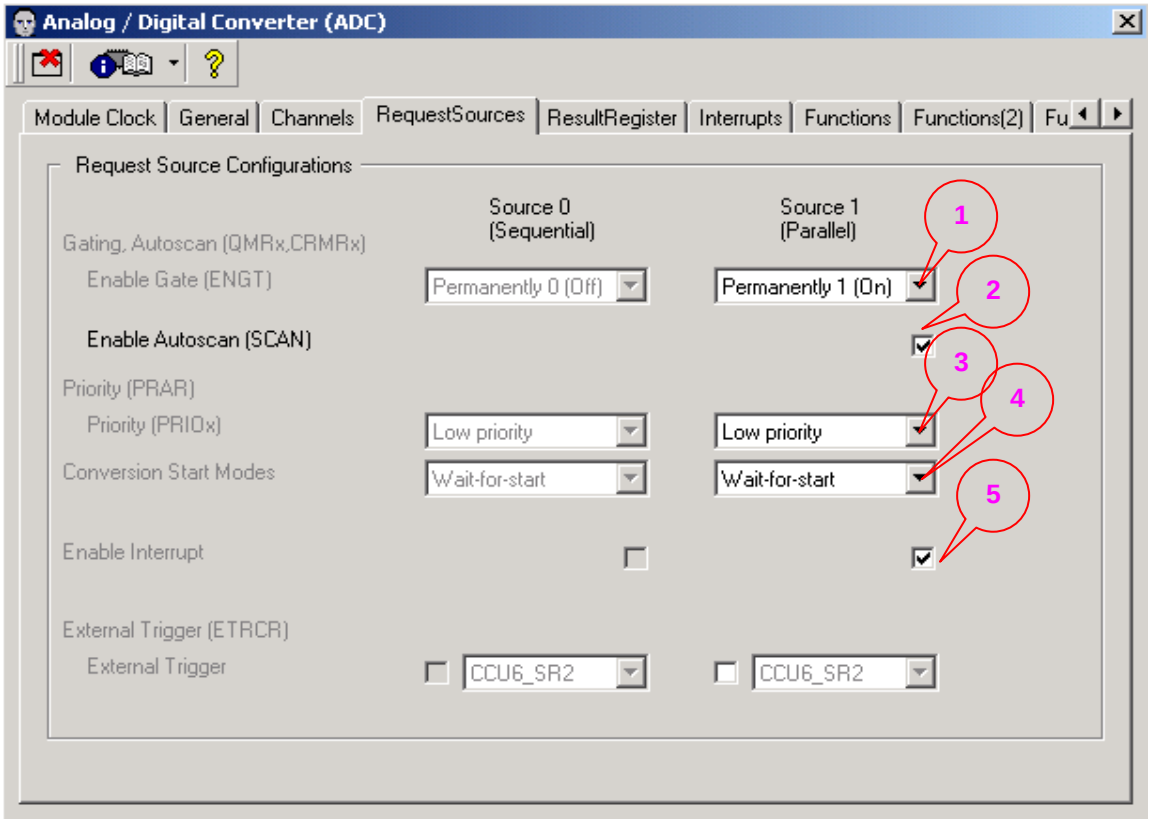


Figure 22 ADC Request Sources Settings Page in Advance Mode

4.3.8 Configuration for the Result Register Page of the ADC in Advance Mode

