ADC_MEASUREMENT

Home

Apps

Here is a list of all modules:

- License Terms and Copyright Information
- Abbreviations and Definitions
- Overview
- Architecture Description
- APP Configuration Parameters
- Enumerations
- Data structures
- Methods
- Usage
- Release History

ADC_MEASUREMENT

License Terms and Copyright Information

License Terms and Copyright Information

(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

To improve the quality of the software, users are encouraged to share modifications, enhancements or bug fixes with Infineon Technologies AG (dave@infineon.com).

ADC_MEASUREMENT

Abbreviations and Definitions

Abbreviations and Definitions

Abbreviations:	
DAVE™	Digital Application Virtual Engineer
APP	DAVE™ Application
API	Application Programming Interface
GUI	Graphical User Interface
MCU	Microcontroller Unit
SW	Software
HW	Hardware
LLD	Low Level Driver
IO	Input Output
ADC	Analog to Digital Conversion
VADC	Versatile Analog to Digital Converter

Definitions:	
Singleton	Only single instance of the APP is permitted
Sharable	Resource sharing with other APPs is permitted
initProvider	Provides the initialization routine
Physical connectivity	Hardware inter/intra peripheral (constant) signal connection
Conditional connectivity	Constrained hardware inter/intra peripheral signal connection
Aggregation	Indicates consumption of low level (dependent) DAVE APPs

ADC_MEASUREMENT

Home

Overview

Overview

The APP measures the connected multiple input signals. It uses the "background request source" of Versatile Analog to Digital Converter (VADC) peripheral to provide the following functionalities.

1. Measures a linear sequence of analog inputs.
2. Provides software and hardware controlled start of measurements.
3. Provides the following interrupt notifications
    - Notification at the end of all measurements (Not applicable for XMC1000).
    - Notification after each individual measurement completion.

Details of provided functionalities

1. **ADC Measurement Sequence:**

   The measurement names configured in the APP GUI "Measurement Table" are assigned to individual channels under background scan request source of VADC peripheral. The assigned channels can be from any groups defined in the VADC hardware. The assigned channels depend on the analog pin selected. All these assigned channels get converted in a fixed linear sequence starting from highest group channel to lowest group channel (G3_CH7 to G0_CH0). This is explained with an example as follows:

   If 6 measurements are configured in the GUI named as CH_A, CH_B, CH_C, CH_D, CH_E, CH_F. The channels assigned to the measurements is as follows

CH_A = GROUP-1 CHANNEL-2
CH_B = GROUP-0 CHANNEL-5
CH_C = GROUP-2 CHANNEL-6
CH_D = GROUP-2 CHANNEL-0
CH_E = GROUP-1 CHANNEL-4
CH_F = GROUP-2 CHANNEL-1

The order of conversion sequence for these measurements is as shown in the following figure

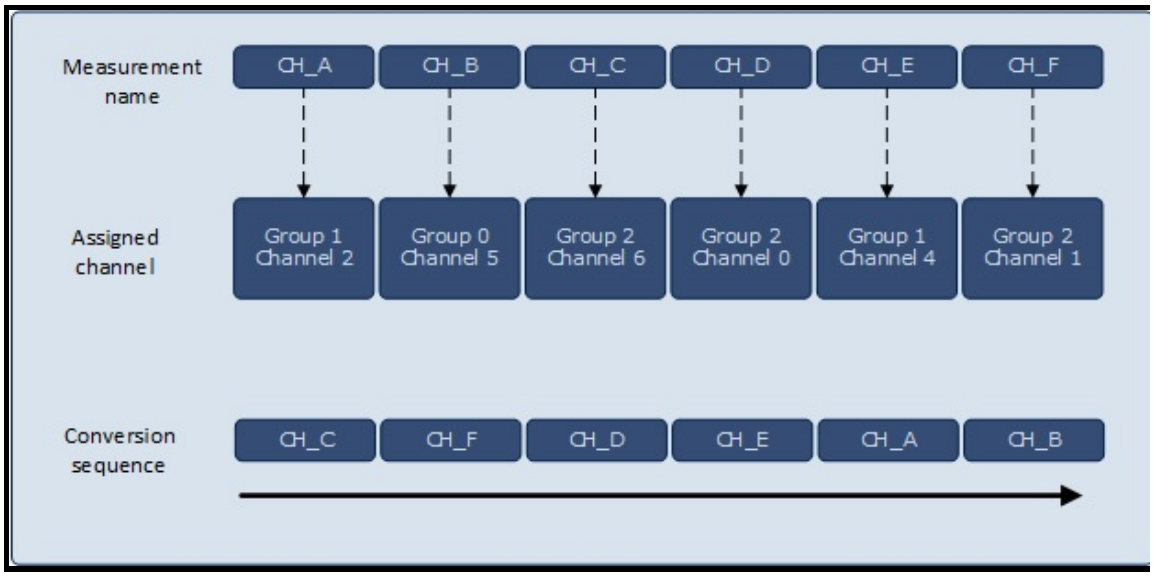


Figure 1 : Conversion Sequence

2. **ADC Measurement start mode:**
   To start the ADC measurements, the APP provides the following four different modes.

   - **Hardware Trigger Single Shot Mode**
   - **Hardware Trigger Continuous mode**
   - **Software Start Single Shot Mode**
   - **Software Start Continuous Mode**

   Software Start mode need **ADC_MEASUREMENT_StartConversion()** API to start the conversions of assigned channels. The software start can also be triggered in the APP initialization by setting the " Start conversion

after initialization" parameter in the General Settings TAB.

Hardware Start mode needs external trigger signal to start the conversions of the assigned channels. This mode can be enabled by setting "Trigger Edge Selection" field in the General Settings TAB to any one edge (Rising/ Falling / Both edges). The trigger signal can be applied from other APPS such as PWM / EVENT_GENERATOR.

Both software and hardware start modes can configure the ADC to measure the inputs repeatedly by enabling the "Enable continuous conversion" option in the GUI. If single shot mode is selected, subsequent conversion (sequence) must be triggered again.

The details of four start modes are explained in the following figure

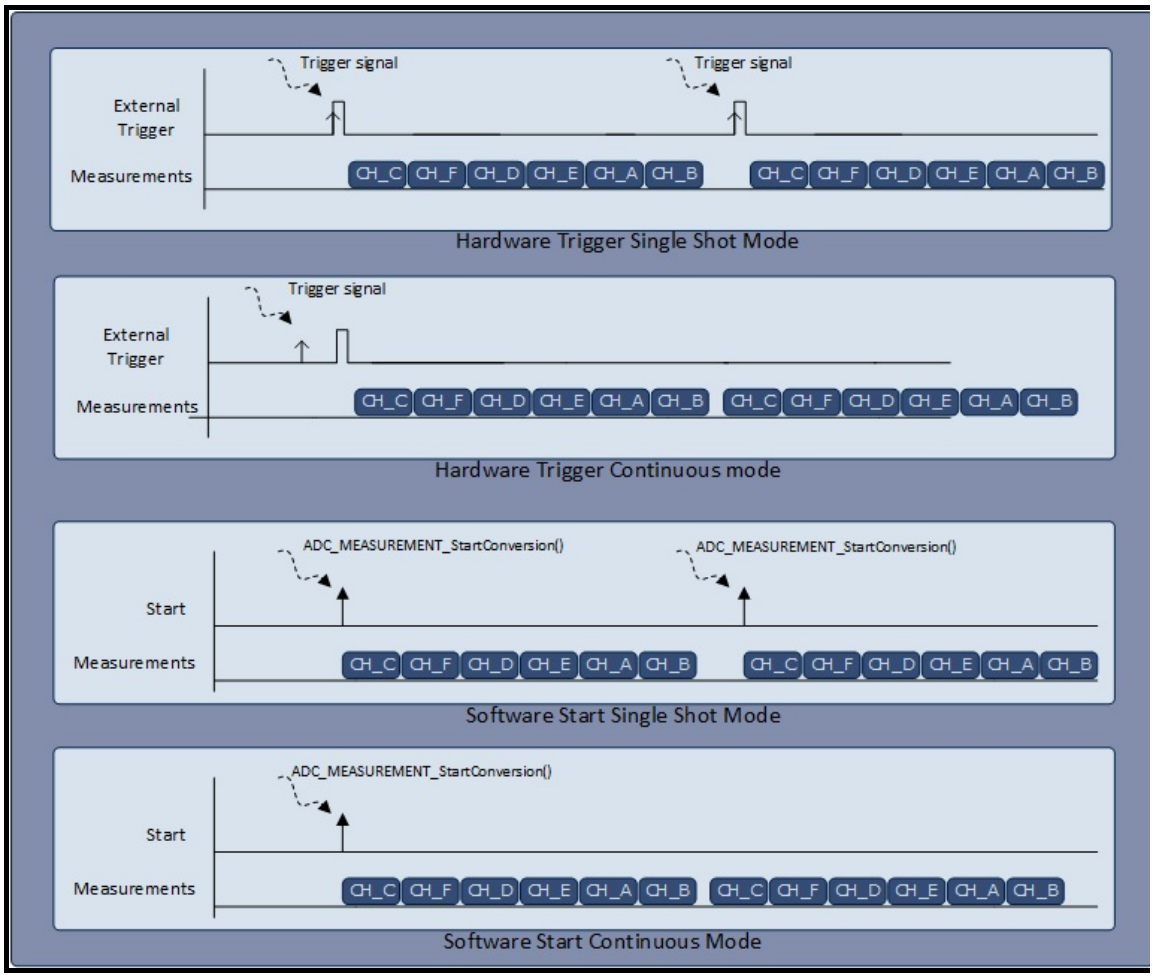


Figure 2 : ADC Measurement start mode

3. **Interrupt Notifications:**

    - XMC1100 Series
      If enabled, the interrupt is generated after each measurement is completed(one channel conversion). To enable the interrupt notification, select GUI field "Enable interrupt after each measurement" under "Interrupt Settings" TAB. It is mandatory to read the converted results in the interrupt notification function callback to allow the next conversions to happen. All the measurement channels share a common GLOBAL result register in "wait for read mode" of operation.

    - Other XMC Series
      If enabled, an interrupt is raised after all the measurements are completed once(all channels are converted once). To enable the interrupt notification, select GUI field "Enable end of measurement interrupt" under "Interrupt Settings" TAB. Additionally on each measurement completion (one result available) an result event notification can be triggered by enabling the "Result event" field associated with the particular measurement name.

    The detailed analysis of the interrupt notifications is explained in the following figure.

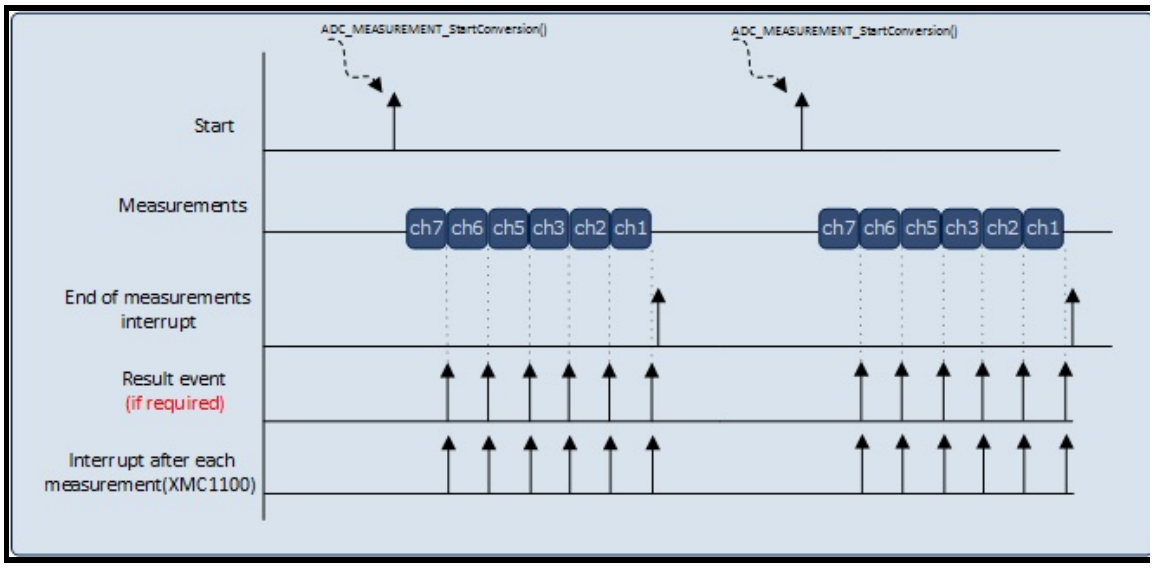


Figure 3 : Interrupt Notifications

APP Structure

Figure 4 , shows how the APP is structured in DAVE. XMC controllers provides the VADC module for analog to digital conversion. The XMC Lib layer provides abstraction for these hardware modules. The **ADC_MEASUREMENT** APP uses VADC and SCU LLDs and other dependent APPS such as GLOBAL_ADC ,ANALOG_IO and CLOCK_XMCx for the functional execution.

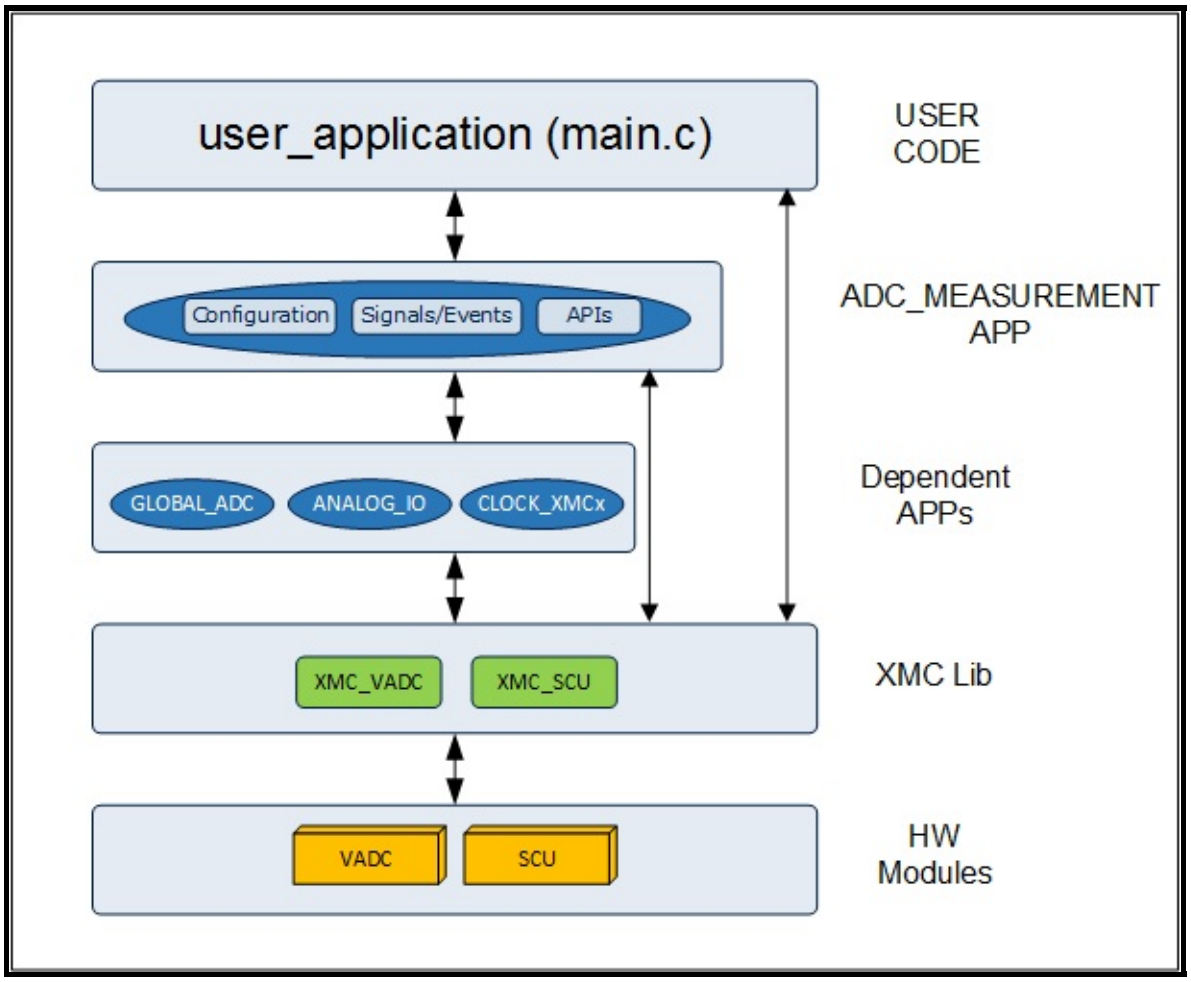


Figure 4 : Hardware and Software connectivity of **ADC_MEASUREMENT** APP

Limitations:

- The available channels depends on the device that is selected and it might not be possible to achieve the maximum value shown for the "Number of measurements" in the UI of the APP.

