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
License Terms and Copyright Information

Copyright (c) 2015, Infineon Technologies AG All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(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).
## Abbreviations and Definitions

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

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

![Conversion Sequence Diagram]

**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 / EVENTGENERATOR.

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
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.
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
5. XMC4200 / XMC4100 Reference Manual
7. XMC1300 Reference Manual
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

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Input/Output</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_background_source</td>
<td>Output</td>
<td>Always</td>
</tr>
</tbody>
</table>
event_result\_channel\_x
Where \textit{channel}\_x represents the channels that are being used.

\textbf{Output} \hspace{1cm} \textbf{Conditional Trigger selection signal}
trigger_input  
Input  
Always  

Always Used to configure the trigger signal for the conversion to start. Can be connected to various sources to act as a trigger for the conversion (e.g. CCU4, CCU8, ERU etc).

Input  
Always  

Always Gating selection signal Used to configure the gating signal for the background request source. Can be connected to various sources to act as a gating for the ADC (e.g. CCU4, CCU8, ERU etc).

Always  

Provides an interface to select a specific VADC module to which the background would belong to. (By default always connected to the GLOBAL_ADC APPs global_signal)

Input  
Always  

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:**

   \[
   \text{Actual Sample Time} = (2 + \text{STC}) \times t\text{ADCI}
   \]
   
   where,
   
   \[
   \begin{align*}
   \text{STC} & : \text{Sample time control (Value - 0 to 256)} \\
   f\text{ADCI} & : \text{Analog clock frequency} \\
   t\text{ADCI} & = 1/f\text{ADCI}
   \end{align*}
   \]

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 \(2/f\text{ADC}\) (GLOBAL_ADC APP).
   
   1. **Standard Conversion Mode:**

   \[
   \text{Total Conversion time} = (2 + \text{STC} + \text{N} + \text{DM} + \text{PC}) \times t\text{ADCI} + 2 \times t\text{ADC}
   \]
   
   Where,
   
   \[
   \begin{align*}
   \text{N} & = 8, 10, 12 \text{ for } n \text{ bit resolution.} \\
   t\text{ADC} & = \text{ADC module clock } = \text{system clock} \\
   t\text{ADCI} & = \text{Analog clock} \\
   \text{STC} & = \text{Sample time control (Value - 0 to 256)} \\
   \text{DM} & = \text{The selected duration of the MSB conversion (DM = 0 or 1)} \\
   \text{PC} & = \text{The post-calibration time PC, if selected (PC = 2)}
   \end{align*}
   \]

   2. **Fast Compare Mode:**

   \[
   \text{Total Conversion time} = (2 + \text{STC} + 2) \times t\text{ADCI} + 2 \times t\text{ADC}
   \]
   
   Where,
   
   \[
   \begin{align*}
   t\text{ADC} & = \text{ADC module clock } = \text{system clock} \\
   t\text{ADCI} & = \text{Analog clock} \\
   \text{STC} & = \text{Sample time control (Value - 0 to 256)}
   \end{align*}
   \]

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

   1. **Standard Conversion Mode:**

   \[
   \text{Total Conversion time} = (2 + \text{STC}) \times t\text{ADCI} + (4 \times t\text{SH}) + (\]
Where,
N							= 8, 10, 12 for n bit resolution.
tSH					=	Sample and Hold clock (Converter clock time period)
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 period)
tADCI				=	Analog clock
STC					=	Sample time control (Value - 0 to 256)
Note: FCRT value is configured as 0.
## App Configuration Parameters

### General Settings

<table>
<thead>
<tr>
<th>Measurement Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of measurements:</td>
</tr>
<tr>
<td>Trigger edge selection:</td>
</tr>
<tr>
<td>Enable continuous conversion</td>
</tr>
<tr>
<td>Start conversion after initialization</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conversion class Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion mode:</td>
</tr>
<tr>
<td>Desired sample time [nsec]:</td>
</tr>
<tr>
<td>Actual sample time [nsec]:</td>
</tr>
<tr>
<td>Total conversion time [nsec]:</td>
</tr>
</tbody>
</table>

Figure 1: General Settings
Figure 2: Measurements

<table>
<thead>
<tr>
<th>Measurement names</th>
<th>Expose pin</th>
<th>Result event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel_A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_J</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel_M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 3: Measurements Cont’d

<table>
<thead>
<tr>
<th>Measurement names</th>
<th>Expose pin</th>
<th>Result event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interrupt Settings

- Enable end of measurements interrupt
- Interrupt handler name: *Adc_Measurement_Handler*
- Preemption priority: 63
- Subpriority: 0

**Figure 4: Interrupt Settings**
Figure 5: Interrupt Settings

Applicable for XMC1100 only
### ADC_MEASUREMENT

#### Enumerations

<table>
<thead>
<tr>
<th>Type</th>
<th>ADC_MEASUREMENT</th>
<th>ADC_MEASUREMENT</th>
<th>ADC_MEASUREMENT</th>
<th>ADC_MEASUREMENT</th>
<th>ADC_MEASUREMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum</td>
<td>ADC_MEASUREMENT</td>
<td>ADC_MEASUREMENT</td>
<td>ADC_MEASUREMENT</td>
<td>ADC_MEASUREMENT</td>
<td>ADC_MEASUREMENT</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>Return value of an AP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typedef enum</td>
<td>ADC_MEASUREMENT_STATUS</td>
<td>ADC_MEASUREMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return value of an AP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 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