1. No Filter is selected
2. Wait-For-Read mode is selected
3. Valid-Flag-Reset is selected

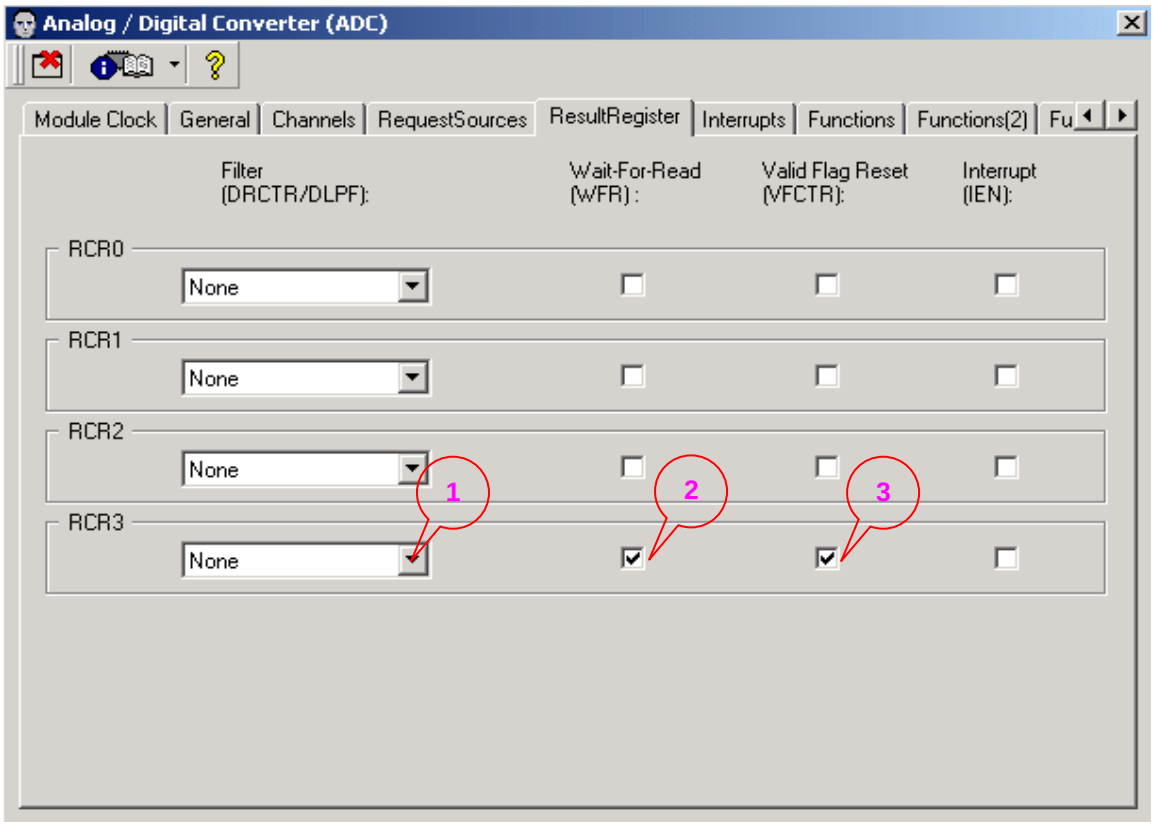
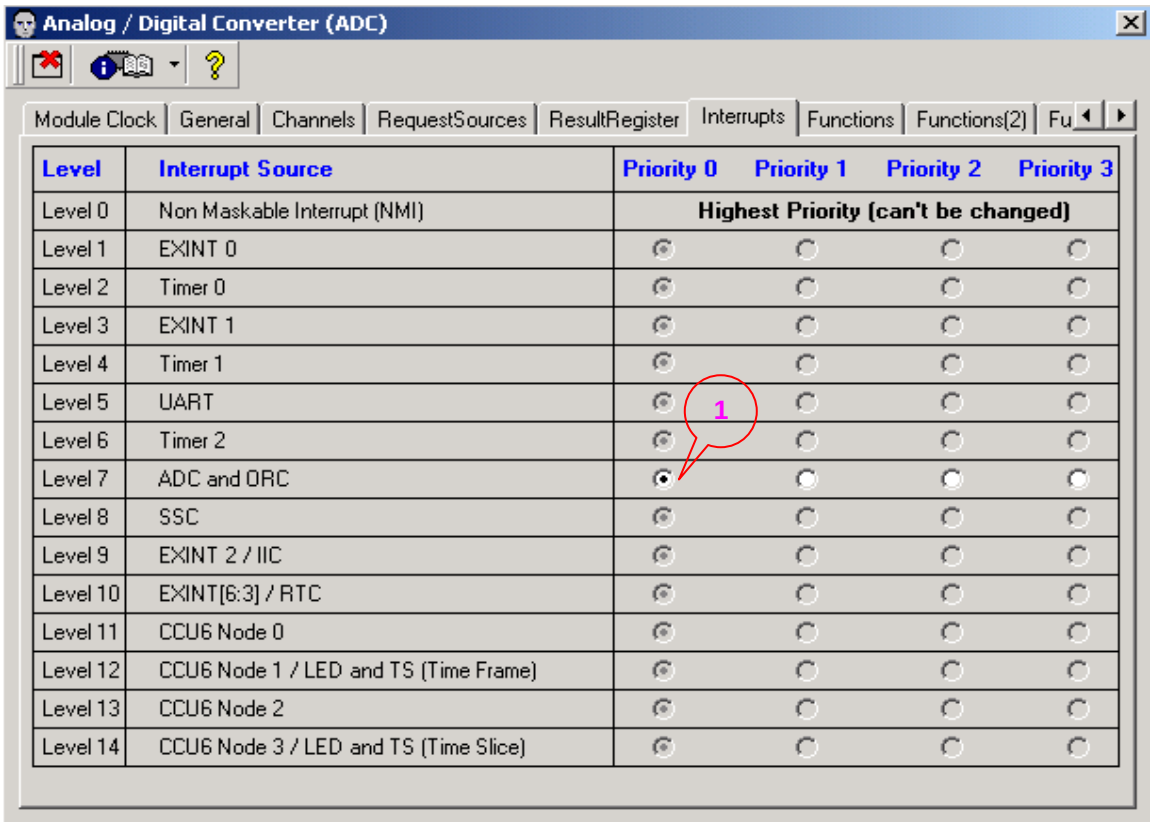