Supported Devices

The APP supports below devices:

1. XMC4800 / XMC4700 Series
2. XMC4500 Series
3. XMC4400 Series
4. XMC4300 Series

5. XMC4200 / XMC4100 Series
6. XMC1400 Series
7. XMC1300 Series
8. XMC1200 Series
9. XMC1100 Series

Reference

1. XMC4800 / XMC4700 Reference Manual
2. XMC4500 Reference Manual
3. XMC4400 Reference Manual
4. XMC4300 Reference Manual
5. XMC4200 / XMC4100 Reference Manual
6. XMC1400 Reference Manual
7. XMC1300 Reference Manual
8. XMC1200 Reference Manual
9. XMC1100 Reference Manual

ADC_MEASUREMENT

Home

Architecture Description

Architecture Description

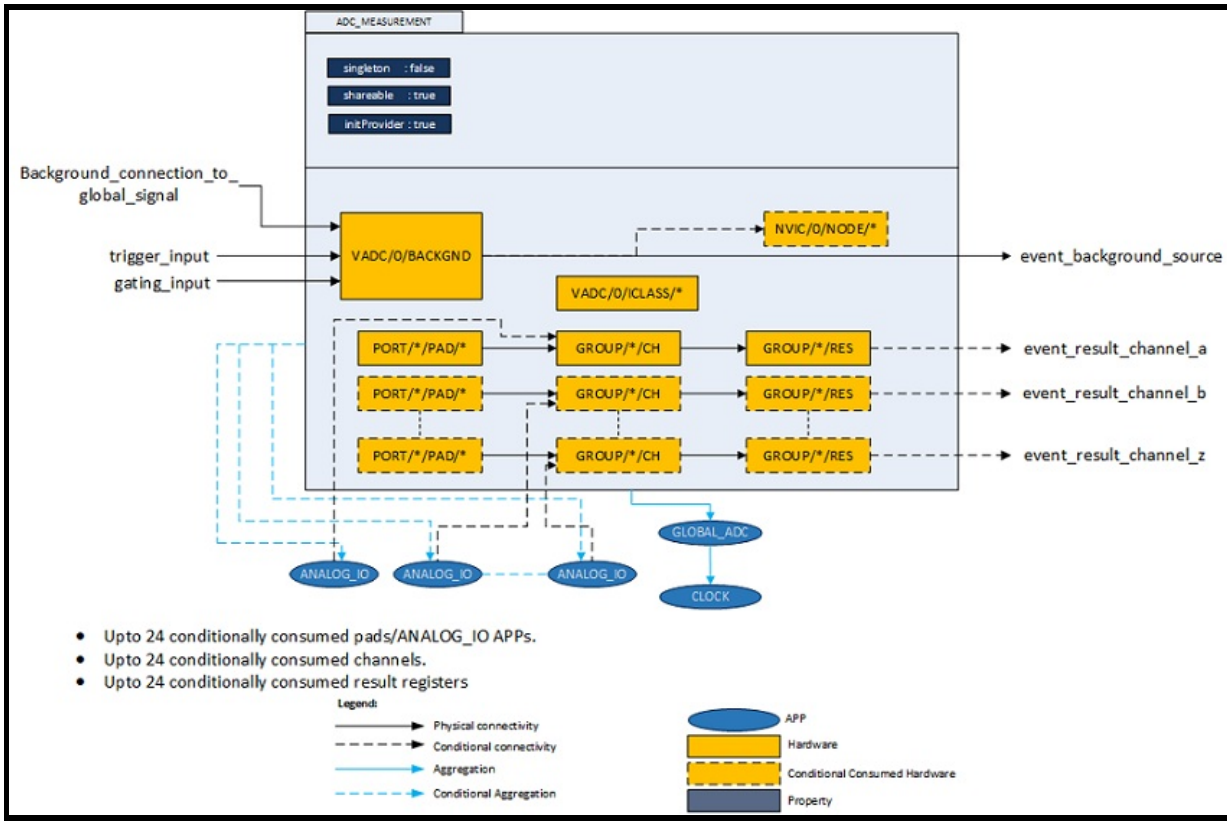


Figure 1 : Architecture of **ADC_MEASUREMENT** APP

The diagram above represents the internal software architecture of the **ADC_MEASUREMENT** APP. The figure shows the consumed hardware resources, dependent APPs and various signals which are exported out. A **ADC_MEASUREMENT** APP instance exists in a DAVE™ project with fixed attributes as shown and uses the VADC peripheral's background request source for converting a channel. This in addition requires the consumption of the GLOBAL_ADC and CLOCK APPS for functional configurations. The **ADC_MEASUREMENT** APP

also provides output signals, these are described in Table-1.

ANALOG_IO APP is conditionally used by **ADC_MEASUREMENT** APP when an "Expose pin"" is selected in the UI. This is applicable for all the channels. By using the ANALOG_IO, the **ADC_MEASUREMENT** can share the pin with other APPs such as DAC, ACMP_CONFIG etc. It is possible to connect the same ANALOG_IO APP to multiple channels. This involves the use of the ALIAS feature of the ADC channels. In this the same pin gets converted by multiple channels from the same group. For Example: Assume that GROUP-1 CH-2 is connected to P14.2 and also to GROUP-0 CH-2. The Alias feature of the ADC enables the Channels 0 and 1 to convert any pin of the group. Hence the same ANALOG_IO APP can be shared between GROUP-1 CH-2, GROUP-1 CH-0, GROUP-1 CH-1. Also the same is applicable for the other group also GROUP-0 CH-2, GROUP-0 CH-0, GROUP-0 CH-1. The following figure shows that resources consumed by the **ADC_MEASUREMENT** APP when the same ANALOG_IO is shared.

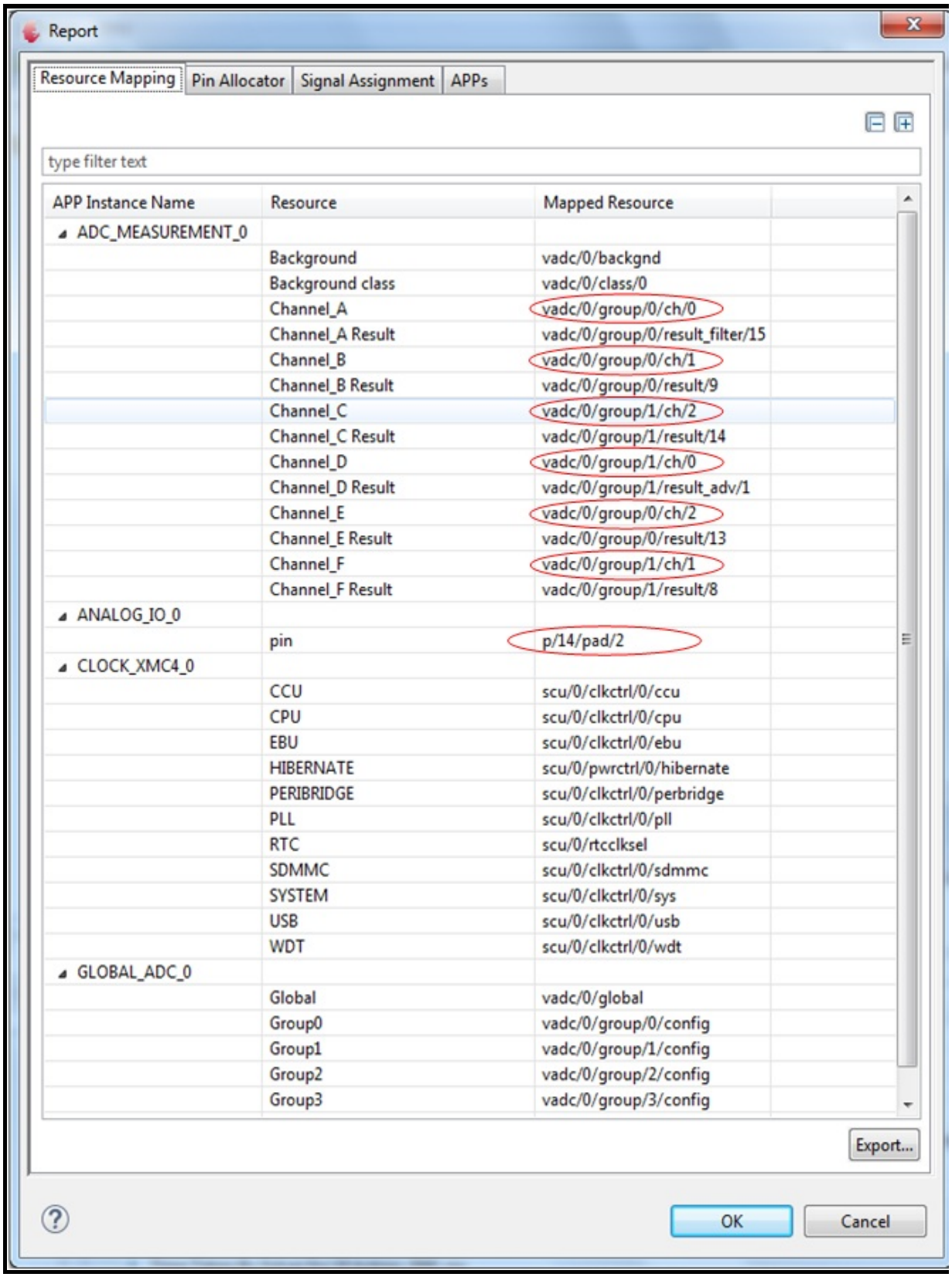


Figure 2 : Example for ALIAS and ANALOG_IO

Result Registers: Each channel is mapped to one result register(excluding XMC1100). There are 3 different categories of result registers based on different functions it provides.

- **result_adv**: Provide boundary flag outputs.
- **result_filter**: Provide filtered output.
- **result**: Provide accumulation(1x, 2x, 3x, 4x)/subtraction mode.

For the working of **ADC_MEASUREMENT** APP all the 3 results are perfectly the same. There will be no difference in the output result received after conversion.

Internally a NVIC node is consumed for background request source event (for XMC1100, global result event) when "Enable end of measurement interrupt" (for XMC1100, "Enable interrupt after each measurement") is enabled in the UI of the APP.

Signals:

The following table presents the signals provided by the APP for connection. It also gives the flexibility to configure and extend the connectivity to other APPs.
Table 1: APP IO signals

Signal Name	Input/Output	Availability	
event_background_source	Output	Always	E s It N th th e s s T b

			a s c
event_result_*channel_x* Where *channel_x* represents the channels that are being used.	Output	Conditional	F *c*
			T s

trigger_input	Input	Always	U tl tl s c v a tl C e
gating_input	Input	Always	G s U tl tl r b v a tl C
Background_connection_to_global_signal	Input	Always	G s F ir s n b b d c G A

Timing Calculations:

ADC_MEASUREMENT APP uses the following equations to calculate the sample time and Total conversion time.
Refer the reference manual for the detailed information.

1. **Sample Time:**

```
Actual Sample Time = (2 + STC) * tADCI
where,
STC   : Sample time control (Value - 0 to 256)
fADCI : Analog clock frequency
tADCI = 1/fADCI
```

2. **Total Conversion time: XMC4000 devices**

   Note: PC value is configured as 2 i.e with post calibration always enabled.
   If post calibration is disabled (in GLOBAL_ADC APP), the total conversion time will be reduced by 2/fADC (GLOBAL_ADC APP).

   1. **Standard Conversion Mode:**

```
Total Conversion time =  (2 + STC + N + DM + PC) * tADCI +
Where,
N       = 8, 10, 12 for n bit resolution.
tADC    = ADC module clock = system clock
tADCI   = Analog clock
STC     = Sample time control (Value - 0 to 256)
DM      = The selected duration of the MSB conversion (DM =
PC      = The post-calibration time PC, if selected (PC = 2
```

   2. **Fast Compare Mode:**

```
Total Conversion time = (2 + STC + 2) * tADCI + 2 * tADC
Where,
tADC    = ADC module clock = system clock
tADCI   = Analog clock
STC     = Sample time control (Value - 0 to 256)
```

3. **Total Conversion time: XMC1000 devices**

   1. **Standard Conversion Mode:**

```
Total Conversion time = ( 2 + STC) * tADCI + (4 * tSH) + (
```

Where,
N = 8, 10, 12 for n bit resolution.
tSH = Sample and Hold clock (Converter clock time peric
tADC = ADC module clock = system clock
tADCI = Analog clock
STC = Sample time control (Value - 0 to 256)

2. **Fast Compare Mode:**

Total Conversion time = (FCRT + 1) * 2 * tADCI + (2 + STC)
Where,
FCRT = Fast Compare Mode Response Time (Value - 0 to 15)
tSH = Sample and Hold clock (Converter clock time peric
tADCI = Analog clock
STC = Sample time control (Value - 0 to 256)
Note: FCRT value is configured as 0.