### Data structures

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Structure Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef void(*)</td>
<td>ADC_MEASUREMENT (void)</td>
</tr>
<tr>
<td>typedef struct ADC_MEASUREMENT_ISR</td>
<td>ADC_MEASUREMENT</td>
</tr>
<tr>
<td>typedef struct ADC_MEASUREMENT_CHANNEL</td>
<td>ADC_MEASUREMENT</td>
</tr>
<tr>
<td>typedef struct ADC_MEASUREMENT_CHANNEL_ARRAY</td>
<td>ADC_MEASUREMENT</td>
</tr>
<tr>
<td>typedef struct ADC_MEASUREMENT</td>
<td>ADC_MEASUREMENT</td>
</tr>
</tbody>
</table>
Typedef Documentation

typedef struct ADC_MEASUREMENT_CHANNEL_ARRAY ADC_MEASUREMENT_CHANNEL_ARRAY_t

Structure to hold channels handles that are configured

typedef struct ADC_MEASUREMENT_CHANNEL ADC_MEASUREMENT_CHANNEL_t

Structure to initialize ADC channels.

typedef struct ADC_MEASUREMENT_ISR ADC_MEASUREMENT_ISR_t

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.
## Methods

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>ADC_MEASUREMENT_GetAppVersion</td>
<td>Get ADC_MEASUREMENT APP version.</td>
</tr>
<tr>
<td>ADC_MEASUREMENT_STATUS_t</td>
<td>ADC_MEASUREMENT_Init(ADC_MEASUREMENT_t *const handle_ptr)</td>
<td>Initializes the APP to measure a set of analog inputs.</td>
</tr>
<tr>
<td>void</td>
<td>ADC_MEASUREMENT_StartConversion(ADC_MEASUREMENT_t *const handle_ptr)</td>
<td>Starts the conversion of the required measurements.</td>
</tr>
<tr>
<td>XMC_VADC_RESULT_SIZE_t</td>
<td>ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_CHANNEL_t handle_ptr)</td>
<td>Returns the converted value for a specific channel. Not Applicable for XMC1100.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_CHANNEL_t handle_ptr)</td>
<td>Returns a detailed conversion result. Not Applicable for XMC1100.</td>
</tr>
<tr>
<td>XMC_VADC_RESULT_SIZE_t</td>
<td>ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_t *const handle_ptr)</td>
<td>Returns the converted value from the global result register. Only Applicable for XMC1100.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Applicability</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>uint32_t ADC_MEASUREMENT_GetDetailedResult(const ADC_MEASUREMENT_t *handle_ptr)</td>
<td>Returns a detailed conversion result. Only applicable for XMC1100.</td>
<td></td>
</tr>
<tr>
<td>__STATIC_INLINE XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetGlobalResult</td>
<td>Returns the converted value from the global result register. Only applicable for XMC1100.</td>
<td></td>
</tr>
<tr>
<td>__STATIC_INLINE uint32_t ADC_MEASUREMENT_GetGlobalDetailedResult(void)</td>
<td>Returns a detailed conversion result. Only applicable for XMC1100.</td>
<td></td>
</tr>
</tbody>
</table>

**Methods**
Function Documentation

DAVE_APP_VERSION_t ADC_MEASUREMENT_GetAppVersion (void)

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:

```c
#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 within DAVE_Init().
    init_status = DAVE_Init();
    version = ADC_MEASUREMENT_GetAppVersion();
    if (version.major != 1U) {
        // Probably, not the right version.
    }

    // More code here
    while(1) {
    }
}
```


```c
uint32_t ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_CHANNEL_t handle_ptr);
```

Definition at line 111 of file `ADC_MEASUREMENT.c`.

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.

```c
#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(&ADC_MEASUREMENT_Channel_A);
        if((bool)(result >> VADC_G_RES_VF_Pos))
        {
            valid_result = (bool)true;
            detailed_result.channel_num = (result & VADC_G_RES_CHNR_Msk) >> VADC_G_RES_CHNR_Pos;
            detailed_result.group_num = ADC_MEASUREMENT_Channel_A.group_index;
            detailed_result.conversion_result = result & VADC_G_RES_RESULT_Msk;
        }
    #endif
}

int main(void)
{
    DAVE_Init();
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_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_MEASUREMENT_t

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 microcontrollers where group result registers are not available. Hence for these microcontroller 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_MEASUREMENT_CHANNEL_t 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(&ADC_MEASUREMENT_0);
    #endif
    if((bool)(result >> VADC_GLOBRES_VF_Pos))
```
{ 
    valid_result = (bool)true;
    detailed_result[index].channel_num = (result & VADC_GLOBRES_CHNR_Msk) >> VADC_GLOBRES_CHNR_Pos;
    detailed_result[index].group_num = ADC_MEASUREMENT_Channel_A.group_index;
    detailed_result[index].conversion_result = (result & VADC_GLOBRES_RESULT_Msk) >> ((uint32_t)ADC_MEASUREMENT_0.iclass_config_handle->conversion_mode_standard * (uint32_t)2);
    index++;
}

int main(void)
{
    DAVE_Init();
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_0);
    while(1);
    return 0;
}

Definition at line 283 of file ADC_MEASUREMENT.c.