Figure 23 ADC Request Sources Settings Page in Advance Mode

4.3.9 Configuration for the Interrupt Page of the ADC in Advance Mode

1. ADC Parallel Source Interrupt is selected



Level	Interrupt Source	Priority 0	Priority 1	Priority 2	Priority 3
Level 0	Non Maskable Interrupt (NMI)	Highest Priority (can't be changed)			
Level 1	EXINT 0	○	○	○	○
Level 2	Timer 0	○	○	○	○
Level 3	EXINT 1	○	○	○	○
Level 4	Timer 1	○	○	○	○
Level 5	UART	○	○	○	○
Level 6	Timer 2	○	○	○	○
Level 7	ADC and ORC	●	○	○	○
Level 8	SSC	○	○	○	○
Level 9	EXINT 2 / IIC	○	○	○	○
Level 10	EXINT[6:3] / RTC	○	○	○	○
Level 11	CCU6 Node 0	○	○	○	○
Level 12	CCU6 Node 1 / LED and TS (Time Frame)	○	○	○	○
Level 13	CCU6 Node 2	○	○	○	○
Level 14	CCU6 Node 3 / LED and TS (Time Slice)	○	○	○	○

Figure 24 ADC Interrupt Page in Advance Mode

4.3.10 Configuration for the Functions Page of the ADC in Advance Mode

The following functions are selected

1. ADC_vInit
2. ADC_viIsr
3. ADC_vSetLoadEvent
4. ADC_vExtStartParReqChNum
5. ADC_ubBusy
6. ADC_uwGetResultData2
7. ADC_uwGetResultData3
8. ADC_ubGetResultChNum2
9. ADC_ubGetResultChNum3