ADC_MEASUREMENT

Home

APP Configuration Parameters

App Configuration Parameters

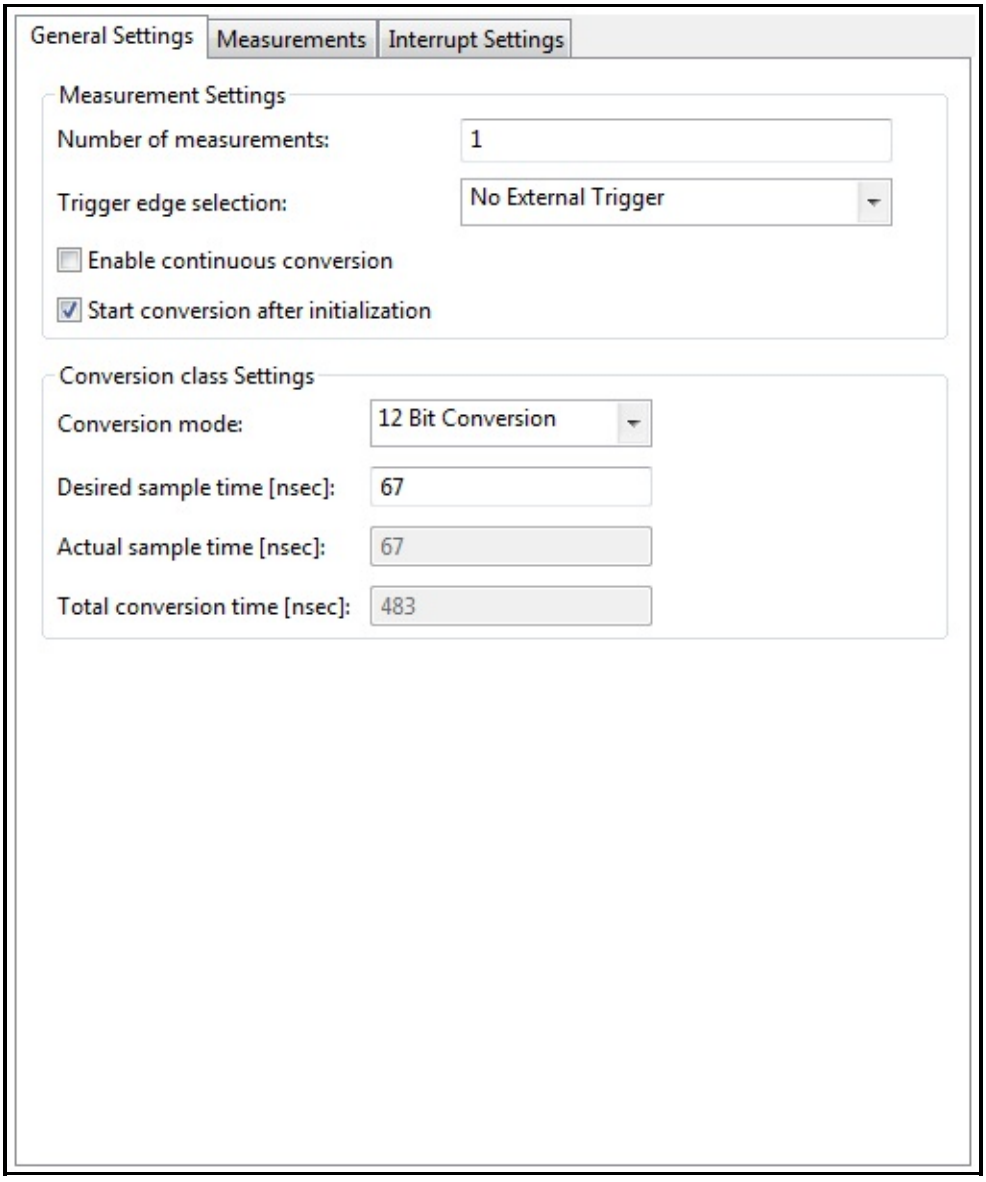


Figure 1: General Settings

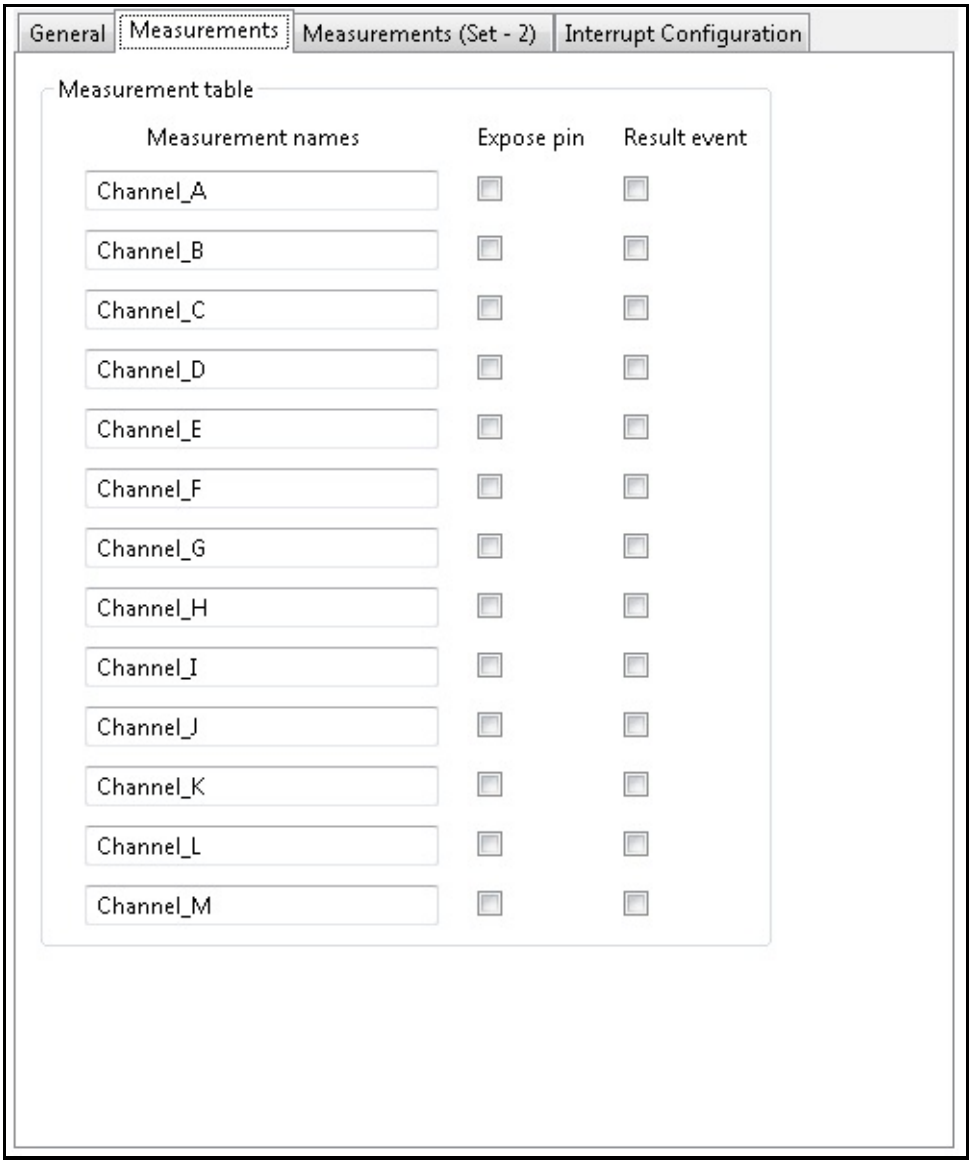


Figure 2: Measurements

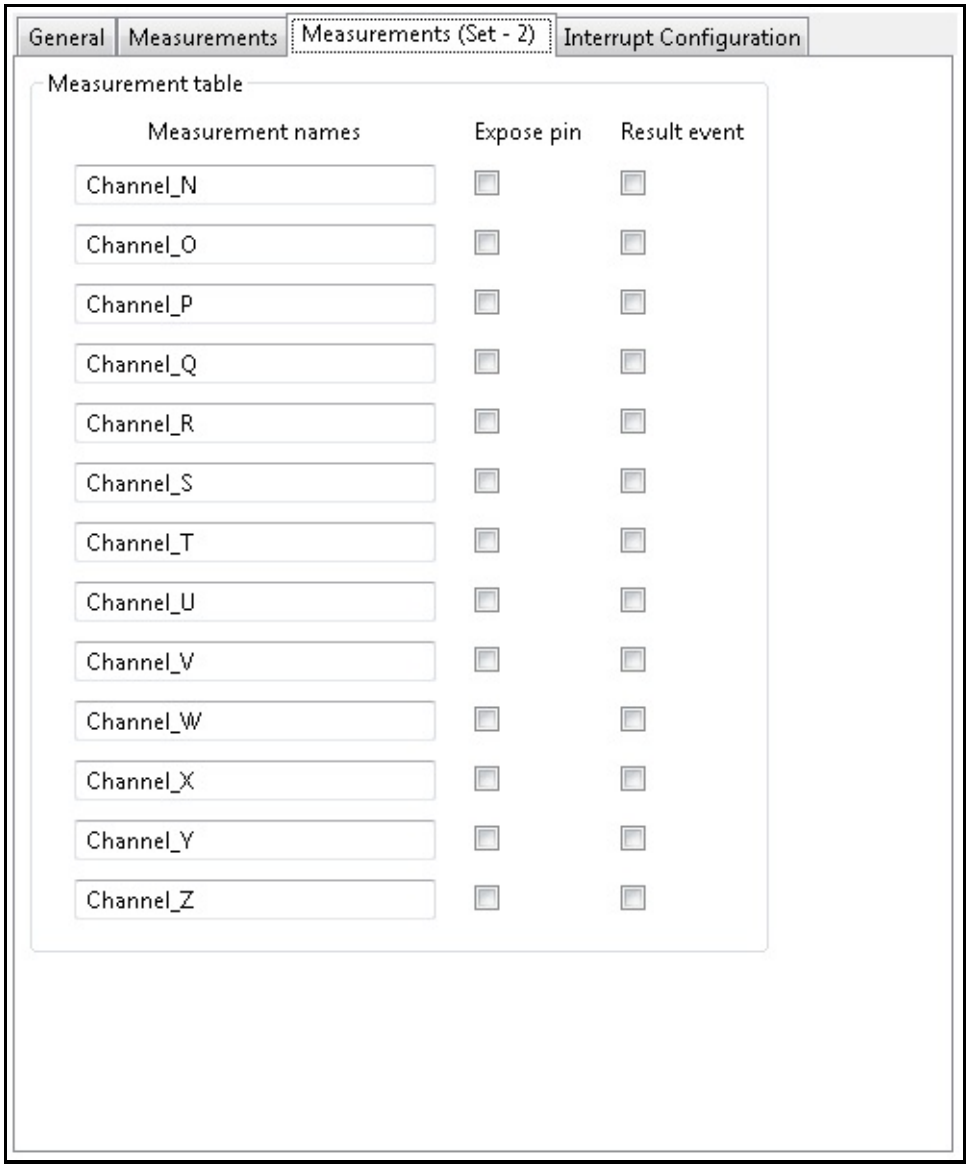


Figure 3: Measurements Cont'd

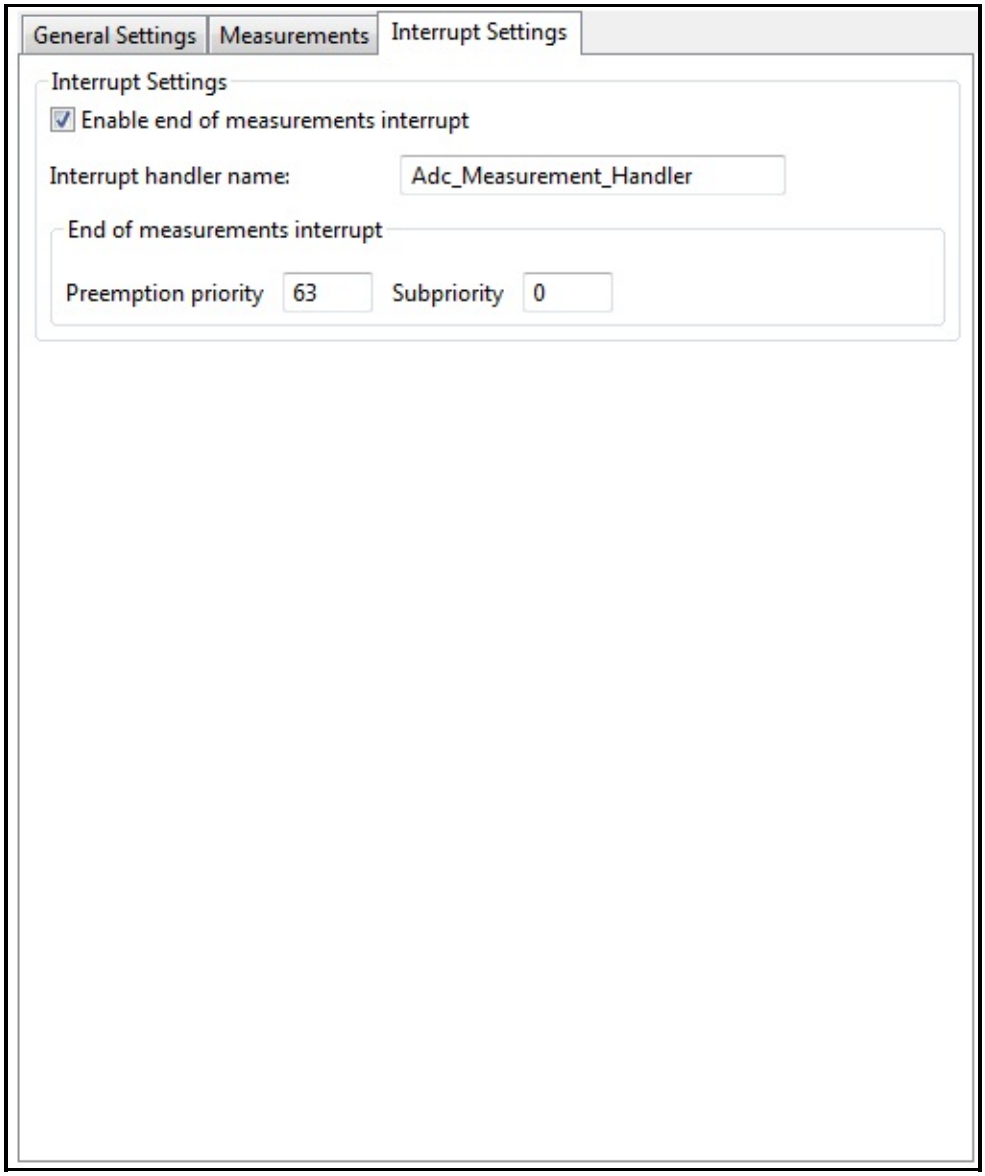


Figure 4: Interrupt Settings

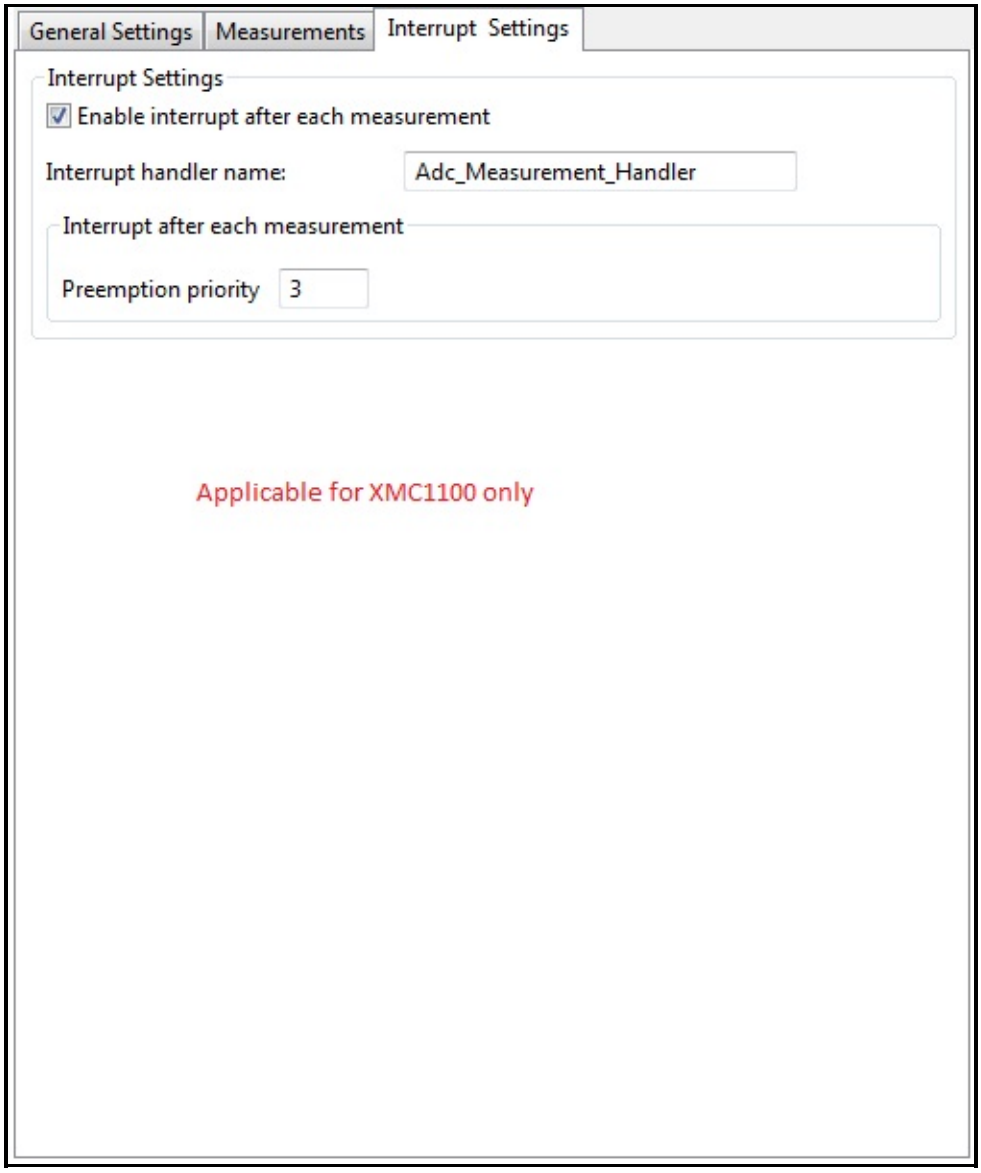


Figure 5: Interrupt Settings

ADC_MEASUREMENT

Home

Enumerations

enum	**ADC_MEASUREMEN** **ADC_MEASUREMEN** **ADC_MEASUREMEN** **ADC_MEASUREMEN** } Return value of an AP
typedef enum **ADC_MEASUREMENT_STATUS**	**ADC_MEASUREMEN** Return value of an AP

Enumeration Type Documentation

enum ADC_MEASUREMENT_STATUS

Return value of an API.