References ADC_MEASUREMENT_GetGlobalDetailedResult().

__STATIC_INLINE uint32_t ADC_MEASUREMENT_GetGlobalDetailedResult()

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 microcontrollers where group result registers are not available. Hence for these microcontroller 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_MEASUREMENT_CHANNEL_t *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
{

```c
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_GetGlobalDetailedResult();
#endif
    if((bool)(result >> VADC_GLOBRES_VF_Pos))
    {
        valid_result = (bool)true;
        detailed_result[index].channel_num = (result & VADC_GLOBRES_CHNR_Msk) >> VADC_GLOBRES_CHNR_Pos;
        detailed_result[index].group_num = ADC_MEASUREMENT_Channel_A.group_index;
        detailed_result[index].conversion_result = (result & VADC_GLOBRES_RESULT_Msk) >> ((uint32_t)ADC_MEASUREMENT_0.iclass_config_handle->conversion_mode_standard * (uint32_t)2);
    }
    index++;
}

int main(void)
```
{  
    DAVE_Init();  
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMEN
t_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_MEASUREMEN

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 microcontrollers where group result registers are not available. Hence for these microcontroller 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.

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

XMC_VADC_RESULT_SIZE_t result;
void Adc_Measurement_Handler()
{
    if(UC_SERIES == XMC11)
        result = ADC_MEASUREMENT_GetGlobalResult();
    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_MEASUREMENT_0);
    while(1);
    return 0;
}
```

Definition at line 614 of file `ADC_MEASUREMENT.h`. 
XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_CHANNEL_t handle_ptr);

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 microcontrollers 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 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 has been enabled
#include <DAVE.h>

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

int main(void)
{
  DAVE_Init();
  ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_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 ( handle_ptr)

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 microcontrollers where group result registers are not available. Hence for these microcontroller 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.

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

XMC_VADC_RESULT_SIZE_t result;
void Adc_Measurement_Handler()
{
    #if (UC_SERIES == XMC11)
        result = ADC_MEASUREMENT_GetResult(&ADC_MEASUREMENT_0);
    ```
```c
#ifndef
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_MEASUREMENT_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
- ADC_MEASUREMENT_STATUS_FAILURE otherwise

**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 [BRSEL]. 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.

```c
#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::backgd_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,
void ADC_MEASUREMENT_StartConversion (ADC_MEASUREMENT_t handle_ptr)

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 has been enabled
#include <DAVE.h>

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


result = \texttt{ADC\_MEASUREMENT\_GetResult}(&ADC\_MEASUREMENT\_Channel\_A);
\textbf{else}
result = \texttt{ADC\_MEASUREMENT\_GetGlobalResult}();
\textbf{endif}

\textbf{int} main(\textbf{void})
{
    \texttt{DAVE\_Init}();
    \texttt{ADC\_MEASUREMENT\_StartConversion}(&ADC\_MEASUREMENT\_0);
    \textbf{while}(1);
    \textbf{return} 0;
}

Definition at line 227 of file \texttt{ADC\_MEASUREMENT.c}.

References \texttt{ADC\_MEASUREMENT::global\_handle}. 
## 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.

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
   \textbf{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.

\textbf{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.

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

\textbf{Sample Application (main.c)}

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

XMC_VADC_RESULT_SIZE_t result;
void Adc_Measurement_Handler()
```
{  
#if(UC_SERIES != XMC11)
  result = ADC_MEASUREMENT_GetResult(&ADC_MEASUREMENT_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 while loop below can be replaced with an user error handler */
    XMC_DEBUG(("DAVE Apps initialization failed with status %d\n", status));
    while(1U)
    {
    }
  }
}
ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMET_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_CALOC0 = SHS_CALLOC0_CLEAR_OFFSET;
\*SHS0_CALOC1 = 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 conversion*/
  VADC->BRSSEL[0] = VADC_BRSSEL_CHSELG0_Msk;

  /*Clear the DPCAL0, DPCAL1 and SUCAL bits*/
  VADC->GLOBCFG &= ~(VADC_GLOBCFG_DPCAL0_Msk | VADC_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)
adc_result_aux = VADC->GLOBRES;

/* Clear offset calibration values*/
CLEAR_OFFSET_CALIB_VALUES;

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

/* Wait until last gain calibration step is finished */
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 | VADC_GLOBCFG_DPCAL1_Msk);
VADC->BRSMR = 0x00;
VADC->BRSEL[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 1920 cycles 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

Release History

Release History
Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>Data Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_MEASUREMENT</td>
</tr>
<tr>
<td>ADC_MEASUREMENT_CHANNEL</td>
</tr>
<tr>
<td>ADC_MEASUREMENT_CHANNEL_ARRAY</td>
</tr>
<tr>
<td>ADC_MEASUREMENT_ISR</td>
</tr>
</tbody>
</table>
ADC_MEASUREMENT

<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>const ADC_MEASUREMENT_CHANNEL_ARRAY t *const array</td>
<td></td>
</tr>
<tr>
<td>const XMC_VADC_BACKGROUND_CONFIG t *const backgnd_config_handle</td>
<td></td>
</tr>
<tr>
<td>const XMC_VADC_GLOBAL_CLASS t *const iclass_config_handle</td>
<td></td>
</tr>
<tr>
<td>const GLOBAL_ADC_t *const global_handle</td>
<td></td>
</tr>
<tr>
<td>const ADC_MEASUREMENT_ISR t *const req_src_intr_handle</td>
<td></td>
</tr>
<tr>
<td>const ADC_MEASUREMENT_ISR t *const result_intr_handle</td>
<td></td>
</tr>
<tr>
<td>ADC_MEASUREMENT_MUX_CONFIG t mux_config</td>
<td></td>
</tr>
<tr>
<td>ADC_MEASUREMENT_STATUS t init_state</td>
<td></td>
</tr>
<tr>
<td>const XMC_VADC_SR t srv_req_node</td>
<td></td>
</tr>
<tr>
<td>const bool start_conversion</td>
<td></td>
</tr>
</tbody>
</table>
Field Documentation

**const ADC_MEASUREMENT_CHANNEL_ARRAY_t* const ADC_MEASUREMENT::array**

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_MEASUREMENT::backgnd_config_handle**

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_MEASUREMENT::iclass_config_handle**

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::mux_config

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::req_src_intr_handle

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::result_intr_handle

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

## 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMC_VADC_CHANNEL_CONFIG_t *</td>
<td>ch_handle</td>
</tr>
<tr>
<td>XMC_VADC_RESULT_CONFIG_t *</td>
<td>res_handle</td>
</tr>
<tr>
<td>XMC_VADC_GROUP_t *</td>
<td>group_handle</td>
</tr>
<tr>
<td>ANALOG_IO_t *</td>
<td>analog_io_config</td>
</tr>
<tr>
<td>uint8_t</td>
<td>group_index</td>
</tr>
<tr>
<td>uint8_t</td>
<td>ch_num</td>
</tr>
</tbody>
</table>
Field Documentation

**ANALOG_IO_t* ADC_MEASUREMENT_CHANNEL::analog_io_config**

This holds 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_CHANNEL::ch_handle**

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_handle**

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::res_handle**

This holds 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`
<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
<th>ADC_MEASUREMENT_CHANNEL_ARRAY Struct Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_MEASUREMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>const $\text{ADC}<em>\text{MEASUREMENT}</em>{\text{CHANNEL}_\text{t}}$</th>
<th>channel_array $[\text{ADC}<em>\text{MEASUREMENT}</em>{\text{MAXC}}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{XMC}<em>\text{VADC}</em>{\text{RESULT}<em>{\text{CONFIG}}</em>\text{t}}$ *</td>
<td>res_handle</td>
</tr>
</tbody>
</table>
Field Documentation

const ADC_MEASUREMENT_CHANNEL_t* const ADC_MEASUREMENT_CHANNEL_ARRAY::channel_array

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_ARRAY::res_handle

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

<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
</tr>
</thead>
</table>

## 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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>node_id</td>
</tr>
<tr>
<td>uint32_t</td>
<td>priority</td>
</tr>
<tr>
<td>uint32_t</td>
<td>sub_priority</td>
</tr>
<tr>
<td>uint8_t</td>
<td>irqctrl</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>ADC_MEASUREMENT</th>
<th>ADC_MEASUREMENT_CHANNEL</th>
<th>ADC_MEASUREMENT_CHANNEL_ARRAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th></th>
<th></th>
</tr>
</thead>
</table>
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]
<table>
<thead>
<tr>
<th>Home</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>Globals</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC_MEASUREMENT.c File Reference</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td><code>ADC_MEASUREMENT_GetAppVersion(void)</code></td>
<td>Get <code>ADC_MEASUREMENT</code> APP version.</td>
</tr>
<tr>
<td>ADC_MEASUREMENT_STATUS_t</td>
<td><code>ADC_MEASUREMENT_Init(ADC_MEASUREMENT_t *const handle_ptr)</code></td>
<td>Initializes the APP to measure a set of analog inputs.</td>
</tr>
<tr>
<td></td>
<td><code>ADC_MEASUREMENT_StartConversion(ADC_MEASUREMENT_t *const handle_ptr)</code></td>
<td>Starts the conversion of the required measurements.</td>
</tr>
<tr>
<td>XMC_VADC_RESULT_SIZE_t</td>
<td><code>ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr)</code></td>
<td>Returns the converted value for a specific channel. Not Applicable for XMC1100.</td>
</tr>
<tr>
<td>uint32_t</td>
<td><code>ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr)</code></td>
<td>Returns a detailed conversion result. Only Applicable for XMC1100.</td>
</tr>
<tr>
<td>XMC_VADC_RESULT_SIZE_t</td>
<td><code>ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_t *const handle_ptr)</code></td>
<td>Returns the converted value from the global result register. Only Applicable for XMC1100.</td>
</tr>
</tbody>
</table>
uint32_t ADC_MEASUREMENT_GetDetailedResult (ADC_MEASUREMENT_t *const handle_ptr)
Returns a detailed conversion result.
Applicable for XMC1100.
Function Documentation

uint32_t ADC_MEASUREMENT_GetDetailedResult (ADC_MEASUREMENT_CHANNEL_t handle_ptr)

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_MEASUREMENT_t *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(&ADC_MEASUREMENT_Channel_A);
    if((bool)(result >> VADC_G_RES_VF_Pos))
    {
        valid_result = (bool)true;
        detailed_result.channel_num = (result & VADC_G_RES_CHNR_Msk) >> VADC_G_RES_CHNR_Pos;
        detailed_result.group_num = ADC_MEASUREMENT_Channel_A.group_index;
        detailed_result.conversion_result = result & VADC_G_RES_RESULT_Msk;
    }
    #endif
}

int main(void)
{
    DAVE_Init();
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_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.

```c
uint32_t ADC_MEASUREMENT_GetDetailedResult (ADC_MEASUREMENT_t
```

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 microcontrollers where group result registers are not available. Hence for these microcontroller 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_MEASUREMENT_CHANNEL_t *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.

```c
#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(&ADC_MEASUREMENT_0);
    #endif

    if((bool)(result >> VADC_GLOBRES_VF_Pos))
    {
        valid_result = (bool)true;
        detailed_result[index].channel_num = (result & VADC_GLOBRES_CHNR_Msk) >> VADC_GLOBRES_CHNR_Pos;
    }
```
detailed_result[index].group_num = ADC_MEASUREMENT_Channel_A.group_index;

detailed_result[index].conversion_result = (result & VADC_GLOBRES_RESULT_Msk) >>
((uint32_t)ADC_MEASUREMENT_0.iclass_config_handle->conversion_mode_standard * (uint32_t)2);

index++;

int main(void)
{
    DAVE_Init();
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_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 (AI

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 microcontrollers 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.

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

XMC_VADC_RESULT_SIZE_t result;
void Adc_Measurement_Handler()
{
    #if(UC_SERIES != XMC11)
        result = ADC_MEASUREMENT_GetResult(&ADC_MEASUREMENT_Channel_A);
    #endif
}
```
int main(void)
{
    DAVE_Init();
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_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 (AI

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 microcontrollers where group result registers are not available. Hence for these microcontroller 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.

```c
#include <DAVE.h>

XMC_VADC_RESULT_SIZE_t result;
void Adc_Measurement_Handler()
{
  #if(UC_SERIES == XMC11)
    result = ADC_MEASUREMENT_GetResult(&ADC_MEASUREMENT_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_MEASUREMENT_0);
  while(1);
  return 0;
}
```
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.

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

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.

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

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

int main(void)
{
    DAVE_Init();
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_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.
<table>
<thead>
<tr>
<th>Home</th>
<th>File List</th>
<th>Globals</th>
<th>Data Structures</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>struct</th>
<th>ADC_MEASUREMENT_ISR</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>ADC_MEASUREMENT_CHANNEL</td>
</tr>
<tr>
<td>struct</td>
<td>ADC_MEASUREMENT_CHANNEL_ARRAY</td>
</tr>
<tr>
<td>struct</td>
<td>ADC_MEASUREMENT</td>
</tr>
</tbody>
</table>
## Typedefs

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef void(*) ADC_MEASUREMENT_(void)</code></td>
<td></td>
</tr>
<tr>
<td><code>typedef struct ADC_MEASUREMENT_ISR</code></td>
<td>ADC_MEASUREMENT_</td>
</tr>
<tr>
<td><code>typedef struct ADC_MEASUREMENT_CHANNEL</code></td>
<td>ADC_MEASUREMENT_</td>
</tr>
<tr>
<td><code>typedef struct ADC_MEASUREMENT_CHANNEL_ARRAY</code></td>
<td>ADC_MEASUREMENT_</td>
</tr>
<tr>
<td><code>typedef struct ADC_MEASEUREMENT</code></td>
<td>ADC_MEASUREMENT_</td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>ADC_MEASUREMENT_GetAppVersion</td>
<td>Initializes the APP to measure a set of analog inputs.</td>
</tr>
<tr>
<td>ADC_MEASUREMENT_STATUS_t</td>
<td>void</td>
<td>ADC_MEASUREMENT_Init(ADC_MEASUREMENT_t) Starts the conversion of the required measurements.</td>
</tr>
<tr>
<td>XMC_VADC_RESULT_SIZE_t</td>
<td>ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_CHANNEL_t handle_ptr)</td>
<td>Returns the converted value for a specific channel. Not Applicable for XMC1100.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT CHANNEL_t handle_ptr)</td>
<td>Returns a detailed conversion result. Not Applicable for XMC1100. Applicable for XMC1100.</td>
</tr>
<tr>
<td>XMC_VADC_RESULT_SIZE_t</td>
<td>ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_DEPRECATED)</td>
<td>Returns the converted value from the global result register. Only Applicable for XMC1100.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_DEPRECATED)</td>
<td>Returns a detailed conversion result. Only Applicable for XMC1100.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>ADC_MEASUREMENT_GetGlobalResult</td>
<td>Returns the converted value from the global result register. Only Applicable for XMC1100.</td>
<td></td>
</tr>
<tr>
<td>ADC_MEASUREMENT_GetGlobalDetailedResult</td>
<td>Returns a detailed conversion result. Only Applicable for XMC1100.</td>
<td></td>
</tr>
<tr>
<td>ADC_MEASUREMENT_GetGlobalResult</td>
<td>Returns the converted value from the global result register. Only Applicable for XMC1100.</td>
<td></td>
</tr>
</tbody>
</table>

```c
__STATIC_INLINE XMC_VADC_RESULT_SIZE_t
ADC_MEASUREMENT_GetGlobalResult

__STATIC_INLINE uint32_t
ADC_MEASUREMENT_GetGlobalDetailedResult(void)
```

```c
typedef enum ADC_MEASUREMENT_STATUS
```

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>Return value of an API.</td>
</tr>
<tr>
<td>FAILURE</td>
<td>Return value of an API.</td>
</tr>
<tr>
<td>UNINITIALIZED</td>
<td>Return value of an API.</td>
</tr>
</tbody>
</table>

Go to the source code of this file.
ADC_MEASUREMENT

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

<table>
<thead>
<tr>
<th>Home</th>
<th>File List</th>
<th>Globals</th>
<th>All</th>
<th>Functions</th>
<th>Typedefs</th>
<th>Enumerations</th>
<th>Enumerator</th>
</tr>
</thead>
</table>

- 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_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
/*CODE_BLOCK_BEGIN*/
#ifndef ADC_MEASUREMENT_H
#define ADC_MEASUREMENT_H

/*******************************************
**************************************************
**************************
* HEADER FILES
*******************************************
**************************************************
**************************
#include "GLOBAL_ADC/global_adc.h"
#include "adc_measurement_conf.h"

/******************************************
**************************************************
**************************
* MACROS
**************************************************
**************************
#if (!((XMC_LIB_MAJOR_VERSION == 2U) &&
((XMC_LIB_MINOR_VERSION >= 0U) &&
(XMC_LIB_PATCH_VERSION >= 0U))))
#error "ADC_MEASUREMENT requires XMC Peripheral Library v2.0.0 or higher"

```c
#include <stdio.h>
```
typedef struct ADC_MEASUREMENT_CHANNEL
{
    #if( XMC_VADC_CHANNEL_CONFIG_CONFIGABLE == 1U)
    XMC_VADC_CHANNEL_CONFIG_CONFIGABLE_t *ch_handle;
    #endif

    #if( XMC_VADC_RESULT_CONFIG_CONFIGABLE == 1U)
    XMC_VADC_RESULT_CONFIG_CONFIGABLE_t *res_handle;
    #endif

    #endif

    #if( XMC_VADC_GROUP_AVAILABLE == 1U)
    XMC_VADC_GROUP t *group_handle;
    #endif

    #endif

    #endif

    #ifdef ADC_MEASUREMENT_ANALOG_IO_USED
    ANALOG_IO_t *analog_io_config;
    #endif

    uint8_t group_index;

    uint8_t ch_num;

} ADC_MEASUREMENT_CHANNEL_t;

typedef struct ADC_MEASUREMENT_CHANNEL_ARRAY
{
    const ADC_MEASUREMENT_CHANNEL t *const cha
nnnel_array[ADC_MEASUREMENT_MAXCHANNELS];
#if( XMC_VADC_GROUP_AVAILABLE == 0U)
    XMC_VADC_RESULT_CONFIG_t *res_handle;
#endif
} ADC_MEASUREMENT_CHANNEL_ARRAY_t;

typedef struct ADC_MEASUREMENT
{
    const ADC_MEASUREMENT_CHANNEL_ARRAY_t *const array;
    const XMC_VADC_BACKGROUND_CONFIG_t *const backgnd_config_handle;
    const XMC_VADC_GLOBAL_CLASS_t *const iclas_config_handle;
    GLOBAL_ADC_t *const global_handle;
    #if (UC_SERIES != XMC11)
    const ADC_MEASUREMENT_ISR_t *const req_src_intr_handle;
    #else
    const ADC_MEASUREMENT_ISR_t *const result_intr_handle;
    #endif
    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;
} ADC_MEASUREMENT_t;

#endif __cplusplus
extern "C" {
#endif

/*****************************/
*/
DAVE_APP_VERSION_t ADC_MEASUREMENT_GetAppVersion(void);
ADC_MEASUREMENT_STATUS_t ADC_MEASUREMENT_Init(ADC_MEASUREMENT_t *const handle_ptr);
void ADC_MEASUREMENT_StartConversion(ADC_MEASUREMENT_t *const handle_ptr);

#if(XMC_VADC_GROUP_AVAILABLE == 1U)
XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr);
uint32_t ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr);
#else /* Applicable for XMC1100 devices*/
XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_t *const handle_ptr)	ADC_MEASUREMENT_DEPRECATED;
uint32_t ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_t *const handle_ptr)	ADC_MEASUREMENT_DEPRECATED;
#endif

__STATIC_INLINE XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetGlobalResult(void) {
	XMC_VADC_RESULT_SIZE_t result;
result = XMC_VADC_GLOBAL_GetDetailedResult(ADC_MEASUREMENT_MODULE_PTR);
return (result);

__STATIC_INLINE uint32_t ADC_MEASUREMENT_GetGlobalDetailedResult(void) {
    uint32_t result;
    result = XMC_VADC_GLOBAL_GetDetailedResult(ADC_MEASUREMENT_MODULE_PTR);
    return (result);
}

#include "ADC_MEASUREMENT_Extern.h"
#ifdef __cplusplus
}
#endif
#endif /* ADC_MEASUREMENT_H_ */
Go to the documentation of this file.

00001
00089  /******************************************************************************
00090  ****************************************************************************/
00091  /******************************************************************************
00092  * HEADER FILES
00093  ****************************************************************************/
00094  /******************************************************************************
00095  * MACROS
00096  ****************************************************************************/
00097  ******************************************************************************
00098  * LOCAL DATA
00099  ****************************************************************************/
00100  ******************************************************************************
00101  
00102  #include "adc_measurement.h"
/*This function returns the version of the ADC_MEASUREMENT App*/

DAVE_APP_VERSION_t ADC_MEASUREMENT_GetAppVersion(void)
{
    DAVE_APP_VERSION_t version;

    version.major = (uint8_t) ADC_MEASUREMENT_MAJOR_VERSION;
    version.minor = (uint8_t) ADC_MEASUREMENT_MINOR_VERSION;
    version.patch = (uint8_t) ADC_MEASUREMENT_PATCH_VERSION;

    return version;
}

/* Initialization routine to call ADC LLD API's */
ADC_MEASUREMENT_STATUS_t ADC_MEASUREMENT_Init(ADC_MEASUREMENT_t *const handle_ptr)
{
    const ADC_MEASUREMENT_CHANNEL_t *indexed;
    uint8_t j;
    ADC_MEASUREMENT_STATUS_t status;

    XMC_ASSERT("ADC_MEASUREMENT_Init:Invalid handle_ptr", (handle_ptr != NULL))

    if (ADC_MEASUREMENT_STATUS_UNINITIALIZED == handle_ptr->init_state) {
        /* Call the function to initialise Clock and ADC global functional units*/
        status = (ADC_MEASUREMENT_STATUS_t) GLOBAL_ADC_Init(handle_ptr->global_handle);

        /*Initialize the Global Conversion class 0*/
        XMC_VADC_GLOBAL_InputClassInit(handle_ptr->global_handle->module_ptr,*handle_ptr->iclass_config_handle,
                                     XMC_VADC_GROUP_CONV_STD,ADC_MEASUREMENT_ICLASS_NUM);

        if (UC_SERIES == XMC11)
            /*Initialize the Global Conversion class 1*/
            XMC_VADC_GLOBAL_InputClassInit(handle_ptr->global_handle->module_ptr,*handle_ptr->iclass_config_handle,
                                            XMC_VADC_GROUP_CONV_STD,ADC_MEASUREMENT_ICLASS_NUM_XMC11);
    }

    #endif
}

/* Initialize the Background Scan hardware */
XMC_VADC_GLOBAL_BackgroundInit(handle_ptr->global_handle->module_ptr, handle_ptr->backgnd_config_handle);

#if (XMC_VADC_GROUP_AVAILABLE == 0U)
/* Initialize the global result register */
XMC_VADC_GLOBAL_ResultInit(handle_ptr->global_handle->module_ptr, handle_ptr->array->res_handle);
#endif

for (j = (uint8_t)0; j < (uint8_t)ADC_MEASUREMENT_MAXCHANNELS; j++)
{
    indexed = handle_ptr->array->channel_array[j];
    #if (XMC_VADC_GROUP_AVAILABLE == 1U)
    /* Initialize for configured channels */