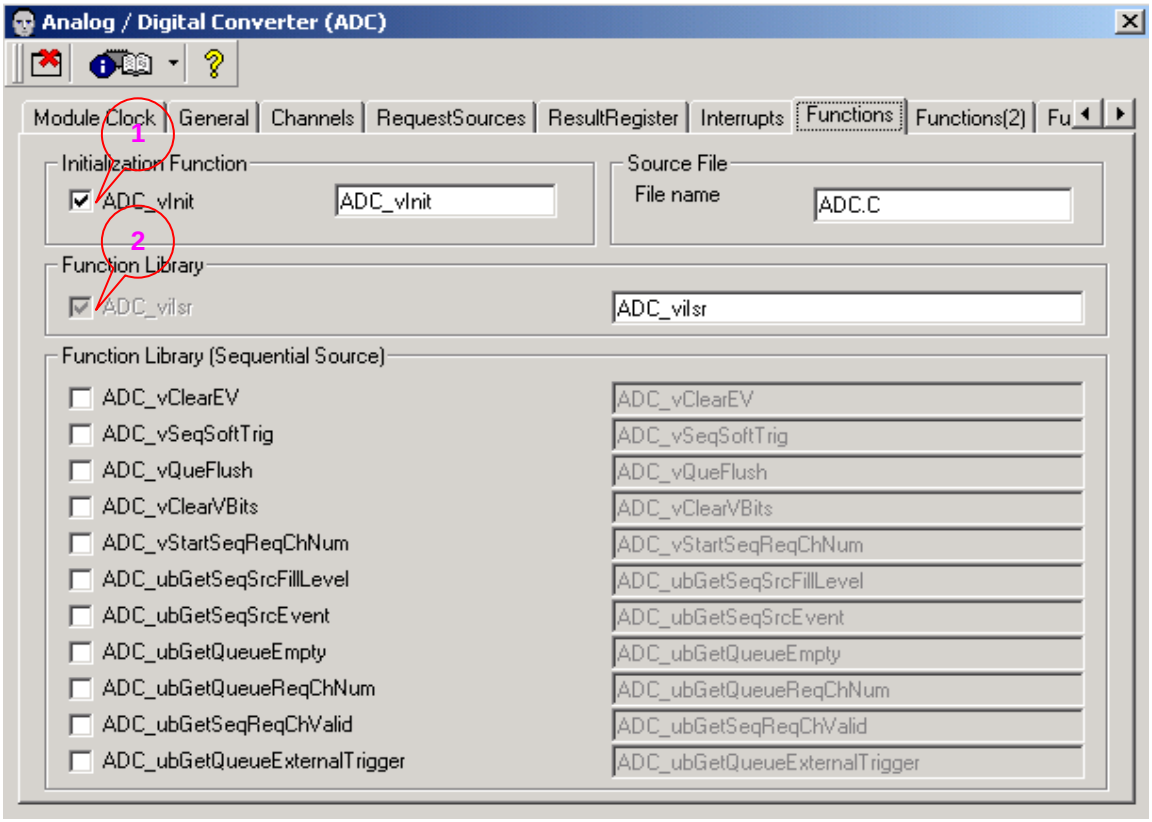


Figure 25 ADC Functions Page in Advance Mode

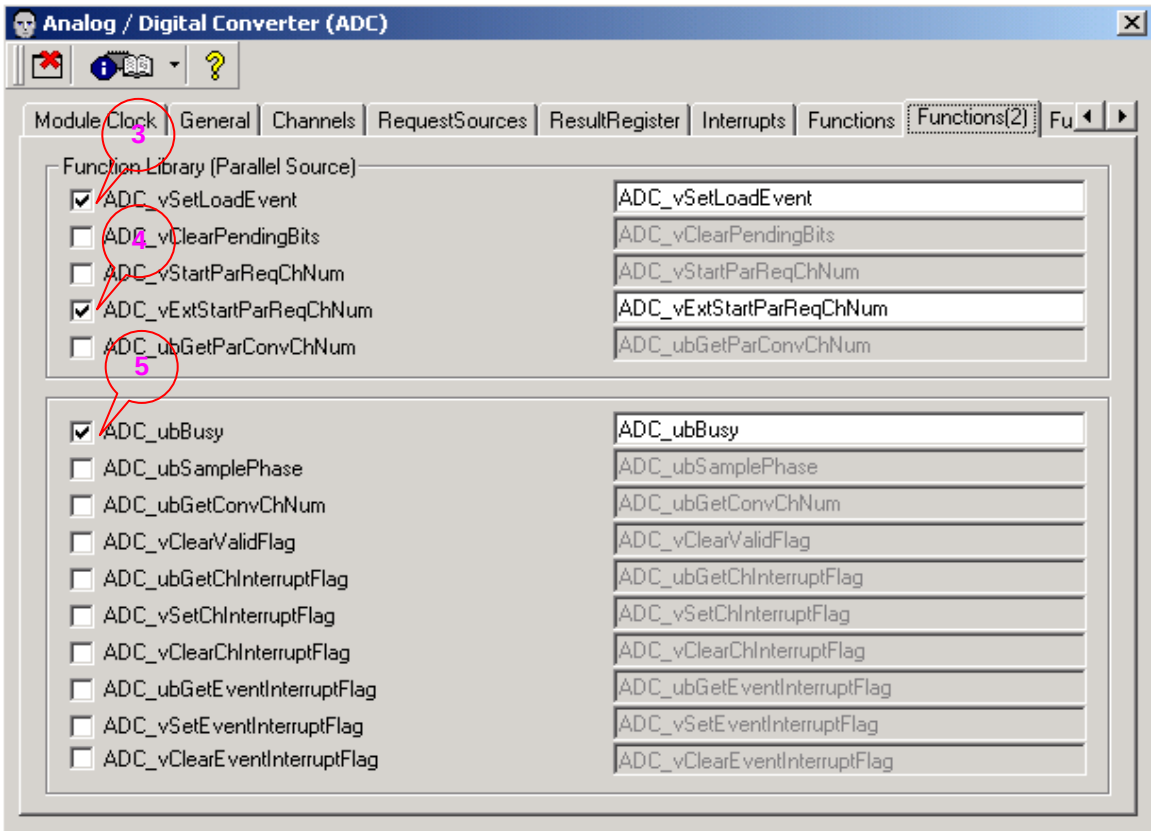


Figure 26 ADC Functions(2) Page in Advance Mode

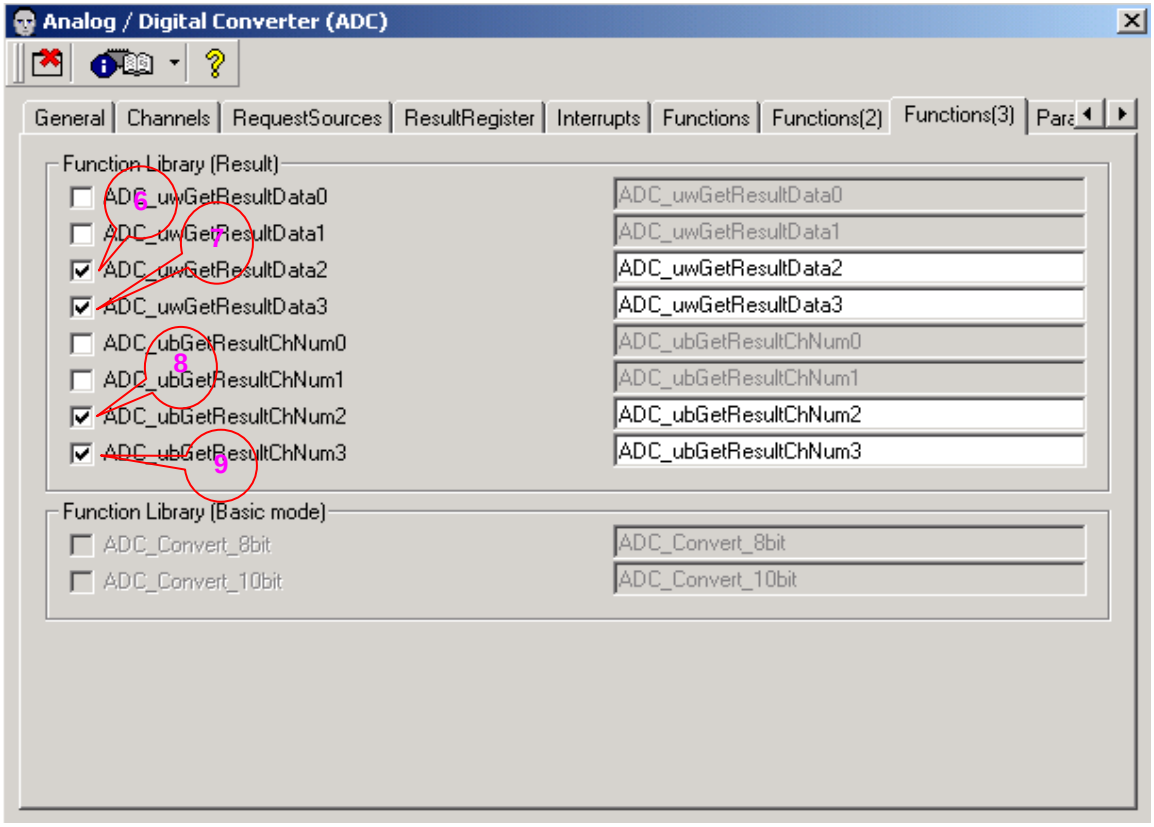


Figure 27 ADC Functions(3) Page in Advance Mode

4.4 Example Code for ADC Advance Mode:

The following C-code shows the handling of Advance mode for ADC module.

The code is for a XC822M device with Keil v8.18 compiler.

Tools Used:

DAvE v2.1r22
XC82xM_Series_v0_2.dip
XC822 Easy Kit Board
XC800_FLOAD_v50d12
Keil Compiler v8.18