Enumerator:

ADC_MEASUREMENT_STATUS_SUCCESS	APP is Initialized
ADC_MEASUREMENT_STATUS_FAILURE	APP Initialization failed
ADC_MEASUREMENT_STATUS_UNINITIALIZED	APP has not been Initialized

Definition at line **127** of file **ADC_MEASUREMENT.h**.

ADC_MEASUREMENT

Home

Data structures

typedef void(*	**ADC_MEASUREMENT_** (void)
typedef struct **ADC_MEASUREMENT_ISR**	**ADC_MEASUREMENT_**
typedef struct **ADC_MEASUREMENT_CHANNEL**	**ADC_MEASUREMENT_**
typedef struct **ADC_MEASUREMENT_CHANNEL_ARRAY**	**ADC_MEASUREMENT_**
typedef struct **ADC_MEASUREMENT**	**ADC_MEASUREMENT_**

Typedef Documentation

typedef struct ADC_MEASUREMENT_CHANNEL_ARRAY ADC_ME

Structure to hold channels handles that are configured

typedef struct ADC_MEASUREMENT_CHANNEL ADC_MEASUREM

Structure to initialize ADC channels.

typedef struct ADC_MEASUREMENT_ISR ADC_MEASUREMENT_IS

Structure to initialize Request Source Interrupt's NVIC Node

typedef void(* ADC_MEASUREMENT_MUX_CONFIG_t)(void)

Function pointer to the mux configuration

Definition at line **145** of file **ADC_MEASUREMENT.h**.

typedef struct ADC_MEASUREMENT ADC_MEASUREMENT_t

Structure to configure **ADC_MEASUREMENT** APP.

ADC_MEASUREMENT

Home

Methods

DAVE_APP_VERSION_t	**ADC_MEASUREMENT_GetAppV** Get **ADC_MEASUREMENT** APP v
ADC_MEASUREMENT_STATUS_t	**ADC_MEASUREMENT_Init** (**ADC_MEASUREMENT_t** *const Initializes the APP to measure a se inputs.
void	**ADC_MEASUREMENT_StartCon** (**ADC_MEASUREMENT_t** *const Starts the conversion of the require measurements.
XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMENT_GetResu** (**ADC_MEASUREMENT_CHANN** handle_ptr) Returns the converted value for a s Not Applicable for XMC1100.
uint32_t	**ADC_MEASUREMENT_GetDetai** (**ADC_MEASUREMENT_CHANN** handle_ptr) Returns a detailed conversion resu Applicable for XMC1100.
XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMENT_GetResu** (**ADC_MEASUREMENT_t** *const ADC_MEASUREMENT_DEPREC Returns the converted value from t register. Only Applicable for XMC1

uint32_t	**ADC_MEASUREMENT_GetDetai** (**ADC_MEASUREMENT_t** *const ADC_MEASUREMENT_DEPREC Returns a detailed conversion resu Applicable for XMC1100.
__STATIC_INLINE XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMENT_GetGloba** Returns the converted value from t register. Only Applicable for XMC1
__STATIC_INLINE uint32_t	**ADC_MEASUREMENT_GetGloba** (void) Returns a detailed conversion resu Applicable for XMC1100.

Methods

Function Documentation

DAVE_APP_VERSION_t ADC_MEASUREMENT_GetAppVersion (vc

Get **ADC_MEASUREMENT** APP version.

Returns:

DAVE_APP_VERSION_t APP version information (major, minor and patch number)

Description:

The function can be used to check application software compatibility with a specific version of the APP.

Example Usage:

```
 #include <DAVE.h>

 int main(void) {
   DAVE_STATUS_t init_status;
   DAVE_APP_VERSION_t version;

   // Initialize ADC_MEASUREMENT APP:
   // ADC_MEASUREMENT_Init() is called from withi
n DAVE_Init().
   init_status = DAVE_Init();

   version = ADC_MEASUREMENT_GetAppVersion();
   if (version.major != 1U) {
     // Probably, not the right version.
   }

   // More code here
   while(1) {

   }
```

```
  return (0);
}
```

Definition at line 111 of file ADC_MEASUREMENT.c.

uint32_t ADC_MEASUREMENT_GetDetailedResult (ADC_MEASUR

Returns a detailed conversion result. Not Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the channel handle structure. (Use the channel handle related macros which are defined in adc_measurement_conf.h)

Returns:

uint32_t The complete result register.

Description:

Returns the 32 bit result register (GxRES[y]) completely. The result of conversion as well as other informations are returned from this API. The detailed result register contains result of the most recent conversion, the channel number requested the conversion, valid flag, converted request source and fast compare result. In polling mechanism the converted result can be read out after checking the valid flag bit. This API can be used in applications where, the channel number associated to the result register is also needed for verification.

Note:

This API is not Applicable for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Use **ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREME *const handle_ptr)** for XMC1100 microcontrollers.

```
#include <DAVE.h>
```

```
  typedef struct detailed_result_struct
  {
        uint8_t channel_num;
        uint8_t group_num;
        uint16_t conversion_result;
  }detailed_result_struct_t;

  uint32_t result;
  bool valid_result;
  detailed_result_struct_t detailed_result;

  void Adc_Measurement_Handler()
  {
        uint32_t result;
    valid_result = (bool)false;
  #if(UC_SERIES != XMC11)
    result = ADC_MEASUREMENT_GetDetailedResult(&A
DC_MEASUREMENT_Channel_A);
    if((bool)(result >> VADC_G_RES_VF_Pos))
    {
       valid_result = (bool)true;
       detailed_result.channel_num = (result & VA
DC_G_RES_CHNR_Msk) >> VADC_G_RES_CHNR_Pos;
       detailed_result.group_num = ADC_MEASUREMEN
T_Channel_A.group_index;
       detailed_result.conversion_result = result
 & VADC_G_RES_RESULT_Msk;
    }
  #endif
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
```

```
    while(1);
    return 0;
}
```

Definition at line **250** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_CHANNEL::ch_handle**, and **ADC_MEASUREMENT_CHANNEL::group_handle**.

uint32_t ADC_MEASUREMENT_GetDetailedResult (ADC_MEASUR

Returns a detailed conversion result. Only Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the APP handle structure.

Returns:

uint32_t The complete Result register.

Description:

Returns the 32 bit result register (GLOBRES) completely. The result of conversion as well as other informations are returned from this API. The detailed result register contains result of the most recent conversion, the channel number requested the conversion, valid flag, converted request source and fast compare result. In polling mechanism the converted result can be read out after checking the valid flag bit. This API can be used in applications where, the channel number associated to the result register is also needed for verification. This API is only used in the microcotrollers where group result registers are not available. Hence for these microntroller devices, all the conversion results are stored in the global result register in shared mode. The wait for read mode hardware option will be enabled for the global result register to avoid overwriting of results. To get a new channel conversion result, it is mandatory to read the previous result using the same API or

ADC_MEASUREMENT_GetResult API.

Note:

- This API is applicable only for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Use **ADC_MEASUREMENT_GetDetailedResult(ADC_MEASURI *const handle_ptr)** for other microcontrollers.
- For either 10Bit or 8Bit ADC resolution the result value needs to be right shifted by either 2 or 4 bits respectively. The 10Bit or 8 bit results are left aligned in the result register, hence a shift operation is needed.

```
  #include <DAVE.h>

  typedef struct detailed_result_struct
  {
        uint8_t channel_num;
        uint8_t group_num;
        uint16_t conversion_result;
  }detailed_result_struct_t;

  uint32_t result;
  bool valid_result;
  detailed_result_struct_t detailed_result[10];

  void Adc_Measurement_Handler()
  {
        static uint8_t index;
        uint32_t result;
    valid_result = (bool)false;
  #if(UC_SERIES == XMC11)
    result = ADC_MEASUREMENT_GetDetailedResult(&A
DC_MEASUREMENT_0);
  #endif

    if((bool)(result >> VADC_GLOBRES_VF_Pos))
```

```
    {
       valid_result = (bool)true;
       detailed_result[index].channel_num = (resu
lt & VADC_GLOBRES_CHNR_Msk) >> VADC_GLOBRES_CHNR_
Pos;
       detailed_result[index].group_num = ADC_MEA
SUREMENT_Channel_A.group_index;
       detailed_result[index].conversion_result =
 (result & VADC_GLOBRES_RESULT_Msk) >>
                                ((uint32_t)ADC_MEASUR
EMENT_0.iclass_config_handle->conversion_mode_sta
ndard * (uint32_t)2);
    }
    index++;
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **283** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_GetGlobalDetailedResult()**.

__STATIC_INLINE uint32_t ADC_MEASUREMENT_GetGlobalDetail

Returns a detailed conversion result. Only Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the APP handle structure.

Returns:

uint32_t The complete Result register.

Description:

Returns the 32 bit result register (GLOBRES) completely. The result of conversion as well as other informations are returned from this API. The detailed result register contains result of the most recent conversion, the channel number requested the conversion, valid flag, converted request source and fast compare result. In polling mechanism the converted result can be read out after checking the valid flag bit. This API can be used in applications where, the channel number associated to the result register is also needed for verification. This API is only used in the microcotrollers where group result registers are not available. Hence for these microntroller devices, all the conversion results are stored in the global result register in shared mode. The wait for read mode hardware option will be enabled for the global result register to avoid overwriting of results. To get a new channel conversion result, it is mandatory to read the previous result using the same API or **ADC_MEASUREMENT_GetResult** API.

Note:

- This API is applicable only for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Use **ADC_MEASUREMENT_GetDetailedResult(ADC_MEASURI *const handle_ptr)** for other microcontrollers.
- For either 10Bit or 8Bit ADC resolution the result value needs to be right shifted by either 2 or 4 bits respectively. The 10Bit or 8 bit results are left aligned in the result register, hence a shift operation is needed.

```
#include <DAVE.h>

typedef struct detailed_result_struct
{
```

```
        uint8_t channel_num;
        uint8_t group_num;
        uint16_t conversion_result;
  }detailed_result_struct_t;

  uint32_t result;
  bool valid_result;
  detailed_result_struct_t detailed_result[10];

  void Adc_Measurement_Handler()
  {
        static uint8_t index;
        uint32_t result;
    valid_result = (bool)false;
  #if(UC_SERIES == XMC11)
    result = ADC_MEASUREMENT_GetGlobalDetailedRes
ult();
  #endif

    if((bool)(result >> VADC_GLOBRES_VF_Pos))
    {
       valid_result = (bool)true;
       detailed_result[index].channel_num = (resu
lt & VADC_GLOBRES_CHNR_Msk) >> VADC_GLOBRES_CHNR_
Pos;
       detailed_result[index].group_num = ADC_MEA
SUREMENT_Channel_A.group_index;
       detailed_result[index].conversion_result =
 (result & VADC_GLOBRES_RESULT_Msk) >>
                              ((uint32_t)ADC_MEASUR
EMENT_0.iclass_config_handle->conversion_mode_sta
ndard * (uint32_t)2);
    }
    index++;
  }

  int main(void)
```

```
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **695** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_GetDetailedResult()**.

__STATIC_INLINE XMC_VADC_RESULT_SIZE_t ADC_MEASUREME

Returns the converted value from the global result register. Only Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the APP handle structure.

Returns:

XMC_VADC_RESULT_SIZE_t conversion result.
Range: [0x0 to 0x3FF] if accumulation of results is switched off.

Description:

Reads the converted result stored in the common result register [GLOBRES], assigned to all the channels. This API is only used in the microcotrollers where group result registers are not available. Hence for these microntroller devices, all the conversion results are stored in the global result register in shared mode. The wait for read mode hardware option will be enabled for the global result register to avoid overwriting of results. To get a new channel conversion result, it is mandatory to read the previous result using the same API or **ADC_MEASUREMENT_GetGlobalDetailedResult** API.

Note:

- This API is only applicable for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Hence this API shall be called with a pointer to the measurement handle of type **ADC_MEASUREMENT_t**.
- For either 10Bit or 8Bit ADC resolution the result value needs to be right shifted by either 2 or 4 bits respectively. The 10Bit or 8 bit results are left aligned in the result register, hence a shift operation is needed.

```
  // Ensure that end of measurements interrupt ha
s been enabled
  #include <DAVE.h>

  XMC_VADC_RESULT_SIZE_t result;
  void Adc_Measurement_Handler()
  {
  #if(UC_SERIES == XMC11)
   result = ADC_MEASUREMENT_GetGlobalResult();
  #endif
   result = result >> ((uint32_t)ADC_MEASUREMENT_
0.iclass_config_handle->conversion_mode_standard
* (uint32_t)2);
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **614** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_GetResult()**.

XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult (AD

Returns the converted value for a specific channel. Not Applicable for XMC1100.

Parameters:

handle_ptr Constant pointer to the channel handle structure of type **ADC_MEASUREMENT_CHANNEL_t**. (Use the channel handle related macros which are defined in adc_measurement_conf.h)

Returns:

XMC_VADC_RESULT_SIZE_t conversion result.
Range: [0x0 to 0x3FF] if accumulation of results is switched off.

Description:

Reads the converted result stored in the result register [GxRESy.RESULT], assigned to the specified channel. This API is only used in the microcotrollers where separate result registers are available for storing each channel results. For these microcontrollers, each channel is configured to a particular group result register. The result register is defined in the channel handle structure **ADC_MEASUREMENT_CHANNEL_t**. Hence this API shall call be called with a pointer to the channel handle of type **ADC_MEASUREMENT_CHANNEL_t** (Directly use the channel handle related macros which are defined in adc_measurement_conf.h).

Note:

This API is not Applicable for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Use **ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_t**

*const handle_ptr) for XMC1100 microcontrollers.

```
  // Ensure that end of measurements interrupt ha
s been enabled
  #include <DAVE.h>

  XMC_VADC_RESULT_SIZE_t result;
  void Adc_Measurement_Handler()
  {
  #if(UC_SERIES != XMC11)
    result = ADC_MEASUREMENT_GetResult(&ADC_MEASU
REMENT_Channel_A);
  #endif
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **237** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_CHANNEL::ch_handle**, and **ADC_MEASUREMENT_CHANNEL::group_handle**.

XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult (AD

Returns the converted value from the global result register. Only Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the APP handle structure.

Returns:

XMC_VADC_RESULT_SIZE_t conversion result.
Range: [0x0 to 0x3FF] if accumulation of results is switched off.

Description:

Reads the converted result stored in the common result register [GLOBRES], assigned to all the channels. This API is only used in the microcotrollers where group result registers are not available. Hence for these microntroller devices, all the conversion results are stored in the global result register in shared mode. The wait for read mode hardware option will be enabled for the global result register to avoid overwriting of results. To get a new channel conversion result, it is mandatory to read the previous result using the same API or **ADC_MEASUREMENT_GetDetailedResult** API.

Note:

- This API is only applicable for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Hence this API shall be called with a pointer to the measurement handle of type **ADC_MEASUREMENT_t**.
- For either 10Bit or 8Bit ADC resolution the result value needs to be right shifted by either 2 or 4 bits respectively. The 10Bit or 8 bit results are left aligned in the result register, hence a shift operation is needed.

```
  // Ensure that end of measurements interrupt ha
s been enabled
  #include <DAVE.h>

  XMC_VADC_RESULT_SIZE_t result;
  void Adc_Measurement_Handler()
  {
  #if(UC_SERIES == XMC11)
   result = ADC_MEASUREMENT_GetResult(&ADC_MEASUR
EMENT_0);
```

```
  #endif
   result = result >> ((uint32_t)ADC_MEASUREMENT_
0.iclass_config_handle->conversion_mode_standard
* (uint32_t)2);
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **266** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_GetGlobalResult()**.

ADC_MEASUREMENT_STATUS_t ADC_MEASUREMENT_Init (ADC

Initializes the APP to measure a set of analog inputs.

Parameters:

handle_ptr constant pointer to the APP handle structure

Returns:

ADC_MEASUREMENT_STATUS_SUCCESS when initialization succeeds else, return ADC_MEASUREMENT_STATUS_FAILURE.

Description:

Initializes the VADC background scan request source, group channels and result registers with the configuration specified in the handle structure. The API configures the conversion timing parameters of VADC, by setting the [GLOBICLASS] register.

The API initializes the channel and result configurations by setting the [GxCHCTRy] and [GxRCRy] registers respectively. It adds all channels into the background request source channel select register [BRSSEL]. For microcontrollers apart from XMC1100, the APP uses background request source event to generate the interrupt. For XMC1100 devices, global result event is used to generate the interrupt. If "Start conversion after initialization" has been selected in the GUI, the ADC conversions starts immediately at the end of this API call.

```
#include <DAVE.h>
int main(void)
{
   DAVE_Init(); //ADC_MEASUREMENT_Init is called within DAVE_Init
   return 0;
}
```

Definition at line **124** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_STATUS_UNINITIALIZED**, **ADC_MEASUREMENT_CHANNEL::analog_io_config**, **ADC_MEASUREMENT::array**, **ADC_MEASUREMENT::backgnd_config_handle**, **ADC_MEASUREMENT_CHANNEL::ch_handle**, **ADC_MEASUREMENT_CHANNEL::ch_num**, **ADC_MEASUREMENT_CHANNEL_ARRAY::channel_array**, **ADC_MEASUREMENT::global_handle**, **ADC_MEASUREMENT_CHANNEL::group_handle**, **ADC_MEASUREMENT_CHANNEL::group_index**, **ADC_MEASUREMENT::iclass_config_handle**, **ADC_MEASUREMENT::init_state**, **ADC_MEASUREMENT_ISR::irqctrl**, **ADC_MEASUREMENT::mux_config**, **ADC_MEASUREMENT_ISR::node_id**, **ADC_MEASUREMENT_ISR::priority**, **ADC_MEASUREMENT::req_src_intr_handle**,

ADC_MEASUREMENT_CHANNEL::res_handle,
ADC_MEASUREMENT_CHANNEL_ARRAY::res_handle,
ADC_MEASUREMENT::result_intr_handle,
ADC_MEASUREMENT::srv_req_node,
ADC_MEASUREMENT::start_conversion, and
ADC_MEASUREMENT_ISR::sub_priority.

void ADC_MEASUREMENT_StartConversion (ADC_MEASUREMEN

Starts the conversion of the required measurements.

Parameters:

handle_ptr Constant pointer to the APP handle structure

Returns:

None

Description:

If "Start conversion after initialization" option is not selected in the GUI, the conversions can be started by calling this API. A call to this API sets the register bit field BRSMR.LDEV to generate a load event. The load event triggers the conversion of selected channels in a fixed sequence. A conversion request can also be raised upon detection of a hardware trigger. Hence, if "Start conversion after initialization" option enabled or "Trigger edge Selection" is configured to any edge, this API call is not mandatory to start the conversions.

```
  // Ensure that end of measurements interrupt ha
s been enabled
  #include <DAVE.h>

  XMC_VADC_RESULT_SIZE_t result;
  void Adc_Measurement_Handler()
  {
  #if(UC_SERIES != XMC11)
```

```
    result = ADC_MEASUREMENT_GetResult(&ADC_MEASU
REMENT_Channel_A);
  #else
    result = ADC_MEASUREMENT_GetGlobalResult();
  #endif
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **227** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT::global_handle**.

ADC_MEASUREMENT

Usage

Usage

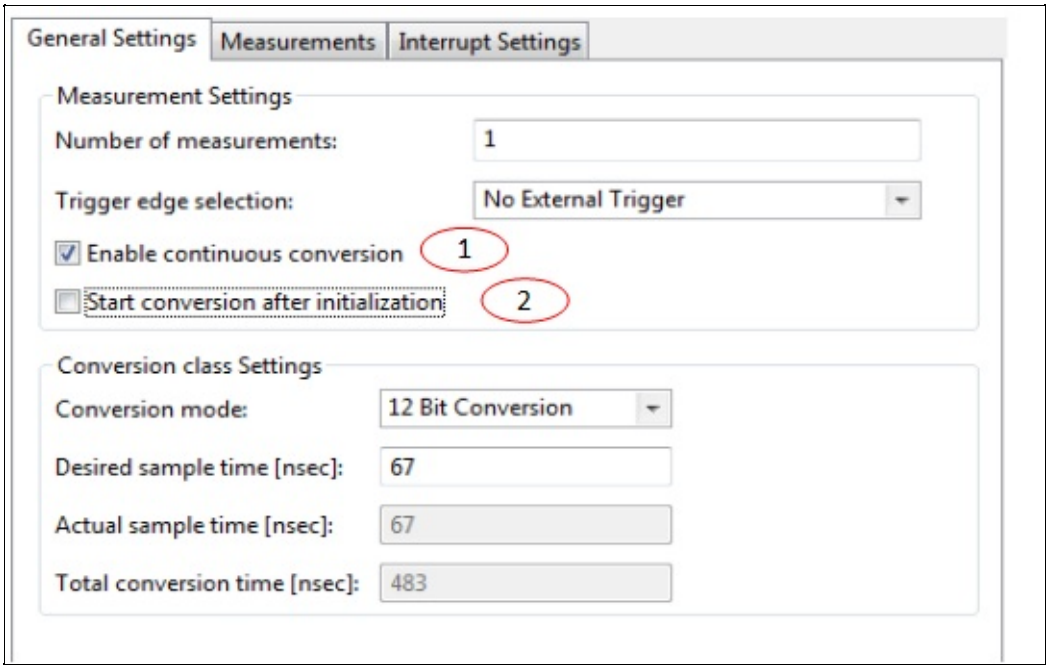
This example demonstrates the conversion of the required ADC channel. If the converted channel is above a threshold voltage then a pin is set to Low else it would be set to High. The port pin is connected to a on-board LED. This would switch on/off the LED depending on the potentiometer value(Potentiometer controls the input voltage for the ADC).

Instantiate the required APPs

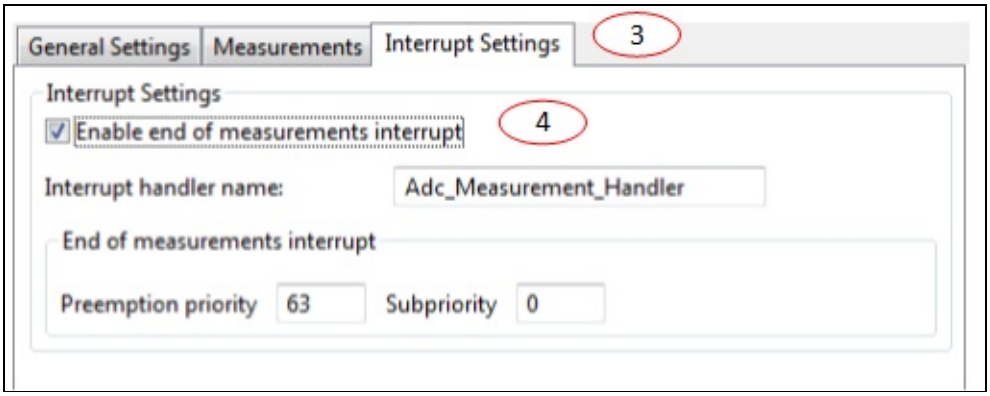
Drag an instance of **ADC_MEASUREMENT** APP and DIGITAL_IO APP. Update the fields in the GUI of these APPs with the following configuration.

Configure the APPs

ADC_MEASUREMENT APP:

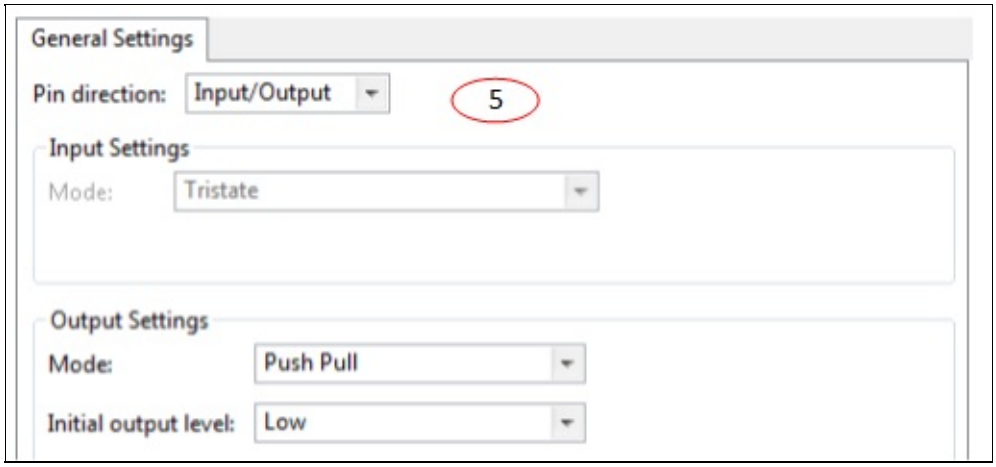


1. Enable continuous conversion.
2. Disable Start conversion after initialization.



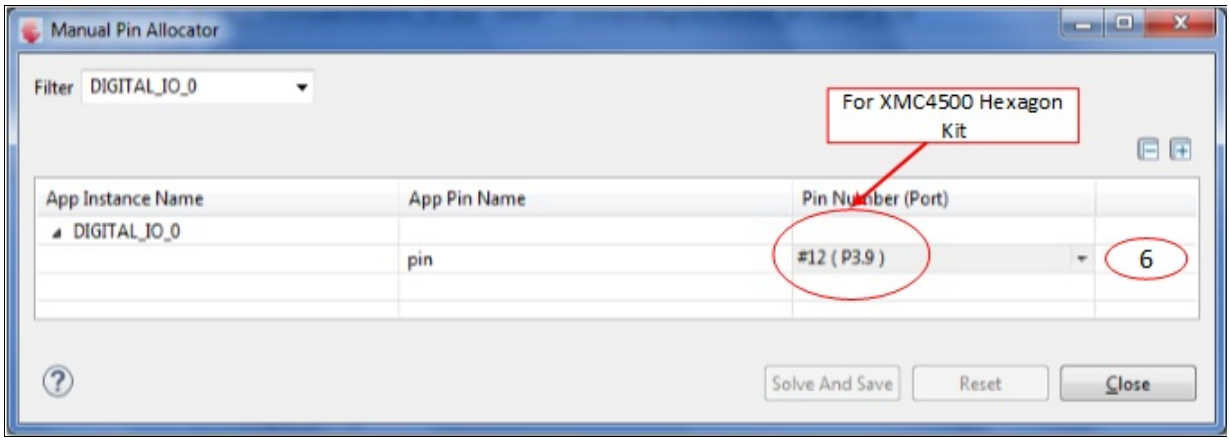
3. Goto Interrupt settings Tab.
4. For XMC1100:
   Enable interrupt after each measurement.
   For other devices:
   Enable end of measurements interrupt.

DIGITAL_IO APP:

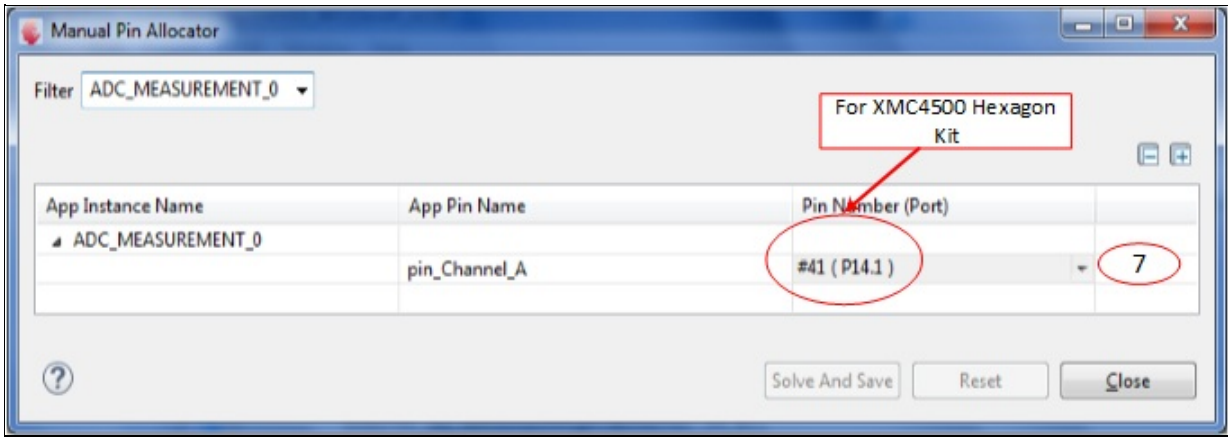


5. Set pin direction to output by choosing - Pin direction : Input/Output

Manual pin allocation



6. Select the pin to be toggled (on-board LED)
   Note: The pin number is specific to the development board chosen to run this example. The pin shown in the image above may not be available on every XMC boot kit. Ensure that a proper pin is selected according to the board.



7. Select the potentiometer Pin present in the boot kit
   Note: The pin number is specific to the development board chosen to run this example. The pin shown in the image above may not be available on every XMC boot kit. Ensure that a proper pin is selected according to the board.

Generate code

Files are generated here: '<project_name>/Dave/Generated/' ('project_name' is the name chosen by the user during project creation). APP instance definitions and APIs are generated only after code generation.

- **Note:** Code must be explicitly generated for every change in the GUI configuration.
  Important: Any manual modification to the APP specific generated files will be overwritten by a subsequent code generation operation.

Sample Application (main.c)

```
 #include <DAVE.h>                 //Declarations from D
AVE Code Generation (includes SFR declaration)

 XMC_VADC_RESULT_SIZE_t result;
 void Adc_Measurement_Handler()
```

```
 {
 #if(UC_SERIES != XMC11)
     result = ADC_MEASUREMENT_GetResult(&ADC_MEASU
REMENT_Channel_A);
 #else
     result = ADC_MEASUREMENT_GetGlobalResult();
 #endif

     if(result >= 2048)
     {
         DIGITAL_IO_SetOutputLow(&DIGITAL_IO_0);
     }
     else
     {
         DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_0);
     }
 }


 int main(void)
 {
   DAVE_STATUS_t status;

   status = DAVE_Init();     /* Initialization of
DAVE Apps  */

   if(status == DAVE_STATUS_FAILURE)
   {
     /* Placeholder for error handler code. The wh
ile loop below can be replaced with an user error
handler */
     XMC_DEBUG(("DAVE Apps initialization failed w
ith status %d\n", status));
     while(1U)
     {
     }
   }
```

```
   ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMEN
T_0);

   while(1U);

   return 1;
 }
```

Build and Run the Project

Observation
Change the potentiometer value connected to the board. If the converted value is greater than 2048(Vcc/2) the LED will be turned on else it would remain switched off.

Calibration work around for XMC1100/XMC1200/XMC1300 AA step devices
NOTE: invoke the API VADC_complete_calibration() to do both startup calibration and gain calibration.

```
 #define SHS0_CALOC0 ((uint32_t *)0x480340E0)
 #define SHS0_CALOC1 ((uint32_t *)0x480340E4)
 #define SHS0_CALCTR ((uint32_t *)0x480340BC)

 #define SHS_CALLOC0_CLEAR_OFFSET (0x8000)
 #define REG_RESET (0x00)
 #define GLOBCFG_CLEAR (0x80030000)
 #define CLEAR_OFFSET_CALIB_VALUES         *SHS0_C
ALOC0 = SHS_CALLOC0_CLEAR_OFFSET;\
                                          *SHS0_CAL
OC1 = SHS_CALLOC0_CLEAR_OFFSET

 void adc_gain_calib(void)
```

```
 {
   uint16_t i = 18000;
   uint32_t adc_result_aux;

   /* ADC_AI.004 errata*/
   *SHS0_CALCTR = 0X3F100400;

   /* add a channel in group-0 for dummy conversio
n*/
   VADC->BRSSEL[0] = VADC_BRSSEL_CHSELG0_Msk;

   /*Clear the DPCAL0, DPCAL1 and SUCAL bits*/
   VADC->GLOBCFG &= ~( VADC_GLOBCFG_DPCAL0_Msk | V
ADC_GLOBCFG_DPCAL1_Msk | VADC_GLOBCFG_SUCAL_Msk);

   /* Clear offset calibration values*/
   CLEAR_OFFSET_CALIB_VALUES;

   VADC->BRSMR = (1 << VADC_BRSMR_ENGT_Pos);
 #if UC_SERIES != XMC11
   VADC_G0->ARBPR = (VADC_G_ARBPR_ASEN2_Msk);
 #endif
   /*Trigger dummy conversion for 9* 2000 times*/

   while(i > 0)
   {
       /*load event */
       VADC->BRSMR |= VADC_BRSMR_LDEV_Msk;
 #if UC_SERIES != XMC11
       /*Wait until a new result is available*/
       while(VADC_G0->VFR == 0);

       /*dummy read of result */
       adc_result_aux = VADC_G0->RES[0];
 #else
       /*Wait untill a new result is available*/
       while((VADC->GLOBRES & VADC_GLOBRES_VF_Msk)
```

```
 == 0);

       /*dummy read of result */
       adc_result_aux = VADC->GLOBRES;
 #endif

       /* Clear offset calibration values*/
       CLEAR_OFFSET_CALIB_VALUES;
       i--;
   }

   /* to avoid a warning*/
   adc_result_aux &= adc_result_aux;

   /* Wait until last gain calibration step is fin
ished */
   while ( (SHS0->SHSCFG & SHS_SHSCFG_STATE_Msk) !
= 0 )
   {
       /* Clear offset calibration values*/
       CLEAR_OFFSET_CALIB_VALUES;
   }
   /* Re enable SUCAL DPCAL */
   VADC->GLOBCFG |= ( VADC_GLOBCFG_DPCAL0_Msk | VA
DC_GLOBCFG_DPCAL1_Msk);
   VADC->BRSMR     = 0x00;
   VADC->BRSSEL[0] = 0x00;
 #if UC_SERIES != XMC11
   VADC_G0->REFCLR = 1U;
   VADC_G0->ARBPR &= ~(VADC_G_ARBPR_ASEN2_Msk);
 #endif
 }

 void VADC_complete_calibration(void)
 {
   uint32_t wait;
   *SHS0_CALOC0 = REG_RESET;
```

```
   *SHS0_CALOC1 = REG_RESET;

   //enable the StartUp calibration in the VADC
   VADC->GLOBCFG |= (1 << VADC_GLOBCFG_SUCAL_Pos &
 VADC_GLOBCFG_SUCAL_Msk)|
                    (1 << VADC_GLOBCFG_DPCAL0_Pos
& VADC_GLOBCFG_DPCAL0_Msk);

   // Wait for 1920cycles or 60us for the startup
calibration to complete
   wait = 20;

   while(wait > 0)
   {
       wait--;
       // Clear offset calibration values
       CLEAR_OFFSET_CALIB_VALUES;
   }
   adc_gain_calib();
 }
```

ADC_MEASUREMENT

Home

Release History

Release History

ADC_MEASUREMENT

Data Structures

Here are the data structures with brief descriptions:

ADC_MEASUREMENT	
ADC_MEASUREMENT_CHANNEL	
ADC_MEASUREMENT_CHANNEL_ARRAY	
ADC_MEASUREMENT_ISR	

ADC_MEASUREMENT

Data Fields

ADC_MEASUREMENT Struct Reference

Detailed Description

Structure to configure **ADC_MEASUREMENT** APP.

Definition at line **203** of file **ADC_MEASUREMENT.h**.

```
#include <ADC_MEASUREMENT.h>
```

Data Fields

const **ADC_MEASUREMENT_CHANNEL_ARRAY_t** *const	**array**
const XMC_VADC_BACKGROUND_CONFIG_t *const	**backgnd_config_han**
const XMC_VADC_GLOBAL_CLASS_t *const	**iclass_config_handle**
GLOBAL_ADC_t *const	**global_handle**
const **ADC_MEASUREMENT_ISR_t** *const	**req_src_intr_handle**
const **ADC_MEASUREMENT_ISR_t** *const	**result_intr_handle**
ADC_MEASUREMENT_MUX_CONFIG_t	**mux_config**
ADC_MEASUREMENT_STATUS_t	**init_state**
const XMC_VADC_SR_t	**srv_req_node**
const bool	**start_conversion**

Field Documentation

const ADC_MEASUREMENT_CHANNEL_ARRAY_t* const ADC_ME

This holds ADC_MEASUREMENT_Channel_HandleArray

Definition at line **205** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

const XMC_VADC_BACKGROUND_CONFIG_t* const ADC_MEASU

This holds the LLD Background Scan Init Structure

Definition at line **207** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

GLOBAL_ADC_t* const ADC_MEASUREMENT::global_handle

This hold the ADC Global APP handle

Definition at line **212** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**, and **ADC_MEASUREMENT_StartConversion()**.

const XMC_VADC_GLOBAL_CLASS_t* const ADC_MEASUREMEN

This holds the adc global ICLASS 0 configuration

Definition at line **210** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

ADC_MEASUREMENT_STATUS_t ADC_MEASUREMENT::init_state

Holds information regarding the APP initialization

Definition at line **222** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

ADC_MEASUREMENT_MUX_CONFIG_t ADC_MEASUREMENT::mu

This hold the pointer to the function that does mux configuration.

Definition at line **220** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

const ADC_MEASUREMENT_ISR_t* const ADC_MEASUREMENT::r

This has the NVIC configuration structure

Definition at line **215** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

const ADC_MEASUREMENT_ISR_t* const ADC_MEASUREMENT::r

This has the NVIC configuration structure

Definition at line **217** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

const XMC_VADC_SR_t ADC_MEASUREMENT::srv_req_node

Service Request Line selected

Definition at line **224** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

const bool ADC_MEASUREMENT::start_conversion

This indicates whether to start at initialization of the APP

Definition at line **226** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

The documentation for this struct was generated from the following file:

- **ADC_MEASUREMENT.h**

ADC_MEASUREMENT

Data Fields

ADC_MEASUREMENT_CHANNEL Struct Reference

Detailed Description

Structure to initialize ADC channels.

Definition at line **166** of file **ADC_MEASUREMENT.h**.

```
#include <ADC_MEASUREMENT.h>
```

Data Fields

XMC_VADC_CHANNEL_CONFIG_t *	**ch_handle**
XMC_VADC_RESULT_CONFIG_t *	**res_handle**
XMC_VADC_GROUP_t *	**group_handle**
ANALOG_IO_t *	**analog_io_config**
uint8_t	**group_index**
uint8_t	**ch_num**

Field Documentation

ANALOG_IO_t* ADC_MEASUREMENT_CHANNEL::analog_io_confi

This hold the address of the ANALOG_IO configuration structure

Definition at line **179** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

XMC_VADC_CHANNEL_CONFIG_t* ADC_MEASUREMENT_CHANN

This holds the VADC Channel LLD struct

Definition at line **169** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_GetDetailedResult()**, **ADC_MEASUREMENT_GetResult()**, and **ADC_MEASUREMENT_Init()**.

uint8_t ADC_MEASUREMENT_CHANNEL::ch_num

This Holds the Channel Number

Definition at line **184** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

XMC_VADC_GROUP_t* ADC_MEASUREMENT_CHANNEL::group_h

This holds the group to which the channel belongs

Definition at line **175** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_GetDetailedResult()**, **ADC_MEASUREMENT_GetResult()**, and **ADC_MEASUREMENT_Init()**.

uint8_t ADC_MEASUREMENT_CHANNEL::group_index

This holds the group index

Definition at line **182** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

XMC_VADC_RESULT_CONFIG_t* ADC_MEASUREMENT_CHANNEl

This hold the VADC LLD Result handler

Definition at line **171** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

The documentation for this struct was generated from the following file:

- **ADC_MEASUREMENT.h**

Data Fields

ADC_MEASUREMENT_CHANNEL_ARRAY Struct Reference

Detailed Description

Structure to hold channels handles that are configured

Definition at line **191** of file **ADC_MEASUREMENT.h**.

```
#include <ADC_MEASUREMENT.h>
```

Data Fields

const **ADC_MEASUREMENT_CHANNEL_t** *const	**channel_array** [ADC_MEASUREMENT_MAXC
XMC_VADC_RESULT_CONFIG_t *	**res_handle**

Field Documentation

const ADC_MEASUREMENT_CHANNEL_t* const ADC_MEASUREM

Array which consists of APPs Channel Handles

Definition at line **193** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

XMC_VADC_RESULT_CONFIG_t* ADC_MEASUREMENT_CHANNEL

This hold the VADC LLD Result handler

Definition at line **196** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

The documentation for this struct was generated from the following file:

- **ADC_MEASUREMENT.h**

ADC_MEASUREMENT

Data Fields

ADC_MEASUREMENT_ISR Struct Reference

Detailed Description

Structure to initialize Request Source Interrupt's NVIC Node

Definition at line **150** of file **ADC_MEASUREMENT.h**.

```
#include <ADC_MEASUREMENT.h>
```

Data Fields

uint32_t	**node_id**
uint32_t	**priority**
uint32_t	**sub_priority**
uint8_t	**irqctrl**

Field Documentation

uint8_t ADC_MEASUREMENT_ISR::irqctrl

This indicates the service request source selected for the consumed NVIC node.

Definition at line **159** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

uint32_t ADC_MEASUREMENT_ISR::node_id

This holds the Node ID of the NVIC.

Definition at line **152** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

uint32_t ADC_MEASUREMENT_ISR::priority

This holds the NVIC priority.

Definition at line **154** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

uint32_t ADC_MEASUREMENT_ISR::sub_priority

This holds the SubPriority of the NVIC. for Only XMC4x Devices

Definition at line **156** of file **ADC_MEASUREMENT.h**.

Referenced by **ADC_MEASUREMENT_Init()**.

The documentation for this struct was generated from the following file:

- **ADC_MEASUREMENT.h**

ADC_MEASUREMENT

Data Structure Index

A

A
ADC_MEASUREMENT
ADC_MEASUREMENT_CHANNEL
ADC_MEASUREMENT_CHANNEL_ARRAY

A

ADC_MEASUREMENT

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- analog_io_config : **ADC_MEASUREMENT_CHANNEL**
- array : **ADC_MEASUREMENT**
- backgnd_config_handle : **ADC_MEASUREMENT**
- ch_handle : **ADC_MEASUREMENT_CHANNEL**
- ch_num : **ADC_MEASUREMENT_CHANNEL**
- channel_array : **ADC_MEASUREMENT_CHANNEL_ARRAY**
- global_handle : **ADC_MEASUREMENT**
- group_handle : **ADC_MEASUREMENT_CHANNEL**
- group_index : **ADC_MEASUREMENT_CHANNEL**
- iclass_config_handle : **ADC_MEASUREMENT**
- init_state : **ADC_MEASUREMENT**
- irqctrl : **ADC_MEASUREMENT_ISR**
- mux_config : **ADC_MEASUREMENT**
- node_id : **ADC_MEASUREMENT_ISR**
- priority : **ADC_MEASUREMENT_ISR**
- req_src_intr_handle : **ADC_MEASUREMENT**
- res_handle : **ADC_MEASUREMENT_CHANNEL** , **ADC_MEASUREMENT_CHANNEL_ARRAY**
- result_intr_handle : **ADC_MEASUREMENT**
- srv_req_node : **ADC_MEASUREMENT**
- start_conversion : **ADC_MEASUREMENT**
- sub_priority : **ADC_MEASUREMENT_ISR**

ADC_MEASUREMENT

- analog_io_config : **ADC_MEASUREMENT_CHANNEL**
- array : **ADC_MEASUREMENT**
- backgnd_config_handle : **ADC_MEASUREMENT**
- ch_handle : **ADC_MEASUREMENT_CHANNEL**
- ch_num : **ADC_MEASUREMENT_CHANNEL**
- channel_array : **ADC_MEASUREMENT_CHANNEL_ARRAY**
- global_handle : **ADC_MEASUREMENT**
- group_handle : **ADC_MEASUREMENT_CHANNEL**
- group_index : **ADC_MEASUREMENT_CHANNEL**
- iclass_config_handle : **ADC_MEASUREMENT**
- init_state : **ADC_MEASUREMENT**
- irqctrl : **ADC_MEASUREMENT_ISR**
- mux_config : **ADC_MEASUREMENT**
- node_id : **ADC_MEASUREMENT_ISR**
- priority : **ADC_MEASUREMENT_ISR**
- req_src_intr_handle : **ADC_MEASUREMENT**
- res_handle : **ADC_MEASUREMENT_CHANNEL** , **ADC_MEASUREMENT_CHANNEL_ARRAY**
- result_intr_handle : **ADC_MEASUREMENT**
- srv_req_node : **ADC_MEASUREMENT**
- start_conversion : **ADC_MEASUREMENT**
- sub_priority : **ADC_MEASUREMENT_ISR**

ADC_MEASUREMENT

File List

Here is a list of all documented files with brief descriptions:

ADC_MEASUREMENT.c [code]
ADC_MEASUREMENT.h [code]

ADC_MEASUREMENT

ADC_MEASUREMENT.c File Reference

Detailed Description

Date:
2016-03-18

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file **ADC_MEASUREMENT.c**.

Functions

DAVE_APP_VERSION_t	**ADC_MEASUREMENT_GetAppV** (void) Get **ADC_MEASUREMENT** APP v
ADC_MEASUREMENT_STATUS_t	**ADC_MEASUREMENT_Init** (**ADC_MEASUREMENT_t** *const handle_ptr) Initializes the APP to measure a se analog inputs.
void	**ADC_MEASUREMENT_StartCon** (**ADC_MEASUREMENT_t** *const handle_ptr) Starts the conversion of the require measurements.
XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMENT_GetResu** (**ADC_MEASUREMENT_CHANN** *const handle_ptr) Returns the converted value for a s channel. Not Applicable for XMC1
uint32_t	**ADC_MEASUREMENT_GetDetai** (**ADC_MEASUREMENT_CHANN** *const handle_ptr) Returns a detailed conversion resu Applicable for XMC1100.
XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMENT_GetResu** (**ADC_MEASUREMENT_t** *const handle_ptr) Returns the converted value from t global result register. Only Applica XMC1100.

uint32_t	**ADC_MEASUREMENT_GetDetai** (**ADC_MEASUREMENT_t** *const handle_ptr) Returns a detailed conversion resu Applicable for XMC1100.

Function Documentation

uint32_t ADC_MEASUREMENT_GetDetailedResult (ADC_MEASUR

Returns a detailed conversion result. Not Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the channel handle structure. (Use the channel handle related macros which are defined in adc_measurement_conf.h)

Returns:

uint32_t The complete result register.

Description:

Returns the 32 bit result register (GxRES[y]) completely. The result of conversion as well as other informations are returned from this API. The detailed result register contains result of the most recent conversion, the channel number requested the conversion, valid flag, converted request source and fast compare result. In polling mechanism the converted result can be read out after checking the valid flag bit. This API can be used in applications where, the channel number associated to the result register is also needed for verification.

Note:

This API is not Applicable for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Use **ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREME *const handle_ptr)** for XMC1100 microcontrollers.

```
#include <DAVE.h>

typedef struct detailed_result_struct
{
```

```
        uint8_t channel_num;
        uint8_t group_num;
        uint16_t conversion_result;
  }detailed_result_struct_t;

  uint32_t result;
  bool valid_result;
  detailed_result_struct_t detailed_result;

  void Adc_Measurement_Handler()
  {
        uint32_t result;
    valid_result = (bool)false;
  #if(UC_SERIES != XMC11)
    result = ADC_MEASUREMENT_GetDetailedResult(&A
DC_MEASUREMENT_Channel_A);
    if((bool)(result >> VADC_G_RES_VF_Pos))
    {
       valid_result = (bool)true;
       detailed_result.channel_num = (result & VA
DC_G_RES_CHNR_Msk) >> VADC_G_RES_CHNR_Pos;
       detailed_result.group_num = ADC_MEASUREMEN
T_Channel_A.group_index;
       detailed_result.conversion_result = result
 & VADC_G_RES_RESULT_Msk;
    }
  #endif
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **250** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_CHANNEL::ch_handle**, and **ADC_MEASUREMENT_CHANNEL::group_handle**.

uint32_t ADC_MEASUREMENT_GetDetailedResult (ADC_MEASUR

Returns a detailed conversion result. Only Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the APP handle structure.

Returns:

uint32_t The complete Result register.

Description:

Returns the 32 bit result register (GLOBRES) completely. The result of conversion as well as other informations are returned from this API. The detailed result register contains result of the most recent conversion, the channel number requested the conversion, valid flag, converted request source and fast compare result. In polling mechanism the converted result can be read out after checking the valid flag bit. This API can be used in applications where, the channel number associated to the result register is also needed for verification. This API is only used in the microcotrollers where group result registers are not available. Hence for these microntroller devices, all the conversion results are stored in the global result register in shared mode. The wait for read mode hardware option will be enabled for the global result register to avoid overwriting of results. To get a new channel conversion result, it is mandatory to read the previous result using the same API or **ADC_MEASUREMENT_GetResult** API.

Note:

- This API is applicable only for XMC1100 microcontroller, because all the channels shares a common result register

called GLOBRES. Use **ADC_MEASUREMENT_GetDetailedResult(ADC_MEASURE *const handle_ptr)** for other microcontrollers.

- For either 10Bit or 8Bit ADC resolution the result value needs to be right shifted by either 2 or 4 bits respectively. The 10Bit or 8 bit results are left aligned in the result register, hence a shift operation is needed.

```
  #include <DAVE.h>

  typedef struct detailed_result_struct
  {
        uint8_t channel_num;
        uint8_t group_num;
        uint16_t conversion_result;
  }detailed_result_struct_t;

  uint32_t result;
  bool valid_result;
  detailed_result_struct_t detailed_result[10];

  void Adc_Measurement_Handler()
  {
        static uint8_t index;
        uint32_t result;
    valid_result = (bool)false;
  #if(UC_SERIES == XMC11)
    result = ADC_MEASUREMENT_GetDetailedResult(&A
DC_MEASUREMENT_0);
  #endif

    if((bool)(result >> VADC_GLOBRES_VF_Pos))
    {
       valid_result = (bool)true;
       detailed_result[index].channel_num = (resu
lt & VADC_GLOBRES_CHNR_Msk) >> VADC_GLOBRES_CHNR_
Pos;
```

```
      detailed_result[index].group_num = ADC_MEA
SUREMENT_Channel_A.group_index;
      detailed_result[index].conversion_result =
 (result & VADC_GLOBRES_RESULT_Msk) >>
                              ((uint32_t)ADC_MEASUR
EMENT_0.iclass_config_handle->conversion_mode_sta
ndard * (uint32_t)2);
    }
    index++;
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **283** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_GetGlobalDetailedResult()**.

XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult (AD

Returns the converted value for a specific channel. Not Applicable for XMC1100.

Parameters:

handle_ptr	Constant pointer to the channel handle structure of type **ADC_MEASUREMENT_CHANNEL_t**. (Use the channel handle related macros which are defined in adc_measurement_conf.h)

Returns:

XMC_VADC_RESULT_SIZE_t conversion result.
Range: [0x0 to 0x3FF] if accumulation of results is switched off.

Description:

Reads the converted result stored in the result register [GxRESy.RESULT], assigned to the specified channel. This API is only used in the microcotrollers where separate result registers are available for storing each channel results. For these microcontrollers, each channel is configured to a particular group result register. The result register is defined in the channel handle structure **ADC_MEASUREMENT_CHANNEL_t**. Hence this API shall call be called with a pointer to the channel handle of type **ADC_MEASUREMENT_CHANNEL_t** (Directly use the channel handle related macros which are defined in adc_measurement_conf.h).

Note:

This API is not Applicable for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Use **ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_t *const handle_ptr)** for XMC1100 microcontrollers.

```
  // Ensure that end of measurements interrupt ha
s been enabled
  #include <DAVE.h>

  XMC_VADC_RESULT_SIZE_t result;
  void Adc_Measurement_Handler()
  {
  #if(UC_SERIES != XMC11)
    result = ADC_MEASUREMENT_GetResult(&ADC_MEASU
REMENT_Channel_A);
  #endif
  }
```

```
  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **237** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_CHANNEL::ch_handle**, and **ADC_MEASUREMENT_CHANNEL::group_handle**.

XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult (AD

Returns the converted value from the global result register. Only Applicable for XMC1100.

Parameters:

handle_ptr constant pointer to the APP handle structure.

Returns:

XMC_VADC_RESULT_SIZE_t conversion result.
Range: [0x0 to 0x3FF] if accumulation of results is switched off.

Description:

Reads the converted result stored in the common result register [GLOBRES], assigned to all the channels. This API is only used in the microcotrollers where group result registers are not available. Hence for these microntroller devices, all the conversion results are stored in the global result register in shared mode. The wait for read mode hardware option will be enabled for the global result register to avoid overwriting of results. To get a new channel conversion result, it is mandatory to read the previous result using the same API or

ADC_MEASUREMENT_GetDetailedResult API.

Note:

- This API is only applicable for XMC1100 microcontroller, because all the channels shares a common result register called GLOBRES. Hence this API shall be called with a pointer to the measurement handle of type **ADC_MEASUREMENT_t**.
- For either 10Bit or 8Bit ADC resolution the result value needs to be right shifted by either 2 or 4 bits respectively. The 10Bit or 8 bit results are left aligned in the result register, hence a shift operation is needed.

```
  // Ensure that end of measurements interrupt ha
s been enabled
  #include <DAVE.h>

  XMC_VADC_RESULT_SIZE_t result;
  void Adc_Measurement_Handler()
  {
  #if(UC_SERIES == XMC11)
   result = ADC_MEASUREMENT_GetResult(&ADC_MEASUR
EMENT_0);
  #endif
   result = result >> ((uint32_t)ADC_MEASUREMENT_
0.iclass_config_handle->conversion_mode_standard
* (uint32_t)2);
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **266** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT_GetGlobalResult()**.

ADC_MEASUREMENT_STATUS_t ADC_MEASUREMENT_Init (ADC

Initializes the APP to measure a set of analog inputs.

Parameters:

handle_ptr constant pointer to the APP handle structure

Returns:

ADC_MEASUREMENT_STATUS_SUCCESS when initialization succeeds else, return ADC_MEASUREMENT_STATUS_FAILURE.

Description:

Initializes the VADC background scan request source, group channels and result registers with the configuration specified in the handle structure. The API configures the conversion timing parameters of VADC, by setting the [GLOBICLASS] register. The API initializes the channel and result configurations by setting the [GxCHCTRy] and [GxRCRy] registers respectively. It adds all channels into the background request source channel select register [BRSSEL]. For microcontrollers apart from XMC1100, the APP uses background request source event to generate the interrupt. For XMC1100 devices, global result event is used to generate the interrupt. If "Start conversion after initialization" has been selected in the GUI, the ADC conversions starts immediately at the end of this API call.

```
#include <DAVE.h>
int main(void)
{
   DAVE_Init(); //ADC_MEASUREMENT_Init is called
within DAVE_Init
```

```
    return 0;
}
```

Definition at line 124 of file ADC_MEASUREMENT.c.

References ADC_MEASUREMENT_STATUS_UNINITIALIZED, ADC_MEASUREMENT_CHANNEL::analog_io_config, ADC_MEASUREMENT::array, ADC_MEASUREMENT::backgnd_config_handle, ADC_MEASUREMENT_CHANNEL::ch_handle, ADC_MEASUREMENT_CHANNEL::ch_num, ADC_MEASUREMENT_CHANNEL_ARRAY::channel_array, ADC_MEASUREMENT::global_handle, ADC_MEASUREMENT_CHANNEL::group_handle, ADC_MEASUREMENT_CHANNEL::group_index, ADC_MEASUREMENT::iclass_config_handle, ADC_MEASUREMENT::init_state, ADC_MEASUREMENT_ISR::irqctrl, ADC_MEASUREMENT::mux_config, ADC_MEASUREMENT_ISR::node_id, ADC_MEASUREMENT_ISR::priority, ADC_MEASUREMENT::req_src_intr_handle, ADC_MEASUREMENT_CHANNEL::res_handle, ADC_MEASUREMENT_CHANNEL_ARRAY::res_handle, ADC_MEASUREMENT::result_intr_handle, ADC_MEASUREMENT::srv_req_node, ADC_MEASUREMENT::start_conversion, and ADC_MEASUREMENT_ISR::sub_priority.

void ADC_MEASUREMENT_StartConversion (ADC_MEASUREMEN

Starts the conversion of the required measurements.

Parameters:

handle_ptr Constant pointer to the APP handle structure

Returns:

None

Description:

If "Start conversion after initialization" option is not selected in the GUI, the conversions can be started by calling this API. A call to this API sets the register bit field BRSMR.LDEV to generate a load event. The load event triggers the conversion of selected channels in a fixed sequence. A conversion request can also be raised upon detection of a hardware trigger. Hence, if "Start conversion after initialization" option enabled or "Trigger edge Selection" is configured to any edge, this API call is not mandatory to start the conversions.

```
  // Ensure that end of measurements interrupt ha
s been enabled
  #include <DAVE.h>

  XMC_VADC_RESULT_SIZE_t result;
  void Adc_Measurement_Handler()
  {
  #if(UC_SERIES != XMC11)
    result = ADC_MEASUREMENT_GetResult(&ADC_MEASU
REMENT_Channel_A);
  #else
    result = ADC_MEASUREMENT_GetGlobalResult();
  #endif
  }

  int main(void)
  {
     DAVE_Init();
     ADC_MEASUREMENT_StartConversion(&ADC_MEASURE
MENT_0);
     while(1);
     return 0;
  }
```

Definition at line **227** of file **ADC_MEASUREMENT.c**.

References **ADC_MEASUREMENT::global_handle**.

Go to the source code of this file.

ADC_MEASUREMENT

ADC_MEASUREMENT.h File Reference

Detailed Description

Date:
2016-03-18

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file **ADC_MEASUREMENT.h**.

Data Structures

struct	**ADC_MEASUREMENT_ISR**
struct	**ADC_MEASUREMENT_CHANNEL**
struct	**ADC_MEASUREMENT_CHANNEL_ARRAY**
struct	**ADC_MEASUREMENT**

Typedefs

typedef void(*	**ADC_MEASUREMENT_** (void)
typedef struct **ADC_MEASUREMENT_ISR**	**ADC_MEASUREMENT_**
typedef struct **ADC_MEASUREMENT_CHANNEL**	**ADC_MEASUREMENT_**
typedef struct **ADC_MEASUREMENT_CHANNEL_ARRAY**	**ADC_MEASUREMENT_**
typedef struct **ADC_MEASUREMENT**	**ADC_MEASUREMENT_**

Functions

DAVE_APP_VERSION_t	**ADC_MEASUREMEN** Get **ADC_MEASURE**
ADC_MEASUREMENT_STATUS_t	**ADC_MEASUREMEN** (**ADC_MEASUREME** Initializes the APP to r inputs.
void	**ADC_MEASUREMEN** (**ADC_MEASUREME** Starts the conversion measurements.
XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMEN** (**ADC_MEASUREME** handle_ptr) Returns the converted Not Applicable for XM
uint32_t	**ADC_MEASUREMEN** (**ADC_MEASUREME** handle_ptr) Returns a detailed cor Applicable for XMC11
XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMEN** (**ADC_MEASUREME** ADC_MEASUREMEN Returns the converted register. Only Applicab
uint32_t	**ADC_MEASUREMEN** (**ADC_MEASUREME**

	ADC_MEASUREMEN Returns a detailed cor Applicable for XMC11(
__STATIC_INLINE XMC_VADC_RESULT_SIZE_t	**ADC_MEASUREMEN** Returns the converted register. Only Applicab
__STATIC_INLINE uint32_t	**ADC_MEASUREMEN** (void) Returns a detailed cor Applicable for XMC11(
enum	**ADC_MEASUREMEN** **ADC_MEASUREMEN** **ADC_MEASUREMEN** **ADC_MEASUREMEN** } Return value of an AP
typedef enum **ADC_MEASUREMENT_STATUS**	**ADC_MEASUREMEN** Return value of an AP

Go to the source code of this file.

ADC_MEASUREMENT

Home

File List | Globals

All | Functions | Typedefs | Enumerations | Enumerator

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- ADC_MEASUREMENT_CHANNEL_ARRAY_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_CHANNEL_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetAppVersion() : **ADC_MEASUREMENT.h** , **ADC_MEASUREMENT.c**
- ADC_MEASUREMENT_GetDetailedResult() : **ADC_MEASUREMENT.h** , **ADC_MEASUREMENT.c** , **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetGlobalDetailedResult() : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetGlobalResult() : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetResult() : **ADC_MEASUREMENT.c** , **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_Init() : **ADC_MEASUREMENT.h** , **ADC_MEASUREMENT.c**
- ADC_MEASUREMENT_ISR_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_MUX_CONFIG_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_StartConversion() : **ADC_MEASUREMENT.c** , **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_STATUS : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_STATUS_FAILURE : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_STATUS_SUCCESS : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_STATUS_t : **ADC_MEASUREMENT.h**

- ADC_MEASUREMENT_STATUS_UNINITIALIZED : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_t : **ADC_MEASUREMENT.h**

ADC_MEASUREMENT

- ADC_MEASUREMENT_GetAppVersion() : **ADC_MEASUREMENT.c** , **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetDetailedResult() : **ADC_MEASUREMENT.c** , **ADC_MEASUREMENT.h** , **ADC_MEASUREMENT.c** , **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetGlobalDetailedResult() : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetGlobalResult() : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_GetResult() : **ADC_MEASUREMENT.c** , **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_Init() : **ADC_MEASUREMENT.h** , **ADC_MEASUREMENT.c**
- ADC_MEASUREMENT_StartConversion() : **ADC_MEASUREMENT.h** , **ADC_MEASUREMENT.c**

ADC_MEASUREMENT

- ADC_MEASUREMENT_CHANNEL_ARRAY_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_CHANNEL_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_ISR_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_MUX_CONFIG_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_STATUS_t : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_t : **ADC_MEASUREMENT.h**

ADC_MEASUREMENT

- ADC_MEASUREMENT_STATUS : **ADC_MEASUREMENT.h**

ADC_MEASUREMENT

- ADC_MEASUREMENT_STATUS_FAILURE : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_STATUS_SUCCESS : **ADC_MEASUREMENT.h**
- ADC_MEASUREMENT_STATUS_UNINITIALIZED : **ADC_MEASUREMENT.h**

ADC_MEASUREMENT

ADC_MEASUREMENT.h

Go to the documentation of this file.

```
00001
00089 /*CODE_BLOCK_BEGIN*/
00090 #ifndef ADC_MEASUREMENT_H
00091 #define ADC_MEASUREMENT_H
00092
00093
00094 /*********************************************
*********************************************
*************************
00095  * HEADER FILES
00096  *********************************************
*********************************************
*************************/
00097
00098 #include "GLOBAL_ADC/global_adc.h"
00099 #include "adc_measurement_conf.h"
00100
00101  /*********************************************
*********************************************
*************************
00102  * MACROS
00103  *********************************************
*********************************************
*************************/
00104 #if (!((XMC_LIB_MAJOR_VERSION == 2U) && \
00105        (XMC_LIB_MINOR_VERSION >= 0U) && \
00106        (XMC_LIB_PATCH_VERSION >= 0U)))
```

```
00107 #error "ADC_MEASUREMENT requires XMC Periphe
ral Library v2.0.0 or higher"
00108 #endif
00109
00110 #if defined (__GNUC__) || defined (__CC_ARM)
00111 #define ADC_MEASUREMENT_DEPRECATED   __attri
bute__((deprecated))
00112 #else
00113 #define ADC_MEASUREMENT_DEPRECATED
00114 #endif
00115
00116 #define ADC_MEASUREMENT_MODULE_PTR ((XMC_VAD
C_GLOBAL_t *) (void *) VADC)
00117
00118  /*********************************************
*************************************************
*************************
00119  * ENUMS
00120  *********************************************
*************************************************
*************************/
00127 typedef enum ADC_MEASUREMENT_STATUS
00128 {
00129   ADC_MEASUREMENT_STATUS_SUCCESS = 0,
00130   ADC_MEASUREMENT_STATUS_FAILURE,
00131   ADC_MEASUREMENT_STATUS_UNINITIALIZED
00132 } ADC_MEASUREMENT_STATUS_t;
00137 /*********************************************
*************************************************
*************************
00138 * DATA STRUCTURES
00139 *********************************************
*************************************************
*************************/
00145 typedef void (*ADC_MEASUREMENT_MUX_CONFIG_t)
(void);
00150 typedef struct ADC_MEASUREMENT_ISR
```

```
00151 {
00152   uint32_t node_id;
00154   uint32_t priority;
00155 #if(UC_FAMILY == XMC4)
00156   uint32_t sub_priority;
00157 #endif
00158 #ifdef ADC_MEASUREMENT_NON_DEFAULT_IRQ_SOURC
E_SELECTED
00159   uint8_t irqctrl;
00160 #endif
00161 } ADC_MEASUREMENT_ISR_t;
00162
00166 typedef struct ADC_MEASUREMENT_CHANNEL
00167 {
00168 #if( XMC_VADC_GROUP_AVAILABLE == 1U)
00169   XMC_VADC_CHANNEL_CONFIG_t *ch_handle;
00171   XMC_VADC_RESULT_CONFIG_t *res_handle;
00172 #endif
00173
00174 #if( XMC_VADC_GROUP_AVAILABLE == 1U)
00175   XMC_VADC_GROUP_t *group_handle;

00176 #endif
00177
00178 #ifdef ADC_MEASUREMENT_ANALOG_IO_USED
00179   ANALOG_IO_t   *analog_io_config;
00180 #endif
00181
00182   uint8_t group_index;

00184   uint8_t ch_num;

00186 } ADC_MEASUREMENT_CHANNEL_t;
00187
00191 typedef struct ADC_MEASUREMENT_CHANNEL_ARRAY
00192 {
00193   const ADC_MEASUREMENT_CHANNEL_t *const cha
```

```
nnel_array[ADC_MEASUREMENT_MAXCHANNELS];
00195 #if( XMC_VADC_GROUP_AVAILABLE == 0U)
00196   XMC_VADC_RESULT_CONFIG_t *res_handle;
00197 #endif
00198 } ADC_MEASUREMENT_CHANNEL_ARRAY_t;
00199
00203 typedef struct ADC_MEASUREMENT
00204 {
00205   const ADC_MEASUREMENT_CHANNEL_ARRAY_t *con
st array;
00207   const XMC_VADC_BACKGROUND_CONFIG_t *const
backgnd_config_handle;
00210   const XMC_VADC_GLOBAL_CLASS_t *const iclas
s_config_handle;
00212   GLOBAL_ADC_t *const global_handle;

00214 #if (UC_SERIES != XMC11)
00215   const ADC_MEASUREMENT_ISR_t *const req_src
_intr_handle;
00216 #else
00217   const ADC_MEASUREMENT_ISR_t *const result_
intr_handle;
00218 #endif
00219
00220   ADC_MEASUREMENT_MUX_CONFIG_t mux_config;
00222   ADC_MEASUREMENT_STATUS_t init_state;
00224   const XMC_VADC_SR_t srv_req_node;
00226   const bool start_conversion;

00227 } ADC_MEASUREMENT_t;
00228
00233 #ifdef __cplusplus
00234 extern "C" {
00235 #endif
00236
00237 /*************************************************
*************************************************
```

```
*************************
00238  * API Prototypes
00239  ****************************************
*****************************************************
*************************/
00279 DAVE_APP_VERSION_t ADC_MEASUREMENT_GetAppVer
sion(void);
00280
00304 ADC_MEASUREMENT_STATUS_t ADC_MEASUREMENT_Init
(ADC_MEASUREMENT_t *const handle_ptr);
00305
00341 void ADC_MEASUREMENT_StartConversion(ADC_MEA
SUREMENT_t *const handle_ptr);
00342
00343 #if(XMC_VADC_GROUP_AVAILABLE == 1U)
00344
00384 XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetRe
sult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr);
00385
00443 uint32_t ADC_MEASUREMENT_GetDetailedResult(A
DC_MEASUREMENT_CHANNEL_t *const handle_ptr);
00444
00445 #else /* Applicable for XMC1100 devices*/
00446
00492 XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetRe
sult(ADC_MEASUREMENT_t *const handle_ptr) ADC_MEAS
UREMENT_DEPRECATED;
00493
00566 uint32_t ADC_MEASUREMENT_GetDetailedResult(A
DC_MEASUREMENT_t *const handle_ptr) ADC_MEASUREMEN
T_DEPRECATED;
00567
00614 __STATIC_INLINE XMC_VADC_RESULT_SIZE_t  ADC_
MEASUREMENT_GetGlobalResult(void)
00615 {
00616         XMC_VADC_RESULT_SIZE_t result;
00617
```

```
00618          result = XMC_VADC_GLOBAL_GetDetailed
Result(ADC_MEASUREMENT_MODULE_PTR);
00619
00620          return (result);
00621 }
00622
00695 __STATIC_INLINE uint32_t ADC_MEASUREMENT_Get
GlobalDetailedResult(void)
00696 {
00697          uint32_t result;
00698          result = XMC_VADC_GLOBAL_GetDetailed
Result(ADC_MEASUREMENT_MODULE_PTR);
00699
00700          return (result);
00701 }
00702 #endif
00703
00708 #include "ADC_MEASUREMENT_Extern.h"
00709 #ifdef __cplusplus
00710 }
00711 #endif
00712
00713 #endif /* ADC_MEASUREMENT_H_ */
```

ADC_MEASUREMENT

ADC_MEASUREMENT.c

Go to the documentation of this file.

```
00001
00089 /*********************************************
*********************************************
*************************
00090  * HEADER FILES
00091  *********************************************
*********************************************
*************************/
00092 #include "adc_measurement.h"
00093
00094 /*********************************************
*********************************************
*************************
00095  * MACROS
00096  *********************************************
*********************************************
*************************/
00097
00098 /*********************************************
*********************************************
*************************
00099  * LOCAL DATA
00100  *********************************************
*********************************************
*************************/
00101
00102 /*********************************************
```

```
**********************************************************
*************************
00103  * LOCAL ROUTINES
00104  **********************************************
**********************************************************
*************************/
00105
00106 /**********************************************
**********************************************************
************************
00107  * API IMPLEMENTATION
00108  **********************************************
**********************************************************
************************/
00109
00110 /*This function returns the version of the A
DC_MEASUREMENT App*/
00111 DAVE_APP_VERSION_t ADC_MEASUREMENT_GetAppVer
sion(void)
00112 {
00113   DAVE_APP_VERSION_t version;
00114
00115   version.major = (uint8_t) ADC_MEASUREMENT_
MAJOR_VERSION;
00116   version.minor = (uint8_t) ADC_MEASUREMENT_
MINOR_VERSION;
00117   version.patch = (uint8_t) ADC_MEASUREMENT_
PATCH_VERSION;
00118
00119   return version;
00120 }
00121
00122 /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~*/
00123 /* Initialization routine to call ADC LLD AP
I's */
```

```
00124 ADC_MEASUREMENT_STATUS_t ADC_MEASUREMENT_Init
(ADC_MEASUREMENT_t *const handle_ptr)
00125 {
00126   const ADC_MEASUREMENT_CHANNEL_t *indexed;
00127   uint8_t j;
00128   ADC_MEASUREMENT_STATUS_t status;
00129
00130   XMC_ASSERT("ADC_MEASUREMENT_Init:Invalid h
andle_ptr", (handle_ptr != NULL))
00131
00132   if (ADC_MEASUREMENT_STATUS_UNINITIALIZED =
= handle_ptr->init_state)
00133   {
00134     /* Call the function to initialise Clock
 and ADC global functional units*/
00135     status = (ADC_MEASUREMENT_STATUS_t) GLOB
AL_ADC_Init(handle_ptr->global_handle);
00136
00137     /*Initialize the Global Conversion class
 0*/
00138     XMC_VADC_GLOBAL_InputClassInit(handle_pt
r->global_handle->module_ptr,*handle_ptr->iclass_c
onfig_handle,
00139
  XMC_VADC_GROUP_CONV_STD,ADC_MEASUREMENT_ICLASS_N
UM);
00140 #if (UC_SERIES == XMC11)
00141     /*Initialize the Global Conversion class
 1*/
00142     XMC_VADC_GLOBAL_InputClassInit(handle_pt
r->global_handle->module_ptr,*handle_ptr->iclass_c
onfig_handle,
00143
  XMC_VADC_GROUP_CONV_STD,ADC_MEASUREMENT_ICLASS_N
UM_XMC11);
00144 #endif
00145
```

```
00146     /* Initialize the Background Scan hardwa
re */
00147     XMC_VADC_GLOBAL_BackgroundInit(handle_pt
r->global_handle->module_ptr, handle_ptr->backgnd_
config_handle);
00148
00149 #if (XMC_VADC_GROUP_AVAILABLE == 0U)
00150     /* Initialize the global result register
 */
00151     XMC_VADC_GLOBAL_ResultInit(handle_ptr->g
lobal_handle->module_ptr,handle_ptr->array->res_ha
ndle);
00152 #endif
00153
00154     for (j = (uint8_t)0; j < (uint8_t)ADC_ME
ASUREMENT_MAXCHANNELS; j++)
00155     {
00156       indexed = handle_ptr->array->channel_a
rray[j];
00157 #if (XMC_VADC_GROUP_AVAILABLE == 1U)
00158       /* Initialize for configured channels*/

00159       XMC_VADC_GROUP_ChannelInit(indexed->gr
oup_handle,(uint32_t)indexed->ch_num, indexed->ch_
handle);
00160
00161       /* Initialize for configured result re
gisters */
00162       XMC_VADC_GROUP_ResultInit(indexed->gro
up_handle, (uint32_t)indexed->ch_handle->result_re
g_number,
00163                                 indexed->res
_handle);
00164 #endif
00165       /* Add all channels into the Backgroun
d Request Source Channel Select Register */
00166       XMC_VADC_GLOBAL_BackgroundAddChannelTo
```

```
Sequence(handle_ptr->global_handle->module_ptr,
00167
         (uint32_t)indexed->group_index, (uint32_t
)indexed->ch_num);
00168
00169 #ifdef ADC_MEASUREMENT_ANALOG_IO_USED
00170       /* ANALOG_IO initialization for the ch
annel*/
00171       if(indexed->analog_io_config != NULL)
00172       {
00173         status |= (ADC_MEASUREMENT_STATUS_t)
 ANALOG_IO_Init(indexed->analog_io_config);
00174       }
00175 #endif
00176     }
00177 #if(UC_SERIES != XMC11)
00178     if ((handle_ptr->backgnd_config_handle->
req_src_interrupt) && (handle_ptr->req_src_intr_ha
ndle != NULL ))
00179     {
00180 #if (UC_FAMILY == XMC1)
00181       NVIC_SetPriority((IRQn_Type)handle_ptr
->req_src_intr_handle->node_id,
00182                        handle_ptr->req_src_
intr_handle->priority);
00183 #else
00184       NVIC_SetPriority((IRQn_Type)handle_ptr
->req_src_intr_handle->node_id,
00185                        NVIC_EncodePriority(
NVIC_GetPriorityGrouping(),
00186                        handle_ptr->req_src_
intr_handle->priority, handle_ptr->req_src_intr_ha
ndle->sub_priority));
00187 #endif
00188       /* Connect background Request Source E
vent to NVIC node */
00189       XMC_VADC_GLOBAL_BackgroundSetReqSrcEve
```

```
ntInterruptNode(handle_ptr->global_handle->module_
ptr,
00190
             (XMC_VADC_SR_t) handle_ptr->srv_req_n
ode);
00191
00192       /* Enable Background Scan Request sour
ce IRQ */
00193       NVIC_EnableIRQ((IRQn_Type)handle_ptr->
req_src_intr_handle->node_id);
00194 #ifdef ADC_MEASUREMENT_NON_DEFAULT_IRQ_SOURC
E_SELECTED
00195   XMC_SCU_SetInterruptControl(handle_ptr->re
q_src_intr_handle->node_id,
00196                                   ((handle_ptr->
req_src_intr_handle->node_id << 8) | handle_ptr->r
eq_src_intr_handle->irqctrl));
00197 #endif
00198     }
00199 #else /* Selected device is XMC11*/
00200     XMC_VADC_GLOBAL_SetResultEventInterruptN
ode(handle_ptr->global_handle->module_ptr, handle_
ptr->srv_req_node );
00201 #ifdef ADC_MEASUREMENT_CPU_1X /* End of sing
le measurement is enabled*/
00202     NVIC_SetPriority((IRQn_Type)handle_ptr->
result_intr_handle->node_id,
00203                                 handle_ptr->resu
lt_intr_handle->priority);
00204
00205     /* Enable Background Scan Request source
 IRQ */
00206     NVIC_EnableIRQ((IRQn_Type)handle_ptr->re
sult_intr_handle->node_id);
00207 #endif
00208 #endif
00209     /* Mux Configuration is done*/
```

```
00210     if (handle_ptr->mux_config != NULL)
00211     {
00212         (handle_ptr->mux_config)();
00213     }
00214
00215     if (handle_ptr->start_conversion != (bool
)false)
00216     {
00217       /* Start conversion manually using loa
d event trigger*/
00218       XMC_VADC_GLOBAL_BackgroundTriggerConve
rsion(handle_ptr->global_handle->module_ptr);
00219     }
00220     handle_ptr->init_state = status;
00221   }
00222   return (handle_ptr->init_state);
00223 }
00224 /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~*/
00225
00226 /* This API will Software trigger ADC Backgr
ound request source and starts conversion*/
00227 void ADC_MEASUREMENT_StartConversion(ADC_MEA
SUREMENT_t *const handle_ptr)
00228 {
00229   XMC_ASSERT("ADC_MEASUREMENT_Start:Invalid
handle_ptr", (handle_ptr != NULL))
00230
00231   /* Generate a load event to start backgrou
nd request source conversion*/
00232   XMC_VADC_GLOBAL_BackgroundTriggerConversio
n(handle_ptr->global_handle->module_ptr);
00233 }
00234 /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~*/
```

```
00235 #if(XMC_VADC_GROUP_AVAILABLE == 1U)
00236 /* This API will get the result of a convers
ion for a specific channel*/
00237 XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetRe
sult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr)
00238 {
00239   XMC_VADC_RESULT_SIZE_t result;
00240
00241   XMC_ASSERT("ADC_MEASUREMENT_GetResult:Inva
lid handle_ptr", (handle_ptr != NULL))
00242
00243   result = XMC_VADC_GROUP_GetResult(handle_p
tr->group_handle, handle_ptr->ch_handle->result_re
g_number);
00244
00245   return (result);
00246 }
00247 /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~*/
00248
00249 /* This API will get the result of a convers
ion for a specific channel. It will return the com
plete result register*/
00250 uint32_t ADC_MEASUREMENT_GetDetailedResult(A
DC_MEASUREMENT_CHANNEL_t *const handle_ptr)
00251 {
00252   uint32_t result;
00253
00254   XMC_ASSERT("ADC_MEASUREMENT_GetDetailedRes
ult:Invalid handle_ptr", (handle_ptr != NULL))
00255
00256   result = XMC_VADC_GROUP_GetDetailedResult(
handle_ptr->group_handle, handle_ptr->ch_handle->r
esult_reg_number);
00257
00258   return (result);
```

```
00259 }
00260
00261 #else /* Applicable for XMC1100 devices */
00262 /* This API will get the result of the conve
rsion from the global result
00263  *
00264  * This API has been deprecated. Use ADC_MEA
SUREMENT_GetGlobalResult() to get the global resul
t.
00265  * */
00266 XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetRe
sult(ADC_MEASUREMENT_t *const handle_ptr)
00267 {
00268   XMC_VADC_RESULT_SIZE_t result;
00269
00270   XMC_ASSERT("ADC_MEASUREMENT_GetResult:Inva
lid handle_ptr", (handle_ptr != NULL))
00271
00272   XMC_UNUSED_ARG(handle_ptr);
00273
00274   result = ADC_MEASUREMENT_GetGlobalResult()
;
00275   return (result);
00276 }
00277 /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~*/
00278
00279 /* This API will get the result of a convers
ion for a specific channel. It will return the glo
bal result register
00280  *
00281  * This API has been deprecated. Use ADC_MEA
SUREMENT_GetGlobalDetailedResult() to get the glob
al result.
00282  * */
00283 uint32_t ADC_MEASUREMENT_GetDetailedResult(A
```

```
DC_MEASUREMENT_t *const handle_ptr)
00284 {
00285   uint32_t result;
00286
00287   XMC_ASSERT("ADC_MEASUREMENT_GetDetailedRes
ult:Invalid handle_ptr", (handle_ptr != NULL))
00288
00289   XMC_UNUSED_ARG(handle_ptr);
00290
00291   /* Needed only for XMC1100 devices to read
 global result register*/
00292   result = ADC_MEASUREMENT_GetGlobalDetailed
Result();
00293   return (result);
00294 }
00295 #endif
```