    XMC_VADC_GROUP_ChannelInit(indexed->group_handle, (uint32_t)indexed->ch_num, indexed->ch_handle);
    /* Initialize for configured result registers */
    XMC_VADC_GROUP_ResultInit(indexed->group_handle, (uint32_t)indexed->ch_handle->result_reg_number, indexed->res_handle);
    #endif
    /* Add all channels into the Background Request Source Channel Select Register */
    XMC_VADC_GLOBAL_BackgroundAddChannelTo
Sequence(handle_ptr->global_handle->module_ptr,
        (uint32_t)indexed->group_index, (uint32_t)indexed->ch_num);

#ifdef ADC_MEASUREMENT_ANALOG_IO_USED
  /* ANALOG_IO initialization for the channel*/
  if (indexed->analog_io_config != NULL)
    {
      status |= (ADC_MEASUREMENT_STATUS_t)ANALOG_IO_Init(indexed->analog_io_config);
    }
#endif
#if(UC_SERIES != XMC11)
  if ((handle_ptr->backgnd_config_handle->req_src_interrupt) && (handle_ptr->req_src_intr_handle != NULL ))
    {
      #if (UC_FAMILY == XMC1)
        NVIC_SetPriority((IRQn_Type)handle_ptr->req_src_intr_handle->node_id,
                         handle_ptr->req_src_intr_handle->priority);
      #else
        NVIC_SetPriority((IRQn_Type)handle_ptr->req_src_intr_handle->node_id,
                         NVIC_EncodePriority(NVIC_GetPriorityGrouping(),
                         handle_ptr->req_src_intr_handle->priority, handle_ptr->req_src_intr_handle->sub_priority));
      #endif
    }
  /* Connect background Request Source Event to NVIC node */
  XMC_VADC_GLOBAL_BackgroundSetReqSrcEve
ntInterruptNode(handle_ptr->global_handle->module_ptr,
00190
        (XMC_VADC_SR_t) handle_ptr->srv_req_node);
00191
00192 /* Enable Background Scan Request source IRQ */
00193       NVIC_EnableIRQ((IRQn_Type)handle_ptr->
00194       req_src_intr_handle->node_id);
00195 #ifdef ADC_MEASUREMENT_NON_DEFAULT_IRQ_SOURCE_SELECTED
00196       XMC_SCU_SetInterruptControl(handle_ptr->req_src_intr_handle->node_id,
00197       ((handle_ptr->req_src_intr_handle->node_id << 8) | handle_ptr->
00198       req_src_intr_handle->irqctrl));
00199 #endif
00200 }
00201 #else /* Selected device is XMC11*/
00202       XMC_VADC_GLOBAL_SetResultEventInterruptNode(handle_ptr->
00203       global_handle->module_ptr, handle_ptr->
00204       srv_req_node);
00205 #ifdef ADC_MEASUREMENT_CPU_1X /* End of single measurement is enabled*/
00206       NVIC_SetPriority((IRQn_Type)handle_ptr->
00207       result_intr_handle->node_id,
00208       handle_ptr->result_intr_handle->priority);
00209 /* Enable Background Scan Request source IRQ */
00210       NVIC_EnableIRQ((IRQn_Type)handle_ptr->result_intr_handle->node_id);
00211 #endif
00212 #endif
00213 /* Mux Configuration is done*/
if (handle_ptr->mux_config != NULL)
{
    (handle_ptr->mux_config)();
}

if (handle_ptr->start_conversion != (bool)false)
{
    /* Start conversion manually using load event trigger*/
    XMC_VADC_GLOBAL_BackgroundTriggerConversion(handle_ptr->global_handle->module_ptr);
}
handle_ptr->init_state = status;
}
return (handle_ptr->init_state);

/* This API will Software trigger ADC Background request source and starts conversion*/
void ADC_MEASUREMENT_StartConversion(ADC_MEASUREMENT_t *const handle_ptr)
{
    XMC_ASSERT("ADC_MEASUREMENT_Start:Invalid handle_ptr", (handle_ptr != NULL))
    /* Generate a load event to start background request source conversion*/
    XMC_VADC_GLOBAL_BackgroundTriggerConversion(handle_ptr->global_handle->module_ptr);
}

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~*/
#if(XMC_VADC_GROUP_AVAILABLE == 1U)

/* This API will get the result of a conversion for a specific channel*/

XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr)
{
    XMC_VADC_RESULT_SIZE_t result;

    XMC_ASSERT("ADC_MEASUREMENT_GetResult:Invalid handle_ptr", (handle_ptr != NULL))

    result = XMC_VADC_GROUP_GetResult(handle_ptr->group_handle, handle_ptr->ch_handle->result_reg_number);

    return (result);
}

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~*/

/* This API will get the result of a conversion for a specific channel. It will return the complete result register*/

uint32_t ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_CHANNEL_t *const handle_ptr)
{
    uint32_t result;

    XMC_ASSERT("ADC_MEASUREMENT_GetDetailedResult:Invalid handle_ptr", (handle_ptr != NULL))

    result = XMC_VADC_GROUP_GetDetailedResult(handle_ptr->group_handle, handle_ptr->ch_handle->result_reg_number);

    return (result);
#else /* Applicable for XMC1100 devices */
/* This API will get the result of the conversion from the global result */
/* This API has been deprecated. Use ADC_MEASUREMENT_GetGlobalResult() to get the global result. */

XMC_VADC_RESULT_SIZE_t ADC_MEASUREMENT_GetResult(ADC_MEASUREMENT_t *const handle_ptr)
{
    XMC_VADC_RESULT_SIZE_t result;
    XMC_ASSERT("ADC_MEASUREMENT_GetResult:Invalid handle_ptr", (handle_ptr != NULL))
    XMC_UNUSED_ARG(handle_ptr);
    result = ADC_MEASUREMENT_GetGlobalResult();
    return (result);
}

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~*/
/* This API will get the result of a conversion for a specific channel. It will return the global result register */
/* This API has been deprecated. Use ADC_MEASUREMENT_GetGlobalDetailedResult() to get the global result. */

uint32_t ADC_MEASUREMENT_GetDetailedResult(ADC_MEASUREMENT_t *const handle_ptr)
DC_MEASUREMENT_t *const handle_ptr)  
00284  {
00285  uint32_t result;
00286
00287  XMC_ASSERT("ADC_MEASUREMENT_GetDetailedResult: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_GetGlobalDetailedResult();
00293  return (result);
00294  }
00295  #endif