The following code is the main function in which the ADC conversion sequence is done and the result is read using Advance Mode functions. This main function is available in the MAIN.C file of DAvE generated code.

```
void main(void)
{
  // USER CODE BEGIN (MAIN_Main,2)
        uword uwResult;
        ubyte ubChannel;
  // USER CODE END

  MAIN_vInit();

  // USER CODE BEGIN (MAIN_Main,3)
     TI = 1;
     Printf("-------------------------------------------------\n");
     printf("\r\n");
     printf("\r\n");
     printf("-------------------------------------------------\r\n");
     printf("Microcontroller  :Infineon XC82X \r\n");
     printf("Module           :ADC \r\n");
     printf("Example          :ADC_AdvanceMode_Example\r\n");
     printf("-------------------------------------------------\r\n");
     printf("\r\n");
     printf("Demonstration of ADC Advance Mode..\r\n");

     // Start Parallel conversion for channels 2 and 3.
      ADC_vExtStartParReqChNum(0x0C);
     // Software trigger for parallel conversion.
      ADC_vSetLoadEvent();
     // Wait until conversion is complete.
      while(ADC_ubBusy());
     // Wait until parallel source interrupt occur.
       while(!ubParSrcIntFlg);

     // Get converted channel number from result register 2.
       ubChannel = ADC_ubGetResultChNum2();
       printf("\r\ nConverted channel number  = 0x%x\r\n",(int)ubChannel);

    // Get converted result from result register 2.
       uwResult = ADC_uwGetResultData2();
       printf("\r\nConverted 8 bit result for channel 2 = 0x%x\r\n",uwResult);

   // Get converted channel number from result register 3.
      ubChannel = ADC_ubGetResultChNum3();
```

```
      printf("\r\ nConverted channel number  = 0x%x\r\n",(int)ubChannel);

  // Get converted result from result register 3.
     uwResult = ADC_uwGetResultData3();
     printf("\r\nConverted 8 bit result for channel 3 = 0x%x\r\n",uwResult);

// USER CODE END
   while(1)
   {

       // USER CODE BEGIN (MAIN_Main,4)


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

The following code shows the ADC Advance mode configuration. This code is available in the ADC.C file.

```
void ADC_vInit(void)
{
  // USER CODE BEGIN (ADC_Init,2)

  // USER CODE END

  ///  -----------------------------------------------------------------------
  ///  Configuration of Global Control:
  ///  -----------------------------------------------------------------------
  ///  - the ADC module clock is enabled
  ///  - the ADC module clock = 48.00 MHz
  ///

  ///  - the ADC Advance mode is selected
  ///

  ///  - the result is 8 bits wide
  ///  --- Conversion Timing -----------------
  ///  - conversion time (CTC)    = 00.94 us

  ///  - Configure global control functions

  SFR_PAGE(_ad0, noSST);         // switch to page 0

  ADC_GLOBCTR   =  0x50;         // load global control register

  ///  -----------------------------------------------------------------------
  ///  Configuration of Priority and Arbitration:
  ///  -----------------------------------------------------------------------
  ///  - 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 permanent arbitration mode is selected
  ///  - Arbitration Slot 0 is disabled
  ///  - Arbitration Slot 1 is enabled

  ADC_PRAR      =  0x80;         // load Priority and Arbitration register

  SFR_PAGE(_ad1, noSST);     // switch to page 1

  ///  -----------------------------------------------------------------------
  ///  Configuration of Channel Control Registers:
  ///  -----------------------------------------------------------------------
  ///  Configuration of Channel 2
  ///  - the result register2 is selected
  ///  - the limit check 0 is selected

  ADC_CHCTR2    =  0x02;       // load channel control register
```

```
/// Configuration of Channel 3
/// - the result register3 is selected
/// - the limit check 0 is selected

ADC_CHCTR3   = 0x03;      // load channel control register

SFR_PAGE(_ad0, noSST);    // switch to page 0

/// ---------------------------------------------------------------------
/// Configuration of Sample Time Control:
/// ---------------------------------------------------------------------

ADC_INPCR0   = 0x00;      // load input class register

/// ---------------------------------------------------------------------
/// Configuration of Out of range comparator:
/// ---------------------------------------------------------------------

ADC_ENORC    = 0x00;      // load out of range comparator register

SFR_PAGE(_ad4, noSST);    // switch to page 4

/// ---------------------------------------------------------------------
/// Configuration of Out of range comparator edge trigger:
/// ---------------------------------------------------------------------

ADC_CNF      = 0x00;      // load out of range comparator trigger edge
                          // select register


/// ---------------------------------------------------------------------
/// Configuration of alias register:
/// ---------------------------------------------------------------------

ADC_ALR0     = 0x00;      // load alias register 0

/// ---------------------------------------------------------------------
/// Configuration of Result Control Registers:
/// ---------------------------------------------------------------------
/// Configuration of Result Control Register 0
/// - the data reduction filter is disabled
/// - the digital low pass filter is disabled
/// - the event interrupt is disabled
/// - the wait-for-read mode is disabled
/// - the VF unchanged by read access to RESRxH

ADC_RCR0     = 0x00;      // load result control register 0

/// Configuration of Result Control Register 1
/// - the data reduction filter is disabled
/// - the digital low pass filter is disabled
/// - the event interrupt is disabled
/// - the wait-for-read mode is disabled
/// - the VF unchanged by read access to RESRxH

ADC_RCR1     = 0x00;      // load result control register 1

/// Configuration of Result Control Register 2
/// - the data reduction filter is disabled
```

```
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled
///  - the VF unchanged by read access to RESRxH

ADC_RCR2     =  0x00;        // load result control register 2

///  Configuration of Result Control Register 3
///  - the data reduction filter is disabled
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is enabled
///  - the VF reset by read access to RESRxH

ADC_RCR3     =  0xC0;        // load result control register 3


///  ---------------------------------------------------------------------
///  Channel Interrupt Node Pointer:
///  ---------------------------------------------------------------------
///  - the SR 0 line become activated if any channel  interrupt is generated

///  ---------------------------------------------------------------------
///  Out of range comparator Interrupt Node Pointer:
///  ---------------------------------------------------------------------
///  - the SR 1 line become activated if out of range comparator  interrupt
///  for any channel is generated

///  ---------------------------------------------------------------------
///  Event Interrupt Node Pointer:
///  ---------------------------------------------------------------------
///  - the SR 0 line become activated if the event 0-1  interrupt is
///  generated
///  - the SR 0 line become activated if the event 4-7  interrupt is
///  generated

SFR_PAGE(_ad0, noSST);   // switch to page 0

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

ADC_LCBR0    =  0x70;       // load limit check boundary register 0
ADC_LCBR1    =  0xB0;       // load limit check boundary register 1

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

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

ADC_QMR0     =  0x00;       // load queue mode register

///  ---------------------------------------------------------------------
///  Configuration of Conversion Request Mode Registers:
///  ---------------------------------------------------------------------
///  - the gating line is permanently 1
```

```
///  - the external trigger is disabled
///  - the source interrupt is enabled
///  - the autoscan functionality is enabled

ADC_CRMR1     =  0x19;     // load conversion request mode register 1

SFR_PAGE(_ad0, noSST);  // switch to page 0

ADC_GLOBCTR  |=  0x80;   // turn on Analog part

///  - Out of range comparator -Interrupt (ORCIEN) remains disabled

///  - Channel limit checking -Interrupt (CLCIEN) remains disabled

///  - ADC -Interrupt (EADC) remains enabled

IEN1   |= 0x01;                  //  Enable ADC-interrupt (EADC)


// USER CODE BEGIN (ADC_Init,3)

// USER CODE END

} //  End of function ADC_vInit
```

This is one example of how the Advance mode of the ADC module can be used based on the application requirements.

5 Conclusion, Related Documents and Links

The Basic and Advance mode handling of the XC82x\XC83x Microcontroller ADC is supported by DAvE for easy use of the module. In Basic mode, the DAvE user doesn't need to bother with the configurations as DAvE configures basic settings automatically. In Advance mode, DAvE users can configure different settings as per the application requirements to make full use of the ADC capabilities.

RELATED DOCUMENTS AND LINKS:

User's Manual v1.0, February 2010 XC82x_um_v1.0.pdf

XC82x/XC83x Product Information:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304323b87bc20123dd1efe2f7011

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304323b87bc20123dd13eca1700f

Starter Kit (Board Manual), XC82x/XC83x Development Tools and Software:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304323b87bc20123dd1efe2f7011&tab=2

DAvE for the Infineon XC82x/XC83x microcontroller Family:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304319c6f18c0119ecd0ed63538e&parentChannelRef=db3a3043243b5f1701245d90b0944284_db3a3043243b5f1701245d90b0944284_db3a3043243b5f1701245d90b0944284

Programming the Analog-to-Digital Converter on XC800 family of Microcontrollers:
ap0806310_Programming_ADC_XC800_Family_Microcontrollers.pdf

http://www.infineon.com/dgdl/AP0806310_Programming_ADC_XC800_Family_Microcontrollers.pdf?folderId=db3a30431375fb1a01138c57204603bd&fileId=db3a30431be39b97011c1d1fefeb77af

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG