

ADC_MEASUREMENT_ADV

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_ADV

Home

License Terms and Copyright Information

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).

ADC_MEASUREMENT_ADV

Home

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_ADV

Home

Overview

Overview

Scope: This APP measures multiple analog signals using the Versatile Analog to Digital Converter (VADC) peripheral. It provides advanced features such as sequencing, post processing, synchronized measurements and hardware boundary check. Some configurations are set in a required APP (ADC_SCAN or ADC_QUEUE). These configurations are read only in this APP. The features in detailed are:

1. Up to 8 analog channels (more with sync. conversion)
2. Measures a linear sequence of analog inputs (Scan request source).
3. Measures a flexible sequence of analog inputs (Queue request source).
4. Supports internal and external consumption of the request sources.
5. Software and hardware controlled start of measurements.
6. Results post processing with FIFO and Filter.
7. Synchronously channels measurement.
8. Boundary configurations for channels
9. Supports events (Request source event, Result event, Channel event, boundary flag).

Scope delimitation:

To avoid cross configuration, the background request source is not in the scope of this APP. This feature can be added without cross configuration by adding the ADC_MEASUREMENT APP to the project.

Details of provided functionalities:

1. Number of channels and conversion details:

The number of channels can be configured in the APP GUI "Overview". Also the type of conversion can be selected. The conversion details (Timing, Resolution...) are read from required APP.

2. Conversion sequence:

The channel names configured in the APP GUI "Channel Configuration". Also for each channel the "Wait for read" can be enabled.

◦ Scan sequence:

The scan sequence is a linear sequence starting with the highest enabled channel (CH7 to CH0). The assignment of the "Channel name" to the physical channel CHx is done by assigning the channel resource or by assigning the input pin. 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 = CHANNEL-2
CH_B = CHANNEL-5
CH_C = CHANNEL-6
CH_D = CHANNEL-0
CH_E = CHANNEL-4
CH_F = CHANNEL-1

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

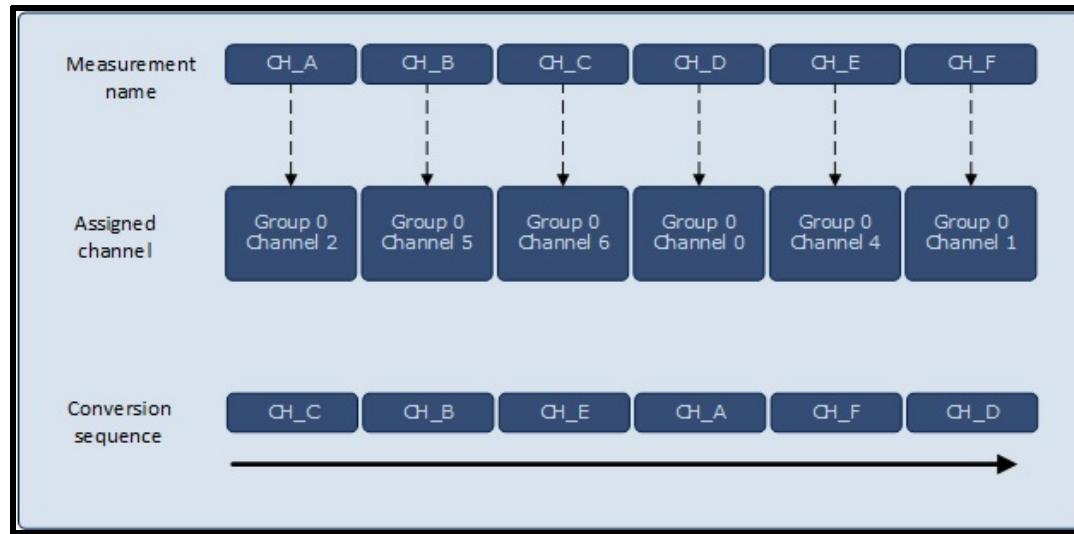


Figure 1 : Scan Conversion Sequence

- **Queue sequence:**

The queue sequence is a flexible 8 stages sequence. Each channel can be placed in any position and also multiple entries of one channel are possible. The TAB "Sequence Plan" in the APP GUI allows configuring the sequence. The assignment of the "Channel name" to the physical channel CHx is done by assigning the channel resource or by assigning the input pin. 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 = CHANNEL-2
 CH_B = CHANNEL-5
 CH_C = CHANNEL-6
 CH_D = CHANNEL-0
 CH_E = CHANNEL-4
 CH_F = CHANNEL-1

And the Queue sequence is selected as follows:

CH_B = CHANNEL-5
CH_B = CHANNEL-5
CH_B = CHANNEL-5

CH_A = CHANNEL-2

CH_C = CHANNEL-6

CH_D = CHANNEL-0

CH_F = CHANNEL-1

CH_E = CHANNEL-4

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

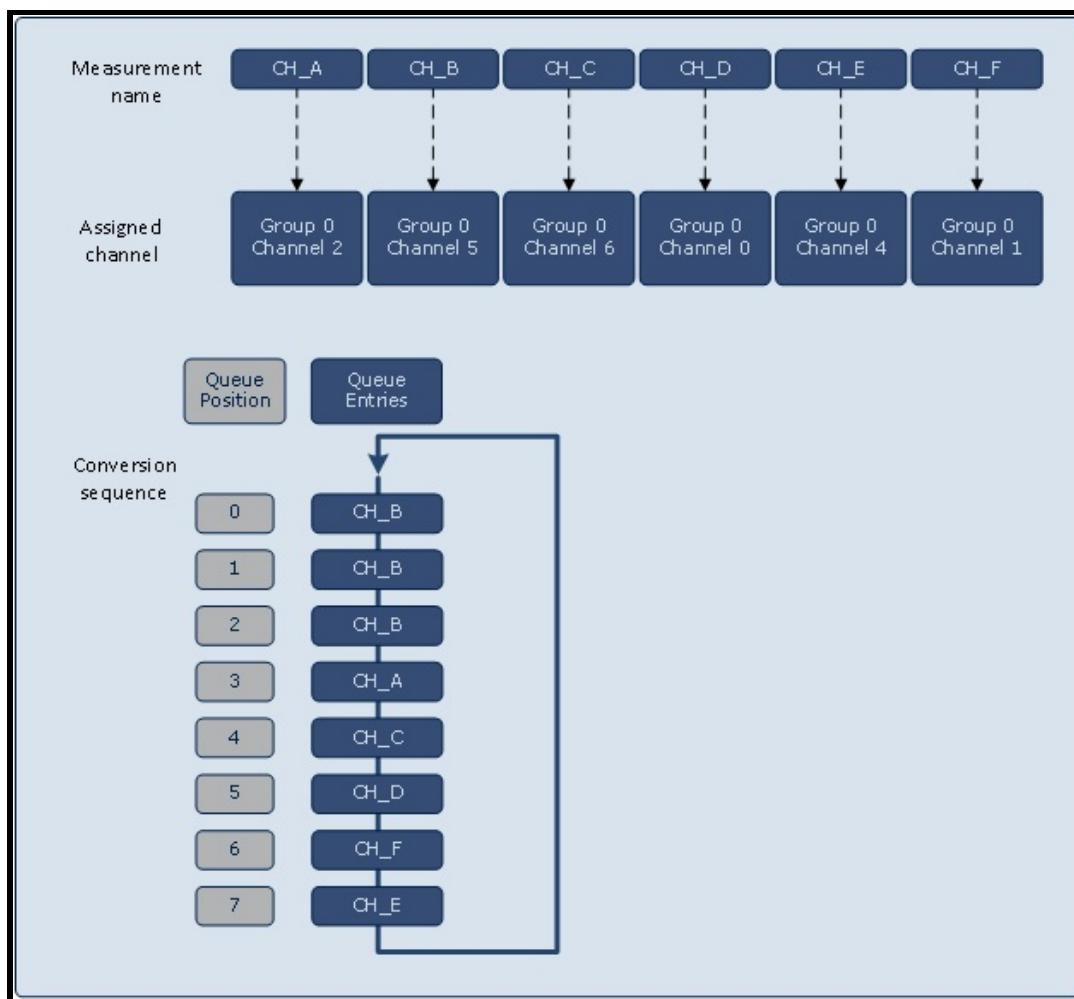


Figure 2 : Queue Conversion Sequence

3. Expose APP:

The APP supports the consumption of the request source inside or outside the APP. This feature shall be selected by selecting the "Expose APP" option in the overview TAB of the APP GUI. If Expose APP is

enabled, the APP consumes the required request source APP either ADC_SCAN or ADC_QUEUE depending on the selection being made. If the expose APP option is disabled then the request source will be consumed inside the APP. Use the "Expose APP" option when multiple use case APPs like **ADC_MEASUREMENT_ADV** and BUCK need to share the request source.

4. Synchronous conversion:

Two or more measurements can be requested at the same time when they share the same trigger. If one of the ADC groups is already converting at this moment this measurement will be delayed compared to the other measurements. In some applications the measurement has to be done completely synchronous. For this case the APP supports the synchronous conversion mode. The synchronous conversion is implemented as master/ slave approach. If one group is a master the following group(s) can be slaves.

To configure a group as sync master the number of slaves need to be selected in the TAB "Sync. Conversion". After this configuration each channel can be configured as master channel. A master channel will allocate the selected amount of slave channels. The master and slave group is different but the channel number is the same. Every time the master channel starts a conversion, the slave channels also start a conversion.

For example G0 is a master group with 1 slave. This makes G1 to a slave group. As soon as "Channel_A" (from G0) is configured as Sync Master and assigned to CH3, G1 CH3 becomes a slave channel. When the master G0 CH3 is requested to convert also the slave G1 CH3 will be converted.

Sync slave channels are named with the <master channel>_slave_<slave token>. Slave channels are exclusive consumed by the APP. The class selection, post processing and boundary configurations are copied from the master to all slaves.

If required, the class configuration can be copied via APP API. This API can overwrite a class configuration if it is configured by another APP.

Note: Ensure that the "Arbitration mode" for all the sync groups is set to "Run Permanently" in GLOBAL_ADC APP.

If 6 measurements are configured in the GUI named as CH_A, CH_B, CH_C, CH_D, CH_E, CH_F slaves groups needed as 3(Group-0 is the master). CH_F(from Group-0) and CH_B(from Group-0) are configured as sync master channels. The channels in the master are using scan request source for conversion and the order of conversion is as shown in the Figure 1. The sync conversion with scan request source is as shown in the following figure.

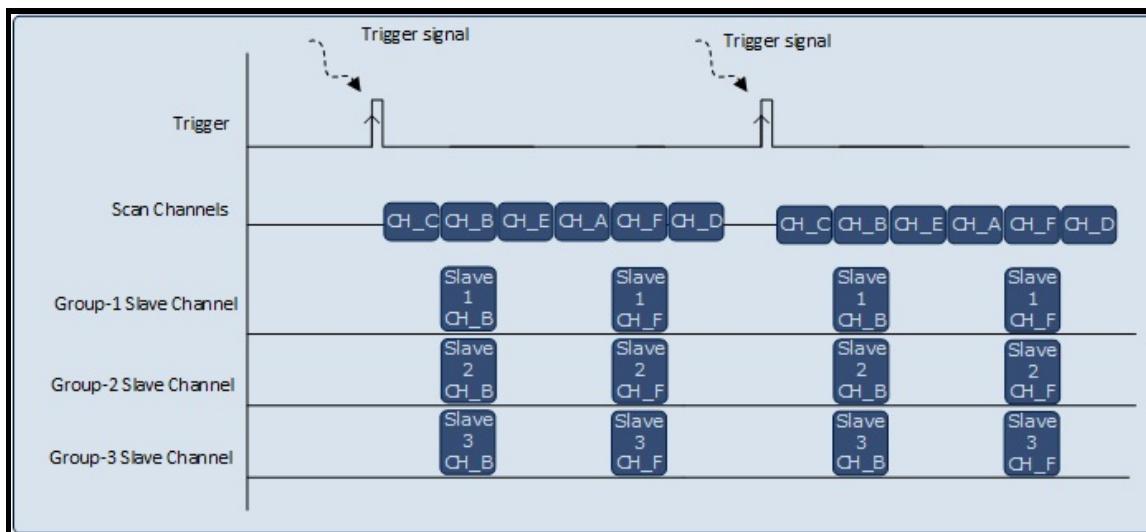


Figure 3 : Sync conversion with scan request source

5. Result processing:

The APP supports result post processing in the hardware. The post processing options can be selected from the APP GUI tab "Post Processing". The options for post processing include the following options:

- FIFO buffer

- This allows up to 16 stage FIFO buffer for the selected channel.
- Result filtering
 - Allows either FIR filter or IIR filter for filtering of conversion results.
- Result accumulation
 - Allows up to 4 accumulations of conversion results.
- Result subtraction
 - Subtracts the conversion result by the value stored in the GxRES0 register. The result register is not allocated when this mode is selection.

6. Limit Checking (Boundary):

The APP supports result monitoring with limit checking.

In normal mode the converted value is automatically checked with two boundary values. If required, a channel event is generated on every result or if the result is inside or outside the boundary.

In fast compare mode the result is checked with a compare value. In this mode a channel event can be generated if the result is lower or higher the compare value or if the value is crossing the compare value.

The fast compare mode can be selected in the required APP ADC_SCAN or ADC_QUEUE. Additional for both modes and up to 4 channels a boundary flag is available.

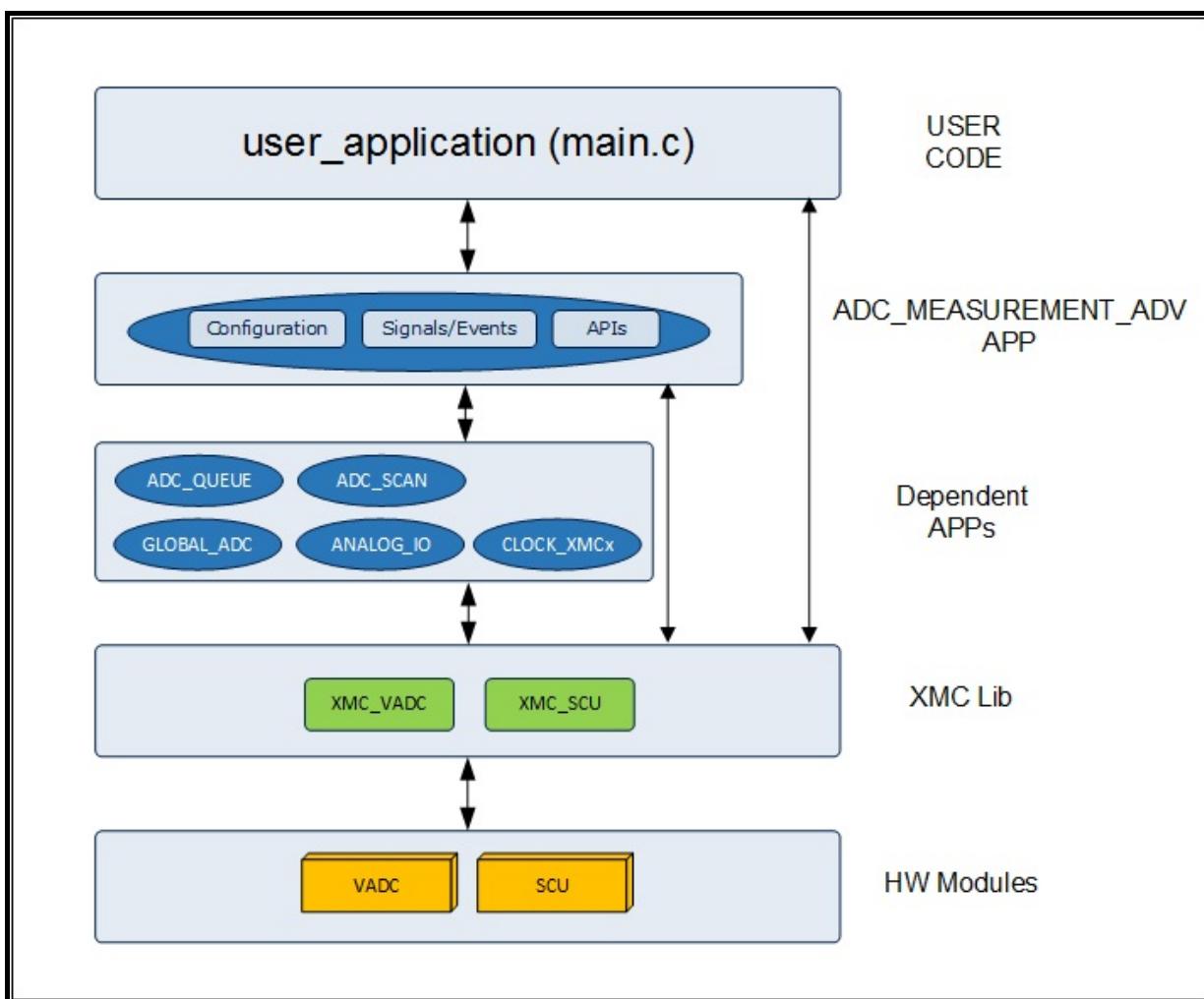
7. Interrupt Notifications:

- Result event:
 - A notification or an event is generated when a new valid result is present in the result register
- Channel event:
 - A notification or an event is generated when either a channel has finished its conversion or the channel's converted result is inside/outside the boundary values.
- Request source interrupt:
 - A interrupt is generated when all the channels are converted

once. For a scan request source the interrupt is generated after the least channel number gets converted. For a queue request source the interrupt is generated after a specific entry finishes its conversion.

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_ADV** APP uses VADC and SCU LLDs and other dependent APPS such as ADC_QUEUE, ADC_SCAN, GLOBAL_ADC ,ANALOG_IO and CLOCK_XMCx for the functional execution.



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

Limitations:

- Channel events are only supported for boundary channels, hence is limited to maximum 4.
- The sync. slave channel events are not available for signal connection.
- For XMC1x devices when alias feature is used, it causes the gain value to be configured for the source channel and not for the alias channel.

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

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
-



ADC_MEASUREMENT_ADV

Home

Architecture Description

Architecture Description

The following diagrams represents the internal software architecture of the **ADC_MEASUREMENT_ADV** APP. The figure shows the consumed hardware resources, dependent APPs and various signals which are exported.

- Figure 1 represents the architecture of the **ADC_MEASUREMENT_ADV** APP with expose APP option enabled in the GUI.
- Figure 2 represents the architecture of the **ADC_MEASUREMENT_ADV** APP with expose APP option disabled in the GUI. This configuration consumes the request source inside **ADC_MEASUREMENT_ADV** itself.

A **ADC_MEASUREMENT_ADV** APP instance exists in a DAVE™ project with fixed attributes as shown and uses the VADC peripheral's scan or queue 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_ADV** APP also provides output signals, these are described in Table-1.

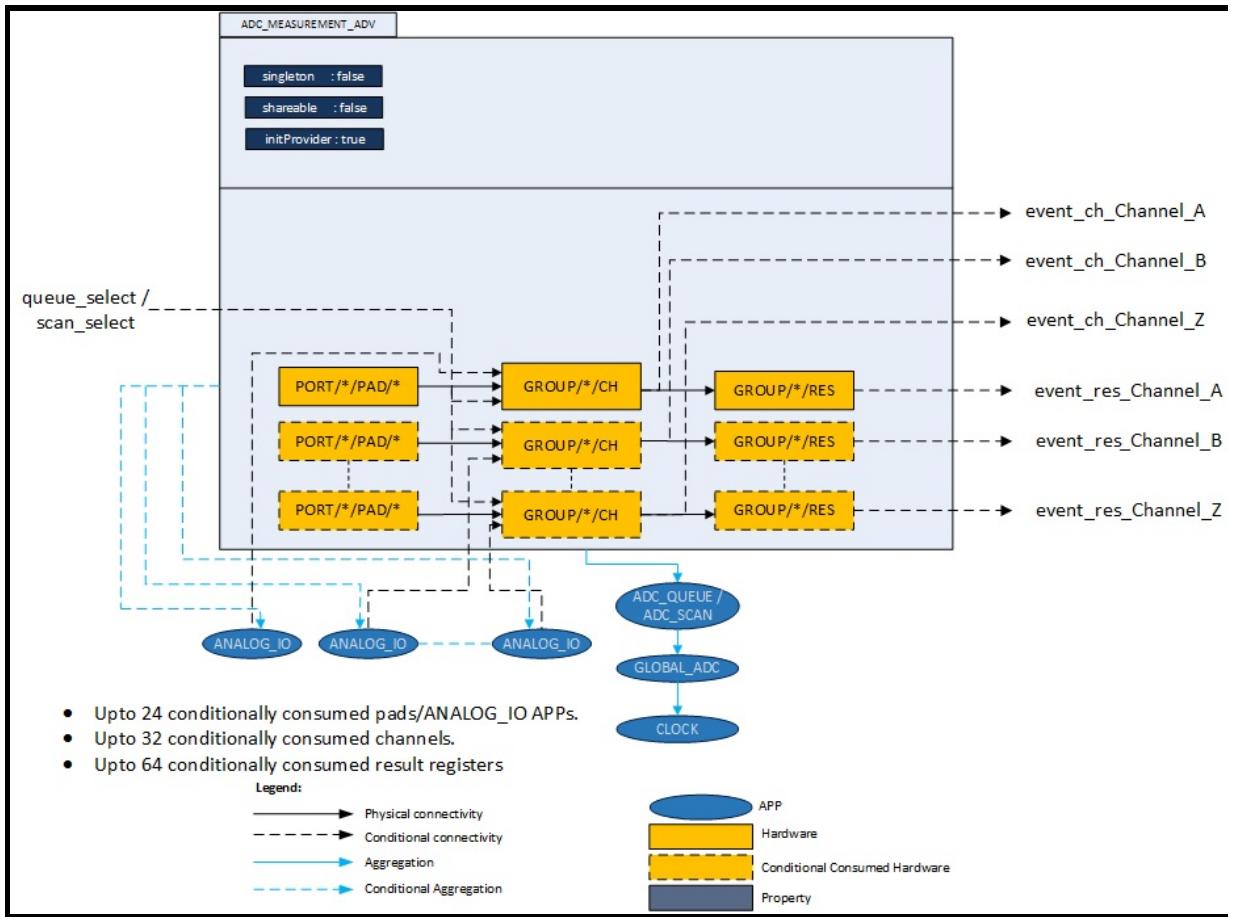


Figure 1 : Architecture of the **ADC_MEASUREMENT_ADV** APP with exposed request source APP.

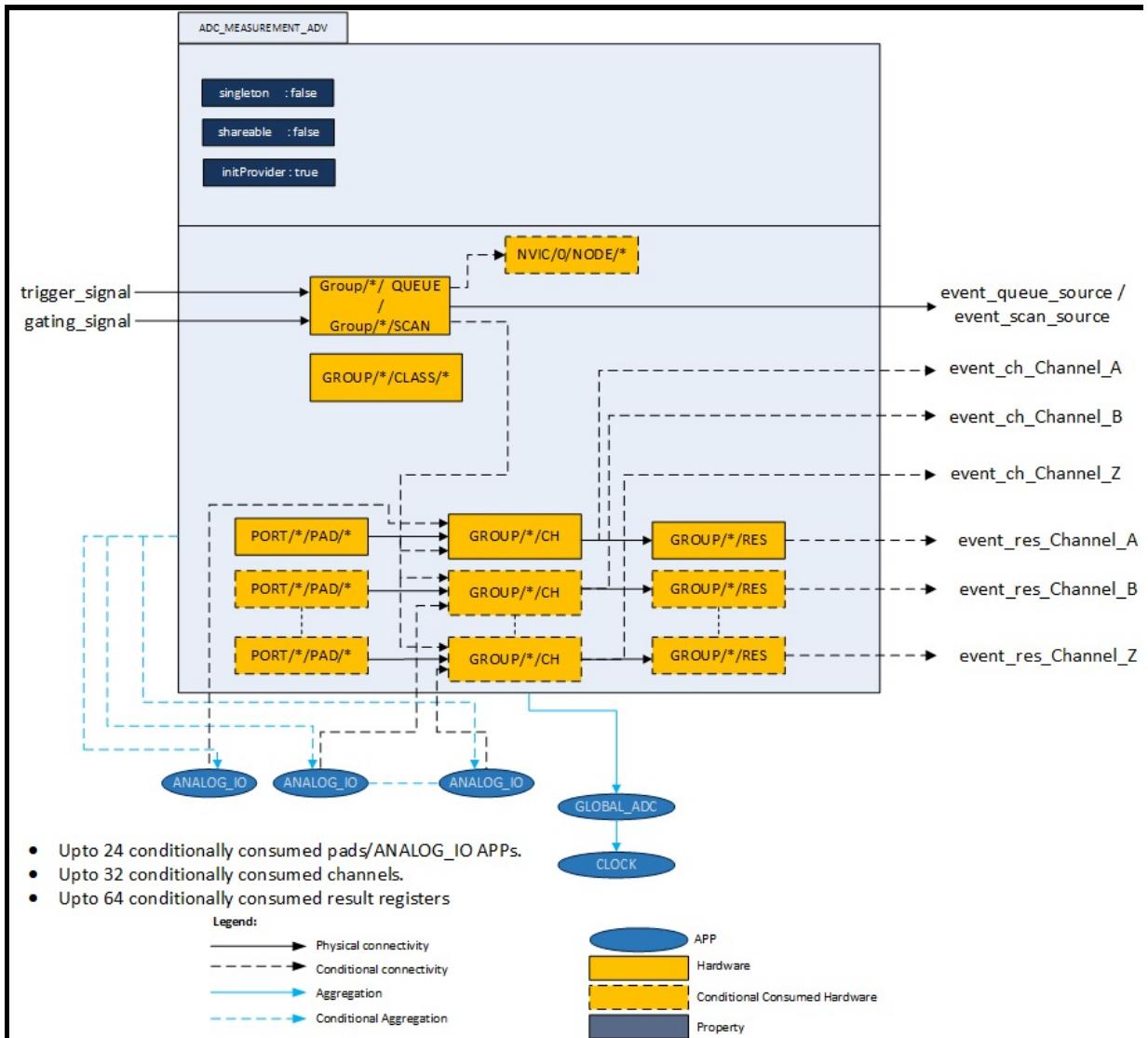


Figure 2 : Architecture of the **ADC_MEASUREMENT_ADV APP** with internally consumed request source.

Following are the features of the **ADC_MEASUREMENT_ADV APP**.

1. **Pin Sharing**
2. **Result Registers**
3. **Interrupt/Event Generation**
4. **Queue Sequencer**
5. **Insert channels**
6. **Sample and Hold Gain Configuration**
7. **Limit checking(Boundary configuration)**

Detailed descriptions are as follows.

1. Pin Sharing:

ANALOG_IO APP is conditionally used by **ADC_MEASUREMENT_ADV** 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_ADV** 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: Two instances of the **ADC_MEASUREMENT_ADV** are present in the project. In **ADC_MEASUREMENT_ADV_0**, 3 channels are used from group-0 and in **ADC_MEASUREMENT_ADV_1** another 3 channels are used from Group-1. The example is taken with reference to XMC4500. The pins and channels distribution are shown in the following figure.

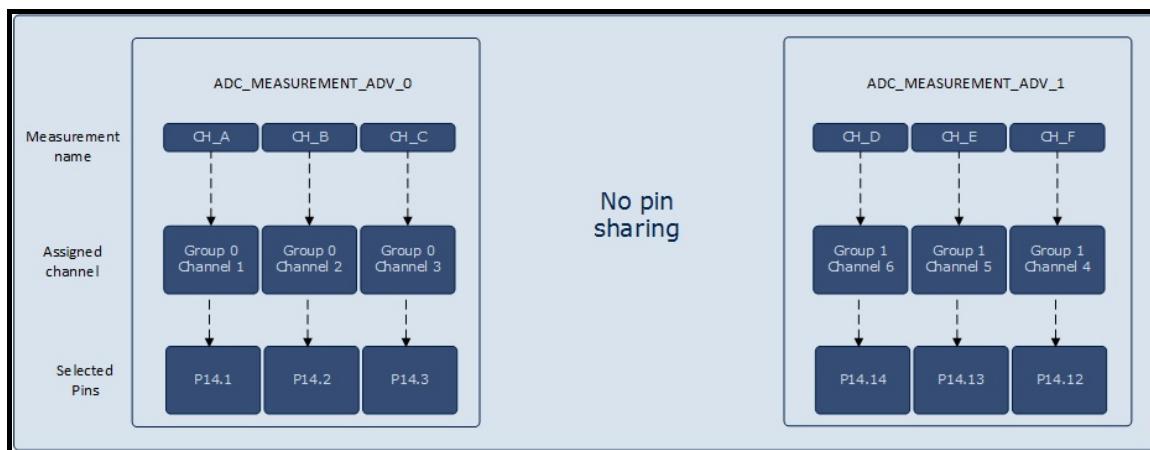


Figure 3 : Pins and channel consumption without ANALOG_IO

The alias feature of the VADC allows the Channels 0 and 1 to convert any pin available in the same group. In XMC4500 the pin P14.7 is only connected to Channel-7 of Group-0. With the alias feature we can use the Group-0 Channel-0 to convert P14.7.

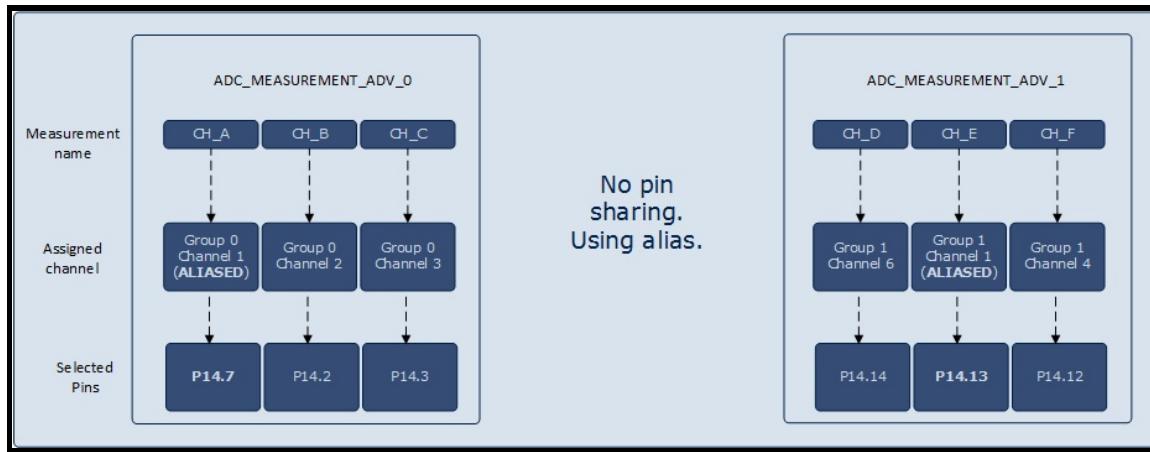


Figure 4 : Pins and channel consumption with alias

The alias feature of the VADC allows the conversion of any given pin of the group by the alias channels Channel-0 or Channel-1. This means that the Channel-0 and Channel-1 can convert not only the pins assigned to it but also the other pins available for the entire group. This feature can be used to convert a pin multiple times. In order to do this a pin must get shared with the other channels as well. This is only possible through the consumption of the ANALOG_IO APP. The following figure depicts how the channel numbers get changed when the same pin is being shared between channels.

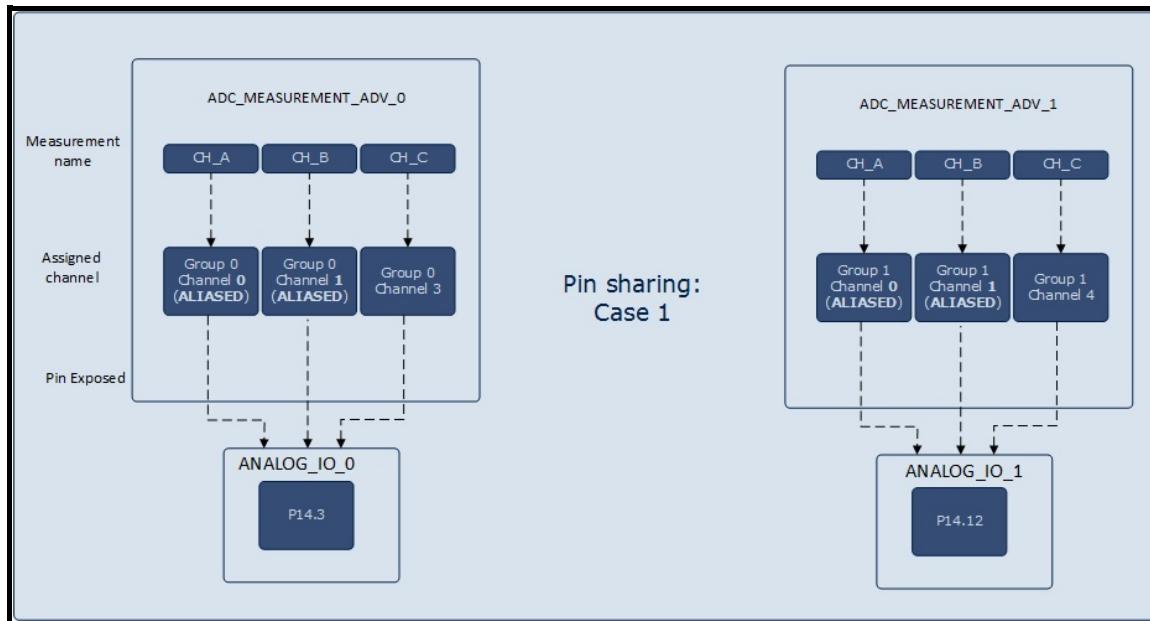


Figure 5 : Pins and channel consumption with ANALOG_IO with a group

In different groups of the VADC the channels can have the same pin number. For example in XMC4500 P14.3 is shared between Group-0 Channel-3 and Group-1 Channel-3. Thus two instances of the **ADC_MEASUREMENT_ADV** APP can convert the same pin. The following figure shows that it is possible to connect the Group-1 Channel-3 to the ANALOG_IO that is consuming P14.3. With the alias feature in the Group-1 is also possible to convert that pin using Group-1 Channel-0 and Group-1 Channel-1.

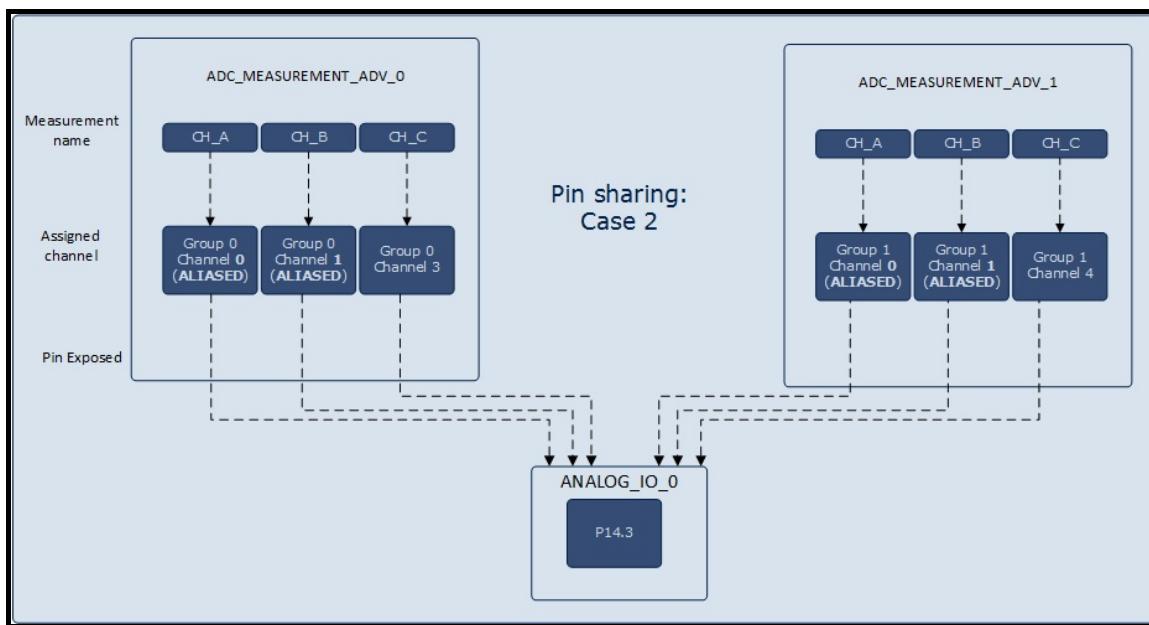


Figure 6 : Pins and channel consumption with ANALOG_IO shared between groups

2. Result Registers:

Each channel is mapped to one result register(excluding FIFO related configuration). 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.

In **ADC_MEASUREMENT_ADV** APP the *result_adv* is used when Boundary is used for Normal Comparison Mode. The *result_filter* is used when result post processing requires Result Filtering Mode .This is either FIR (Finite Impulse Response) or the IIR(Infinite Impulse Response) filters. The *result* is used when ever whenever Standard Data Reduction Mode or Subtraction Mode is selected.

3. Interrupt/Event Generation:

- **Request source interrupt:**

In the Interrupts TAB the **ADC_MEASUREMENT_ADV** doesn't consume a NVIC node for interrupt. But it uses a callback registration mechanism that is called whenever the request source interrupt is raised. This callback is registered with the request source APP, ADC_QUEUE/ADC_SCAN. The callback is registered in the queue/scan entry by considering the following rules.

- For queue request source the callback would be registered to the entry that has the "Source event" enabled in the Sequence Plan TAB.
- For scan request source the callback is registered with the lowest numbered channel number. Since the lowest channel numbered entry is the last channel to get executed in the scan sequence.

- **Result Event:**

Receive an event every time a new result is available in the result register. This event has to be enabled in the "Channel Configuration" TAB. To generate an interrupt from this event, connect the result event signal from the **ADC_MEASUREMENT_ADV** APP to INTERRUPT APP using the H/W Signal connections. This setup would provide result interrupts for the channels.

- **Channel Event:**

This event has to be enabled in the "Boundary Settings" TAB. To enable the channel events, select the channel event combo box for the required channel. To generate an interrupt from this event, connect the result event signal from the **ADC_MEASUREMENT_ADV** APP to INTERRUPT APP using the H/W Signal connections.

In Normal Conversion Mode it is possible to get a channel event only if the converted value is inside/outside the boundary values or on each new result. The boundary values must be configured using the API `ADC_MEASUREMENT_ADV_SetBoundary()`.

In Fast Compare Mode it is possible to get a channel event only if the measured value is above or below a compare value, or if it crosses the compare value. The compare value is located in the result register and must be configured with the API `ADC_MEASUREMENT_ADV_SetFastCompareValue()`

4. Queue Sequencer:

In the Queue sequence it is possible to have a flexible sequence of channels for conversion.

The "Sequence Plan" allows a flexible configuration of the sequence. Each Channel can be placed on any position. Also multiple selections are allowed. Additional for each position the configurations "Wait for Trigger", "Refill" and "Source Event" are available. With the "Wait for Trigger" "configuration" the queue is waiting for a trigger before the selected channel is converted. The "Refill" option allows a one time measurement. This position is not refilled in the queue and will not execute in the next rounds. The "Source Event" configuration will execute a source event when the conversion is finished.

The following examples explain the usage of the "Wait for Trigger"

and "Refill" configuration.

In the first example the sequence is starting with CH_C. This channel has the configuration "Wait for Trigger" therefore the sequence is waiting for a Trigger before it starts with the conversion of CH_C. No other channel is configured with "Wait for Trigger" therefore the sequence is immediately continuing with the conversion. When the sequence is reaching the end all channels with "Refill" are refilled and the sequence starts again as new. In this case it is waiting again for a trigger.

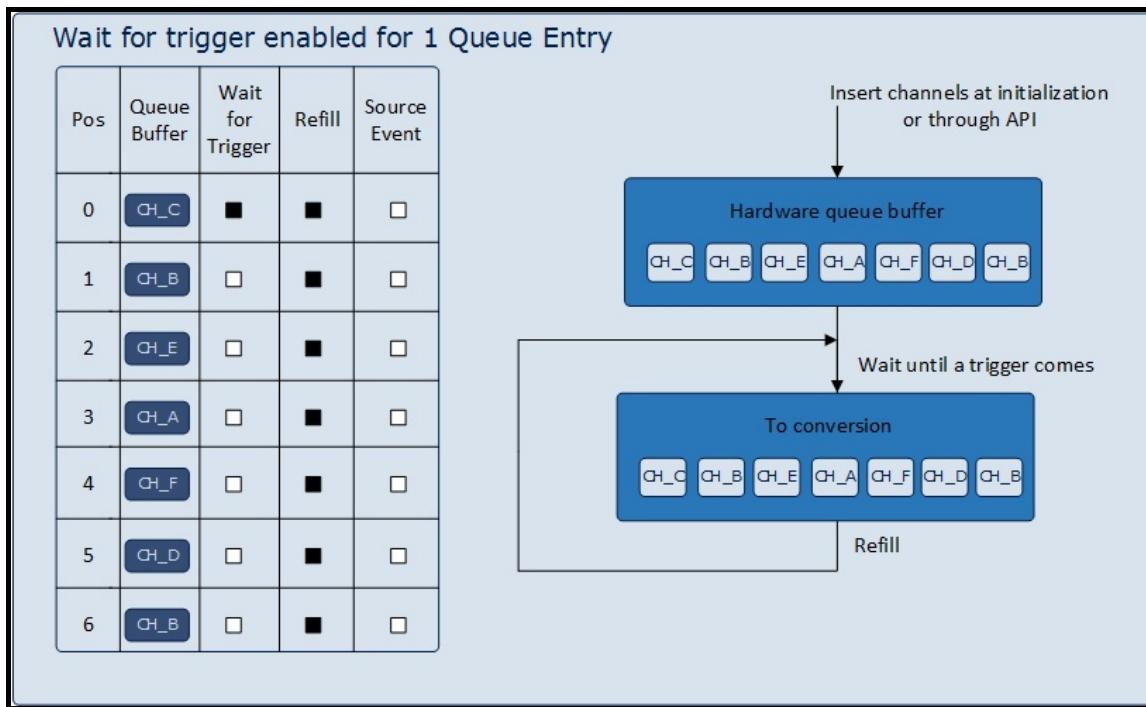


Figure 7 : Queue sequence: Wait for trigger enabled for 1 Queue Entry

In the second example additional CH_A is configured with "Wait for Trigger". This results in a two steps sequence. First the sequence is waiting for a trigger, after this trigger CH_C, CH_B and CH_E are converted. Now the sequence is again waiting for a trigger to continue with CH_A. After a trigger CH_A, CH_F, CH_D and CH_B are converted. Hence all channels provide the configuration "Refill" the sequence is repeated.

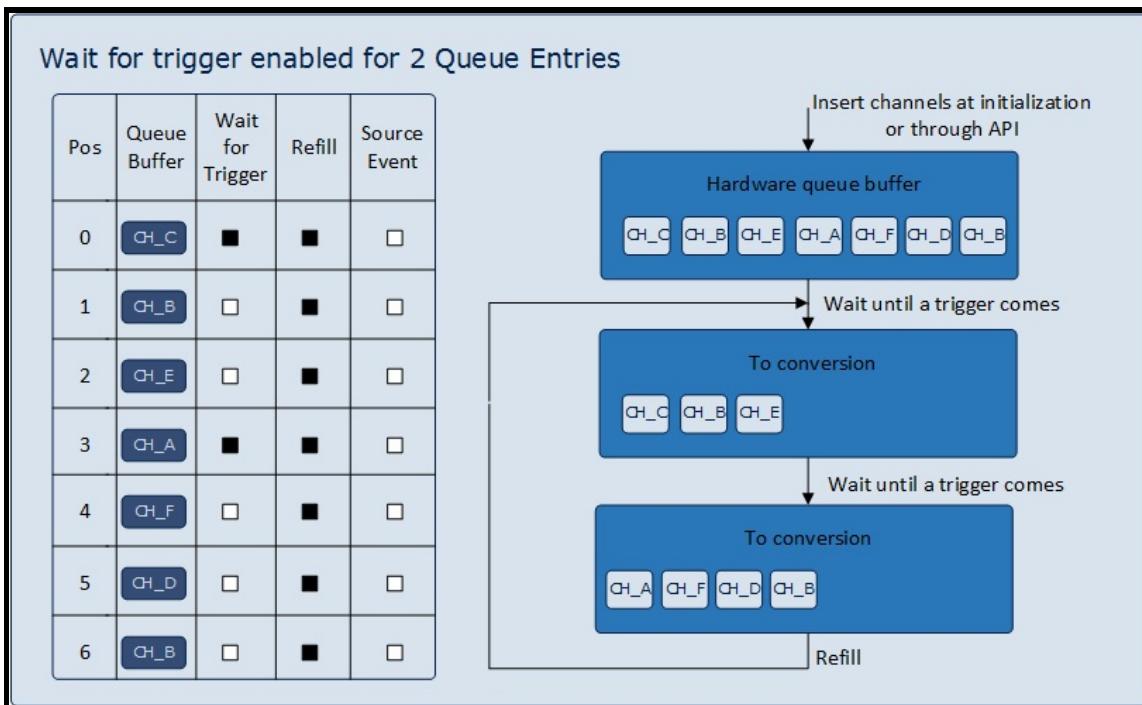


Figure 8 : Queue sequence: Wait for trigger enabled for 2 Queue Entries

In the third example again only CH_C provides a "Wait for Trigger" configuration but know don't has a "Refill" configuration. This means the sequence is waiting for a trigger. After the trigger it start with the conversion of CH_C and continuous with all other channels. At the end of the sequence all channels with "Refill" configuration are refilled. Hence CH_C don't has a "Refill" it will not copied in the sequence. This means after the second round there is no entry with "Wait for Read" and the sequence repeat endless.

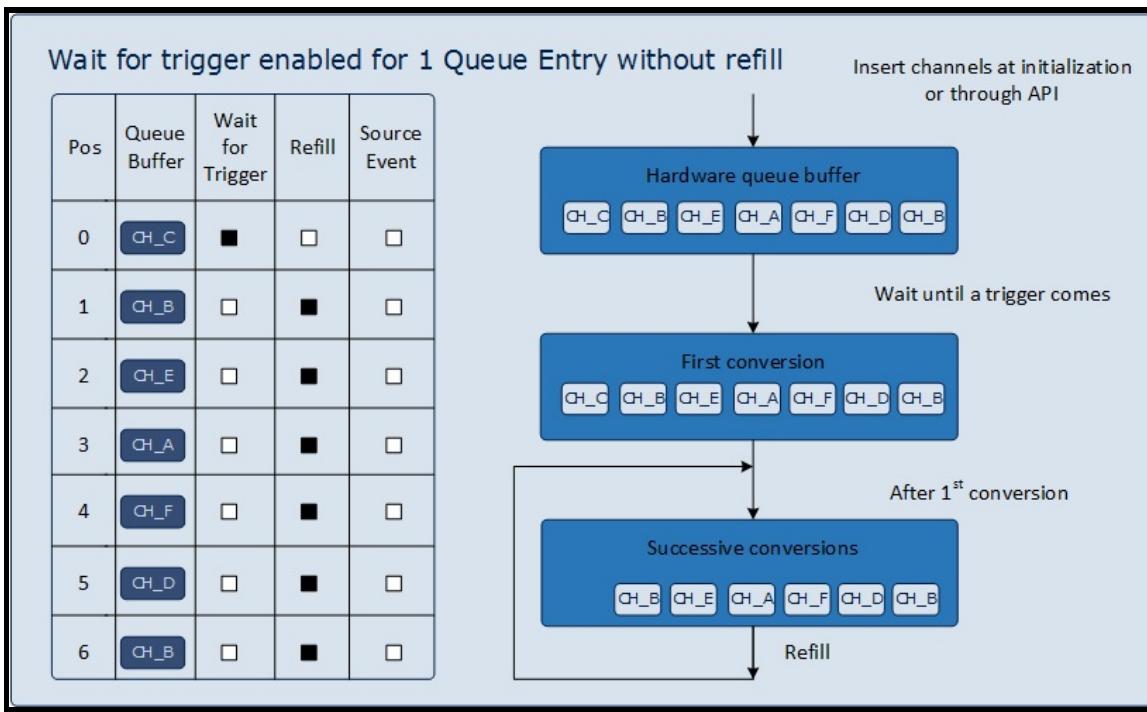


Figure 9 : Queue sequence: Wait for trigger enabled for 1 Queue Entry without refill

5. Insert channels :

The insert operation in **ADC_MEASUREMENT_ADV** binds the channels to the selected request source in the hardware. Since the ADC_QUEUE is shared it is possible to assign the same queue position from other instances of **ADC_MEASUREMENT_ADV** (or from other APPs). In order to prevent such operations, a MACRO is constructed with the following format.

VADC_QUEUE_GROUP_<group_number>_POSITION_<position>

This would prevent the APPs that are sharing the ADC_QUEUE APP from trying to push a queue entry into the same location. If the same position is used a error message is displayed. If such a error message is received then the queue sequence needs to be modified appropriately in the GUI.

- **ADC_QUEUE**

When a ADC_MEASUREMENT_ADV_InsertChannels() API is invoked or the GUI check box "Insert channels at initialization" is enabled with the "Wait for Trigger", the channels will wait until a trigger event occurs. A trigger event can be either a ADC_MEASUREMENT_ADV_StartConversion() or an external trigger edge. When a trigger event occurs the particular channel will go for conversion.

When a ADC_MEASUREMENT_ADV_InsertChannels() API is invoked or the GUI check box "Insert channels at initialization" is enabled without the "Wait for Trigger", The channel being inserted will get converted immediately. And if the refill is enabled for the channel then it will continuously keep converting.

- **ADC_SCAN**

When a ADC_MEASUREMENT_ADV_InsertChannels() API is invoked or the GUI check box "Insert channels at initialization" is enabled, the channels will be inserted into the hardware.

There these channels will continue to wait until a trigger event occurs. A trigger event can be either a ADC_MEASUREMENT_ADV_StartConversion() or an external trigger edge. The trigger event will make all the channels to go for conversion. When all the channels are converted it will go back to wait state when "continuous conversion" is disabled. If "Continuous conversion" is enabled the last channel to get converted will trigger all the channels to get converted again.

6. Sample and Hold Gain Configuration:

The gain related configuration is applicable for only the XMC1x series. This feature allows the input signal to be amplified by the required factor. The gain values can be selected in the GUI for the required channel or by invoking the API

ADC_MEASUREMENT_ADV_SetChannelGain(). The following example explains how the gain is configured when alias feature is used. The following figure explains the gain configuration by taking

3 channels as an example. The configuration is as follows, CH_A is Group-0 Channel-3, CH_B and CH_C are the alias channels which consume group-0 channel-0 and group-0 channel-1. CH_A and CH_B need gain of 1:1 and CH_C needs 1:3 as the gain ratio. The source channel in this case is Group-0 Channel-3 and the source pin is P14.3 and the other 2 are aliased to this channel. When alias feature is used and gain configuration is needed for the alias channel it is necessary to configure the gain of the source channel rather than the gain of the alias channel. In this example in-order to configure the gain for the CH_C it is necessary to configure the gain in the source channel i.e. group-0 channel-3. Due to such a configuration of the SHS the CH_A and CH_B would also convert with a gain factor of 1:3.

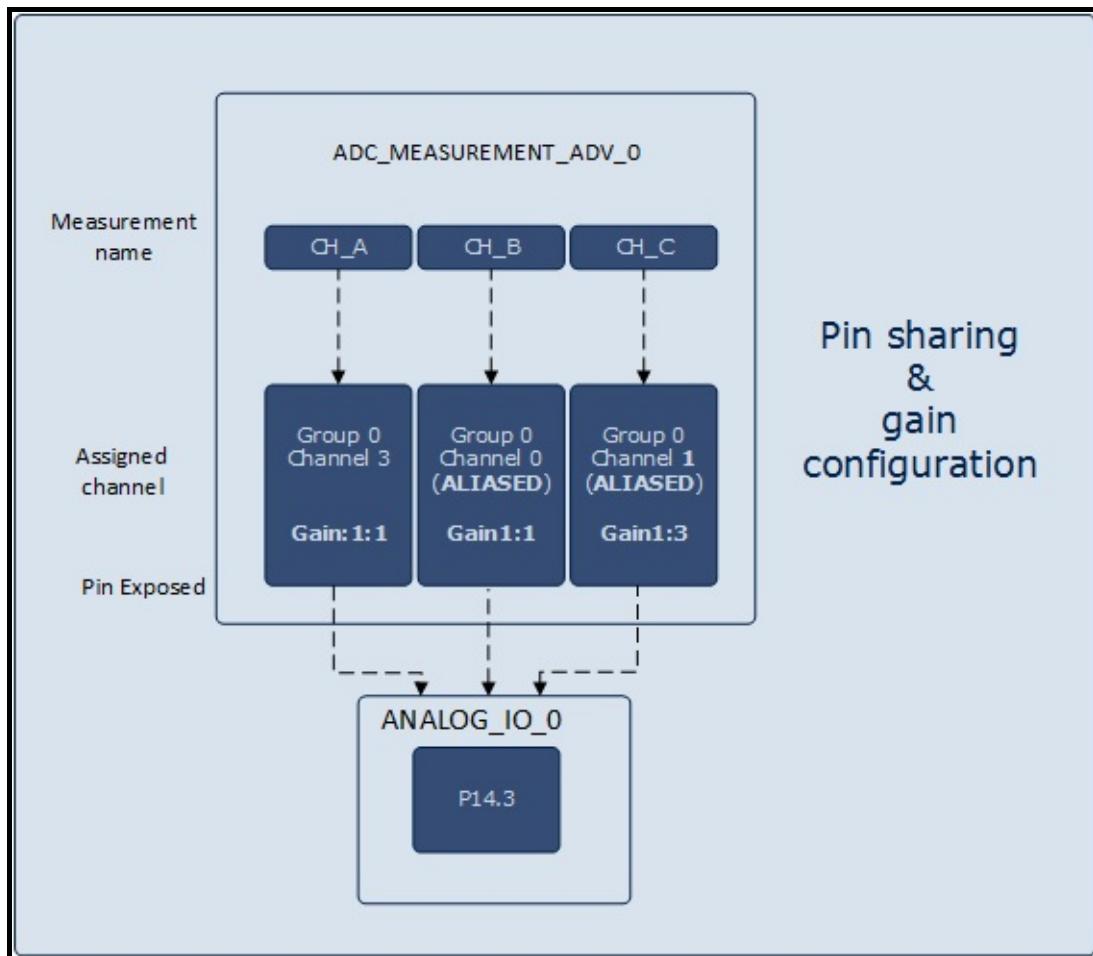


Figure 10 : Sample and Hold gain configuration example with alias enabled

7. Limit checking(Boundary configuration):

The Limit checking can automatically compare each conversion to an upper or lower bound values. Accordingly a channel event can be triggered if the result is inside/outside the user-defined band. This limit checking can only be done by specific registers with boundary being configured in GxBOUND or GLOBBOUND. In XMC45 device the limit checking can be done the first 4 channels in each group and in all the other devices the limit checking is done by the first 4 result registers in each group. Hence in the UI the boundary configuration is limited to only 4.

The Boundary values can only be configured by invoking the API **ADC_MEASUREMENT_ADV_SetBoundaryUpper()** and **ADC_MEASUREMENT_ADV_SetBoundaryLower()**. These API would configure the respective boundary registers mentioned in the "Boundary Settings" Tab in APP's GUI.

Note: The value to be configured in the boundary registers have to be left aligned compare values. For example, a compare value in 10 bit resolution is 500 in order to configure this in the boundary registers it has to be first left aligned(left alignment is done by left shift operation). After the left shift operation the resulting value is 2000 this value has to be passed as an argument into the **ADC_MEASUREMENT_ADV_SetBoundaryUpper()** or **ADC_MEASUREMENT_ADV_SetBoundaryLower()** API.

Signals:

The following table provides all APP signals for connection.

Table 1: APP IO signals

Signal Name	Input/Output	Availability	Description
			Result event

	<i>channel_x</i> :
	<ul style="list-style-type: none"> ◦ It can be connected to INTERF APP to generate interrupt the result generated by a particular channel.
event_res_	<i>channel_x</i>
Where <i>channel_x</i> represents the channels that are being used.	Output
	Unconditional

Channel event
channel_x:

- It can be connected to INTERF APP.
- The channel event is

`event_ch_channel_x`

Where *channel_x* represents the channels that are being used.

Output

Unconditional

available in the combobox selected in the UI "Boundary Settings" tab of the part channel enabled.

- The label *channel_x* can be configured in the "Channel Configuration Tab" of the APP GUI.

`bound_fl_channel_x`

Where *channel_x* represents the channels that are being used.

Output

Conditional

Boundary flag output for *channel_x*:

- It can be connected to other peripherals like CCI, POSIF, etc.
- The boundary flag signal is available when the combobox selection in the UI "Boundary Settings" tab of the part

channel
enabled
◦ The label
channel_x
can be
configured
the "Ch
Configura
Tab of th
APP GU

Input connec
from ANALOG
APP pin to g
channel:

- The pin
channel
connect
visible c
when
"Expose
option is
selected
channel
configur
under
"Chann
Configura
TAB".
- The Ex
option a
the pin t
shared
between
peripher
◦ The lab
channel_x
can be

channel_x_input

Where *channel_x*
represents the channels
that are being exposed.

Conditional

Input

configuring
the "Ch
Configura
Tab of the
APP GU

Input connect
from ANALC
APP pin to s
group channel
◦ The pin
channel
connect
visible c
when
"Expose
option is
selected
synchro
channel
configur
under
"Chann
Configu
TAB".

- The Ex
option a
the pin i
shared
between
peripher
- The lab
channel
can be
configur
the "Ch
Configu

channel_x_slavey_input

Where *channel_x*
represents the slave Input channels that are being exposed.

Conditional

Tab of the
APP GU

Input connected
from request
source APP
(ADC_SCAN
ADC_QUEUE)
channel:

- It selects
particular
group
channel
establishes
connection
from group
request
source to
selected
channel
- The label
channel_x
can be
configured
in the "Chan-
Config" Tab of the
APP GU

channel_x_sel

Where *channel_x*
represents the channels
that are being used.

Unconditional

Timing Calculations:

ADC_MEASUREMENT_ADV APP uses the following equations to calculate the sample time and Total conversion time.

Refer the reference manual for the detailed information.

1. 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. 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. 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 (PC)

PC = The post-calibration time PC, if selected (PC)

3. Fast Compare Mode:

Total Conversion time = (2 + STC + 2) * tADCI + 2 * tAI

Where,

tADC = ADC module clock = system clock

tADCI = Analog clock

STC = Sample time control (Value - 0 to 256)

2. Total Conversion time: XMC1000 devices

- "Total conversion time" for XMC1400/XMC1300/XMC1200 devices are calculated with post calibration.

1. Sample Time:

Minimum sample time is calculated to in accordance with Errata ADC_AI.H007. To ensure proper operation of the internal control logic, tS must be at least four cycles of the prescaled converter clock fSH, i.e. tS \geq 4 tCONV x

(DIVS+1).

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

2. 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)
tADC = ADC module clock = system clock
tADCI = Analog clock
STC = Sample time control (Value - 0 to 256)

tSH = tCONV * (DIVS + 1)
Where,
tSH = Sample and Hold clock (Converter clock time)
tCONV = Converter clock time period (SHS Clock).
DIVS = Divider Factor for the SHS Clock

3. Fast Compare Mode:

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

ADC_MEASUREMENT_ADV

Home

APP Configuration Parameters

App Configuration Parameters

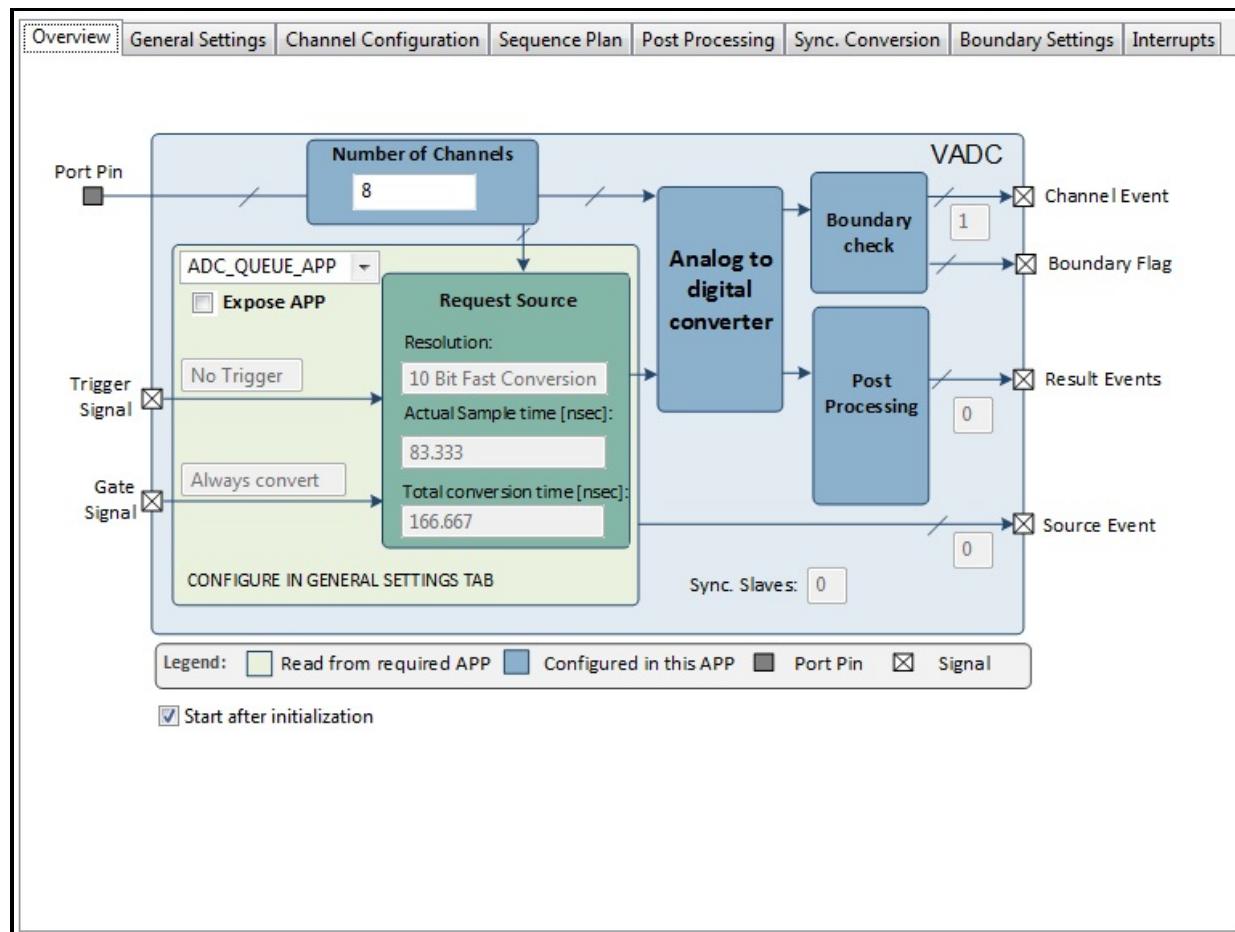


Figure 1: Overview

Overview General Settings Channel Configuration Sequence Plan Post Processing Sync. Conversion Boundary Settings Interrupts

Request Source Settings

Trigger edge selection: No External Trigger

Gating selection: All Conversion Requests are Issued

Priority of queue source: Priority-0 (Lowest Priority)

Conversion start mode: Wait For Start Mode

Class Settings

Conversion mode: 10 Bit Fast Conversion

Desired sample time [nsec]: 100

Actual sample time [nsec]: 83.333

Total conversion time [nsec]: 166.667

Figure 2: General Settings

Overview General Settings Channel Configuration Sequence Plan Post Processing Sync. Conversion Boundary Settings Interrupts

Channel configuration

Channel name:	Expose Pin:	Wait for read:	Result Event:	Reference GND:	Gain:
Channel_A	<input type="checkbox"/> ALL	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1
Channel_B	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1
Channel_C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1
Channel_D	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1
Channel_E	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1
Channel_F	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1
Channel_G	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1
Channel_H	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Vss	1:1

Figure 3: Channel Configuration

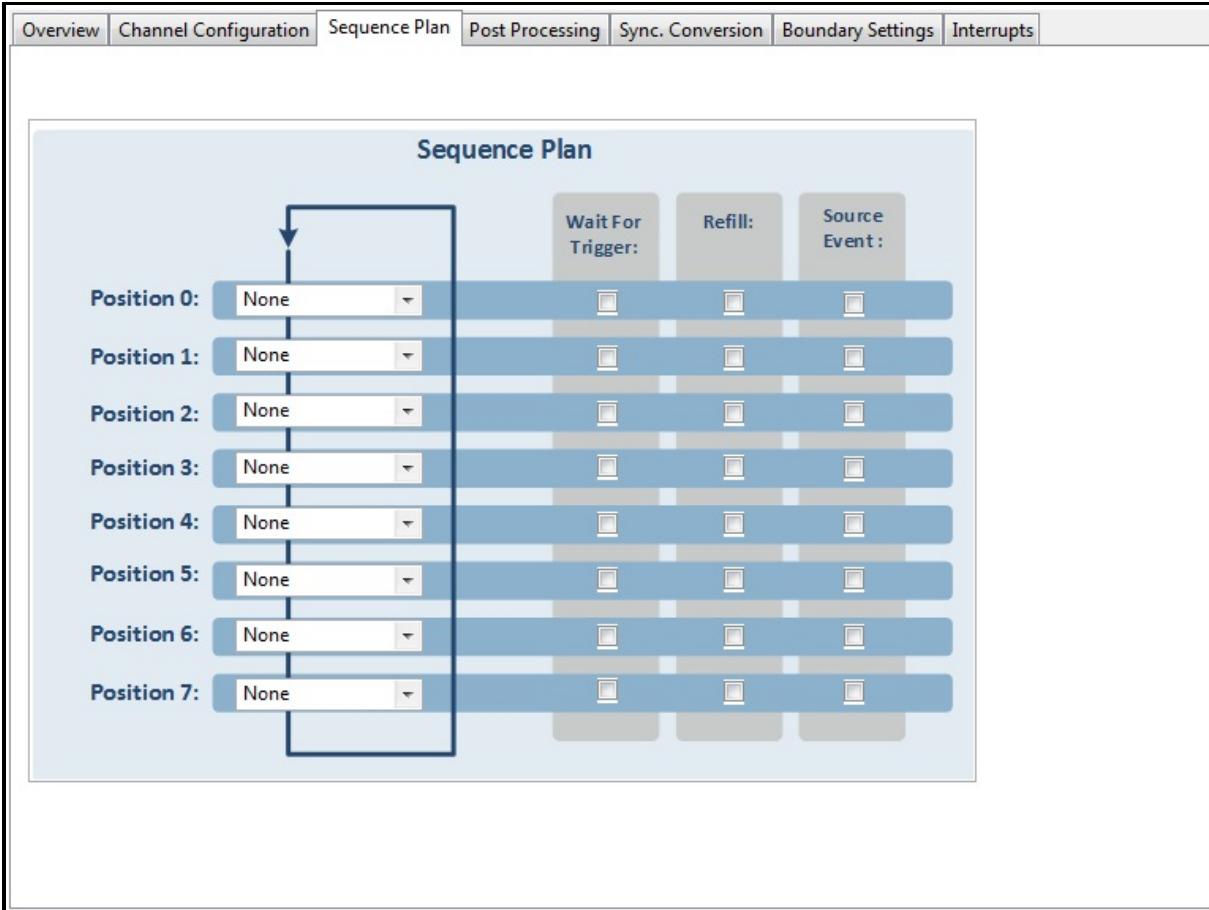
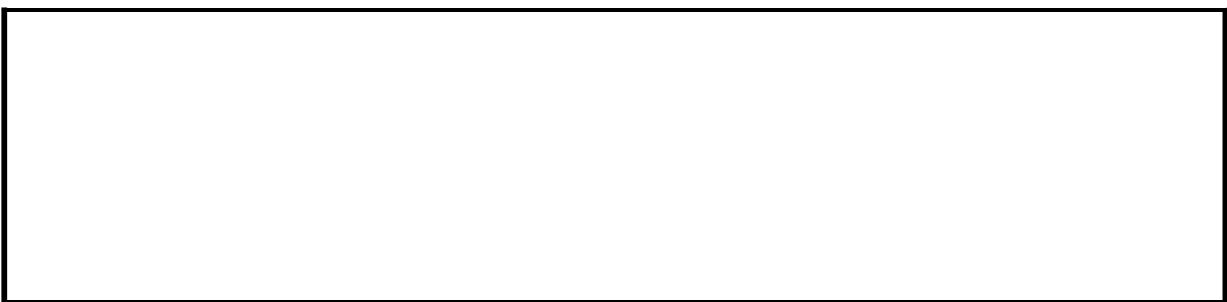


Figure 4: Sequence Plan

Overview	Channel Configuration	Sequence Plan	Post Processing	Sync. Conversion	Boundary Settings	Interrupts																																				
<table border="1"><thead><tr><th colspan="2">FIFO configuration:</th><th colspan="2">Filter configuration:</th></tr></thead><tbody><tr><td>Channel_A</td><td>No</td><td>No</td><td>No</td></tr><tr><td>Channel_B</td><td>No</td><td>No</td><td>No</td></tr><tr><td>Channel_C</td><td>No</td><td>No</td><td>No</td></tr><tr><td>Channel_D</td><td>No</td><td>No</td><td>No</td></tr><tr><td>Channel_E</td><td>No</td><td>No</td><td>No</td></tr><tr><td>Channel_F</td><td>No</td><td>No</td><td>No</td></tr><tr><td>Channel_G</td><td>No</td><td>No</td><td>No</td></tr><tr><td>Channel_H</td><td>No</td><td>No</td><td>No</td></tr></tbody></table>							FIFO configuration:		Filter configuration:		Channel_A	No	No	No	Channel_B	No	No	No	Channel_C	No	No	No	Channel_D	No	No	No	Channel_E	No	No	No	Channel_F	No	No	No	Channel_G	No	No	No	Channel_H	No	No	No
FIFO configuration:		Filter configuration:																																								
Channel_A	No	No	No																																							
Channel_B	No	No	No																																							
Channel_C	No	No	No																																							
Channel_D	No	No	No																																							
Channel_E	No	No	No																																							
Channel_F	No	No	No																																							
Channel_G	No	No	No																																							
Channel_H	No	No	No																																							

Figure 5: Post Processing



Overview General Settings Channel Configuration Sequence Plan Post Processing Sync. Conversion Boundary Settings Interr

Synchronised groups: Configure global ICLASS 1

	Sync master	Sync slave A	Sync slave B	Sync slave C
Channel_A	<input type="checkbox"/>	Channel_A_sync_slaveA	Channel_A_sync_slaveB	Channel_A_sync_slaveC
Channel_B	<input type="checkbox"/>	Channel_B_sync_slaveA	Channel_B_sync_slaveB	Channel_B_sync_slaveC
Channel_C	<input type="checkbox"/>	Channel_C_sync_slaveA	Channel_C_sync_slaveB	Channel_C_sync_slaveC
Channel_D	<input type="checkbox"/>	Channel_D_sync_slaveA	Channel_D_sync_slaveB	Channel_D_sync_slaveC

Note: Ensure that the "Arbitration mode" for all the sync groups are set to "Runs Permanently" in GLOBAL_ADC APP.

Figure 6: Sync. Conversion



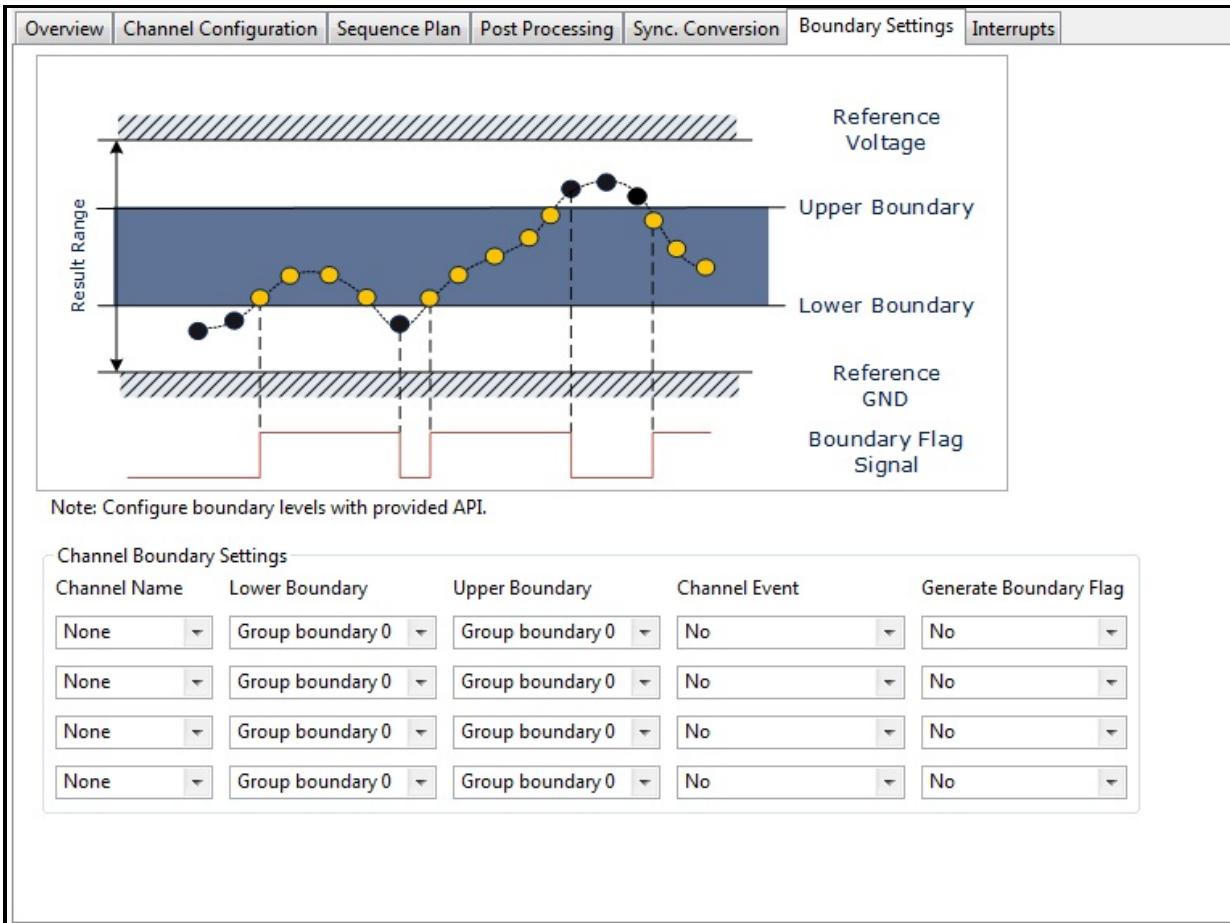


Figure 7: Boundary Settings

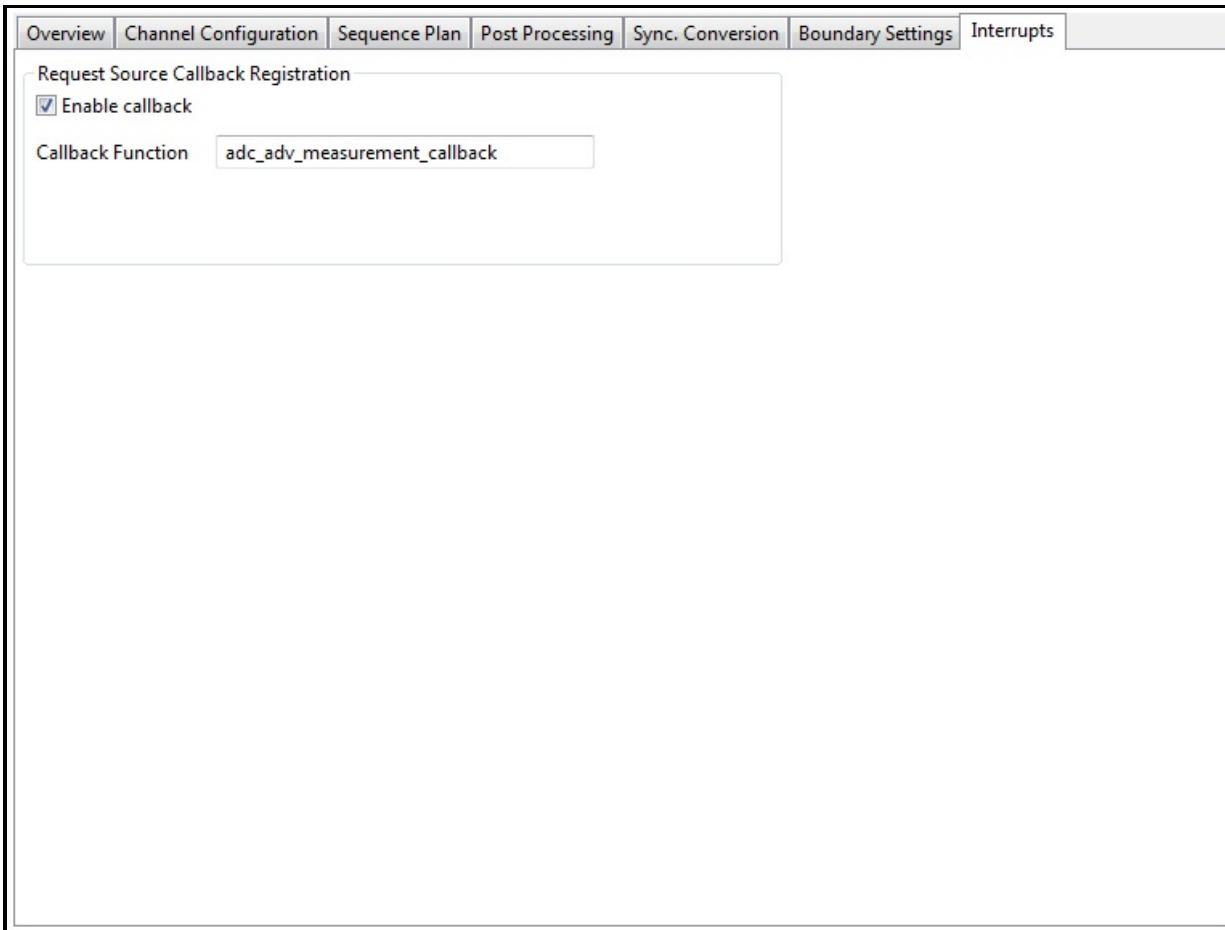
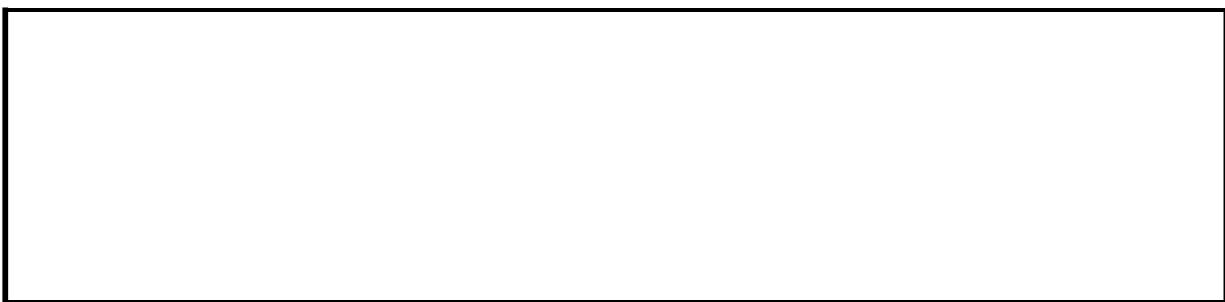


Figure 8: Interrupts



ADC_MEASUREMENT_ADV

Home

Enumerations

enum	ADC_MEASURE ADC_MEASURE ADC_MEASURE ADC_MEASURE Return value of a
enum	ADC_MEASURE ADC_MEASURE ADC_MEASURE ADC_MEASURE ADC_MEASURE }
enum	ADC_MEASURE ADC_MEASURE ADC_MEASURE ADC_MEASURE The selected Rec
enum	ADC_MEASURE ADC_MEASURE ADC_MEASURE ADC_MEASURE The gain applied
enum	ADC_MEASURE ADC_MEASUR = 0U, ADC_MEASURE = 0U, ADC_MEASURE 2U, ADC_MEASURE = 0U, ADC_MEASUR 4U,

		ADC_MEASURE
		OU
		}
		Alignment options
	enum	ADC_MEASURE
		The result of the
	enum	ADC_MEASURE
		The result of the
		ADC_MEASURE
		Return value of a
		ADC_MEASURE
		The selected Rec
		ADC_MEASURE
		The gain applied
		ADC_MEASURE
		Alignment options
		ADC_MEASURE
		The result of the
		ADC_MEASURE
		The result of the

Enumeration Type Documentation

enum ADC_MEASUREMENT_ADV_STATUS

Return value of an API.

Enumerator:

ADC_MEASUREMENT_ADV_STATUS_SUCCESS

The API call is success

ADC_MEASUREMENT_ADV_STATUS_FAILURE

The API call is failed

ADC_MEASUREMENT_ADV_STATUS_UNINITIALIZED

APP has not been Initialize

Definition at line [107](#) of file **ADC_MEASUREMENT_ADV.h**.

ADC_MEASUREMENT_ADV

Home

Data structures

typedef void(*	ADC_MEASUREMENT
) (void)	
typedef struct	ADC_MEASUREMENT
ADC_MEASUREMENT_ADV_NVIC_CONFIG	NVIC Configuration stru interrupt.
typedef struct	ADC_MEASUREMENT
ADC_MEASUREMENT_ADV_SCAN	Configuration Data struc source.
typedef struct	ADC_MEASUREMENT
ADC_MEASUREMENT_ADV_QUEUE	Configuration Data struc source.
typedef struct	ADC_MEASUREMENT
ADC_MEASUREMENT_ADV_CHANNEL	
typedef struct ADC_MEASUREMENT_ADV	ADC_MEASUREMENT

Typedef Documentation

typedef struct ADC_MEASUREMENT_ADV_CHANNEL ADC_MEASUREMENT_ADV_CHANNEL_t

Structure to configure the channels in the **ADC_MEASUREMENT_ADV** APP.

typedef void(* ADC_MEASUREMENT_ADV_EVENT_CONFIG_t)(void)

Function pointer to the mux configuration

Definition at line **190** of file **ADC_MEASUREMENT_ADV.h**.

typedef struct ADC_MEASUREMENT_ADV ADC_MEASUREMENT_ADV_t

Structure to configure **ADC_MEASUREMENT_ADV** APP.

ADC_MEASUREMENT_ADV

Home

Methods

DAVE_APP_VERSION_t **ADC_MEASURE**

Get **ADC_MEASU**

void **ADC_MEASURE**

ADC_MEASURE

Starts the VADC re

void **ADC_MEASURE**

ADC_MEASURE

Starts the conversi

ADC_MEASUREMENT_ADV_STATUS_t **ADC_MEASURE**

ADC_MEASURE

Initializes the **ADC**

Instance.

ADC_MEASURE

ADC_MEASURE

handle_ptr)

Returns the conver

ADC_MEASURE

(const **ADC_MEAS**

*const handle_ptr)

Returns the comple

ADC_MEASURE

ADC_MEASURE

handle_ptr)

Returns the conver

ADC_MEASURE

(const **ADC_MEAS**

*const handle_ptr)

Returns the comple

	<pre>__STATIC_INLINE ADC_MEASUREMENT_ADV_FAST_COMPARE_t</pre>	<pre>(const ADC_MEASUREMENT_ADV_FAST_COMPARE_t *const handle_ptr)</pre>	Returns the result.
	<pre>ADC_MEASUREMENT_ADV_STATUS_t</pre>	<pre>(const ADC_MEASUREMENT_ADV_STATUS_t *const handle_ptr,</pre>	Sets the fast conversion status.
void	<pre>ADC_MEASURE</pre>	<pre>(const ADC_MEASUREMENT_ADV_FAST_COMPARE_t handle_ptr,</pre>	<pre>ADC_MEASURE</pre>
	<pre>ADC_MEASURE</pre>	<pre>subtraction_alignment)</pre>	Set the subtraction alignment required.
void	<pre>ADC_MEASURE</pre>	<pre>(const ADC_MEASUREMENT_ADV_FAST_COMPARE_t *const handle_ptr,</pre>	<pre>XMC_VADC_GRO</pre>
	<pre>ADC_MEASURE</pre>	<pre>XMC_VADC_GRO_time).</pre>	Configures the input time.
void	<pre>ADC_MEASURE</pre>	<pre>(const ADC_MEASUREMENT_ADV_FAST_COMPARE_t *const handle_ptr,</pre>	<pre>XMC_VADC_CHA</pre>
	<pre>ADC_MEASURE</pre>	<pre>Selects alternate reference.</pre>	
void	<pre>ADC_MEASURE</pre>	<pre>(ADC_MEASURE</pre>	
	<pre>ADC_MEASURE</pre>	<pre>handle_ptr, const ADC_MEASURE</pre>	
	<pre>ADC_MEASURE</pre>	<pre>gain_factor)</pre>	Sets the channel gain factor.
void	<pre>ADC_MEASURE</pre>	<pre>(ADC_MEASURE</pre>	
	<pre>ADC_MEASURE</pre>	<pre>handle_ptr, XMC_VADC_BOUNDARY,</pre>	
	<pre>ADC_MEASURE</pre>	<pre>XMC_VADC_BOUNDARY_SELECTION)</pre>	

Select the boundary for the upper channel.

ADC_MEASURE_SET_UPPER_CHANNEL_ALIAS

void ADC_MEASURE_SET_UPPER_CHANNEL_ALIAS(
 const ADC_MEASURE_t *const handle_ptr,
 Sets the upper boundary for the channel.

ADC_MEASURE_SET_LOWER_CHANNEL_ALIAS

void ADC_MEASURE_SET_LOWER_CHANNEL_ALIAS(
 const ADC_MEASURE_t *const handle_ptr,
 Sets the lower boundary for the channel.

ADC_MEASURE_GET_CONFIG

XMC_VADC_CHANNEL_ALIAS_t ADC_MEASURE_GET_CONFIG(
 handle_ptr)
Returns the configuration for the channel.

void ADC_MEASURE_ENABLE_UNIFORM_CLOCKING(
 ADC_MEASURE_t *const handle_ptr)
Enables uniform clocking for all slaves.

Methods

Function Documentation

```
void ADC_MEASUREMENT_ADV_ConfigureChannelClass ( const A
                                                const X
                                                )
```

Configures the input class (Resolution and Sampling time).

Parameters:

iclass_selection	Select the input class to be configured Pass XMC_VADC_CHANNEL_CONV_GROUP_CLA to configure group input class 0. Pass XMC_VADC_CHANNEL_CONV_GROUP_CLA to configure group input class 1.
config	Constant pointer to the input class configuration
group_num	The group number whose input class needs to be configured.

Returns:

None

Description:

Configures the input class for standard conversion (GxICLASS[0] OR GxICLASS[1]). A call to this API would configure the Resolution and sampling time for standard conversion. The group-specific input class registers define the sample time and data conversion mode for each channel of the respective group. And each channel can use these by selecting the input class in GxCHCTRY.ICLSEL.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.
```



```
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\n");
}

while(1U)
{
}

ADC_MEASUREMENT_ADV_ConfigureChannelClass(&ADC_MEASUREMENT_ADV_0_Channel_A,&res_8bit);

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

// Start the queue conversion sequence
ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);

while(1U)
{
}
}
```

Definition at line **968** of file **ADC_MEASUREMENT_ADV.c**.

References **ADC_MEASUREMENT_ADV_CHANNEL::ch_handle**,
and **ADC_MEASUREMENT_ADV_CHANNEL::group_index**.

XMC_VADC_CHANNEL_ALIAS_t ADC_MEASUREMENT_ADV_GetA

Returns the configured alias value.

Parameters:

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

Returns:

XMC_VADC_CHANNEL_ALIAS_t returns
XMC_VADC_CHANNEL_ALIAS_DISABLED if the alias is not
applicable to the channel or if alias is not enabled
else it returns the alias value.

Description:

Return the alias value for the channel. If the alias feature is enabled then the channels CH-0 or CH-1 can convert any other channel's input signal. The API returns XMC_VADC_CHANNEL_ALIAS_DISABLED if the Channel is neither CH-0 nor CH-1. Also the value XMC_VADC_CHANNEL_ALIAS_DISABLED is returned when the CH-0 or CH-1 is not configured with alias. If either CH-0 or CH-1 is configured with alias then the appropriate aliased channel number is returned. A call to this API would access the register GxALIAS.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.  
// Select the request source APP as ADC_SCAN.  
// Generate the code and build.  
// Replace this in the main.c.  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)
```

```
#define MAX_LOCAL_BUFFER (16U)
uint16_t result[16];
uint16_t i = 0U;

void channel_event_callback(void)
{
    result[i++] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);
    if(MAX_LOCAL_BUFFER == i)
    {
        i = 0U;
    }
}

int main(void)
{
    DAVE_STATUS_t status;
    XMC_VADC_CHANNEL_ALIAS_t alias_ch;

    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\n");
    }

    while(1U)
    {
    }
}
```

```

// Check if the given channel is aliased.
alias_ch = ADC_MEASUREMENT_ADV_GetAliasValue(&ADC_MEASUREMENT_ADV_0_Channel_A);

if(XMC_VADC_CHANNEL_ALIAS_DISABLED == alias_ch)
{
    // do something
}

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

while(1U)
{
    // Continuously re-trigger the scan conversion sequence
    ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);
}

```

Definition at line **1054** of file **ADC_MEASUREMENT_ADV.c**.

References **ADC_MEASUREMENT_ADV_CHANNEL::ch_num**, and **ADC_MEASUREMENT_ADV_CHANNEL::group_index**.

DAVE_APP_VERSION_t ADC_MEASUREMENT_ADV_GetAppVersion()

Get **ADC_MEASUREMENT_ADV** 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_ADV APP:
    // ADC_MEASUREMENT_ADV_Init() is called from within DAVE_Init().
    init_status = DAVE_Init();

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

    // More code here
    while(1) {

    }
    return (0);
}
```

Definition at line [682](#) of file [ADC_MEASUREMENT_ADV.c](#).

__STATIC_INLINE XMC_VADC_DETAILED_RESULT_t ADC_MEASU

Returns the complete conversion result.

Parameters:

handle_ptr constant pointer to the channel handle structure.

(Use the channel handle related macros which are defined in adc_measure_adv_conf.h)

Returns:

XMC_VADC_DETAILED_RESULT_t returns the complete result register

Description:

Return the completely 32 bit result register (GxRESy). In the APP each channel is configured to a particular group result register (excluding FIFO). The result of conversion as well as other information is returned from this API. The detailed result contains result of the most recent conversion, the channel number requested the conversion, valid flag, converted request source, fast compare result, the result data reduction counter and the EMUX channel number (if GxRES[0] only). In polling mechanism the converted result can be read out after checking the valid flag bit. The result register is defined in the channel handle structure **ADC_MEASUREMENT_ADV_CHANNEL_t**. Hence this API shall call be called with a pointer to the channel handle of type **ADC_MEASUREMENT_ADV_CHANNEL_t** (Directly use the channel handle related macros which are defined in adc_measure_adv_conf.h).

Note:

This API is not Applicable for reading the result from the result FIFO registers Use **ADC_MEASUREMENT_ADV_GetFifoDetailedResult** in order to read the FIFO result.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.  
// Select the request source APP from ADC_SCAN to ADC_QUEUE.  
// Goto the sequence plan and select Channel_A at position-0 and Channel_B at position-1.
```

```

// Enable Wait for trigger for the Channel_A.
// Goto interrupts tab enable request source interrupt.
// Goto the ADC_QUEUE APP and enable the request source interrupt.
// Generate the code and build.
// Replace this in the main.c.
#include <DAVE.h>                                //Declarations
from DAVE Code Generation (includes SFR declaration)
#define QUEUE_SRC (0U)

XMC_VADC_DETAILED_RESULT_t result[2];
uint32_t queue_flag = 0U;

void adc_measurement_adv_callback(void)
{
    // Use the channel handle parameter in this format "<APP Name>_<CHANNEL Name>"
    result[0] = ADC_MEASUREMENT_ADV_GetDetailedResult(&ADC_MEASUREMENT_ADV_0_Channel_A);
    result[1] = ADC_MEASUREMENT_ADV_GetDetailedResult(&ADC_MEASUREMENT_ADV_0_Channel_B);

    if((result[0].converted_request_source == QUEUE_SRC) && (QUEUE_SRC == result[1].converted_request_source))
    {
        queue_flag++;
    }
}

int main(void)
{
    DAVE_STATUS_t status;

```

```
    status = DAVE_Init();                      // Initialization
on 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\n"
    );

    while(1U)
    {

    }
}

ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_A
DV_0);

// Start the queue conversion sequence
ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUR
EMENT_ADV_0);

while(queue_flag != 0U);
// do something

while(1U)
{
}
}
```

Definition at line 731 of file [ADC_MEASUREMENT_ADV.h](#).

References [ADC_MEASUREMENT_ADV_CHANNEL::ch_handle](#),
and [ADC_MEASUREMENT_ADV_CHANNEL::group_index](#).

__STATIC_INLINE ADC_MEASUREMENT_ADV_FAST_COMPARE_t

Returns the result of fast conversion.

Parameters:

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

Returns:

XMC_VADC_FAST_COMPARE_t fast conversion result.

Returns

::ADC_MEASUREMENT_ADV_FAST_COMPARE_HIGH if the sampled signal is greater than the compare value. Returns ::ADC_MEASUREMENT_ADV_FAST_COMPARE_LOW if the sampled signal is lower than the compare value. Returns ::ADC_MEASUREMENT_ADV_FAST_COMPARE_INVALID if there is no valid result available.

Description:

Returns the fast conversion result stored in the result register [GxRESy.FCR]. In the APP each channel is configured to a particular group result register (excluding FIFO). The result register is defined in the channel handle structure

ADC_MEASUREMENT_ADV_CHANNEL_t. Hence this API shall call be called with a pointer to the channel handle of type **ADC_MEASUREMENT_ADV_CHANNEL_t** (Directly use the channel handle related macros which are defined in adc_measure_adv_conf.h).

Note:

This API is only applicable to read fast compare result of the particular channel.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set th
```

```

e number of required channels to 1.
// Select the request source APP from ADC_SCAN to
ADC_QUEUE.
// Goto the sequence plan and select Channel_A at
position-0 and enable the refill.
// Enable Wait for trigger for the Channel_A.
// Goto interrupts tab enable result event for Ch
annel_A.
// Instantiate the interrupt APP.
// In the UI of the interrupt APP change the inte
rrupt handler to "channel_event_callback"
// goto HW signal connectivity and connect event_
result_Channel_A to interrupt APP.
// Generate the code and build.
// Replace this in the main.c.

#include <DAVE.h> //Declarations
from DAVE Code Generation (includes SFR declarati
on)

#define MAX_LOCAL_BUFFER (16U)
ADC_MEASUREMENT_ADV_FAST_COMPARE_t result[16];
uint16_t i = 0U;

void channel_event_callback(void)
{
    result[i++] = ADC_MEASUREMENT_ADV_GetFastCompar
eResult(&ADC_MEASUREMENT_ADV_0_Channel_A);
    if(MAX_LOCAL_BUFFER == i)
    {
        i = 0U;
    }
}

int main(void)
{
    DAVE_STATUS_t status;

```

```

    status = DAVE_Init();                      // Initialization
on 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\n"
    );

    while(1U)
    {

    }
}

// Set the threshold value as Vdd/2
ADC_MEASUREMENT_ADV_SetFastCompareValue(&ADC_MEASUREMENT_ADV_0_Channel_A, 512U);

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

while(1U)
{
    // Continuously trigger the queue conversion sequence
    ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);
}
}

```

Definition at line **1001** of file **ADC_MEASUREMENT_ADV.h**.

References **ADC_MEASUREMENT_ADV_CHANNEL::ch_handle**,

and [**ADC_MEASUREMENT_ADV_CHANNEL::group_index**](#).

__STATIC_INLINE XMC_VADC_DETAILED_RESULT_t ADC_MEASU

Returns the complete conversion result.

Parameters:

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

Returns:

XMC_VADC_DETAILED_RESULT_t returns the complete result register

Description:

Return the completely 32 bit FIFO result register (GxRESy). If result FIFO is configured then the results are available in the FIFO tail register. The result of conversion as well as other information is returned from this API. The detailed result contains result of the most recent conversion, the channel number requested the conversion, valid flag, converted request source, fast compare result, the result data reduction counter and the EMUX channel number (if GxRES[0] only). In polling mechanism the converted result can be read out after checking the valid flag bit. The result register is defined in the channel handle structure [**ADC_MEASUREMENT_ADV_CHANNEL_t**](#). Hence this API shall call be called with a pointer to the channel handle of type [**ADC_MEASUREMENT_ADV_CHANNEL_t**](#) (Directly use the channel handle related macros which are defined in adc_measure_adv_conf.h).

Note:

This API is not Applicable for reading the result from channels which done use FIFO. Use

[**ADC_MEASUREMENT_ADV_GetDetailedResult**](#) in order to

read from a single result register.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.  
// Select the request source APP from ADC_SCAN to ADC_QUEUE.  
// Goto the sequence plan and select Channel_A at position-0 and Channel_B at position-1.  
// Enable Wait for trigger for the Channel_A.  
// Goto interrupts tab enable request source interrupt.  
// Goto the ADC_QUEUE APP and enable the request source interrupt.  
// Generate the code and build.  
// Replace this in the main.c.  
  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
  
#define MAX_FIFO (16U)  
XMC_VADC_DETAILED_RESULT_t result[16];  
uint16_t i = 0U;  
void adc_measurement_adv_callback(void)  
{  
    // Use the channel handle parameter in this format "<APP Name>_<CHANNEL Name>"  
    result[i++] = ADC_MEASUREMENT_ADV_GetFifoDetail  
edResult(&ADC_MEASUREMENT_ADV_0_Channel_A);  
    result[i++] = ADC_MEASUREMENT_ADV_GetFifoDetail  
edResult(&ADC_MEASUREMENT_ADV_0_Channel_B);  
  
    if(MAX_FIFO == i)  
    {  
        i = (uint32_t)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\n");
    }

    while(1U)
    {

    }
}

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

// Start the queue conversion sequence
ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);

while(1U)
{
}
}
```

Definition at line 908 of file **ADC_MEASUREMENT_ADV.h**.

References

[ADC_MEASUREMENT_ADV_CHANNEL::group_index](#), and
[ADC_MEASUREMENT_ADV_CHANNEL::result_fifo_tail_number](#).

__STATIC_INLINE uint16_t ADC_MEASUREMENT_ADV_GetFifoRes

Returns the conversion result.

Parameters:

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

Returns:

uint16_t conversion result.

Range: [0x0 to 0xFFFF] without any filters/ accumulation/
subtraction enabled.

Description:

Returns the converted result stored in the result FIFO register [GxRESy.RESULT]. If result FIFO is configured then the results are available in the FIFO tail register. The result register can only be read at the tail of the FIFO, This result register number is defined in the channel handle structure

[ADC_MEASUREMENT_ADV_CHANNEL_t](#). Hence this API shall call be called with a pointer to the channel handle of type [ADC_MEASUREMENT_ADV_CHANNEL_t](#) (Directly use the channel handle related macros which are defined in adc_measure_adv_conf.h).

Note:

This API is not Applicable for reading the result from channels which do not use FIFO. Use

[ADC_MEASUREMENT_ADV_GetResult](#) in order to read from a single result register.

Example Usage:

```

// Initialize the ADC_MEASUREMENT_ADV APP. Set the
// number of required channels to 2.
// Select the request source APP from ADC_SCAN to
// ADC_QUEUE.
// Goto the sequence plan and select Channel_A at
// position-0 and Channel_B at position-1.
// Enable Wait for trigger for the Channel_A.
// Goto postprocessing Tab and select 8 stages FI
// FO for both Channel_A and Channel_B
// Goto interrupts tab enable request source inte
// rrupt.
// Goto the ADC_QUEUE APP and enable the request
// source interrupt.
// Generate the code and build.
// Replace this in the main.c.

#include <DAVE.h> //Declarations
from DAVE Code Generation (includes SFR declarati
on)

#define MAX_FIFO (16U)
uint16_t result[16];
uint16_t i;
void adc_measurement_adv_callback(void)
{
    // Use the channel handle parameter in this for
    mat "<APP Name>_<CHANNEL Name>"
    result[i++] = ADC_MEASUREMENT_ADV_GetFifoResult
(&ADC_MEASUREMENT_ADV_0_Channel_A);
    result[i++] = ADC_MEASUREMENT_ADV_GetFifoResult
(&ADC_MEASUREMENT_ADV_0_Channel_B);

    if(MAX_FIFO == i)
    {
        i = (uint32_t)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\n");
    }

    while(1U)
    {

    }
}

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

// Start the queue conversion sequence
ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);

while(1U)
{
}
}
```

Definition at line [821](#) of file [ADC_MEASUREMENT_ADV.h](#).

References

[**ADC_MEASUREMENT_ADV_CHANNEL::group_index**](#), and
[**ADC_MEASUREMENT_ADV_CHANNEL::result_fifo_tail_number**](#).

__STATIC_INLINE uint16_t ADC_MEASUREMENT_ADV_GetResult()

Returns the conversion result.

Parameters:

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

Returns:

uint16_t conversion result.
Range: [0x0 to 0xFFFF] without any filters/ accumulation/
subtraction enabled.

Description:

Return the converted result stored in the result register
[GxRESy.RESULT]. In the APP each channel is configured to a
particular group result register (excluding FIFO). The result
register is defined in the channel handle structure
[**ADC_MEASUREMENT_ADV_CHANNEL_t**](#). Hence this API
shall call be called with a pointer to the channel handle of type
[**ADC_MEASUREMENT_ADV_CHANNEL_t**](#) (Directly use the
channel handle related macros which are defined in
adc_measure_adv_conf.h).

Note:

This API is not Applicable for reading the result from the result
FIFO registers Use
[**ADC_MEASUREMENT_ADV_GetFifoResult**](#) in order to read
the FIFO result.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.  
// Select the request source APP from ADC_SCAN to ADC_QUEUE.  
// Goto the sequence plan and select Channel_A at position-0 and Channel_B at position-1.  
// Enable Wait for trigger for the Channel_A.  
// Goto interrupts tab enable request source interrupt.  
// Goto the ADC_QUEUE APP and enable the request source interrupt.  
// Generate the code and build.  
// Replace this in the main.c.  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
uint16_t result[2];  
void adc_measurement_adv_callback(void)  
{  
    // Use the channel handle parameter in this format "<APP Name>_<CHANNEL Name>"  
    result[0] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);  
    result[1] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_B);  
}  
  
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\n"
);

    while(1U)
{
}

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_A
DV_0);

// Start the queue conversion sequence
ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUR
EMENT_ADV_0);

    while(1U)
{
}
}
```

Definition at line 638 of file **ADC_MEASUREMENT_ADV.h**.

References **ADC_MEASUREMENT_ADV_CHANNEL::ch_handle**,
and **ADC_MEASUREMENT_ADV_CHANNEL::group_index**.

ADC_MEASUREMENT_ADV_STATUS_t ADC_MEASUREMENT_ADV

Initializes the **ADC_MEASUREMENT_ADV** APP Instance.

Parameters:

handle_ptr constant pointer to the APP handle structure

Returns:

None

Description:

Initialize the ADC and all the required configurations. A call to this API would initialize the queue request source or the scan request source depending on the GUI selection. The initialization is taken up by calling ADC_QUEUE_Init() or ADC_SCAN_Init(). Following this the synchronous conversion related initializations are taken up. In the sync initialization the slave groups and the master group are powered down and either GxSYNCTR.STSEL (if Slave) or GxSYNCTR.EVALRy (if master) are configured. After the sync related configurations are completed the master group alone is powered on. Following this the result event or channel event related service request node configurations are done (if required). Then the GxCHCTR configurations are completed. After the channel initialization the result handling initializations are done. This entails configuring the GxRCR registers for result filtering, accumulation, subtraction and FIFO. After all these initialization are completed the channels configured in the GUI is inserted into the appropriate ADC_QUEUE or ADC_SCAN APP buffer. If the GUI check box "Insert channels at initialization" is enable then these entries is pushed to the Hardware.

Example Usage:

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

Definition at line 796 of file **ADC_MEASUREMENT_ADV.c**.

References `ADC_MEASUREMENT_ADV_StartADC()`,
`ADC_MEASUREMENT_ADV_STATUS_UNINITIALIZED`,
`ADC_MEASUREMENT_ADV_CHANNEL::analog_io_config`,
`ADC_MEASUREMENT_ADV_CHANNEL::ch_handle`,
`ADC_MEASUREMENT_ADV_CHANNEL::ch_num`,
`ADC_MEASUREMENT_ADV::channel_array`,
`ADC_MEASUREMENT_ADV::event_config`,
`ADC_MEASUREMENT_ADV_CHANNEL::group_index`,
`ADC_MEASUREMENT_ADV::init_state`,
`ADC_MEASUREMENT_ADV::req_src`,
`ADC_MEASUREMENT_ADV_CHANNEL::shs_gain_factor`,
`ADC_MEASUREMENT_ADV::start_at_initialization`,
`ADC_MEASUREMENT_ADV::total_number_of_channels`, and
`ADC_MEASUREMENT_ADV::total_number_of_entries`.

```
void ADC_MEASUREMENT_ADV_SelectBoundary ( const ADC_MEASURE_XMC_VADC_Boundary_t boundary, XMC_VADC_Cfg_t cfg )
```

Select the boundary for the channel.

Parameters:

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

boundary The lower boundary or upper boundary of the channel to be configured.

boundary_selection Boundary register selection for the particular channel.

Returns:

None

Description:

Select the boundary for the channel. This API will select either lower boundary or the upper boundary depending on **boundary** and configures the **boundary_selection** value into the GxCHCTRy.BNDSELL or GxCHCTRy.BNDSELU accordingly.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 1.  
// Select the request source APP from ADC_SCAN to ADC_QUEUE.  
// Goto the sequence plan and select Channel_A at position-0 and enable the refill.  
// Enable Wait for trigger for the Channel_A.  
// Goto the Boundary Settings tab and select the channel name as Channel_A.  
// Change the channel event from No to "If Result Inside Band".  
// Change the "Generate boundary flag" to Yes/Non-Inverted.  
// Instantiate the interrupt APP.  
// In the UI of the interrupt APP change the interrupt handler to "channel_event_callback"  
// goto HW signal connectivity and connect event_channel_Channel_A to interrupt APP.  
// Generate the code and build.  
// Replace this in the main.c.  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
#define MAX_LOCAL_BUFFER (16U)  
uint16_t result[16];  
uint16_t i = 0U;  
  
void channel_event_callback(void)
```

```

{
    result[i++] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);
    if(MAX_LOCAL_BUFFER == i)
    {
        i = 0U;
    }
}

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\n");
    }

    while(1U)
    {

    }
}

// Set the boundary selection for Channel_A lower Bound as Group Bound-1
ADC_MEASUREMENT_ADV_SelectBoundary(&ADC_MEASUREMENT_ADV_0_Channel_A, XMC_VADC_BOUNDARY_SELECT_LOWER_BOUND,
                                    XMC_VADC_CH
ANNEL_BOUNDARY_GROUP_BOUND1);

```

```

    // Set the boundary selection for Channel_A upper Bound as Group Bound-0 and also the value as 2048U
    ADC_MEASUREMENT_ADV_SetBoundaryUpper(&ADC_MEASUREMENT_ADV_0_ChannelA, 2048U);

    // Start the selected request source
    ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

    while(1U)
    {
        // Continuously trigger the queue conversion sequence
        ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);
    }
}

```

Definition at line [1014](#) of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV_CHANNEL::ch_num](#), and [ADC_MEASUREMENT_ADV_CHANNEL::group_index](#).

```

void ADC_MEASUREMENT_ADV_SetAlternateReference ( const AD
                                                const XM
)

```

Selects alternate reference voltage for the channel.

Parameters:

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

reference_select Voltage reference for the channel.

Returns:

None

Description:

Select the reference voltage for conversion. For XMC4000 series, an internal voltage reference (VRef) or an external voltage reference fed to Ch-0 can serve as a voltage reference for conversions. For XMC1000 series, an internal ground reference (Vss) or an external reference ground from CH-0 can serve as an alternate reference. A call to this API would configure the register bit field GxCHCTR.REFSEL.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the
// number of required channels to 2.
// Goto interrupts tab enable request source interrupt.
// Goto the ADC_SCAN APP and enable the request source interrupt.
// Generate the code and build.
// Replace this in the main.c.

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

uint16_t result[2];
void adc_measurement_adv_callback(void)
{
    result[0] = ADC_MEASUREMENT_ADV_GetResult(&ADC_
MEASUREMENT_ADV_0_Channel_A);
    result[1] = ADC_MEASUREMENT_ADV_GetResult(&ADC_
MEASUREMENT_ADV_0_Channel_B);
}
```

```
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\n");
    }

    while(1U)
    {

    }

    // Start the selected request source
    ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

    // connect the alternate reference to the channel-0 pin of Channel_A's group.
    ADC_MEASUREMENT_ADV_SetAlternateReference(&ADC_MEASUREMENT_ADV_0_Channel_A, XMC_VADC_CHANNEL_REF_ALT_CH0);

    while(1U)
    {
        // Continuously re-trigger the scan conversion sequence
        ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);
```

```
    }  
}
```

Definition at line 979 of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV_CHANNEL::ch_num](#), and [ADC_MEASUREMENT_ADV_CHANNEL::group_index](#).

```
void ADC_MEASUREMENT_ADV_SetBoundaryLower ( const ADC_  
                                            uint32_t  
                                            )
```

Sets the lower boundary value for the channel.

Parameters:

handle_ptr	constant pointer to the channel handle structure. (Use the channel handle related macros which are defined in adc_measure_adv_conf.h)
boundary_value	The boundary value that needs to be configured in the lower boundary register.

Returns:

None

Description:

Sets the lower boundary value for the channel. This API will set the lower boundary value depending on boundary selected for the channel in the GUI of the APP. Thus also configured in the API is either the register GLOBBOUND or GxBOUND.

Note:

This API will configure the lower boundary for the channel according to the selection done in the GUI. Runtime change of the lower boundary selection will not be handled by this API.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 1.  
// Select the request source APP as ADC_SCAN.  
// Goto the Boundary Settings tab and select the channel name as Channel_A.  
// Select the Upper boundary as Group boundary 1.  
// Change the channel event from No to "If Result Inside Band".  
// Change the "Generate boundary flag" to Yes/Non-Inverted.  
// Instantiate the interrupt APP.  
// In the UI of the interrupt APP change the interrupt handler to "channel_event_callback"  
// goto HW signal connectivity and connect event_channel_Channel_A to interrupt APP.  
// Generate the code and build.  
// Replace this in the main.c.  
  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
#define MAX_LOCAL_BUFFER (16U)  
uint16_t result[16];  
uint16_t i = 0U;  
  
void channel_event_callback(void)  
{  
    result[i++] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);  
    if(MAX_LOCAL_BUFFER == i)  
    {  
        i = 0U;  
    }  
}
```

```
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\n");
    }

    while(1U)
    {

    }
}

// Set the boundary selection for Channel_A lower Bound as Group Bound-1
ADC_MEASUREMENT_ADV_SetBoundaryLower(&ADC_MEASUREMENT_ADV_0_Channel1_A,1024);

// Set the boundary selection for Channel_A upper Bound as Group Bound-0 and also the value as 2048U
ADC_MEASUREMENT_ADV_SetBoundaryUpper(&ADC_MEASUREMENT_ADV_0_Channel1_A,2048);

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

while(1U)
```

```

    {
        // Continuously re-trigger the scan conversion sequence
        ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);
    }
}

```

Definition at line **1040** of file **ADC_MEASUREMENT_ADV.c**.

References **ADC_MEASUREMENT_ADV_CHANNEL::ch_handle**.

```

void ADC_MEASUREMENT_ADV_SetBoundaryUpper ( const ADC_
                                            uint32_t
                                            )

```

Sets the upper boundary value for the channel.

Parameters:

handle_ptr	constant pointer to the channel handle structure. (Use the channel handle related macros which are defined in adc_measure_adv_conf.h)
boundary_value	The boundary value that needs to be configured in the upper boundary register.

Returns:

None

Description:

Sets the upper boundary value for the channel. This API will set the upper boundary value depending on boundary selected for the channel in the GUI of the APP. Thus also configured in the API is either the register GLOBBOUND or GxBOUND.

Note:

This API will configure the upper boundary for the channel according to the selection done in the GUI. Runtime change of the Upper boundary selection will not be handled by this API.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 1.  
// Select the request source APP as ADC_SCAN.  
// Goto the Boundary Settings tab and select the channel name as Channel_A.  
// Select the Upper boundary as Group boundary 1.  
// Change the channel event from No to "If Result Inside Band".  
// Change the "Generate boundary flag" to Yes/Non-Inverted.  
// Instantiate the interrupt APP.  
// In the UI of the interrupt APP change the interrupt handler to "channel_event_callback"  
// goto HW signal connectivity and connect event_channel_Channel_A to interrupt APP.  
// Generate the code and build.  
// Replace this in the main.c.  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
#define MAX_LOCAL_BUFFER (16U)  
uint16_t result[16];  
uint16_t i = 0U;  
  
void channel_event_callback(void)  
{  
    result[i++] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);  
    if(MAX_LOCAL_BUFFER == i)  
    {
```

```
    i = 0U;
}
}

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\n");
    }

    while(1U)
    {

    }
}

// Set the boundary selection for Channel_A lower Bound as Group Bound-1
ADC_MEASUREMENT_ADV_SetBoundaryLower(&ADC_MEASUREMENT_ADV_0_Channel_A,1024);

// Set the boundary selection for Channel_A upper Bound as Group Bound-0 and also the value as 2048U
ADC_MEASUREMENT_ADV_SetBoundaryUpper(&ADC_MEASUREMENT_ADV_0_Channel_A,2048);

// Start the selected request source
```

```
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_A  
DV_0);  
  
    while(1U)  
    {  
        // Continuously re-trigger the scan conversion sequence  
        ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEAS  
UREMENT_ADV_0);  
    }  
}
```

Definition at line **1027** of file **ADC_MEASUREMENT_ADV.c**.

References **ADC_MEASUREMENT_ADV_CHANNEL::ch_handle**.

```
void ADC_MEASUREMENT_ADV_SetChannelGain ( const ADC_ME  
                                         const ADC_ME  
                                         )
```

Sets the channel gain.

Parameters:

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

gain_factor The gain factor value.

Returns:

None

Description:

Set the gain value for the particular channel. This API would set the SHS gain factor for the channel. The input voltage will get a gain proportional to the selected **gain_factor** then will be converted by the ADC.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.  
// Goto interrupts tab enable request source interrupt.  
// Goto the ADC_SCAN APP and enable the request source interrupt.  
// Generate the code and build.  
// Replace this in the main.c.  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
uint16_t result[2];  
void adc_measurement_adv_callback(void)  
{  
    result[0] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);  
    result[1] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_B);  
}  
  
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\n");  
    }  
}
```

```
    while(1U)
    {
        }

    // Start the selected request source
    ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_A
DV_0);

    // select the gain value of 12
    ADC_MEASUREMENT_ADV_SetChannelGain(&ADC_MEASURE
MENT_ADV_0_Channel_A, ADC_MEASUREMENT_ADV_GAIN_12
);

    while(1U)
    {
        // Continuously re-trigger the scan conversio
n sequence
        ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEAS
UREMENT_ADV_0);
    }
}
```

Definition at line 992 of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV_CHANNEL::ch_handle](#), [ADC_MEASUREMENT_ADV_CHANNEL::ch_num](#), and [ADC_MEASUREMENT_ADV_CHANNEL::group_index](#).

[ADC_MEASUREMENT_ADV_STATUS_t ADC_MEASUREMENT_ADV](#)

Sets the fast conversion value.

Parameters:

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

Returns:

None

Description:

Returns the converted result stored in the result register [GxRESy.RESULT]. In the APP each channel is configured to a particular group result register. If FIFO is enabled for the particular channel this reads the fast compare result from the FIFO tail register. The result register is defined in the channel handle structure **ADC_MEASUREMENT_ADV_CHANNEL_t**. Hence this API shall call be called with a pointer to the channel handle of type **ADC_MEASUREMENT_ADV_CHANNEL_t** (Directly use the channel handle related macros which are defined in adc_measure_adv_conf.h).

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 1.  
// Select the request source APP from ADC_SCAN to ADC_QUEUE.  
// Goto the sequence plan and select Channel_A at position-0 and enable the refill.  
// Enable Wait for trigger for the Channel_A.  
// Goto interrupts tab enable result event for Channel_A.  
// Instantiate the interrupt APP.  
// In the UI of the interrupt APP change the interrupt handler to "result_event_callback"
```

```
// goto HW signal connectivity and connect event_
result_Channel_A to interrupt APP.
// Generate the code and build.
// Replace this in the main.c.
#include <DAVE.h> //Declarations
from DAVE Code Generation (includes SFR declarati
on)

#define MAX_LOCAL_BUFFER (16U)
ADC_MEASUREMENT_ADV_FAST_COMPARE_t result[16];
uint16_t i = 0U;

void result_event_callback(void)
{
    result[i++] = ADC_MEASUREMENT_ADV_GetFastCompar
eResult(&ADC_MEASUREMENT_ADV_0_Channel_A);
    if(MAX_LOCAL_BUFFER == i)
    {
        i = 0U;
    }
}

int main(void)
{
    DAVE_STATUS_t status;

    status = DAVE_Init(); // Initializati
on 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\n"
);
```

```

    while(1U)
    {
        }

    // Set the threshold value as Vdd/2
    ADC_MEASUREMENT_ADV_SetFastCompareValue(&ADC_MEASUREMENT_ADV_0_Channel_A, 512U);

    // Start the selected request source
    ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

    while(1U)
    {
        // Continuously trigger the queue conversion sequence
        ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);
    }
}

```

Definition at line [921](#) of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV_STATUS_FAILURE](#), [ADC_MEASUREMENT_ADV_STATUS_SUCCESS](#), [ADC_MEASUREMENT_ADV_CHANNEL::ch_handle](#), and [ADC_MEASUREMENT_ADV_CHANNEL::group_index](#).

void ADC_MEASUREMENT_ADV_SetIclass (const ADC_MEASURE

Enables uniform conversion configurations across slaves.

Parameters:

handle_ptr constant pointer to the channel handle structure.

(Use the channel handle related macros which are defined in adc_measure_adv_conf.h)

Returns:

None

Description:

Enables uniform conversion configurations across slaves. The ADC_QUEUE configures the input class settings for the master group. When slaves need to convert the input signals at the same configuration as the master group then a call to this API is needed. A call to this API will ensure that the master and the slave channels are converting the input signals at the same resolution and sampling time.

Note:

- ADC_ADVANCE_MEASURE APP will configure the input class used by ADC_QUEUE or ADC_SCAN in the channel configuration. Thus the input class is either GxICLASS[0] or GxICLASS[1]. The slaves channels are also configured by the same input class number in the GxCHCTR.ICLSEL. A call to this API will copy the configurations from the master groups input class(GxICLASS[z]) to the slaves groups input class (GyICLASS[z], where in x is the master group, y is the slave group and z is the input class number which is common across master and slave).
- The channel iclass can be changed to global iclass at any time by calling runtime APIs. This API cannot support such a situation.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 1.  
// Select the request source APP from ADC_SCAN to ADC_QUEUE.  
// Goto the sequence plan and select Channel_A at
```

```
position-0 and enable the refill.  
// Enable Wait for trigger for the Channel_A.  
// Goto the Sync. Conversion tab and select the synchronized groups as 1 slave.  
// Goto the Sync Master check box for the Channel_A and enable it.  
// Goto interrupts tab enable request source interrupt.  
// Goto the ADC_QUEUE APP and enable the request source interrupt.  
// Generate the code and build.  
// Replace this in the main.c.  
  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
#define MAX_LOCAL_BUFFER (16U)  
uint16_t result[16],sync_result[16];  
uint16_t i = 0U;  
  
void adc_measurement_adv_callback(void)  
{  
    result[i++] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);  
    sync_result[i++] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A_SLAVE_A);  
    if(MAX_LOCAL_BUFFER == i)  
    {  
        i = 0U;  
    }  
}  
  
int main(void)  
{  
    DAVE_STATUS_t status;  
  
    status = DAVE_Init(); // Initialization
```

```
on 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\n");
    }

    while(1U)
    {

    }
}

//Set the same conversion characteristics for the slave groups channels as well
ADC_MEASUREMENT_ADV_SetIcclass(&ADC_MEASUREMENT_ADV_0);

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

while(1U)
{
}
}
```

Definition at line [1090](#) of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV::group_index](#).

void ADC_MEASUREMENT_ADV_SetSubtractionValue (const ADC

```
ADC_MEASUREMENT_ADV_APP_t
{
    uint16_t subtraction_value;
}
```

Set the subtraction value if a result difference mode is required.

Parameters:

handle_ptr	constant pointer to the APP handle structure
subtraction_alignment	The result alignment in the result register.
subtraction_value	constant value that is used for subtraction.

Returns:

None

Description:

Set the subtraction value in the result register 0 [GxRES[0].RESULT]. In the subtraction mode the result register 0 is used as the subtrahend. Any channel in the GUI which has selected the subtraction mode will have its converted value subtracted from the value stored in the result register-0. This can be used as an offset for the converted values.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.
// Select the request source APP from ADC_SCAN to ADC_QUEUE.
// Goto the sequence plan and select Channel_A at position-0 and Channel_B at position-1.
// Enable Wait for trigger for the Channel_A.
// Goto interrupts tab enable request source interrupt.
```

```
// Goto the ADC_QUEUE APP and enable the request
source interrupt.
// Generate the code and build.
// Replace this in the main.c.
#include <DAVE.h> //Declarations
from DAVE Code Generation (includes SFR declarati
on)

#define VOLTAGE_OFFSET (50U)
uint16_t result[2];
void adc_measurement_adv_callback(void)
{
    result[0] = ADC_MEASUREMENT_ADV_GetResult(&ADC_
MEASUREMENT_ADV_0_Channel_A);
    result[1] = ADC_MEASUREMENT_ADV_GetResult(&ADC_
MEASUREMENT_ADV_0_Channel_B);
}

int main(void)
{
    DAVE_STATUS_t status;

    status = DAVE_Init(); // Initializati
on 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\n"
);

        while(1U)
        {

        }
    }
}
```

```

    }

    // Start the selected request source
    ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_A
DV_0);

    ADC_MEASUREMENT_ADV_SetSubtractionValue(&ADC_ME
ASUREMENT_ADV_0, ADC_MEASUREMENT_ADV_SUBTRACTION_12
BIT_RIGHT_ALIGN,
                                         VOLTAG
E_OFFSET);

    // Start the queue conversion sequence
    ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUR
EMENT_ADV_0);

    while(1U)
    {
    }
}

```

Definition at line [941](#) of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV::group_index](#).

void ADC_MEASUREMENT_ADV_SoftwareTrigger (const ADC_ME

Starts the conversion of the required channels.

Parameters:

handle_ptr constant pointer to the APP handle structure

Returns:

None

Description:

Trigger a load event for the required channels thus starting the conversion of the ADC channels. If scan request source is selected then this API would write to GxASMR.LDEV bit, causing the conversion to start. If queue request source is selected then this API would write to GxQMR0.TREV bit. At the time of writing to TREV bit for the queue. If there was queue entry waiting in the queue buffer for a hardware trigger, writing to TREV bit triggers the conversion for that entry.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.  
// Goto interrupts tab enable request source interrupt.  
// Goto the ADC_SCAN/ADC_QUEUE APP and enable the request source interrupt.  
// Generate the code and build.  
// Replace this in the main.c.  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
uint16_t result[2];  
void adc_measurement_adv_callback(void)  
{  
    result[0] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);  
    result[1] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_B);  
}  
  
int main(void)  
{  
    DAVE_STATUS_t status;  
  
    status = DAVE_Init(); // Initialization
```

```

on 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\n");
    }

    while(1U)
    {

    }
}

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_A
DV_0);

while(1U)
{
    // Continuously re-trigger the scan conversio
    n sequence
    ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEAS
UREMENT_ADV_0);
}
}

```

Definition at line [718](#) of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV::group_index](#), and [ADC_MEASUREMENT_ADV::req_src](#).

void ADC_MEASUREMENT_ADV_StartADC (const ADC_MEASURE

Starts the VADC request source.

Parameters:

handle_ptr constant pointer to the APP handle structure

Returns:

None

Description:

This API would start the request source of the VADC. After this API is invoked the ADC is ready for accepting conversion requests. If scan request source is selected then this API would write to GxARBPR.ASEN1 bit. If queue request source is selected then this API would write to GxARBPR.ASEN0 bit. At the time of writing to ASEN0 bit for the queue, If there was queue entry waiting in the queue buffer without waiting for a hardware trigger, this API would start the conversion of such a queue.

Example Usage:

```
// Initialize the ADC_MEASUREMENT_ADV APP. Set the number of required channels to 2.  
// Goto interrupts tab enable request source interrupt.  
// Goto the ADC_SCAN/ADC_QUEUE APP and enable the request source interrupt.  
// Generate the code and build.  
// Replace this in the main.c.  
#include <DAVE.h> //Declarations  
from DAVE Code Generation (includes SFR declaration)  
  
uint16_t result[2];  
void adc_measurement_adv_callback(void)  
{
```

```
    result[0] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_A);
    result[1] = ADC_MEASUREMENT_ADV_GetResult(&ADC_MEASUREMENT_ADV_0_Channel_B);
}

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\n");
    }

    while(1U)
    {

    }
}

// Start the selected request source
ADC_MEASUREMENT_ADV_StartADC(&ADC_MEASUREMENT_ADV_0);

while(1U)
{
    // Continuously re-trigger the scan conversion sequence
    ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT_ADV_0);
```

```
    }  
}
```

Definition at line 695 of file [ADC_MEASUREMENT_ADV.c](#).

References [ADC_MEASUREMENT_ADV::group_index](#), and
[ADC_MEASUREMENT_ADV::req_src](#).

Referenced by [ADC_MEASUREMENT_ADV_Init\(\)](#).

ADC_MEASUREMENT_ADV

Home

Usage

Usage

The following examples demonstrate some of the use cases of the **ADC_MEASUREMENT_ADV APP**.

1. **Example to illustrate the autoscan mode (Software start continuous mode).**
2. **Example to illustrate the software triggered conversions (Software start single shot mode).**
3. **Example to illustrate the hardware triggered conversions (Hardware trigger single shot mode).**
4. **Example to illustrate synchronous conversions.**

Use case 1: The use case illustrates the use of the "autoscan mode" (continuous conversion). An API call starts the conversion round. After one conversion round (2 Channels) the round is automatically repeated. Hence no additional start of the conversion round is necessary. The result is cautiously read in the main loop.

In this example the "ADC_SCAN_APP" is used. See following configuration.

Note: The conversion result is read continuously and it is independent of ADC result being ready.

The use case can also be handled with "ADC_QUEUE_APP" by disabling all "Wait for trigger" in the "Sequence Plan" tab.

Note: This configuration will immediately start the conversion after the ADC start (GUI checkbox "Start after initialization" or API for start ADC). This use case is not shown in this example.

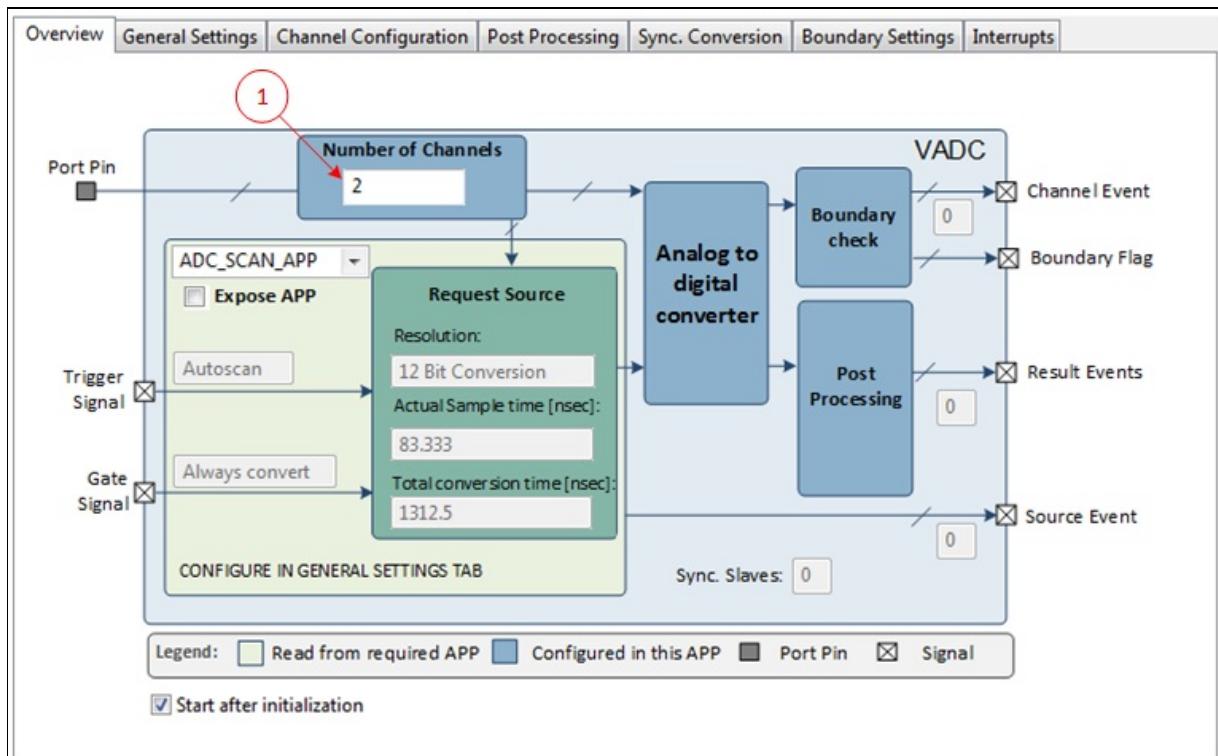
Instantiate the required APPs

Drag an instance of **ADC_MEASUREMENT_ADV** APP. Update the fields in the GUI of the APP with the following configuration.

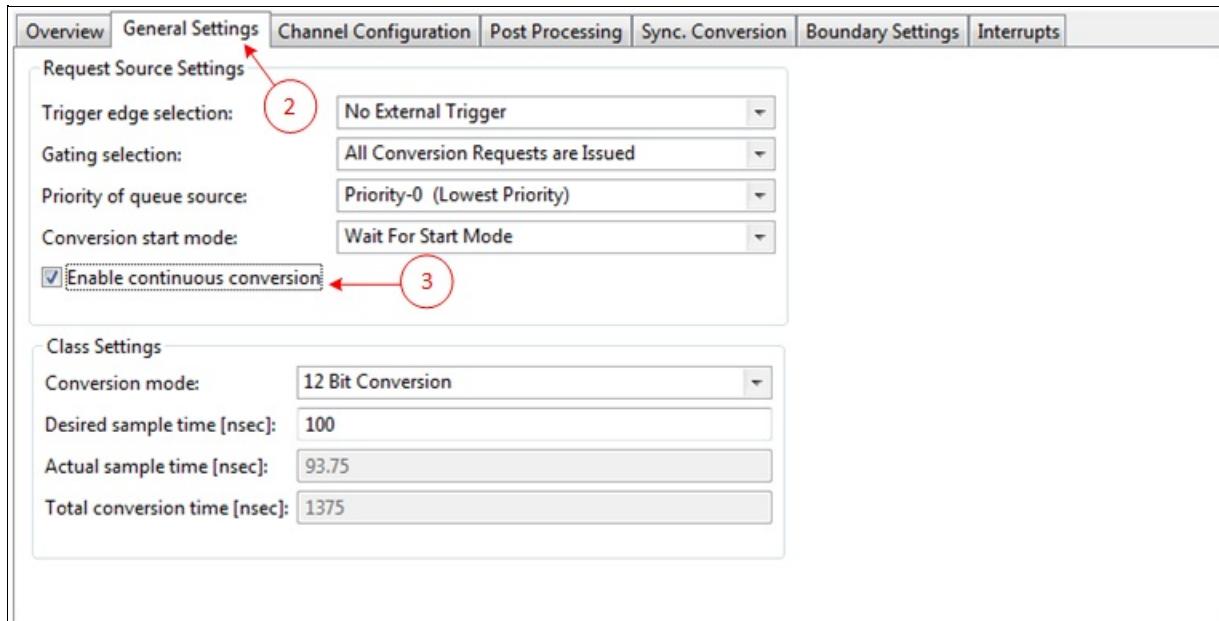
Configure the APP:

ADC_MEASUREMENT_ADV_0 APP:

1. Set the number of channels to 2 .

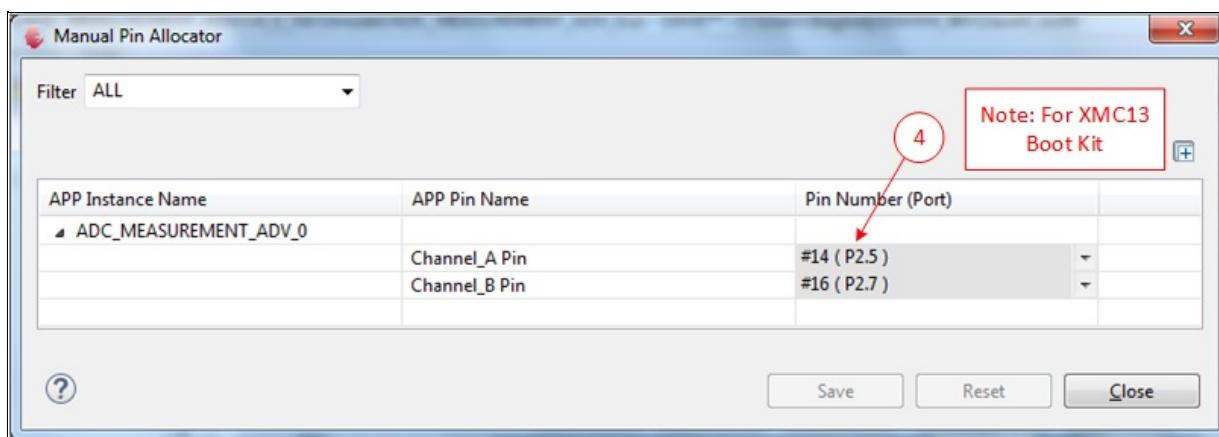


2. Goto General settings Tab.
3. Check the "Enable continuous Conversion" check box.



Manual pin allocation

4. Select the potentiometer Pin present in the boot kit and also another pin in available for the select board.
- 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 DAVE Code Generation (includes SFR declaratio
n)

int main(void)
{
    DAVE_STATUS_t status;
    uint16_t resultA, resultB;

    status = DAVE_Init();                          /* Initializati
on 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\n"
    );

    while(1U)
    {
```

```

    }

}

// One Software trigger to start the conversion
of the 2 channels.
// After each conversion round(i.e after 2 chan-
nels) the conversion will be automatically repeate-
d.
ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUR-
EMENT_ADV_0);
while(1U)
{
    //continuously read the result value.
    //NOTE: The result read doesn't wait for the
new result to be ready.
    resultA = ADC_MEASUREMENT_ADV_GetResult(&ADC_
MEASUREMENT_ADV_0_Channel_A);
    resultB = ADC_MEASUREMENT_ADV_GetResult(&ADC_
MEASUREMENT_ADV_0_Channel_B);
    XMC_UNUSED_ARG(resultA);
    XMC_UNUSED_ARG(resultB);
}
}

```

Build and Run the Project

Observation

The measurement is started with the **ADC_MEASUREMENT_ADV_SoftwareTrigger()** API call. The conversion results are read continuously and stored in "resultA" and "resultB" variable.

Use case 2:

The use case illustrates the software triggered conversions. In this use case a sequence of 2 ADC Channels will be converted. The conversion round is started with an API. After one conversion round (2 Channels)

the conversion is stopped and waits for the next software start. After each conversion round a result event is generated and the ADC results are read in the ISR. After the results have been read, a new conversion is started with an API call.

In this example the "ADC_QUEUE_APP" is used. See following configuration.

The use case can also be handled with "ADC_SCAN_APP". This use case is not shown in this example.

Note: This usecase uses an INTERRUPT APP for the result events.

For those configurations refer to INTERRUPT APP help.

Note: In "ADC_SCAN_APP" the conversion sequence depends on the channels consumed. If Channel_A consumes a lower channel number than Channel_B then Channel_B would be converted first. Hence care needs to be taken when a similar setup is done while using ADC_SCAN_APP.

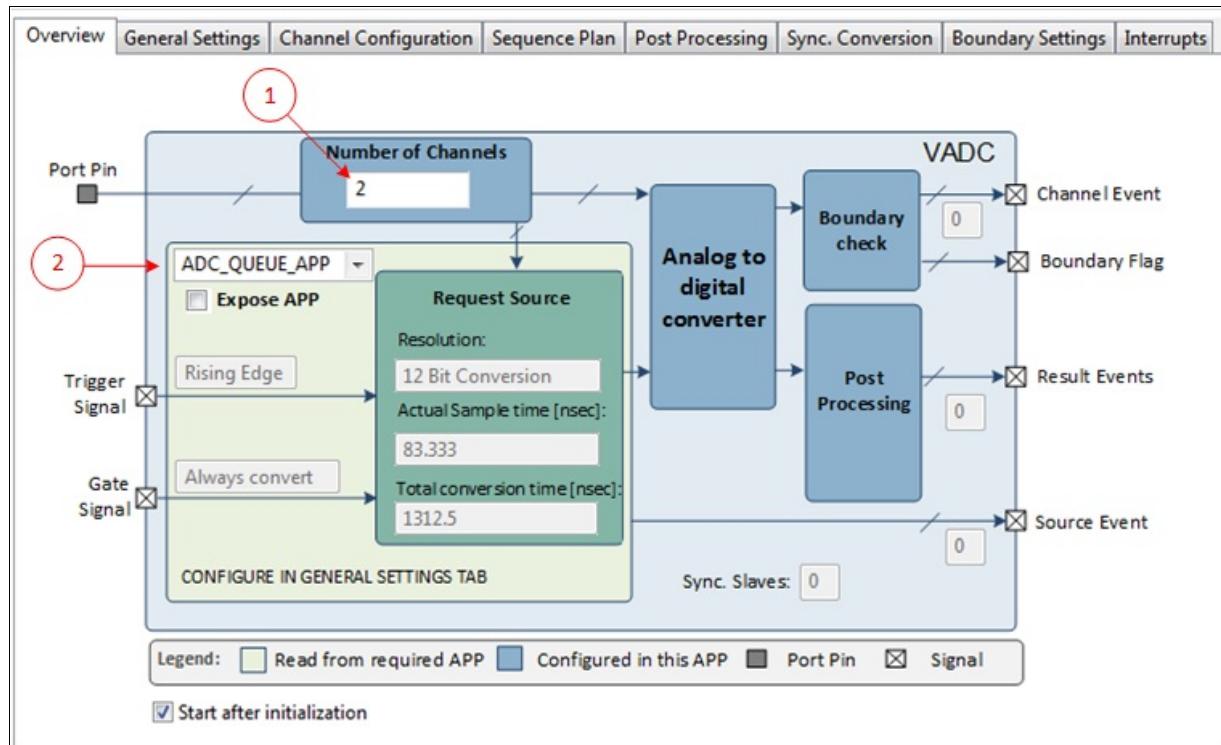
Instantiate the required APPs

Drag an instance of **ADC_MEASUREMENT_ADV** APP and one instance of INTERRUPT APP. Update the fields in the GUI of the APP with the following configuration.

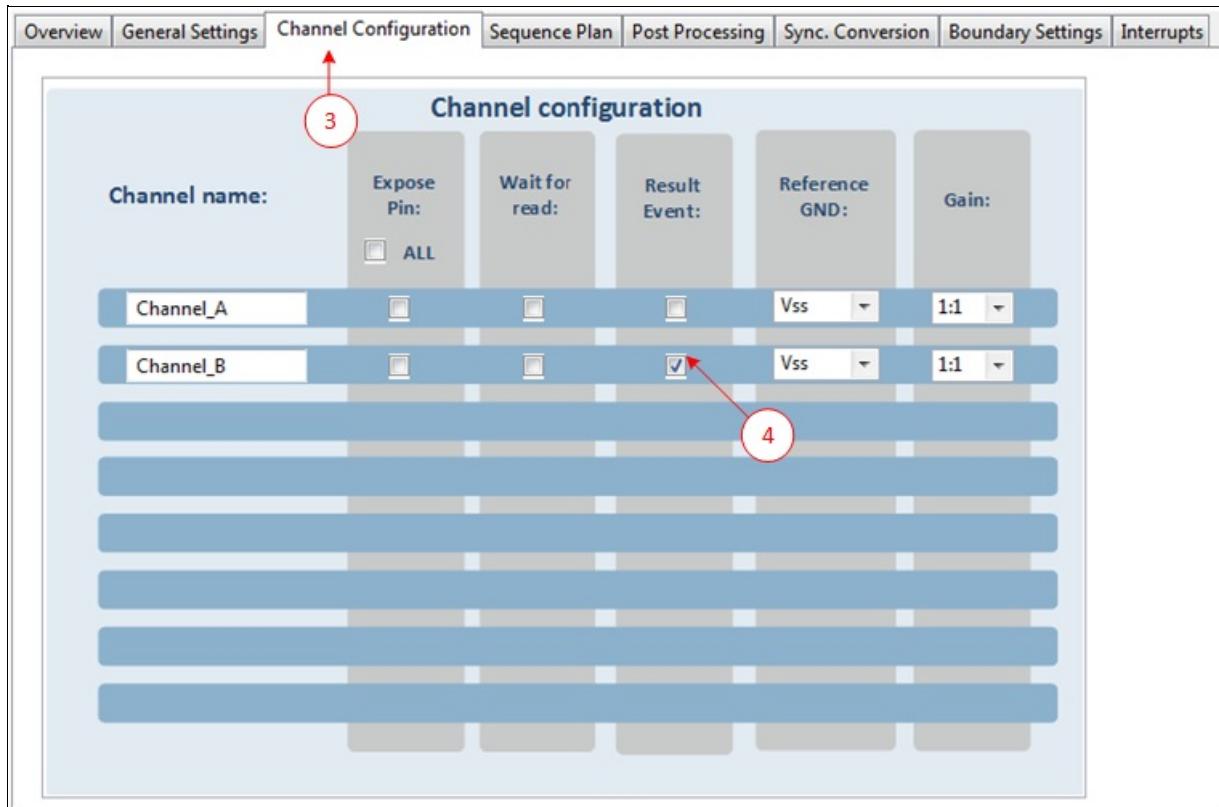
Configure the APP:

ADC_MEASUREMENT_ADV_0 APP:

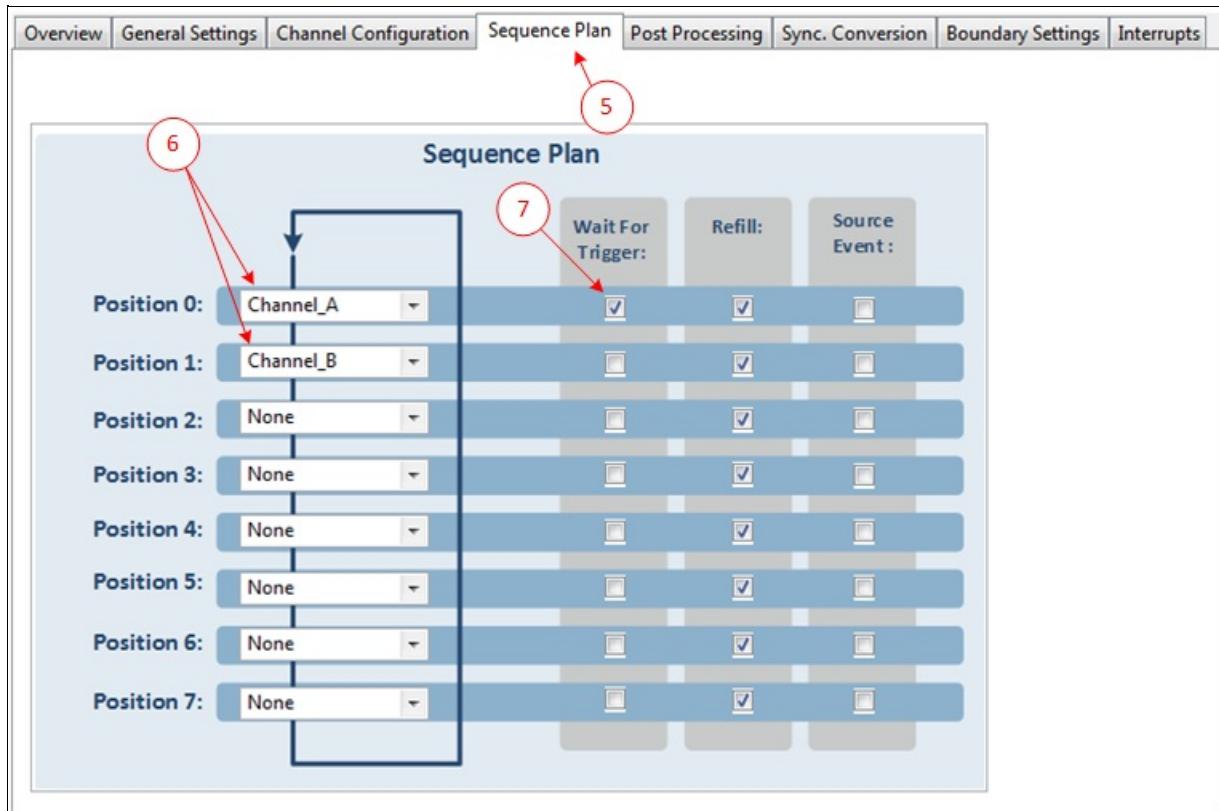
1. Set the number of channels to 2 .
2. Select the request source as ADC_QUEUE_APP .



3. Goto Channel configuration Tab.
4. Enable the result event for Channel_B.



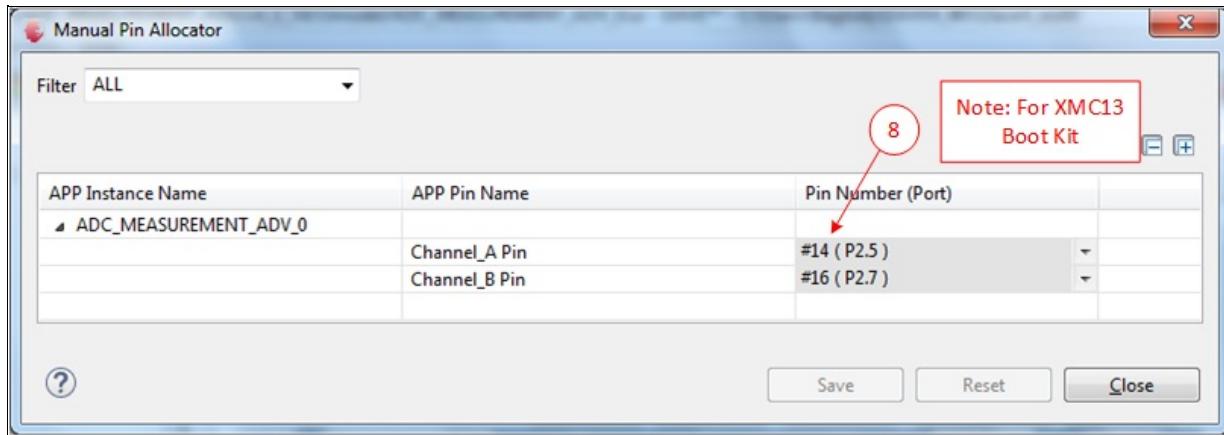
5. Goto Sequence Plan Tab.
6. Select the Queue Position-0 as Channel-A, Position-1 as Channel_B.
7. Select "Wait for Trigger" for Channel-A.



Manual pin allocation

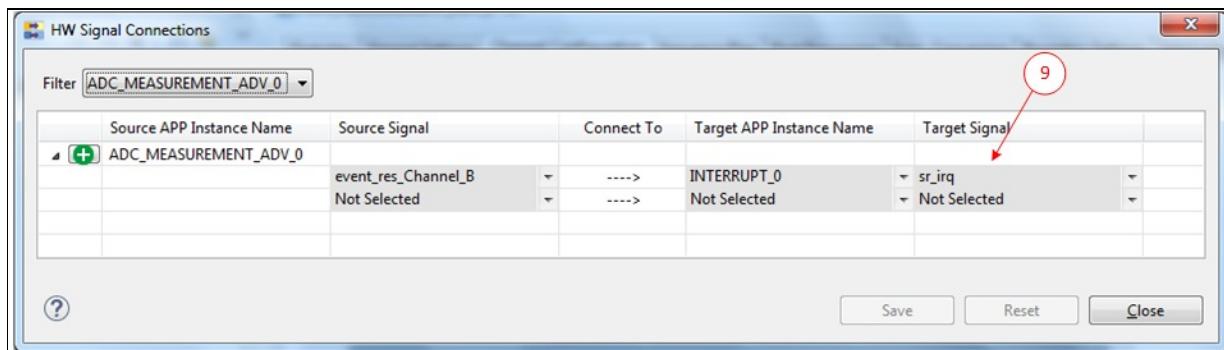
8. Select the potentiometer Pin present in the boot kit and also another pin in available for the select board.

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.



HW Signal Connections

9. Connect the ADC_MEASUREMENT_ADV_0 event_res_Channel_B signal to INTERRUPT_0 sr_irq



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 DAVE Code Generation (includes SFR declaratio
n)

uint16_t resultA,resultB;

// Result event for Channel_B
void UserIRQHandler(void)
{
    // read the results of the conversion
    resultA = ADC_MEASUREMENT_ADV_GetResult(&AD
C_MEASUREMENT_ADV_0_Channel_A);
    resultB = ADC_MEASUREMENT_ADV_GetResult(&AD
C_MEASUREMENT_ADV_0_Channel_B);

    //Start the next round of conversion
    ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_ME
ASUREMENT_ADV_0);
}

int main(void)
{
    DAVE_STATUS_t status;

    status = DAVE_Init();                          /* Initializati
on 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\n"
    }
}
```

```

);

    while(1U)
    {

    }

    // One Software trigger to start the conversion
    // of the 2 channels.
    ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUR
MENT_ADV_0);
    while(1U)
    {

    }
}

```

Build and Run the Project

Observation

The measurement is started with the

[ADC_MEASUREMENT_ADV_SoftwareTrigger\(\)](#) API call. After Channel_B is converted an event is triggered and executes the ISR UserIRQHandler(). In this ISR the results of Channel_A and Channel_B are stored in "resultA" and "resultB" variables. The API call ADC_MEASUREMENT_ADV_SoftwareTrigger(&ADC_MEASUREMENT in the ISR starts a new conversion round.

Use case 3:

The use case illustrates hardware triggered conversions. A hardware trigger starts the conversion round of 3 Channels. When the conversion round is finished a source interrupt is executed. In the ISR the results are read. The ADC waits now for a new hardware trigger.

In this example the "ADC_QUEUE_APP" is used, this allows a user defined sequence. See following configuration.

The use case can also be handled with "ADC_SCAN_APP". In that case it is mandatory to disable the "autoscan mode". Also the "result event" can be used to read the ADC results. Both the use cases are not handled in this example.

Note: The 10kHz hardware trigger is generated with a PWM APP. For PWM configuration refer to the PWM APP help.

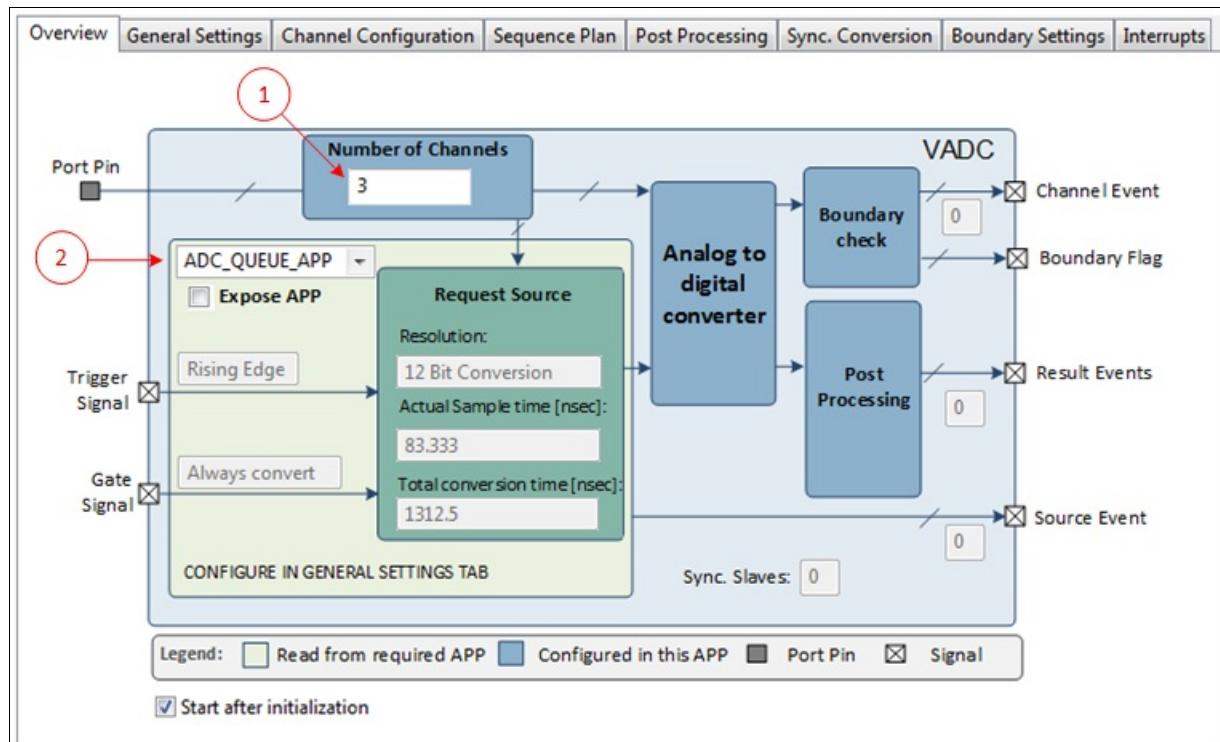
Instantiate the required APPs

Drag an instance of **ADC_MEASUREMENT_ADV** APP and one instance of PWM APP. Update the fields in the GUI of the APP with the following configuration.

Configure the APP:

ADC_MEASUREMENT_ADV_0 APP:

1. Set the number of channels to 3 .
2. Set the request source to ADC_QUEUE_APP .



3. Goto General Settings Tab.

4. Select the Trigger edge as Rising Edge.

The screenshot shows the 'Channel Configuration' tab selected in a software interface. The 'Request Source Settings' section contains the following configuration:

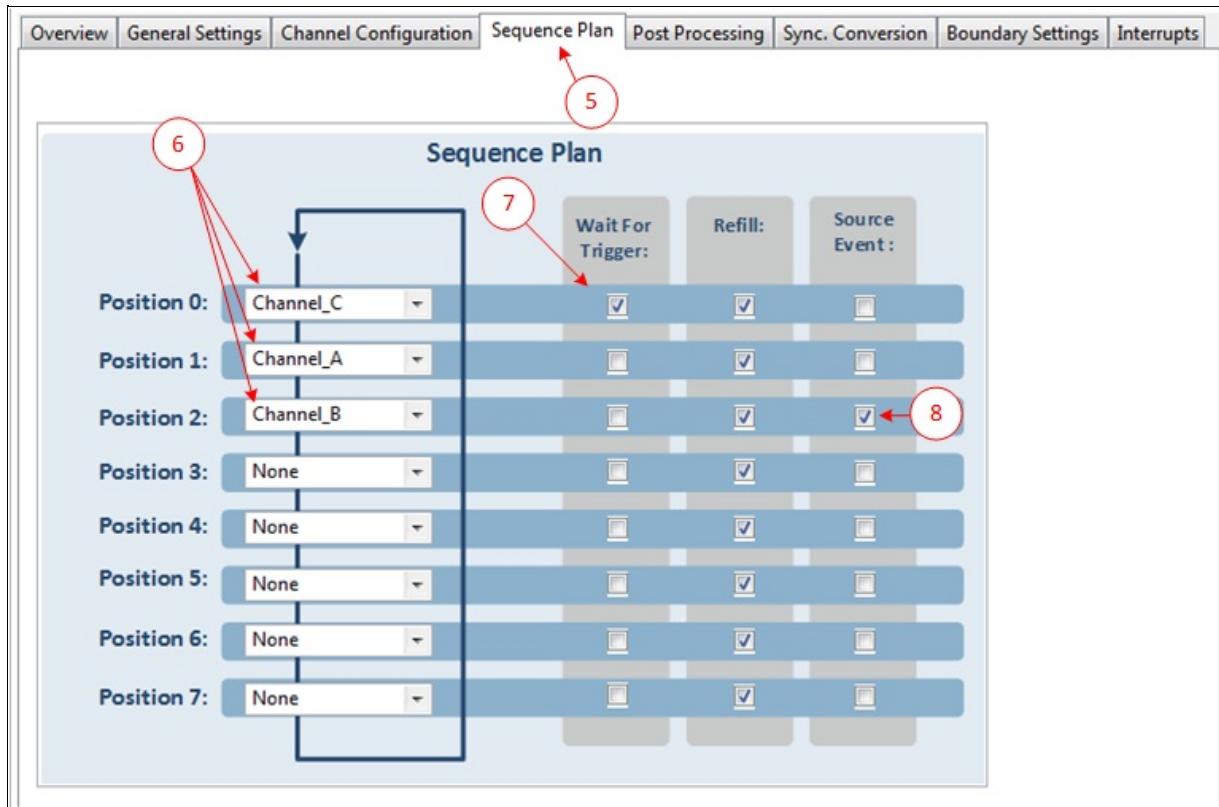
- Trigger edge selection: External Trigger Upon Rising Edge (highlighted with a red circle labeled '3')
- Gating selection: All Conversion Requests are Issued (highlighted with a red arrow labeled '4')
- Priority of queue source: Priority-0 (Lowest Priority)
- Conversion start mode: Wait For Start Mode

Below this, the 'Class Settings' section includes:

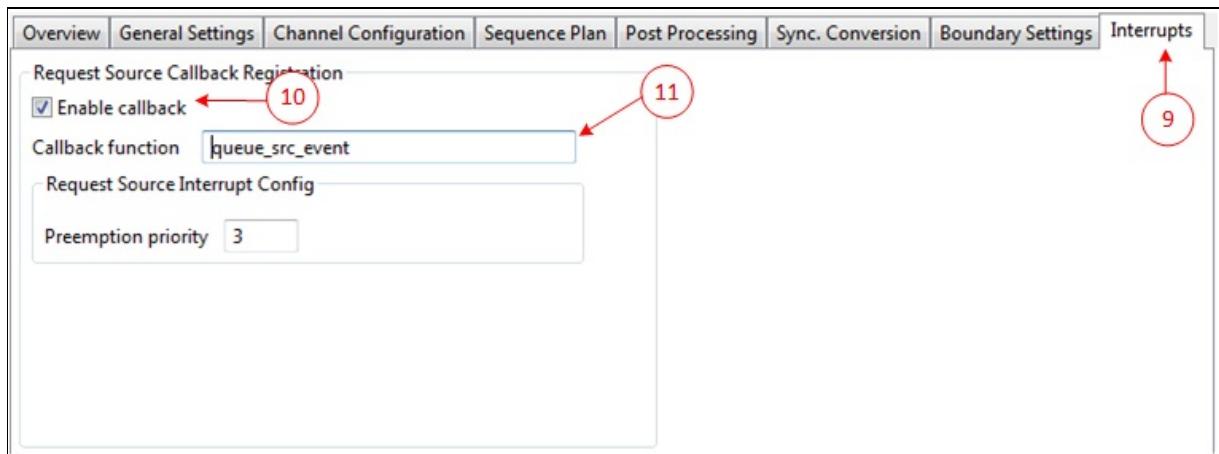
- Conversion mode: 12 Bit Conversion
- Desired sample time [nsec]: 100
- Actual sample time [nsec]: 83.333
- Total conversion time [nsec]: 1312.5

A note at the bottom states: "Note: Total conversion time is always calculated with post calibration enabled."

5. Goto Sequence Plan Tab.
6. Select the Queue Position-0 as Channel-C, Position-1 as Channel-A, Position-2 as Channel_B.
7. Select "Wait for Trigger" for Channel-C.
8. Select "Source Event" for Channel-B.



9. Goto Interrupts Tab.
10. Check the "Enable callback" check box.
11. Change the Callback function to queue_src_event.

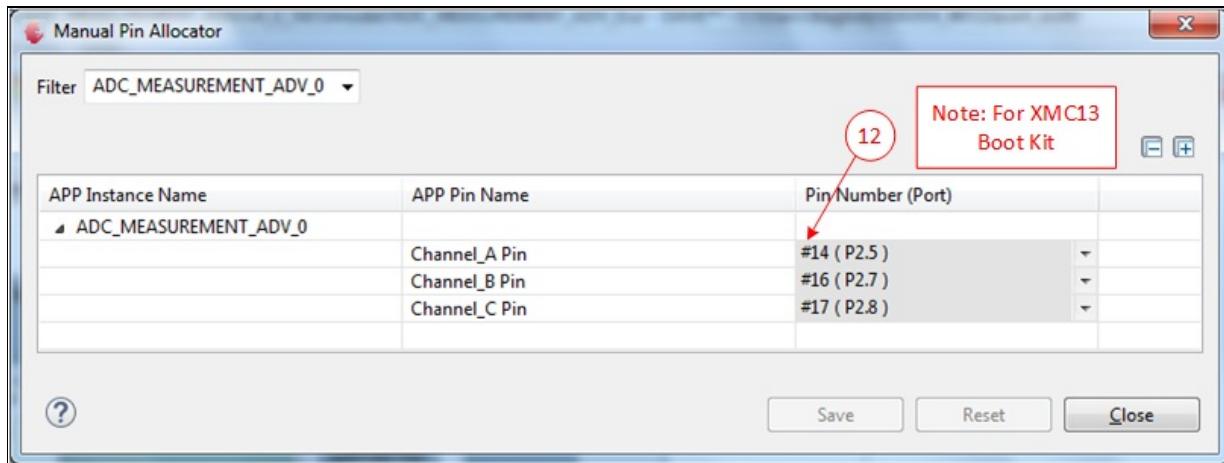


Manual pin allocation

12. Select the potentiometer Pin present in the boot kit and also the

other pins which are available for the select board.

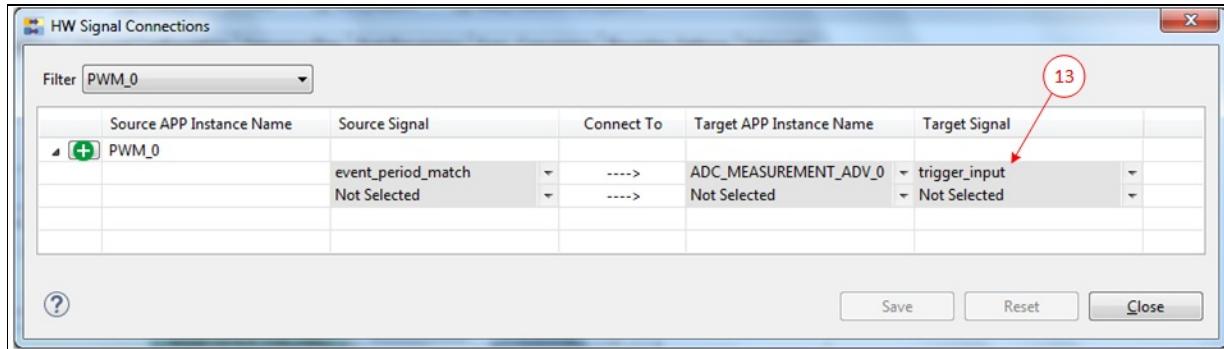
Note: The pin number is specific to the development board chosen to run this example. The pin shown in the image above may not be available on every XMC boot kit. Ensure that a proper pin is selected according to the board.



HW Signal Connections

5. Connect the PWM_0 event_period_match signal to ADC_MEASUREMENT_ADV_0 trigger_input.

Note: Ensure that the period match event in the PWM_0 is enabled.



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 DAVE Code Generation (includes SFR declaratio
n)

uint16_t resultA, resultB, resultC;

// Request source event for Channel_B
void queue_src_event(void)
{
    // read the results of the conversion
    resultA = ADC_MEASUREMENT_ADV_GetResult(&ADC_ME
ASUREMENT_ADV_0_Channel_A);
    resultB = ADC_MEASUREMENT_ADV_GetResult(&ADC_ME
ASUREMENT_ADV_0_Channel_B);
    resultC = ADC_MEASUREMENT_ADV_GetResult(&ADC_ME
ASUREMENT_ADV_0_Channel_C);
}

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\n");
}

while(1U)
{
}

// Start the PWM trigger.
PWM_Start(&PWM_0);
while(1U)
{
}

}

```

Build and Run the Project

Observation

The measurement is started after initialization and waits for a hardware trigger. With a hardware trigger Channel_C, Channel_A and Channel_B are converted in this sequence. The source event for Channel_B is enabled and execute the ISR queue_src_event(void). In this ISR the results of Channel_A, Channel_B and Channel_C are stored in "resultA", "resultB" and "resultC" variables. A new sequence is started with the next hardware trigger

Use case 4:

The use case illustrates synchronous conversions.

The synchronous conversion is used when one or more slaves are

busy while the master starts a request. In this case the master group cancel the slave group conversion, wait until all slaves are ready and starts a conversion. This provides an exact synchronous conversion.

In this use case one **ADC_MEASUREMENT_ADV** APPs are used. The APP is used to configure the master which is controlled by a hardware trigger. The result event of the master is generating an interrupt where both results are read.

The **ADC_MEASUREMENT_ADV** APP for master is using the "ADC_QUEUE_APP" with a hardware trigger input. A Sync. Conversion with one slave is configured, see following configuration. The **ADC_MEASUREMENT_ADV** APP for master is triggered by a hardware trigger. **See Use Case 3 for configuration.**

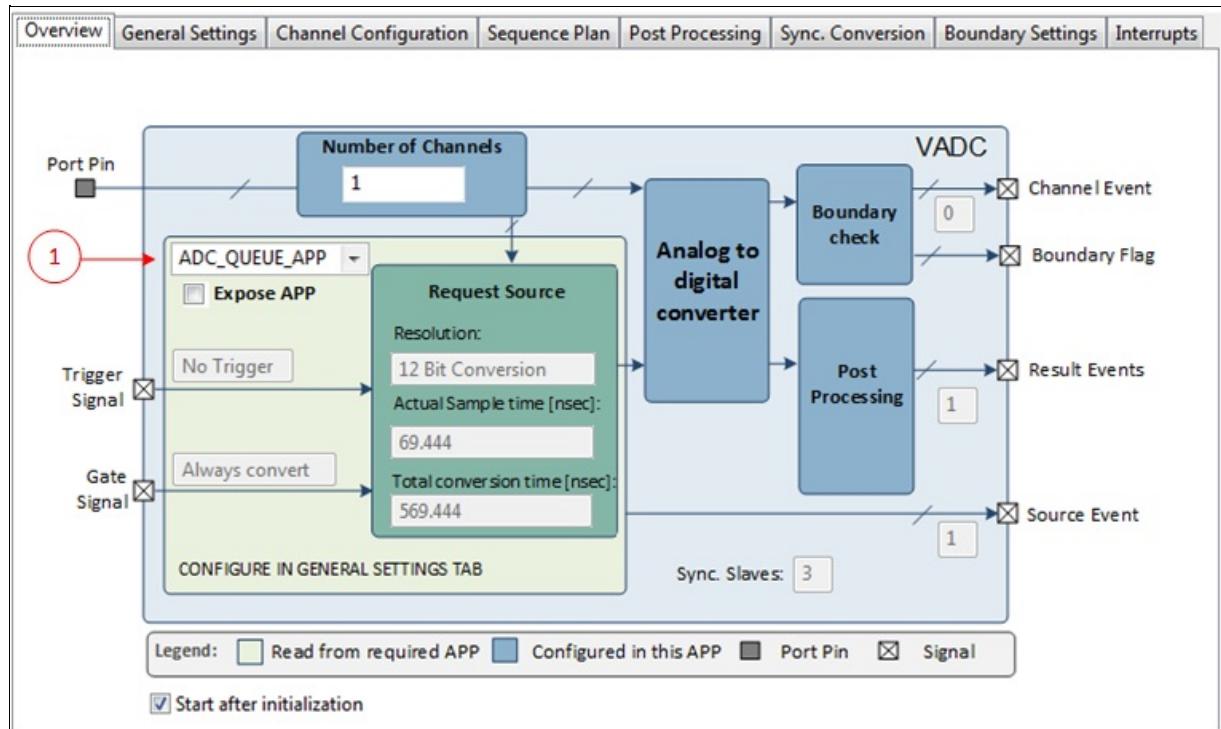
Instantiate the required APPs

Drag an instance of **ADC_MEASUREMENT_ADV** APP, one instance of PWM APP and INTERRUPT APP. Update the fields in the GUI of the APP with the following configuration.

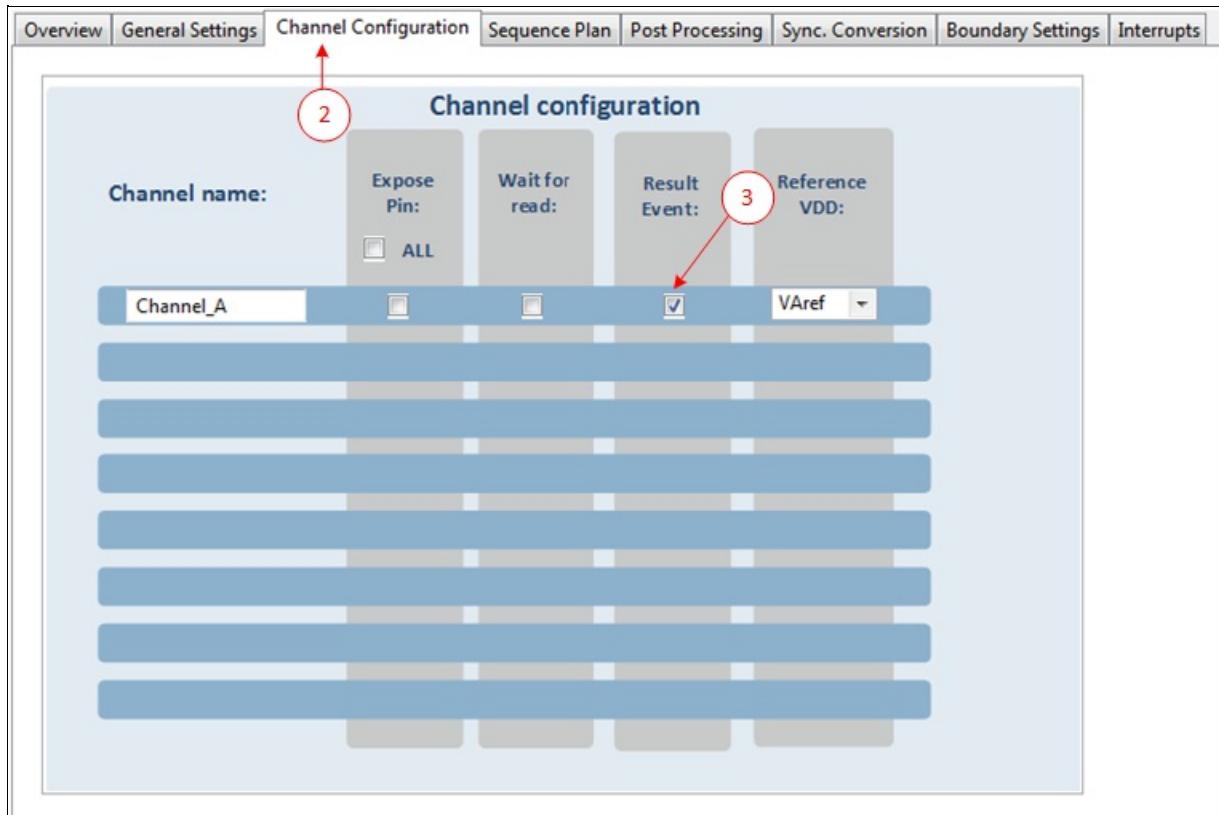
Configure the APP:

ADC_MEASUREMENT_ADV_0 APP:

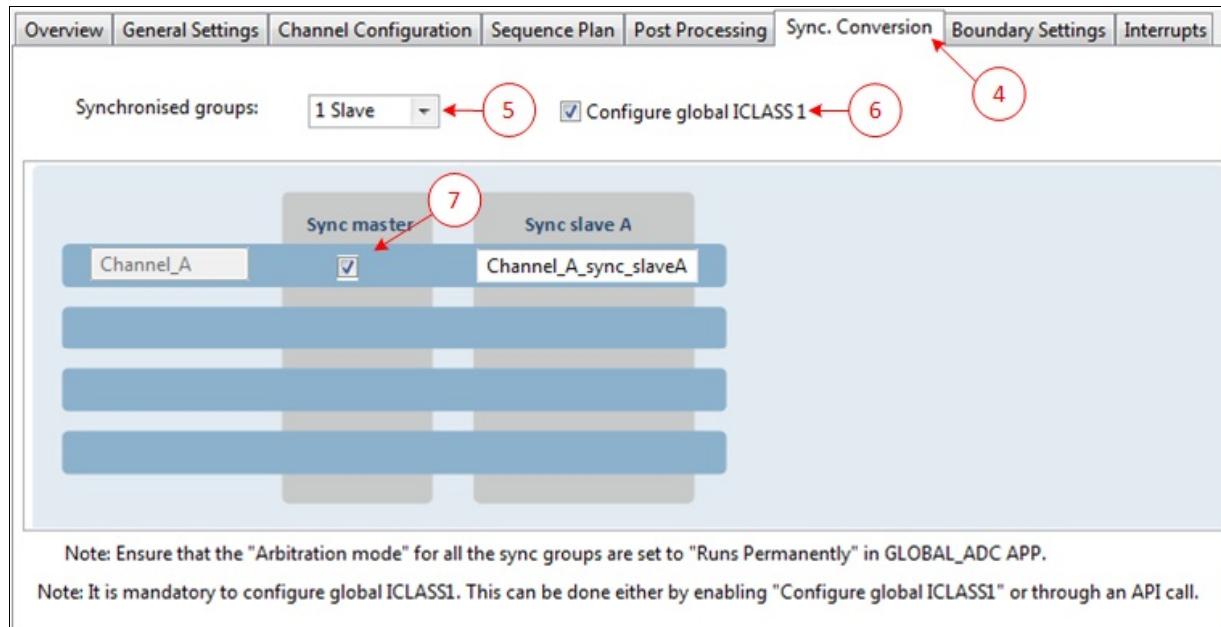
1. Set the request source to ADC_QUEUE_APP .



2. Goto Channel configuration Tab.
3. Enable the result event for the Channel_A.



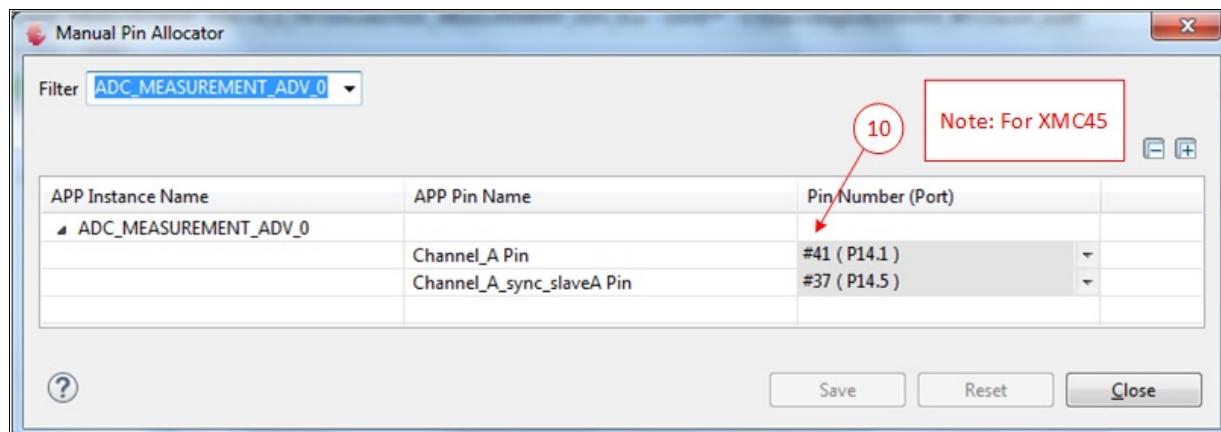
9. Goto Sync. conversion Tab.
10. Select the synchronized groups to "1 Slaves" .
11. Enable the Configure global ICLASS-1.
12. Enable the sync master checkbox for Channel_A.



Manual pin allocation

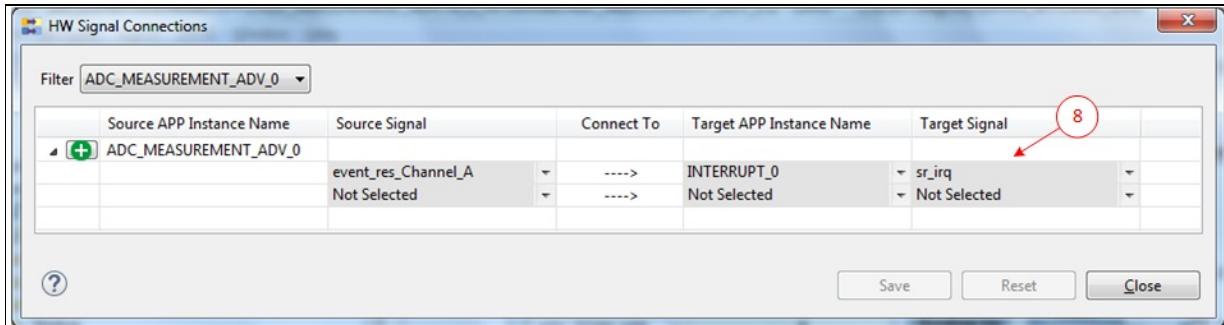
12. Select the potentiometer Pin present in the boot kit and also the other pins which are available for the select board.

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.

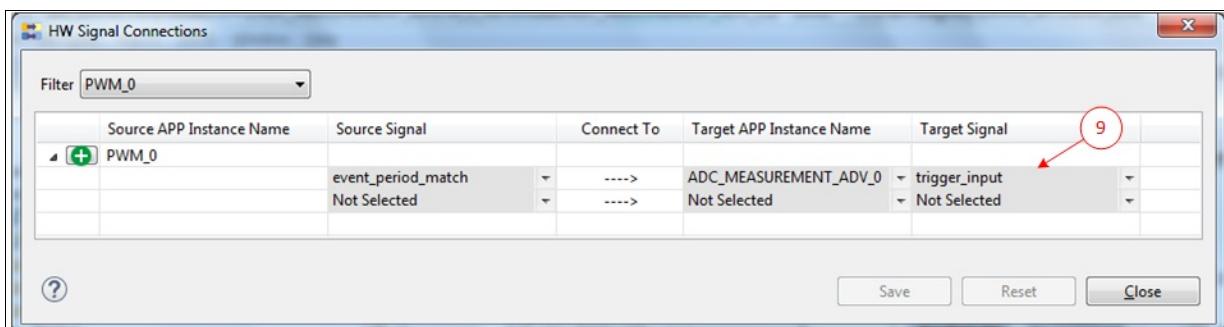


HW Signal Connections

5. Connect the PWM_0 event_period_match signal to ADC_MEASUREMENT_ADV_0 trigger_input.
Note: Ensure that the event_period_match in the PWM_0 is enabled.



5. Connect the ADC_MEASUREMENT_ADV_0 event_res_Channel_A signal to INTERRUPT_0 sr_irq.



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 DAVE Code Generation (includes SFR declaratio
n)

uint16_t resultA,result_slaveA,result_slaveB,resu
lt_slaveC;

// Result event for Channel_A
void UserIRQHandler(void)
{
    // read the results of the conversion
    resultA = ADC_MEASUREMENT_ADV_GetResult(&ADC_ME
ASUREMENT_ADV_0_Channel_A);
    result_slaveA = ADC_MEASUREMENT_ADV_GetResult(&
ADC_MEASUREMENT_ADV_0_Channel_A_sync_slaveA);
}

int main(void)
{
    DAVE_STATUS_t status;

    status = DAVE_Init();                            /* Initializati
on 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\n"
    );

    while(1U)
```

```
{  
}  
}  
  
// start the trigger signal for the ADC_MEASURE  
MENT_ADV_0  
PWM_Start(&PWM_0);  
  
while(1U)  
{  
}  
}  
}
```

Build and Run the Project

Observation

The voltage at sync pins are converted for each rising edge of the PWM signal. The conversion results are stored in "resultA" and "result_slaveA"variables.



ADC_MEASUREMENT_ADV

Home

Release History

Release History



ADC_MEASUREMENT_ADV

Home	
Data Structures	Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

ADC_MEASUREMENT_ADV	
ADC_MEASUREMENT_ADV_CHANNEL	
ADC_MEASUREMENT_ADV_NVIC_CONFIG	NVIC Configuration structure for request source interrupt
ADC_MEASUREMENT_ADV_QUEUE	Configuration Data structure of queue request source
ADC_MEASUREMENT_ADV_SCAN	Configuration Data structure of scan request source

ADC_MEASUREMENT_ADV

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

ADC_MEASUREMENT_ADV Struct Reference

Detailed Description

Structure to configure **ADC_MEASUREMENT_ADV APP**.

Definition at line **286** of file **ADC_MEASUREMENT_ADV.h**.

```
#include <ADC_MEASUREMENT_ADV.h>
```

Data Fields

const	
ADC_MEASUREMENT_ADV_CHANNEL_t	channel_array
**const	
ADC_MEASUREMENT_ADV_EVENT_CONFIG_t	event_config
ADC_MEASUREMENT_ADV_STATUS_t *	init_state
ADC_MEASUREMENT_ADV_REQUEST_SOURCE_t	req_src
const uint8_t	group_index
const uint8_t	total_number_
const uint8_t	total_number_
const bool	start_at_initial
const bool	configure_glo
const ADC_SCAN_ENTRY_t **const	scan_entries
const ADC_QUEUE_ENTRY_t **const	queue_entries
const XMC_VADC_QUEUE_ENTRY_t	local_queue_e
**const	
ADC_SCAN_t *const	scan_handle
ADC_QUEUE_t *const	queue_handle
ADC_MEASUREMENT_ADV_SCAN_t *const	local_scan_ha
ADC_MEASUREMENT_ADV_QUEUE_t *const	local_queue_h
uint8_t	sync_slave_g

Field Documentation

const ADC_MEASUREMENT_ADV_CHANNEL_t const ADC_MEASUREMENT_ADV::channelList**

This holds an array of channels configured by the current instance of the **ADC_MEASUREMENT_ADV APP**

Definition at line **288** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by **ADC_MEASUREMENT_ADV_Init()**.

const bool ADC_MEASUREMENT_ADV::configure_globiclass1

Copy the master channels conversion parameters to the global iclass 1. Hence the slave channels are using the same features as that of the master.

Definition at line **356** of file **ADC_MEASUREMENT_ADV.h**.

ADC_MEASUREMENT_ADV_EVENT_CONFIG_t ADC_MEASUREMENT_ADV::muxConfig

This hold the pointer to the function that does mux configuration. Which entails channel node and result node configuration

Definition at line **306** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by **ADC_MEASUREMENT_ADV_Init()**.

const uint8_t ADC_MEASUREMENT_ADV::group_index

The group index number for the APP

Definition at line **345** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by [ADC_MEASUREMENT_ADV_SetIclass\(\)](#),
[ADC_MEASUREMENT_ADV_SetSubtractionValue\(\)](#),
[ADC_MEASUREMENT_ADV_SoftwareTrigger\(\)](#), and
[ADC_MEASUREMENT_ADV_StartADC\(\)](#).

ADC_MEASUREMENT_ADV_STATUS_t* ADC_MEASUREMENT_AD

This enumeration gives information about the status of the APP

Definition at line [325](#) of file [ADC_MEASUREMENT_ADV.h](#).

Referenced by [ADC_MEASUREMENT_ADV_Init\(\)](#).

const XMC_VADC_QUEUE_ENTRY_t const ADC_MEASUREMENT**

Holds the pointer to the queue entries.

Definition at line [301](#) of file [ADC_MEASUREMENT_ADV.h](#).

ADC_MEASUREMENT_ADV_QUEUE_t* const ADC_MEASUREMEN

Pointer to the queue handle

Definition at line [321](#) of file [ADC_MEASUREMENT_ADV.h](#).

ADC_MEASUREMENT_ADV_SCAN_t* const ADC_MEASUREMENT

Pointer to the scan handle

Definition at line [318](#) of file [ADC_MEASUREMENT_ADV.h](#).

const ADC_QUEUE_ENTRY_t const ADC_MEASUREMENT_ADV:**

Holds the pointer to the queue entries.

Definition at line [298](#) of file [ADC_MEASUREMENT_ADV.h](#).

ADC_QUEUE_t* const ADC_MEASUREMENT_ADV::queue_handle

Pointer to the ADC_QUEUE APP handle

Definition at line [315](#) of file [ADC_MEASUREMENT_ADV.h](#).

ADC_MEASUREMENT_ADV_REQUEST_SOURCE_t ADC_MEASUR

The request source used by this instance of the
[ADC_MEASUREMENT_ADV](#) APP

Definition at line [327](#) of file [ADC_MEASUREMENT_ADV.h](#).

Referenced by [ADC_MEASUREMENT_ADV_Init\(\)](#),
[ADC_MEASUREMENT_ADV_SoftwareTrigger\(\)](#), and
[ADC_MEASUREMENT_ADV_StartADC\(\)](#).

const ADC_SCAN_ENTRY_t const ADC_MEASUREMENT_ADV::s**

Holds the pointer to the scan entries.

Definition at line [295](#) of file [ADC_MEASUREMENT_ADV.h](#).

ADC_SCAN_t* const ADC_MEASUREMENT_ADV::scan_handle

Pointer to the ADC_SCAN APP handle

Definition at line [312](#) of file [ADC_MEASUREMENT_ADV.h](#).

const bool ADC_MEASUREMENT_ADV::start_at_initialization

This determines if the insertion of the queue or scan entries should happen after initialization of the APP

Definition at line **353** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by **ADC_MEASUREMENT_ADV_Init()**.

uint8_t ADC_MEASUREMENT_ADV::sync_slave_g0

If set the group-0 will be configured as the slave group.

Definition at line **335** of file **ADC_MEASUREMENT_ADV.h**.

uint8_t ADC_MEASUREMENT_ADV::sync_slave_g1

If set the group-1 will be configured as the slave group

Definition at line **336** of file **ADC_MEASUREMENT_ADV.h**.

uint8_t ADC_MEASUREMENT_ADV::sync_slave_g2

If set the group-2 will be configured as the slave group

Definition at line **337** of file **ADC_MEASUREMENT_ADV.h**.

uint8_t ADC_MEASUREMENT_ADV::sync_slave_g3

If set the group-3 will be configured as the slave group

Definition at line **338** of file **ADC_MEASUREMENT_ADV.h**.

const uint8_t ADC_MEASUREMENT_ADV::total_number_of_chann

Indicates the total number of channels configured in the current APP instance

Definition at line **350** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by **ADC_MEASUREMENT_ADV_Init()**.

const uint8_t ADC_MEASUREMENT_ADV::total_number_of_entries

Indicates the total number of entries configured in the current APP instance

Definition at line **347** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by **ADC_MEASUREMENT_ADV_Init()**.

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

- **ADC_MEASUREMENT_ADV.h**
-

ADC_MEASUREMENT_ADV

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

ADC_MEASUREMENT_ADV_CHANNEL Struct Reference

Detailed Description

Structure to configure the channels in the **ADC_MEASUREMENT_ADV APP.**

Definition at line **257** of file **ADC_MEASUREMENT_ADV.h**.

```
#include <ADC_MEASUREMENT_ADV.h>
```

Data Fields

XMC_VADC_CHANNEL_CONFIG_t *	ch_handle
XMC_VADC_RESULT_CONFIG_t *	res_handle
	[ADC_MEASUREMENT_ADV_
ANALOG_IO_t *	analog_io_config
uint8_t	max_fifo_required
uint8_t	result_fifo_tail_number
uint8_t	group_index
uint8_t	ch_num
ADC_MEASUREMENT_ADV_GAIN_t	shs_gain_factor

Field Documentation

ANALOG_IO_t* ADC_MEASUREMENT_ADV_CHANNEL::analog_io

This hold the address of the ANALOG_IO configuration structure

Definition at line **265** of file [ADC_MEASUREMENT_ADV.h](#).

Referenced by [ADC_MEASUREMENT_ADV_Init\(\)](#).

XMC_VADC_CHANNEL_CONFIG_t* ADC_MEASUREMENT_ADV_CI

This holds the VADC Channel LLD structures

Definition at line **259** of file [ADC_MEASUREMENT_ADV.h](#).

Referenced by

[ADC_MEASUREMENT_ADV_ConfigureChannelClass\(\)](#),
[ADC_MEASUREMENT_ADV_GetDetailedResult\(\)](#),
[ADC_MEASUREMENT_ADV_GetFastCompareResult\(\)](#),
[ADC_MEASUREMENT_ADV_GetResult\(\)](#),
[ADC_MEASUREMENT_ADV_Init\(\)](#),
[ADC_MEASUREMENT_ADV_SetBoundaryLower\(\)](#),
[ADC_MEASUREMENT_ADV_SetBoundaryUpper\(\)](#),
[ADC_MEASUREMENT_ADV_SetChannelGain\(\)](#), and
[ADC_MEASUREMENT_ADV_SetFastCompareValue\(\)](#).

uint8_t ADC_MEASUREMENT_ADV_CHANNEL::ch_num

This Holds the Channel Number

Definition at line **275** of file [ADC_MEASUREMENT_ADV.h](#).

Referenced by [ADC_MEASUREMENT_ADV_GetAliasValue\(\)](#),

`ADC_MEASUREMENT_ADV_Init()`,
`ADC_MEASUREMENT_ADV_SelectBoundary()`,
`ADC_MEASUREMENT_ADV_SetAlternateReference()`, and
`ADC_MEASUREMENT_ADV_SetChannelGain()`.

`uint8_t ADC_MEASUREMENT_ADV_CHANNEL::group_index`

This holds the group index

Definition at line **273** of file `ADC_MEASUREMENT_ADV.h`.

Referenced by

`ADC_MEASUREMENT_ADV_ConfigureChannelClass()`,
`ADC_MEASUREMENT_ADV_GetAliasValue()`,
`ADC_MEASUREMENT_ADV_GetDetailedResult()`,
`ADC_MEASUREMENT_ADV_GetFastCompareResult()`,
`ADC_MEASUREMENT_ADV_GetFifoDetailedResult()`,
`ADC_MEASUREMENT_ADV_GetFifoResult()`,
`ADC_MEASUREMENT_ADV_GetResult()`,
`ADC_MEASUREMENT_ADV_Init()`,
`ADC_MEASUREMENT_ADV_SelectBoundary()`,
`ADC_MEASUREMENT_ADV_SetAlternateReference()`,
`ADC_MEASUREMENT_ADV_SetChannelGain()`, and
`ADC_MEASUREMENT_ADV_SetFastCompareValue()`.

`uint8_t ADC_MEASUREMENT_ADV_CHANNEL::max_fifo_required`

The required number of FIFO elements

Definition at line **269** of file `ADC_MEASUREMENT_ADV.h`.

`XMC_VADC_RESULT_CONFIG_t* ADC_MEASUREMENT_ADV_CHA`

This hold the VADC LLD Result configuration structures

Definition at line **261** of file **ADC_MEASUREMENT_ADV.h**.

uint8_t ADC_MEASUREMENT_ADV_CHANNEL::result_fifo_tail_nu

The tail result register number if FIFO is selected.

Definition at line **271** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by

ADC_MEASUREMENT_ADV_GetFifoDetailedResult(), and
ADC_MEASUREMENT_ADV_GetFifoResult().

ADC_MEASUREMENT_ADV_GAIN_t ADC_MEASUREMENT_ADV_C

The required gain factor for the channel.

Definition at line **278** of file **ADC_MEASUREMENT_ADV.h**.

Referenced by **ADC_MEASUREMENT_ADV_Init()**.

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

- **ADC_MEASUREMENT_ADV.h**

ADC_MEASUREMENT_ADV

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

ADC_MEASUREMENT_ADV_NVIC_CONFIG Struct Reference

Detailed Description

NVIC Configuration structure for request source interrupt.

Definition at line [195](#) of file [ADC_MEASUREMENT_ADV.h](#).

```
#include <ADC_MEASUREMENT_ADV.h>
```

Data Fields

uint32_t	node_id
uint32_t	priority
uint32_t	sub_priority
bool	interrupt_enable
uint8_t	irqctrl

Field Documentation

bool ADC_MEASUREMENT_ADV_NVIC_CONFIG::interrupt_enable

This flag indicates if a Interrupt has been requested.

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

uint8_t ADC_MEASUREMENT_ADV_NVIC_CONFIG::irqctrl

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

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

uint32_t ADC_MEASUREMENT_ADV_NVIC_CONFIG::node_id

This indicates the NVIC Node number.

Definition at line **197** of file **ADC_MEASUREMENT_ADV.h**.

uint32_t ADC_MEASUREMENT_ADV_NVIC_CONFIG::priority

This indicates the NVIC priority.

Definition at line **199** of file **ADC_MEASUREMENT_ADV.h**.

uint32_t ADC_MEASUREMENT_ADV_NVIC_CONFIG::sub_priority

This indicates the NVIC sub priority in XMC4x Devices.

Definition at line **201** of file **ADC_MEASUREMENT_ADV.h**.

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

- **ADC_MEASUREMENT_ADV.h**

ADC_MEASUREMENT_ADV

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

Data Fields

ADC_MEASUREMENT_ADV_QUEUE Struct Reference

Detailed Description

Configuration Data structure of queue request source.

Definition at line [236](#) of file [ADC_MEASUREMENT_ADV.h](#).

```
#include <ADC_MEASUREMENT_ADV.h>
```

Data Fields

const ADC_MEASUREMENT_ADV_NVIC_CONFIG_t	rs_intr_handle
const XMC_VADC_GROUP_CLASS_t	iclass_config_handl
const XMC_VADC_QUEUE_CONFIG_t *const	queue_config_hand
const XMC_VADC_GATEMODE_t	gating_mode
const XMC_VADC_SR_t	srv_req_node
const uint8_t	iclass_num

Field Documentation

const XMC_VADC_GATEMODE_t ADC_MEASUREMENT_ADV_QUE

Gating mode configuration needed for Scan request source

Definition at line [244](#) of file **ADC_MEASUREMENT_ADV.h**.

const XMC_VADC_GROUP_CLASS_t ADC_MEASUREMENT_ADV_

Holds the ICLASS Configurations

Definition at line [240](#) of file **ADC_MEASUREMENT_ADV.h**.

const uint8_t ADC_MEASUREMENT_ADV_QUEUE::iclass_num

Holds the ICLASS ID either ICLASS-0 or ICLASS-1

Definition at line [248](#) of file **ADC_MEASUREMENT_ADV.h**.

const XMC_VADC_QUEUE_CONFIG_t* const ADC_MEASUREMEN

Holds the LLD QUEUE Structure

Definition at line [242](#) of file **ADC_MEASUREMENT_ADV.h**.

const ADC_MEASUREMENT_ADV_NVIC_CONFIG_t ADC_MEASUR

Holds the ISR Handle

Definition at line [238](#) of file **ADC_MEASUREMENT_ADV.h**.

const XMC_VADC_SR_t ADC_MEASUREMENT_ADV_QUEUE::srv_

Source event interrupt node pointer

Definition at line **246** of file **ADC_MEASUREMENT_ADV.h**.

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

- **ADC_MEASUREMENT_ADV.h**

ADC_MEASUREMENT_ADV

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

ADC_MEASUREMENT_ADV_SCAN Struct Reference

Detailed Description

Configuration Data structure of scan request source.

Definition at line **213** of file [ADC_MEASUREMENT_ADV.h](#).

```
#include <ADC_MEASUREMENT_ADV.h>
```

Data Fields

const ADC_MEASUREMENT_ADV_NVIC_CONFIG_t	rs_intr_handle
const XMC_VADC_GROUP_CLASS_t	iclass_config_handl
const XMC_VADC_SCAN_CONFIG_t *const	scan_config_handle
const XMC_VADC_GATEMODE_t	gating_mode
const XMC_VADC_SR_t	srv_req_node
const uint32_t	insert_mask
const uint8_t	iclass_num

Field Documentation

const XMC_VADC_GATEMODE_t ADC_MEASUREMENT_ADV_SCA

Gating mode configuration needed for Scan request source

Definition at line [221](#) of file **ADC_MEASUREMENT_ADV.h**.

const XMC_VADC_GROUP_CLASS_t ADC_MEASUREMENT_ADV_

Holds the ICLASS Configurations

Definition at line [217](#) of file **ADC_MEASUREMENT_ADV.h**.

const uint8_t ADC_MEASUREMENT_ADV_SCAN::iclass_num

Holds the ICLASS ID either ICLASS-0 or ICLASS-1

Definition at line [227](#) of file **ADC_MEASUREMENT_ADV.h**.

const uint32_t ADC_MEASUREMENT_ADV_SCAN::insert_mask

Insert Mask for the scan request source

Definition at line [225](#) of file **ADC_MEASUREMENT_ADV.h**.

const ADC_MEASUREMENT_ADV_NVIC_CONFIG_t ADC_MEASUR

Holds the ISR Handle

Definition at line [215](#) of file **ADC_MEASUREMENT_ADV.h**.

const XMC_VADC_SCAN_CONFIG_t* const ADC_MEASUREMENT_

Holds the LLD SCAN Structure

Definition at line **219** of file **ADC_MEASUREMENT_ADV.h**.

const XMC_VADC_SR_t ADC_MEASUREMENT_ADV_SCAN::srv_re

Source event interrupt node pointer

Definition at line **223** of file **ADC_MEASUREMENT_ADV.h**.

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

- **ADC_MEASUREMENT_ADV.h**

ADC_MEASUREMENT_ADV

Home

Data Structures

Data Structure Index

Data Fields

Data Structure Index

A

A

ADC_MEASUREMENT_ADV_CHAN
ADC_MEASUREMENT_ADV_NVIC_C

ADC_MEASUREMENT_ADV

A



ADC_MEASUREMENT_ADV

Home												
Data Structures					Data Structure Index					Data Fields		
All		Variables										
a	c	e	g	i	l	m	n	p	q	r	s	t

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

- a -

- analog_io_config : [ADC_MEASUREMENT_ADV_CHANNEL](#)

- c -

- ch_handle : [ADC_MEASUREMENT_ADV_CHANNEL](#)
- ch_num : [ADC_MEASUREMENT_ADV_CHANNEL](#)
- channel_array : [ADC_MEASUREMENT_ADV](#)
- configure_globiclass1 : [ADC_MEASUREMENT_ADV](#)

- e -

- event_config : [ADC_MEASUREMENT_ADV](#)

- g -

- gating_mode : [ADC_MEASUREMENT_ADV_SCAN](#) , [ADC_MEASUREMENT_ADV_QUEUE](#)
- group_index : [ADC_MEASUREMENT_ADV_CHANNEL](#) , [ADC_MEASUREMENT_ADV](#)

- i -

- iclass_config_handle : [ADC_MEASUREMENT_ADV_SCAN](#) , [ADC_MEASUREMENT_ADV_QUEUE](#)
- iclass_num : [ADC_MEASUREMENT_ADV_QUEUE](#) ,

ADC_MEASUREMENT_ADV_SCAN

- init_state : **ADC_MEASUREMENT_ADV**
- insert_mask : **ADC_MEASUREMENT_ADV_SCAN**
- interrupt_enable : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**
- irqctrl : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**

- | -

- local_queue_entries : **ADC_MEASUREMENT_ADV**
- local_queue_handle : **ADC_MEASUREMENT_ADV**
- local_scan_handle : **ADC_MEASUREMENT_ADV**

- m -

- max_fifo_required : **ADC_MEASUREMENT_ADV_CHANNEL**

- n -

- node_id : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**

- p -

- priority : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**

- q -

- queue_config_handle : **ADC_MEASUREMENT_ADV_QUEUE**
- queue_entries : **ADC_MEASUREMENT_ADV**
- queue_handle : **ADC_MEASUREMENT_ADV**

- r -

- req_src : **ADC_MEASUREMENT_ADV**
- res_handle : **ADC_MEASUREMENT_ADV_CHANNEL**
- result_fifo_tail_number : **ADC_MEASUREMENT_ADV_CHANNEL**
- rs_intr_handle : **ADC_MEASUREMENT_ADV_SCAN** ,
ADC_MEASUREMENT_ADV_QUEUE

- s -

- scan_config_handle : **ADC_MEASUREMENT_ADV_SCAN**
- scan_entries : **ADC_MEASUREMENT_ADV**
- scan_handle : **ADC_MEASUREMENT_ADV**
- shs_gain_factor : **ADC_MEASUREMENT_ADV_CHANNEL**
- srv_req_node : **ADC_MEASUREMENT_ADV_SCAN** ,
ADC_MEASUREMENT_ADV_QUEUE
- start_at_initialization : **ADC_MEASUREMENT_ADV**
- sub_priority : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**
- sync_slave_g0 : **ADC_MEASUREMENT_ADV**
- sync_slave_g1 : **ADC_MEASUREMENT_ADV**
- sync_slave_g2 : **ADC_MEASUREMENT_ADV**
- sync_slave_g3 : **ADC_MEASUREMENT_ADV**

- t -

- total_number_of_channels : **ADC_MEASUREMENT_ADV**
 - total_number_of_entries : **ADC_MEASUREMENT_ADV**
-

ADC_MEASUREMENT_ADV

Home												
Data Structures					Data Structure Index					Data Fields		
All		Variables										
a	c	e	g	i	l	m	n	p	q	r	s	t

- a -

- analog_io_config : [ADC_MEASUREMENT_ADV_CHANNEL](#)

- c -

- ch_handle : [ADC_MEASUREMENT_ADV_CHANNEL](#)
- ch_num : [ADC_MEASUREMENT_ADV_CHANNEL](#)
- channel_array : [ADC_MEASUREMENT_ADV](#)
- configure_globiclass1 : [ADC_MEASUREMENT_ADV](#)

- e -

- event_config : [ADC_MEASUREMENT_ADV](#)

- g -

- gating_mode : [ADC_MEASUREMENT_ADV_SCAN](#) , [ADC_MEASUREMENT_ADV_QUEUE](#)
- group_index : [ADC_MEASUREMENT_ADV_CHANNEL](#) , [ADC_MEASUREMENT_ADV](#)

- i -

- iclass_config_handle : [ADC_MEASUREMENT_ADV_SCAN](#) , [ADC_MEASUREMENT_ADV_QUEUE](#)
- iclass_num : [ADC_MEASUREMENT_ADV_QUEUE](#) , [ADC_MEASUREMENT_ADV_SCAN](#)

- init_state : **ADC_MEASUREMENT_ADV**
- insert_mask : **ADC_MEASUREMENT_ADV_SCAN**
- interrupt_enable : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**
- irqctrl : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**

- l -

- local_queue_entries : **ADC_MEASUREMENT_ADV**
- local_queue_handle : **ADC_MEASUREMENT_ADV**
- local_scan_handle : **ADC_MEASUREMENT_ADV**

- m -

- max_fifo_required : **ADC_MEASUREMENT_ADV_CHANNEL**

- n -

- node_id : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**

- p -

- priority : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**

- q -

- queue_config_handle : **ADC_MEASUREMENT_ADV_QUEUE**
- queue_entries : **ADC_MEASUREMENT_ADV**
- queue_handle : **ADC_MEASUREMENT_ADV**

- r -

- req_src : **ADC_MEASUREMENT_ADV**
- res_handle : **ADC_MEASUREMENT_ADV_CHANNEL**
- result_fifo_tail_number : **ADC_MEASUREMENT_ADV_CHANNEL**
- rs_intr_handle : **ADC_MEASUREMENT_ADV_SCAN** ,
ADC_MEASUREMENT_ADV_QUEUE

- s -

- scan_config_handle : **ADC_MEASUREMENT_ADV_SCAN**
- scan_entries : **ADC_MEASUREMENT_ADV**
- scan_handle : **ADC_MEASUREMENT_ADV**
- shs_gain_factor : **ADC_MEASUREMENT_ADV_CHANNEL**
- srv_req_node : **ADC_MEASUREMENT_ADV_SCAN** ,
ADC_MEASUREMENT_ADV_QUEUE
- start_at_initialization : **ADC_MEASUREMENT_ADV**
- sub_priority : **ADC_MEASUREMENT_ADV_NVIC_CONFIG**
- sync_slave_g0 : **ADC_MEASUREMENT_ADV**
- sync_slave_g1 : **ADC_MEASUREMENT_ADV**
- sync_slave_g2 : **ADC_MEASUREMENT_ADV**
- sync_slave_g3 : **ADC_MEASUREMENT_ADV**

- t -

- total_number_of_channels : **ADC_MEASUREMENT_ADV**
 - total_number_of_entries : **ADC_MEASUREMENT_ADV**
-

ADC_MEASUREMENT_ADV

[Home](#)

[File List](#)

File List

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

[ADC_MEASUREMENT_ADV.c](#) [code] 

[ADC_MEASUREMENT_ADV.h](#) [code] 

ADC_MEASUREMENT_ADV

[Home](#)

[File List](#)

ADC_MEASUREMENT_ADV.h

```
00001
00078 /*CODE_BLOCK_BEGIN*/
00079 #ifndef ADC_MEASUREMENT_ADV_H
00080 #define ADC_MEASUREMENT_ADV_H
00081
00082
00083 /*****
00084 * HEADER FILES
00085 ****
00086
00087 #include "GLOBAL_ADC/global_adc.h"
00088 #include "adc_measurement_adv_conf.h"
00089
00090 ****
00091 * MACROS
00092 ****
00093 #if (((XMC_LIB_MAJOR_VERSION == 2U) && \
00094         (XMC_LIB_MINOR_VERSION >= 1U) && \
00095         (XMC_LIB_PATCH_VERSION >= 8U)))
00096 #error "ADC_MEASUREMENT_ADV requires XMC Per
```

```
ipheral Library v2.1.8 or higher"
00097 #endif
00098 /*****
***** 
00099 * ENUMS
00100 *****
***** 
***** */
00107 typedef enum ADC_MEASUREMENT_ADV_STATUS
00108 {
00109     ADC_MEASUREMENT_ADV_STATUS_SUCCESS = 0,
00110     ADC_MEASUREMENT_ADV_STATUS_FAILURE,
00111     ADC_MEASUREMENT_ADV_STATUS_UNINITIALIZED
00112 } ADC_MEASUREMENT_ADV_STATUS_t;
00113
00114
00118 typedef enum ADC_MEASUREMENT_ADV_REQUEST_SO
RCE
00119 {
00120     ADC_MEASUREMENT_ADV_REQUEST_SOURCE_SCAN =
0,
00121     ADC_MEASUREMENT_ADV_REQUEST_SOURCE_LOCAL_S
CAN,
00122     ADC_MEASUREMENT_ADV_REQUEST_SOURCE_QUEUE,
00123     ADC_MEASUREMENT_ADV_REQUEST_SOURCE_LOCAL_Q
UEUE,
00124 } ADC_MEASUREMENT_ADV_REQUEST_SOURCE_t;
00125
00129 typedef enum ADC_MEASUREMENT_ADV_GAIN
00130 {
00131     ADC_MEASUREMENT_ADV_GAIN_1 = 0U,
00132     ADC_MEASUREMENT_ADV_GAIN_3 = 1U,
00133     ADC_MEASUREMENT_ADV_GAIN_6 = 2U,
00134     ADC_MEASUREMENT_ADV_GAIN_12 = 3U
00135 } ADC_MEASUREMENT_ADV_GAIN_t;
```

```
00136
00140 typedef enum ADC_MEASUREMENT_ADV_SUBTRACTION
00141 {
00142     ADC_MEASUREMENT_ADV_SUBTRACTION_12BIT_LEFT_
ALIGN = 0U,
00143     ADC_MEASUREMENT_ADV_SUBTRACTION_12BIT_RIGHT
_ALIGN = 0U,
00144     ADC_MEASUREMENT_ADV_SUBTRACTION_10BIT_LEFT_
ALIGN = 2U,
00145     ADC_MEASUREMENT_ADV_SUBTRACTION_10BIT_RIGHT
_ALIGN = 0U,
00146     ADC_MEASUREMENT_ADV_SUBTRACTION_8BIT_LEFT_A
LIGN = 4U,
00147     ADC_MEASUREMENT_ADV_SUBTRACTION_8BIT_RIGHT_
ALIGN = 0U
00148 } ADC_MEASUREMENT_ADV_SUBTRACTION_t;
00149
00153 typedef enum ADC_MEASUREMENT_ADV_FAST_COMPARE
00154 {
00155     ADC_MEASUREMENT_ADV_FAST_COMPARE_LOW      =
0U,
00156     ADC_MEASUREMENT_ADV_FAST_COMPARE_HIGH    =
1U,
00157     ADC_MEASUREMENT_ADV_FAST_COMPARE_INVALID =
2U
00158 } ADC_MEASUREMENT_ADV_FAST_COMPARE_t;
00159
00163 typedef enum ADC_MEASUREMENT_ADV_SYNC_SEQ
00164 {
00165     ADC_MEASUREMENT_ADV_SYNC_SEQ_POWER_DOWN =
0U,
00166     ADC_MEASUREMENT_ADV_SYNC_SEQ_STSEL_CONFIG,
00167     ADC_MEASUREMENT_ADV_SYNC_SEQ_EVAL_CONFIG,
00168 } ADC_MEASUREMENT_ADV_SYNC_SEQ_t;
00169
```

```
00170
00175 /*****
00176 ****
00177 * DATA STRUCTURES
00178 ****
00179 ****/
00182 /*Anonymous structure/union guard start*/
00183 #if defined(__CC_ARM)
00184   #pragma push
00185   #pragma anon_unions
00186 #elif defined(__TASKING__)
00187   #pragma warning 586
00188 #endif
00189
00190 typedef void (*ADC_MEASUREMENT_ADV_EVENT_CONFIG_t)(void);
00195 typedef struct ADC_MEASUREMENT_ADV_NVIC_CONFIG
00196 {
00197   uint32_t node_id;
00199   uint32_t priority;
00200 #if(UC_FAMILY == XMC4)
00201   uint32_t sub_priority;
00202 #endif
00203   bool interrupt_enable;
00204 #ifdef ADC_MEASUREMENT_ADV_NON_DEFAULT_IRQ_SOURCE_SELECTED
00205   uint8_t irqctrl;
00206 #endif
00207 } ADC_MEASUREMENT_ADV_NVIC_CONFIG_t;
00208
00209 #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00210
00213 typedef struct ADC_MEASUREMENT_ADV_SCAN
00214 {
```

```
00215     const ADC_MEASUREMENT_ADV_NVIC_CONFIG_t rs
00216     _intr_handle;
00217     const XMC_VADC_GROUP_CLASS_t iclass_config
00218     _handle;
00219     const XMC_VADC_SCAN_CONFIG_t *const scan_c
00220     onfig_handle;
00221     const XMC_VADC_GATEMODE_t gating_mode;
00222
00223     const XMC_VADC_SR_t srv_req_node;
00224
00225     const uint32_t insert_mask;
00226
00227     const uint8_t iclass_num;
00228
00229 } ADC_MEASUREMENT_ADV_SCAN_t;
00230 #endif
00231
00232 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00233
00234 typedef struct ADC_MEASUREMENT_ADV_QUEUE
00235 {
00236     const ADC_MEASUREMENT_ADV_NVIC_CONFIG_t rs
00237     _intr_handle;
00238     const XMC_VADC_GROUP_CLASS_t iclass_config
00239     _handle;
00240     const XMC_VADC_QUEUE_CONFIG_t *const queue
00241     _config_handle;
00242     const XMC_VADC_GATEMODE_t gating_mode;
00243
00244     const XMC_VADC_SR_t srv_req_node;
00245
00246     const uint8_t iclass_num;
00247
00248 } ADC_MEASUREMENT_ADV_QUEUE_t;
00249 #endif
00250
00251
00252
00253
```

```
00257 typedef struct ADC_MEASUREMENT_ADV_CHANNEL
00258 {
00259     XMC_VADC_CHANNEL_CONFIG_t *ch_handle;
00261     XMC_VADC_RESULT_CONFIG_t *res_handle[ADC_M
EASUREMENT_ADV_RESULT_REG];
00264 #ifdef ADC_MEASUREMENT_ADV_ANALOG_IO_USED
00265     ANALOG_IO_t    *analog_io_config;
00266 #endif
00267
00268 #ifdef ADC_MEASUREMENT_ADV_FIFO_USED
00269     uint8_t max_fifo_required;
00271     uint8_t result_fifo_tail_number;
00272 #endif
00273     uint8_t group_index;
00275     uint8_t ch_num;
00277 #if(UC_FAMILY == XMC1)
00278     ADC_MEASUREMENT_ADV_GAIN_t shs_gain_factor
;
00279 #endif
00280
00281 } ADC_MEASUREMENT_ADV_CHANNEL_t;
00282
00286 typedef struct ADC_MEASUREMENT_ADV
00287 {
00288     const ADC_MEASUREMENT_ADV_CHANNEL_t **const
channel_array;
00290 #if defined(ADC_MEASUREMENT_ADV_ADC_SCAN_USE
D) || defined(ADC_MEASUREMENT_ADV_ADC_QUEUE_USED)
|| \
00291     defined(ADC_MEASUREMENT_ADV_LOCAL_QUEUE_
USED)
00292     union
00293     {
00294 #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00295     const ADC_SCAN_ENTRY_t **const scan_entries
;
00296 #endif
```

```
00297 #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00298     const ADC_QUEUE_ENTRY_t **const queue_entries;
00299 #endif
00300 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00301     const XMC_VADC_QUEUE_ENTRY_t **const local_queue_entries;
00302 #endif
00303 };
00304 #endif
00305
00306     ADC_MEASUREMENT_ADV_EVENT_CONFIG_t event_config;
00307     union
00308     {
00309 #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00310         ADC_SCAN_t *const scan_handle;
00311 #endif
00312 #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00313         ADC_QUEUE_t *const queue_handle;
00314 #endif
00315 #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00316         ADC_MEASUREMENT_ADV_SCAN_t *const local_scan_handle;
00317 #endif
00318 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00319         ADC_MEASUREMENT_ADV_QUEUE_t *const local_queue_handle;
00320 #endif
00321     };
00322 #endif
00323 };
00324
00325     ADC_MEASUREMENT_ADV_STATUS_t *init_state;
00326
00327     ADC_MEASUREMENT_ADV_REQUEST_SOURCE_t req_src;
00328 #ifdef ADC_MEASUREMENT_ADV_SYNC_USED
00329     union
```



```
00378 ****
00379 ****
00380 ****
00381 ****
00382 ****
00383 ****
00384 ****
00385 ****
00386 ****
00387 ****
00388 ****
00389 ****
00390 ****
00391 ****
00392 ****
00393 ****
00394 ****
00395 ****
00396 ****
00397 ****
00398 ****
00399 ****
00400 ****
00401 ****
00402 ****
00403 ****
00404 ****
00405 ****
00406 ****
00407 ****
00408 ****
00409 ****
00410 ****
00411 ****
00412 ****
00413 ****
00414 ****
00415 ****
00416 ****
00417 ****
00418 DAVE_APP_VERSION_t ADC_MEASUREMENT_ADV_GetAp
pVersion(void);
00419
00420 void ADC_MEASUREMENT_ADV_StartADC(const ADC_
MEASUREMENT_ADV_t *const handle_ptr);
00421
00422 void ADC_MEASUREMENT_ADV_SoftwareTrigger(con
st ADC_MEASUREMENT_ADV_t *const handle_ptr);
00423
00424 ADC_MEASUREMENT_ADV_STATUS_t ADC_MEASUREMENT
_ADV_Init(const ADC_MEASUREMENT_ADV_t *const handl
e_ptr);
00425
00426 __STATIC_INLINE uint16_t ADC_MEASUREMENT_ADV
_GetResult(const ADC_MEASUREMENT_ADV_CHANNEL_t *co
nst handle_ptr)
00427 {
00428     uint16_t result;
00429     extern XMC_VADC_GROUP_t *const group_ptrs[
XMC_VADC_MAXIMUM_NUM_GROUPS];
00430     XMC_ASSERT("ADC_MEASUREMENT_ADV_GetResult:
Invalid handle_ptr", (handle_ptr != NULL))
00431
00432     result = XMC_VADC_GROUP_GetResult(group_pt
rs[handle_ptr->group_index],
00433                                         (uint32_
t) handle_ptr->ch_handle->result_reg_number);
00434     return(result);
00435 }
00436
00437
00438 __STATIC_INLINE XMC_VADC_DETAILED_RESULT_t A
DC_MEASUREMENT_ADV_GetDetailedResult(const
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
00999
```

```
00731     NNEL_t *const handle_ptr)
00732 {
00733     XMC_VADC_DETAILED_RESULT_t result_register
00734 ;
00735     extern XMC_VADC_GROUP_t *const group_ptrs[
00736         XMC_VADC_MAXIMUM_NUM_GROUPS];
00737     XMC_ASSERT("ADC_MEASUREMENT_ADV_GetDetailedResult:Invalid handle_ptr", (handle_ptr != NULL))
00738     result_register.res = XMC_VADC_GROUP_GetDetailedResult(group_ptrs[handle_ptr->group_index],
00739
00740             (uint32_t) handle_ptr->ch_handle->result_reg_number);
00741     return(result_register);
00742 #ifdef ADC_MEASUREMENT_ADV_FIFO_USED
00743
00821 __STATIC_INLINE uint16_t ADC_MEASUREMENT_ADV
00822 _GetFifoResult(const ADC_MEASUREMENT_ADV_CHANNEL_t
00823 *const handle_ptr)
00824 {
00825     uint16_t result;
00826     extern XMC_VADC_GROUP_t *const group_ptrs[
00827         XMC_VADC_MAXIMUM_NUM_GROUPS];
00828     XMC_ASSERT("ADC_MEASUREMENT_ADV_GetFifoResult:Invalid handle_ptr", (handle_ptr != NULL))
00829     result = XMC_VADC_GROUP_GetResult(group_ptrs[handle_ptr->group_index], handle_ptr->result_fifo_tail_number);
00830     return(result);
00831 }
00908 __STATIC_INLINE XMC_VADC_DETAILED_RESULT_t ADC_MEASUREMENT_ADV_GetFifoDetailedResult(const
00909
```

ADC_MEASUREMENT_ADV_CHA

```
00910     NNEL_t *const handle_ptr)
00911 {
00912     XMC_VADC_DETAILED_RESULT_t result_register
00913     ;
00914     extern XMC_VADC_GROUP_t *const group_ptrs[
00915         XMC_VADC_MAXIMUM_NUM_GROUPS];
00916     XMC_ASSERT("ADC_MEASUREMENT_ADV_GetFifoDetailedResult:Invalid handle_ptr", (handle_ptr != NULL))
00917     result_register.res = XMC_VADC_GROUP_GetDetailedResult(group_ptrs[handle_ptr->group_index],
00918
00919             handle_ptr->result_fifo_tail_number);
00920     return(result_register);
00921 }
00922
00923 #endif
00924
01001 __STATIC_INLINE ADC_MEASUREMENT_ADV_FAST_COMPARE_t ADC_MEASUREMENT_ADV_GetFastCompareResult(co
01002 nst
01003
01004             ADC_MEASUREMENT_ADV_CHA
01005     NNEL_t *const handle_ptr)
01006 {
01007     ADC_MEASUREMENT_ADV_FAST_COMPARE_t fast_compare_result;
01008     extern XMC_VADC_GROUP_t *const group_ptrs[
01009         XMC_VADC_MAXIMUM_NUM_GROUPS];
01010     XMC_ASSERT("ADC_MEASUREMENT_ADV_GetFastCompareResult:Invalid handle_ptr", (handle_ptr != NULL))
01011
01012     fast_compare_result = (ADC_MEASUREMENT_ADV_FAST_COMPARE_t)XMC_VADC_GROUP_GetFastCompareResul
01013 t(
```

```
01009                                     group_ptrs[ha  
ndle_ptr->group_index],(uint32_t) handle_ptr->ch_h  
andle->result_reg_number);  
01010  
01011     return(fast_compare_result);  
01012 }  
01013  
01089 ADC_MEASUREMENT_ADV_STATUS_t ADC_MEASUREMENT  
_ADV_SetFastCompareValue(const ADC_MEASUREMENT_ADV  
_CHANNEL_t *const handle_ptr,  
01090  
                                uint16_t compare_value);  
01091  
01159 void ADC_MEASUREMENT_ADV_SetSubtractionValue(  
const ADC_MEASUREMENT_ADV_t *const handle_ptr,  
01160  
    ADC_MEASUREMENT_ADV_SUBTRACTION_t subtraction_ali  
gnment,  
01161  
    uint16_t subtraction_value);  
01162  
01236 void ADC_MEASUREMENT_ADV_ConfigureChannelCla  
ss(const ADC_MEASUREMENT_ADV_CHANNEL_t *const hand  
le_ptr,  
01237  
    const XMC_VADC_GROUP_CLASS_t *config);  
01238  
01301 void ADC_MEASUREMENT_ADV_SetAlternateReferen  
ce(const ADC_MEASUREMENT_ADV_CHANNEL_t *const hand  
le_ptr,  
01302  
    const XMC_VADC_CHANNEL_REF_t reference_select)  
;  
01303  
01304 #if (XMC_VADC_SHS_AVAILABLE == 1U)  
01305  
01365 void ADC_MEASUREMENT_ADV_SetChannelGain(const
```

```
    ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_ptr,
01366                                         con
st ADC_MEASUREMENT_ADV_GAIN_t gain_factor);
01367 #endif
01368
01369
01448 void ADC_MEASUREMENT_ADV_SelectBoundary(const
      ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_ptr,
01449                                         XMC
      _VADC_BOUNDARY_SELECT_t boundary,
01450                                         XMC
      _VADC_CHANNEL_BOUNDARY_t boundary_selection);
01451
01531 void ADC_MEASUREMENT_ADV_SetBoundaryUpper(co
nst ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_pt
r,
01532                                         ui
nt32_t boundary_value);
01533
01613 void ADC_MEASUREMENT_ADV_SetBoundaryLower(co
nst ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_pt
r,
01614                                         ui
nt32_t boundary_value);
01615
01616
01694 XMC_VADC_CHANNEL_ALIAS_t ADC_MEASUREMENT_ADV
      _GetAliasValue(const ADC_MEASUREMENT_ADV_CHANNEL_t
      *const handle_ptr);
01695
01696 #ifdef ADC_MEASUREMENT_ADV_SYNC_USED
01697
01780 void ADC_MEASUREMENT_ADV_SetIiclass(const ADC
      _MEASUREMENT_ADV_t *const handle_ptr);
01781 #endif
01782
01783 #include "adc_measurement_adv_extern.h"
```

```
01784 #ifdef __cplusplus
01785 }
01786 #endif
01787
01788 #endif /* _ADC_MEASUREMENT_ADV_H_ */
```



ADC_MEASUREMENT_ADV

[Home](#)

[File List](#)

ADC_MEASUREMENT_ADV.c

```
00001
00078 /*****
***** * * * * *
***** * * * * *
***** * * * * *
00079 * HEADER FILES
00080 ****
***** * * * * *
***** * * * * *
***** * * * * *
***** * * * * /
00081 #include "adc_measurement_adv.h"
00082
00083 /*****
***** * * * * *
***** * * * * *
***** * * * * *
***** * * * * *
***** * * * * /
00084 * MACROS
00085 ****
***** * * * * *
***** * * * * *
***** * * * * *
***** * * * * /
00086 /* Pointer to the VADC GLOBAL*/
00087 #define ADC_MEASUREMENT_ADV_GLOBAL_PTR ((XMC
_VADC_GLOBAL_t *) (void *) VADC)
00088
00089 /* Max value possible with 10 bit resolution
is 1023*/
00090 #define ADC_MEASUREMENT_ADV_10_BIT_MAX_VALUE
((uint32_t)1023)
00091
00092 #if (XMC_VADC_SHS_AVAILABLE == 1U)
00093 /* Pointer to the SHS unit */
```

```
00094 #define ADC_MEASUREMENT_ADV_SHS_PTR ((XMC_
VADC_GLOBAL_SHS_t *) (void *) SHS0)
00095 #endif
00096
00102 #define ADC_MEASUREMENT_ADV_RESERVED_REGISTE
RS ((uint32_t)2)
00103
00109 #define ADC_MEASUREMENT_ADV_HEAD_RESULT_REG_
CONFIG ((uint32_t)0)
00110
00116 #define ADC_MEASUREMENT_ADV_TAIL_RESULT_REG_
CONFIG ((uint32_t)1)
00117
00118 /* Configure the slave input class as global
   input class 1*/
00119 #define ADC_MEASUREMENT_ADV_GLOBICLASS1 ((ui
nt32_t)1)
00120
00121
00122 /* Since the SCU is different for various de
   vices a macro is defined here to enable check of c
   lock-ungating*/
00123 #if UC_FAMILY == XMC1
00124 #define ADC_MEASUREMENT_ADV_CHECK_CLOCK_GA
TING ((uint32_t)1)
00125 #endif
00126
00127 ****
*****  

*****  

*****  

00128 * LOCAL DATA
00129 ****
*****  

*****  

*****  

00130 /* Array of Group pointers*/
00131 XMC_VADC_GROUP_t *const group_ptrs[XMC_VADC
_MAXIMUM_NUM_GROUPS] =
```

```
00132  {
00133      (VADC_G_TypeDef*)(void*) VADC_G0,
00134      (VADC_G_TypeDef*)(void*) VADC_G1
00135 #if (XMC_VADC_MAXIMUM_NUM_GROUPS > 2U)
00136     , (VADC_G_TypeDef*)(void*) VADC_G2,
00137     (VADC_G_TypeDef*)(void*) VADC_G3
00138 #endif
00139 };
00140
00141 #ifdef ADC_MEASUREMENT_ADV_FIFO_USED
00142
00145 static const XMC_VADC_RESULT_CONFIG_t ADC_ME
ASUREMENT_ADV_fifo_intermediate_stage =
00146 {
00147     .data_reduction_control = 0,
00148     .post_processing_mode = 0,
00149     .wait_for_read_mode = 0,
00150     .part_of_fifo = (bool)true,
00151     .event_gen_enable = 0
00152 };
00153 #endif
00154
00155 /*Anonymous structure/union guard start*/
00156 #if defined(__CC_ARM)
00157     #pragma push
00158     #pragma anon_unions
00159 #elif defined(__TASKING__)
00160     #pragma warning 586
00161 #endif
00162
00163 /* Private structure to determine the ALIAS*/
00164 typedef struct ADC_MEASUREMENT_ADV_ALIAS
00165 {
00166     union
00167     {
00168         struct
```

```
00169     {
00170         uint32_t alias0 : 5; /* ALIAS for Chan
nel 0*/
00171         uint32_t      : 3;
00172         uint32_t alias1 : 5; /* ALIAS for chan
nel 1*/
00173         uint32_t      : 19;
00174     };
00175     uint32_t alias;
00176 };
00177 }ADC_MEASUREMENT_ADV_ALIAS_t;
00178
00179 /*Anonymous structure/union guard end*/
00180 #if defined(__CC_ARM)
00181     #pragma pop
00182 #elif defined(__TASKING__)
00183     #pragma warning restore
00184 #endif
00185 ****
00186 ****
00187 ****
00188 ****
00189 ****
00190 ****
00191 ****
00192 ****
00193 ****
```

```
00194     if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
SCAN == handle_ptr->req_src)
00195 #endif
00196 {
00197     ADC_SCAN_InsertScanEntry(handle_ptr->s
can_handle, handle_ptr->scan_entries[ch_num]);
00198 }
00199 #endif
00200 #if defined(ADC_MEASUREMENT_ADV_QUEUE_USED)
&& defined(ADC_MEASUREMENT_ADV_ADC_QUEUE_USED)
00201 #if defined(ADC_MEASUREMENT_ADV_SCAN_USED)
&& defined(ADC_MEASUREMENT_ADV_ADC_SCAN_USED)
00202 else
00203 #endif
00204 {
00205     ADC_QUEUE_InsertQueueEntry(handle_ptr->
queue_handle, handle_ptr->queue_entries[ch_num]);
00206 }
00207 #endif
00208 }
00209 #endif
00210
00211 /*~~~~~
~~~~~~~~~~*/
00212 XMC_VADC_CHANNEL_CONV_t ADC_MEASUREMENT_ADV
_lGetIclass(const ADC_MEASUREMENT_ADV_t *const han
dle_ptr)
00213 {
00214     XMC_VADC_CHANNEL_CONV_t req_iclass;
00215
00216 #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00217     #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00218         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
LOCAL_SCAN >= handle_ptr->req_src)
00219     #endif
00220     {
```

```
00221 #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00222     #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00223         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_SCAN != handle_ptr->req_src)
00224     #endif
00225     {
00226         req_iclass = (XMC_VADC_CHANNEL_CONV_
t)handle_ptr->local_scan_handle->iclass_num;
00227     }
00228 #endif
00229
00230 #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00231     #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00232         else
00233     #endif
00234     {
00235         /* Call the function to initialise C
lock and ADC global functional units*/
00236         req_iclass = (XMC_VADC_CHANNEL_CONV_
t)handle_ptr->scan_handle->iclass_num;
00237     }
00238 #endif
00239     }
00240 #endif
00241
00242 #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00243     #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00244         else
00245     #endif
00246     {
00247 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00248     #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00249         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_QUEUE != handle_ptr->req_src)
00250     #endif
00251     {
00252         req_iclass = (XMC_VADC_CHANNEL_CONV_
```

```
t)handle_ptr->local_queue_handle->iclass_num;
00253         }
00254 #endif
00255
00256 #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00257     #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00258         else
00259     #endif
00260     {
00261         req_iclass = (XMC_VADC_CHANNEL_CONV_
t)handle_ptr->queue_handle->iclass_num;
00262     }
00263 #endif
00264     }
00265 #endif
00266     return (req_iclass);
00267 }
00268
00269 /*~~~~~
~~~~~
~~~~~*/
00270 #ifdef ADC_MEASUREMENT_ADV_SYNC_USED
00271
00272 /* Helper function to configure the eval bit
s in the slave*/
00273 void ADC_MEASUREMENT_ADV_lSyncEvalConfig(uint32_t master_group, uint32_t slave_selected,
00274                                         uint32_t sync_group)
00275 {
00276     int8_t group_index;
00277     sync_group |= (1U << master_group);
00278     sync_group &= ~(1U << slave_selected);
00279     for( group_index = XMC_VADC_MAXIMUM_NUM_GR
00280         - (int32_t)1; group_index >= (int32_t)0 ; gr
00281         oup_index-- )
```

```
00280      {
00281          if ( (bool)false != (bool)((sync_group >
00282              > group_index) & 0x1 ))
00283          {
00284              XMC_VADC_GROUP_SetSyncSlaveReadySignal
00285                  (group_ptrs[slave_selected], slave_selected, group
00286                      _index);
00287          }
00288      }
00289 /*~~~~~*/
00290 /* Helper function to execute the sync init
sequence*/
00291 void ADC_MEASUREMENT_ADV_1SyncSequencer(const
00292 ADC_MEASUREMENT_ADV_t *const handle_ptr,
00293                                     uint
00294                                     32_t sync_group,
00295                                     ADC_
00296                                     MEASUREMENT_ADV_SYNC_SEQ_t sequence)
00297 {
00298     int8_t group_index;
00299     for( group_index = XMC_VADC_MAXIMUM_NUM_GR
00300           OUPS - (int32_t)1; group_index >= (int32_t)0 ; gr
00301           oup_index-- )
00302     {
00303         if ( (bool)false != (bool)((sync_group >
00304             > group_index) & 0x1 ))
00305         {
00306             switch( sequence)
00307             {
00308                 case ADC_MEASUREMENT_ADV_SYNC_SEQ_P0
00309 WER_DOWN:
00310                 XMC_VADC_GROUP_SetPowerMode(gro
00311                     up_ptrs[group_index], XMC_VADC_GROUP_POWERMODE_OFF)
```

```
; 00303             break;
00304         case ADC_MEASUREMENT_ADV_SYNC_SEQ_ST
SEL_CONFIG:
00305             XMC_VADC_GROUP_SetSyncSlave(gro
up_ptrs[group_index], handle_ptr->group_index, gro
up_index);
00306             XMC_VADC_GROUP_CheckSlaveReadin
ess(group_ptrs[handle_ptr->group_index], group_inde
x);
00307             break;
00308         case ADC_MEASUREMENT_ADV_SYNC_SEQ_EV
AL_CONFIG:
00309             ADC_MEASUREMENT_ADV_1SyncEvalCo
nfig(handle_ptr->group_index, group_index, sync_gr
oup);
00310         default:
00311             break;
00312     }
00313 }
00314 }
00315 }
00316
00317 /*~~~~~*/~~~~~*/
00318 /* Initialization of the all the sync relate
d functions */
00319 __STATIC_INLINE void ADC_MEASUREMENT_ADV_1Sy
ncInit(const ADC_MEASUREMENT_ADV_t *const handle_p
tr)
00320 {
00321     uint8_t sync_group;
00322
00323     /* shift to get the 4 bit position needed
to or it with the slave groups */
00324     sync_group = handle_ptr->sync_slaves | ( 1
```

```
<< handle_ptr->group_index);
00325     ADC_MEASUREMENT_ADV_1SyncSequencer(handle_
ptr, sync_group, ADC_MEASUREMENT_ADV_SYNC_SEQ_POWE
R_DOWN);
00326
00327     sync_group = handle_ptr->sync_slaves;
00328     ADC_MEASUREMENT_ADV_1SyncSequencer(handle_
ptr, sync_group, ADC_MEASUREMENT_ADV_SYNC_SEQ_STSE
L_CONFIG);
00329     ADC_MEASUREMENT_ADV_1SyncSequencer(handle_
ptr, sync_group, ADC_MEASUREMENT_ADV_SYNC_SEQ_EVAL
_CONFIG);
00330
00331     /* Configure the iclass settings needed fo
r the sync slaves*/
00332     if( (bool) false != handle_ptr->configure_
globiclass1)
00333     {
00334         ADC_MEASUREMENT_ADV_SetIclass(handle_ptr
);
00335     }
00336
00337     XMC_VADC_GROUP_SetSyncMaster(group_ptrs[ha
ndle_ptr->group_index]);
00338
00339     XMC_VADC_GROUP_SetPowerMode(group_ptrs[han
dle_ptr->group_index],XMC_VADC_GROUP_POWERMODE_NOR
MAL);
00340 }
00341 #endif
00342
00343 /*~~~~~
~~~~~
~~~~~*/
00344 #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00345 __STATIC_INLINE ADC_MEASUREMENT_ADV_STATUS_t
ADC_MEASUREMENT_ADV_1ScanInit(ADC_MEASUREMENT_ADV
```

```
_SCAN_t *const handle_ptr,
00346                                uint8_t group_index)
00347 {
00348     ADC_MEASUREMENT_ADV_STATUS_t status;
00349
00350     /*Initialization of APP 'ADCGroup'*/
00351     status = (ADC_MEASUREMENT_ADV_STATUS_t) GL
00352         OBAL_ADC_Init(ADC_MEASUREMENT_ADV_GLOBAL_HANDLE);
00353     XMC_VADC_GROUP_InputClassInit(group_ptrs[group_index],
00354                                     handle_ptr->iclass_config_handle,
00355                                     XMC_VADC_GROUP_CONV_STD,
00356                                     (uint32_t)handle_ptr->iclass_num);
00357
00358     /*Initialization of scan request source*/
00359     XMC_VADC_GROUP_ScanInit(group_ptrs[group_index],
00360                               handle_ptr->scan_config_handle);
00361
00362     /* Configure the gating mode for Scan*/
00363     XMC_VADC_GROUP_ScanSetGatingMode(group_ptrs[group_index],
00364                                       handle_ptr->gating_mode);
00365
00366     /*Interrupt Configuration*/
00367     if ((bool)true == handle_ptr->rs_intr_handle.interrupt_enable)
00368     {
00369 #if (UC_FAMILY == XMC1)
00370         NVIC_SetPriority((IRQn_Type)handle_ptr->
00371                         rs_intr_handle.node_id,
00372                         handle_ptr->rs_intr_handle.priority);
00373 #else
00374         NVIC_SetPriority((IRQn_Type)handle_ptr->
00375                         rs_intr_handle.node_id,
00376                         NVIC_EncodePriority(NVIC_GetPriorityGrouping(),
```

```

00371                               handle_ptr->rs_intr_ha
ndle.priority,handle_ptr->rs_intr_handle.sub_prior
ity));
00372 #endif
00373 #ifdef ADC_MEASUREMENT_ADV_NON_DEFAULT_IRQ_S
OURCE_SELECTED
00374     XMC_SCU_SetInterruptControl(handle_ptr->
rs_intr_handle.node_id,
00375                                     ((handle_ptr
->rs_intr_handle.node_id << 8) | handle_ptr->rs_in
tr_handle.irqctrl));
00376 #endif
00377
00378     /* Connect RS Events to NVIC nodes */
00379     XMC_VADC_GROUP_ScanSetReqSrcEventInterru
ptNode(group_ptrs[group_index], handle_ptr->srv_re
q_node);
00380 }
00381
00382     return (status);
00383 }
00384 #endif
00385 /*~~~~~*/
00386 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00387 __STATIC_INLINE ADC_MEASUREMENT_ADV_STATUS_t
ADC_MEASUREMENT_ADV_lQueueInit(ADC_MEASUREMENT_AD
V_QUEUE_t *const handle_ptr,
00388
                                uint8_t group_index)
00389 {
00390     ADC_MEASUREMENT_ADV_STATUS_t status;
00391
00392     /*Initialization of APP 'GLOBAL_ADC'*/
00393     status = (ADC_MEASUREMENT_ADV_STATUS_t) GL
OBAL_ADC_Init(ADC_MEASUREMENT_ADV_GLOBAL_HANDLE);

```

```

00394
00395     /*Class Configuration*/
00396     XMC_VADC_GROUP_InputClassInit(group_ptrs[group_index], handle_ptr->iclass_config_handle,
00397                                         XMC_VADC_GROUP_CONV_STD, handle_ptr->iclass_num);
00398
00399     /* Initialize the Queue hardware */
00400     XMC_VADC_GROUP_QueueInit(group_ptrs[group_index], handle_ptr->queue_config_handle);
00401
00402     /* Configure the gating mode for queue*/
00403     XMC_VADC_GROUP_SetGatingMode(group_ptrs[group_index], handle_ptr->gating_mode);
00404
00405     /*Interrupt Configuration*/
00406     if ((bool)true == handle_ptr->rs_intr_handle.interrupt_enable)
00407     {
00408 #if (UC_FAMILY == XMC1)
00409         NVIC_SetPriority((IRQn_Type)handle_ptr->rs_intr_handle.node_id, handle_ptr->rs_intr_handle.priority);
00410 #else
00411         NVIC_SetPriority((IRQn_Type)handle_ptr->rs_intr_handle.node_id,
00412                         NVIC_EncodePriority(NVIC_GetPriorityGrouping(),
00413                                         handle_ptr->rs_intr_handle.priority, handle_ptr->rs_intr_handle.sub_priority));
00414 #endif
00415 #ifdef ADC_MEASUREMENT_ADV_NON_DEFAULT_IRQ_SOURCE_SELECTED
00416     XMC_SCU_SetInterruptControl(handle_ptr->rs_intr_handle.node_id,
00417                                     ((handle_ptr

```

```

->rs_intr_handle.node_id << 8) | handle_ptr->rs_in
tr_handle.irqctrl));
00418 #endif
00419
00420     /* Connect RS Events to NVIC nodes */
00421     XMC_VADC_GROUP_QueueSetReqSrcEventInterr
uptNode(group_ptrs[group_index], (XMC_VADC_SR_t)ha
ndle_ptr->srv_req_node);
00422 }
00423
00424     return (status);
00425 }
00426 #endif
00427 /*~~~~~
~~~~~
~~~~~*/
00428 /* Local function to do the request source i
nitialization.*/
00429 __STATIC_INLINE ADC_MEASUREMENT_ADV_STATUS_t
ADC_MEASUREMENT_ADV_1RequestSrcInit(const ADC_ME
ASUREMENT_ADV_t
00430
                                         *const handle
_ptr)
00431 {
00432     ADC_MEASUREMENT_ADV_STATUS_t status;
00433
00434 #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00435     #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00436         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
LOCAL_SCAN >= handle_ptr->req_src)
00437     #endif
00438     {
00439 #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00440     #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00441         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURC
E_SCAN != handle_ptr->req_src)

```

```
00442 #endif
00443 {
00444     status = ADC_MEASUREMENT_ADV_1ScanIn
it(handle_ptr->local_scan_handle,handle_ptr->group
_index);
00445 }
00446 #endif
00447
00448 #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00449     #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00450         else
00451     #endif
00452     {
00453         /* Call the function to initialise C
lock and ADC global functional units*/
00454         status = (ADC_MEASUREMENT_ADV_STATUS
_t) ADC_SCAN_Init(handle_ptr->scan_handle);
00455     }
00456 #endif
00457 }
00458 #endif
00459
00460 #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00461     #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00462         else
00463     #endif
00464     {
00465 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00466     #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00467         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURC
E_QUEUE != handle_ptr->req_src)
00468     #endif
00469     {
00470         status = ADC_MEASUREMENT_ADV_1QueueI
nit(handle_ptr->local_queue_handle,handle_ptr->gro
up_index);
00471     }
```

```
00472 #endif
00473
00474 #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00475     #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00476         else
00477     #endif
00478     {
00479         /* Call the function to initialise C
lock and ADC global functional units*/
00480         status = (ADC_MEASUREMENT_ADV_STATUS
00481 _t) ADC_QUEUE_Init(handle_ptr->queue_handle);
00481     }
00482 #endif
00483     }
00484 #endif
00485     return (status);
00486 }
00487
00488 /*~~~~~
~~~~~
~~~~~*/
00489 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00490 /* Local function to insert the queue entries into the hardware.*/
00491 __STATIC_INLINE void ADC_MEASUREMENT_ADV_1Qu
eueInsertEntries(const ADC_MEASUREMENT_ADV_t *const
    handle_ptr)
00492 {
00493     uint32_t entry_index;
00494
00495     XMC_VADC_GROUP_t *queue_group_ptr = group_
ptrs[handle_ptr->group_index];
00496     const XMC_VADC_QUEUE_ENTRY_t **const entries_array =
handle_ptr->local_queue_entries;
00497
00498     for(entry_index = 0; entry_index < handle_
```

```
ptr->total_number_of_entries; entry_index++)
00499    {
00500        XMC_VADC_GROUP_QueueInsertChannel(queue_
group_ptr, *entries_array[entry_index]);
00501    }
00502 }
00503 #endif
00504
00505 /*-----*/
-----*/
00506 __STATIC_INLINE bool ADC_MEASUREMENT_ADV_1Ar
bitrationStatus(const ADC_MEASUREMENT_ADV_t *const
handle_ptr)
00507 {
00508     bool clock_reset_check;
00509     bool arbitration_status;
00510
00511 #if !defined(CLOCK_GATING_SUPPORTED) || !def
ined(ADC_MEASUREMENT_ADV_CHECK_CLOCK_GATING)
00512     clock_reset_check = (bool)false;
00513 #endif
00514     arbitration_status = (bool)false;
00515
00516     /* To check if the arbiter is already enab
led. Before checking this ensure that clock and re
set states are correct */
00517 #if defined(CLOCK_GATING_SUPPORTED) && defin
ed(ADC_MEASUREMENT_ADV_CHECK_CLOCK_GATING)
00518     clock_reset_check = !XMC_SCU_CLOCK_IsPerip
heralClockGated(XMC_SCU_PERIPHERAL_CLOCK_VADC);
00519 #endif
00520 #ifdef PERIPHERAL_RESET_SUPPORTED
00521     clock_reset_check |= !XMC_SCU_RESET_IsPeri
pheralResetAsserted(XMC_SCU_PERIPHERAL_RESET_VADC)
;
00522 #endif
```

```
00523     if(clock_reset_check != (bool)false)
00524     {
00525 #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00526     #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00527         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
00528             LOCAL_SCAN >= handle_ptr->req_src)
00529     #endif
00530     {
00531         arbitration_status = XMC_VADC_GROUP_
00532 ScanIsArbitrationSlotEnabled(group_ptrs[handle_ptr
00533 ->group_index]);
00534     }
00535 #endif ADC_MEASUREMENT_ADV_QUEUE_USED
00536     #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00537         else
00538     #endif
00539     {
00540         arbitration_status = XMC_VADC_GROUP_
00541 QueueIsArbitrationSlotEnabled(group_ptrs[handle_pt
00542 r->group_index]);
00543     }
00544 #endif
00545     }
00546 /*~~~~~
~~~~~
~~~~~*/
00547 __STATIC_INLINE void ADC_MEASUREMENT_ADV_lDi
00548 sableArbitration(const ADC_MEASUREMENT_ADV_t *const
00549 handle_ptr,
00550
00551         bool arbitration_status)
00552 }
```

```
00550     if(arbitration_status == (bool)false)
00551     {
00552 #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00553     #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00554         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_LOCAL_SCAN >= handle_ptr->req_src )
00555     #endif
00556     {
00557         XMC_VADC_GROUP_ScanDisableArbitrationSlot(group_ptrs[handle_ptr->group_index]);
00558     }
00559 #endif
00560
00561 #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00562     #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00563         else
00564     #endif
00565     {
00566         XMC_VADC_GROUP_QueueDisableArbitrationSlot(group_ptrs[handle_ptr->group_index]);
00567     }
00568 #endif
00569 }
00570 }
00571
00572 /*~~~~~
~~~~~
~~~~~*/
00573 /* Insert channels into the hardware*/
00574 void ADC_MEASUREMENT_ADV_lInsertChannels(const ADC_MEASUREMENT_ADV_t *const handle_ptr)
00575 {
00576     XMC_ASSERT("ADC_MEASUREMENT_ADV_InsertChannels:Invalid handle_ptr", (handle_ptr != NULL))
00577
00578 #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00579     #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
```

```
00580     if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
LOCAL_SCAN >= handle_ptr->req_src)
00581 #endif
00582 {
00583 #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00584     #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00585         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
SCAN != handle_ptr->req_src)
00586     #endif
00587 {
00588     XMC_VADC_GROUP_ScanAddMultipleChanne
ls(group_ptrs[handle_ptr->group_index], handle_ptr
->local_scan_handle->insert_mask);
00589 }
00590 #endif
00591
00592 #ifdef ADC_MEASUREMENT_ADV_ADC_SCAN_USED
00593     #ifdef ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED
00594         else
00595     #endif
00596 {
00597     ADC_SCAN_AllEntriesInserted(handle_p
tr->scan_handle);
00598 }
00599 #endif
00600 }
00601 #endif
00602
00603 #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00604     #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00605         else
00606     #endif
00607 {
00608 #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00609     #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00610         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
QUEUE != handle_ptr->req_src)
```

```
00611     #endif
00612         {
00613             ADC_MEASUREMENT_ADV_lQueueInsertEntries(handle_ptr);
00614         }
00615     #endif
00616
00617 #ifdef ADC_MEASUREMENT_ADV_ADC_QUEUE_USED
00618     #ifdef ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED
00619         else
00620     #endif
00621         {
00622             ADC_QUEUE_AllEntriesInserted(handle_
ptr->queue_handle);
00623         }
00624     #endif
00625     }
00626 #endif
00627 }
00628
00629 /*~~~~~
~~~~~
~~~~~*/
00630 /* Select the boundary for a channel and con
figure its value as well.*/
00631 void ADC_MEASUREMENT_ADV_lSetBoundary(const
ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_ptr,
00632                                     XMC_VAD
C_CHANNEL_BOUNDARY_t boundary_select,
00633                                     uint32_
t boundary_value)
00634 {
00635     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetBoundar
y:Invalid handle_ptr", (handle_ptr != NULL))
00636
00637     switch(boundary_select)
```

```
00638 {  
00639     case XMC_VADC_CHANNEL_BOUNDARY_GROUP_BOU  
ND0:  
00640         case XMC_VADC_CHANNEL_BOUNDARY_GROUP_BOU  
ND1:  
00641             XMC_VADC_Group_SetIndividualBoundary(group_ptrs[handle_ptr->group_index], boundary_se  
lect,  
00642             (uint16_t)boundary_value);  
00643             break;  
00644  
00645         case XMC_VADC_CHANNEL_BOUNDARY_GLOBAL_BO  
UND0:  
00646         case XMC_VADC_CHANNEL_BOUNDARY_GLOBAL_BO  
UND1:  
00647             XMC_VADC_Global_SetIndividualBounda  
ry(ADC_MEASUREMENT_ADV_GLOBAL_PTR,  
00648             boundary_select,  
00649             (uint16_t)boundary_value);  
00650             break;  
00651     }  
00652 }  
00653 /*-----  
-----*/  
00654 #ifndef ADC_MEASUREMENT_ADV_SYNC_USED  
00655 /* Address the errata for the incorrect conv  
ersion. */  
00656 void ADC_MEASUREMENT_ADV_1SyncADCClocks(void  
)  
00657 {  
00658     int32_t group_index;  
00659  
00660     for (group_index = (int32_t)XMC_VADC_MAXIM
```

```
UM_NUM_GROUPS - (int32_t)1; group_index >= (int32_
t)0 ; group_index-- )
00661 {
00662     XMC_VADC_GROUP_SetPowerMode(group_ptrs[g
roup_index],XMC_VADC_GROUP_POWERMODE_OFF);
00663 }
00664
00665     for (group_index = (int32_t)XMC_VADC_MAXIM
UM_NUM_GROUPS - (int32_t)1; group_index > (int32_t
)0 ; group_index-- )
00666 {
00667     XMC_VADC_GROUP_SetSyncSlave(group_ptrs[g
roup_index], (uint32_t)0, (uint32_t)group_index);
00668
00669     XMC_VADC_GROUP_CheckSlaveReadiness(group
_ptrs[0U], (uint32_t)group_index);
00670 }
00671
00672     XMC_VADC_GROUP_SetSyncMaster(group_ptrs[0U
]);
00673
00674     XMC_VADC_GROUP_SetPowerMode(group_ptrs[0U]
,XMC_VADC_GROUP_POWERMODE_NORMAL);
00675 }
00676 #endif
00677 ****
***** ****
*****
00678 * API IMPLEMENTATION
00679 ****
***** ****
*****
00680
00681 /*This function returns the version of the A
DC_MEASUREMENT App*/
00682 DAVE_APP_VERSION_t ADC_MEASUREMENT_ADV_GetAp
pVersion(void)
```

```
00683 {
00684     DAVE_APP_VERSION_t version;
00685
00686     version.major = (uint8_t) ADC_MEASUREMENT_
00687     ADV_MAJOR_VERSION;
00688     version.minor = (uint8_t) ADC_MEASUREMENT_
00689     ADV_MINOR_VERSION;
00690     version.patch = (uint8_t) ADC_MEASUREMENT_
00691     ADV_PATCH_VERSION;
00692
00693 /*~~~~~*/
00694 /* Enables the arbiter of the selected request source*/
00695 void ADC_MEASUREMENT_ADV_StartADC(const ADC_
00696     MEASUREMENT_ADV_t *const handle_ptr)
00697 {
00698 #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00699     #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00700         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
00701             LOCAL_SCAN >= handle_ptr->req_src)
00702             XMC_VADC_GROUP_ScanEnableArbitration
00703             Slot(group_ptrs[handle_ptr->group_index]);
00704     }
00705 #endif
00706 #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00707     #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00708         else
00709     #endif
00710     {
```

```
00711         XMC_VADC_GROUP_QueueEnableArbitratio
nSlot(group_ptrs[handle_ptr->group_index]);
00712     }
00713 #endif
00714 }
00715
00716 /*~~~~~
~~~~~*/
00717 /* Starts the ADC conversions by causing a s
oftware start of conversion*/
00718 void ADC_MEASUREMENT_ADV_SoftwareTrigger(const ADC_MEASUREMENT_ADV_t *const handle_ptr)
00719 {
00720     XMC_ASSERT("ADC_MEASUREMENT_ADV_StartConve
rsion:Invalid handle_ptr", (handle_ptr != NULL))
00721
00722 #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00723     #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00724         if ( ADC_MEASUREMENT_ADV_REQUEST_SOURCE_
LOCAL_SCAN >= handle_ptr->req_src)
00725     #endif
00726     {
00727         XMC_VADC_GROUP_ScanTriggerConversion(g
roup_ptrs[handle_ptr->group_index]);
00728     }
00729 #endif
00730 #ifdef ADC_MEASUREMENT_ADV_QUEUE_USED
00731     #ifdef ADC_MEASUREMENT_ADV_SCAN_USED
00732         else
00733     #endif
00734     {
00735         XMC_VADC_GROUP_QueueTriggerConversion(
group_ptrs[handle_ptr->group_index]);
00736     }
00737 #endif
00738 }
```



```
00763 __STATIC_INLINE void ADC_MEASUREMENT_ADC_1Re
sultInit(const ADC_MEASUREMENT_ADV_CHANNEL_t *inde
xed)
00764 {
00765 #ifdef ADC_MEASUREMENT_ADV_FIFO_USED
00766     uint8_t fifo_num_of_intermediate_stages;
00767     uint8_t fifo_index;
00768     uint8_t fifo_head;
00769 #endif
00770 #ifdef ADC_MEASUREMENT_ADV_FIFO_USED
00771     /* If FIFO is selected for the particular
channel then do the FIFO initializations*/
00772     if ( (uint32_t)0 != indexed->max_fifo_requ
ired)
00773     {
00774         /*Excluding the head and tail from the t
otal number of FIFO elements needed*/
00775         fifo_num_of_intermediate_stages = indexe
d->max_fifo_required - ADC_MEASUREMENT_ADV_RESERVE
D_REGISTERS;
00776
00777         fifo_head = (uint8_t)indexed->ch_handle-
>result_reg_number;
00778         for (fifo_index = 1; fifo_index <= fifo_
num_of_intermediate_stages; fifo_index++)
00779         {
00780             XMC_VADC_GROUP_ResultInit(group_ptrs[i
ndexed->group_index], (uint32_t)fifo_head - fifo_i
ndex,
00781                                         &ADC_MEASURE
MENT_ADV_fifo_intermediate_stage);
00782         }
00783
00784         /* For the FIFO Tail configuration*/
00785         XMC_VADC_GROUP_ResultInit(group_ptrs[i
ndexed->group_index], (uint32_t)indexed->result_fi
fo_tail_number,
```

```
00786                                         indexed->res
00787     _handle[ADC_MEASUREMENT_ADV_TAIL_RESULT_REG_CONFIG
00788 ];
00789     }
00790     /* Initialize for configured result re
00791 gisters For FIFO Head configuration*/
00792     XMC_VADC_GROUP_ResultInit(group_ptrs[i
00793 ndexed->group_index], (uint32_t)indexed->ch_handle
00794 ->result_reg_number,
00795                                         indexed->res
00796 _handle[ADC_MEASUREMENT_ADV_HEAD_RESULT_REG_CONFIG
00797 ]);
00798 }
00799
00800 /*~~~~~*/
00801 /* Initialization routine to call ADC LLD AP
00802 I's */
00803 ADC_MEASUREMENT_ADV_STATUS_t ADC_MEASUREMENT
00804 _ADV_Init(const ADC_MEASUREMENT_ADV_t *const handl
00805 e_ptr)
00806 {
00807     XMC_ASSERT("ADC_MEASUREMENT_ADV_Init:Inval
00808 id handle_ptr", (handle_ptr != NULL))
00809
00810     const ADC_MEASUREMENT_ADV_CHANNEL_t *index
00811 ed;
00812     uint8_t ch_num;
00813     uint8_t total_number_of_channels;
00814     ADC_MEASUREMENT_ADV_STATUS_t status;
00815 #ifdef ADC_MEASUREMENT_ADV_SHS_GAIN_NON_DEFA
00816 ULT
00817     uint8_t channel_number;
00818 #endif
00819     bool arbitration_status = (bool)false;
```

```
00808
00809     if (ADC_MEASUREMENT_ADV_STATUS_UNINITIALIZ
ED == *handle_ptr->init_state)
00810     {
00811
00812         arbitration_status = ADC_MEASUREMENT_ADV
 _lArbitrationStatus(handle_ptr);
00813
00814     /* Initialize the scan/queue request sour
ce.*/
00815     status = ADC_MEASUREMENT_ADV_lRequestSrcI
nit(handle_ptr);
00816
00817     /* Disable the Arbitration if no other in
stance has enabled it*/
00818     ADC_MEASUREMENT_ADV_lDisableArbitration(h
andle_ptr,arbitration_status);
00819
00820 #ifdef ADC_MEASUREMENT_ADV_SYNC_USED
00821     #ifndef ADC_MEASUREMENT_ADV_SYNC_NOT_ALL_US
ED
00822         if ((uint32_t)0 != handle_ptr->sync_slav
es)
00823     #endif
00824     {
00825         /* Configure the Sync conversion oper
ation */
00826         ADC_MEASUREMENT_ADV_lSyncInit(handle_p
tr);
00827     }
00828 #else
00829     ADC_MEASUREMENT_ADV_lSyncADCClocks();
00830 #endif
00831
00832     /* Initialize the SR lines for the Chann
el event and the Result event, if required*/
00833 #ifdef ADC_MEASUREMENT_ADV_MUX_USED
```

```
00834 #ifdef ADC_MEASUREMENT_ADV_MUX_NOT_ALL_USED
00835     if (handle_ptr->event_config != NULL)
00836 #endif
00837     {
00838         (handle_ptr->event_config)();
00839     }
00840 #endif
00841
00842     total_number_of_channels = (uint8_t)handle_ptr->total_number_of_channels;
00843     for (ch_num = (uint8_t)0; ch_num < (uint8_t)total_number_of_channels; ch_num++)
00844     {
00845         indexed = handle_ptr->channel_array[ch_num];
00846
00847         /* Initialize for configured channels*/
00848
00849         XMC_VADC_GROUP_ChannelInit(group_ptrs[ indexed->group_index],(uint32_t)indexed->ch_num, indexed->ch_handle);
00850 #if (XMC_VADC_SHS_AVAILABLE == 1U)
00851     #ifdef ADC_MEASUREMENT_ADV_SHS_GAIN_NONFAULT
00852         channel_number = indexed->ch_num;
00853     #ifdef ADC_MEASUREMENT_ADV_SHS_GAIN_ALIAS
00854         if (indexed->ch_handle->alias_channel != XMC_VADC_CHANNEL_ALIAS_DISABLED)
00855             {
00856                 channel_number = indexed->ch_handle->alias_channel;
00857             }
00858     #endif
00859         XMC_VADC_GLOBAL_SHS_SetGainFactor(ADC_
```

```
MEASUREMENT_ADV_SHS_PTR,
00860                                         (uin
t8_t)indexed->shs_gain_factor,
00861                                         (XMC
_VADC_GROUP_INDEX_t)indexed->group_index,
00862                                         chan
nel_number);
00863 #endif
00864 #endif
00865
00866         /* Result Init both with and without F
IFO */
00867         ADC_MEASUREMENT_ADC_1ResultInit(indexe
d);
00868
00869 #ifdef ADC_MEASUREMENT_ADV_ANALOG_IO_USED
00870         /* ANALOG_IO initialization for the ch
annel*/
00871 #ifndef ADC_MEASUREMENT_ADV_ANALOG_IO_ALL_CH
ANNELS_USED
00872         if(indexed->analog_io_config != NULL)
00873 #endif
00874         {
00875             status |= (ADC_MEASUREMENT_ADV_STATU
S_t) ANALOG_IO_Init(indexed->analog_io_config);
00876         }
00877 #endif
00878     }
00879
00880 #if defined(ADC_MEASUREMENT_ADV_ADC_SCAN_USE
D) || defined(ADC_MEASUREMENT_ADV_ADC_QUEUE_USED)
00881     /* Load the queue/scan entries into ADC_
QUEUE/ADC_SCAN.
00882     * This would load the scan/ queue entri
es into the software buffers in the ADC_SCAN/ADC_Q
UEUE APPs.
00883     * A call to this API would only configu
```

```
re the ADC_SCAN/ADC_QUEUE software buffers and will not be
0084      * programmed into the Hardware. The programming into the hardware is taken care by another API.
0085      */
0086 #if defined(ADC_MEASUREMENT_ADV_LOCAL_SCAN_USED) || defined(ADC_MEASUREMENT_ADV_LOCAL_QUEUE_USED)
0087      if( (ADC_MEASUREMENT_ADV_REQUEST_SOURCE_SCAN == handle_ptr->req_src) ||
0088          (ADC_MEASUREMENT_ADV_REQUEST_SOURCE_QUEUE == handle_ptr->req_src))
0089 #endif
0090      {
0091          for (ch_num = (uint8_t)0; ch_num < (uint8_t)handle_ptr->total_number_of_entries; ch_num++)
0092          {
0093              ADC_MEASUREMENT_ADV_lInsertEntry(handle_ptr, ch_num);
0094          }
0095      }
0096 #endif
0097
0098     /* Enables the NVIC node if NVIC node is consumed inside the APP*/
0099     ADC_MEASUREMENT_ADC_lNvicEnable(handle_ptr);
0100
0101     /* Load the queue/scan entries into the hardware */
0102     ADC_MEASUREMENT_ADV_lInsertChannels(handle_ptr);
0103
0104     /*Start the arbiter of the ADC request source after the initialization. */
```

```
00905 #ifdef ADC_MEASUREMENT_ADV_START_ADC
00906     #ifdef ADC_MEASUREMENT_ADV_NOT_ALL_REQ_STA
RT
00907         if ((bool)false != handle_ptr->start_at
_initialization)
00908     #endif
00909     {
00910         ADC_MEASUREMENT_ADV_StartADC(handle_pt
r);
00911     }
00912 #endif
00913
00914     *handle_ptr->init_state = status;
00915 }
00916 return (*handle_ptr->init_state);
00917 }
00918
00919 /*~~~~~
~~~~~
~~~~~*/
00920 /* Set the Fast compare value*/
00921 ADC_MEASUREMENT_ADV_STATUS_t ADC_MEASUREMENT
_ADV_SetFastCompareValue(const ADC_MEASUREMENT_ADV
_CHANNEL_t
00922
                                         *const handle_ptr, uint16
_t compare_value)
00923 {
00924     ADC_MEASUREMENT_ADV_STATUS_t status;
00925     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetFastCom
pareValue:Invalid handle_ptr", (handle_ptr != NULL
))
00926
00927     status = ADC_MEASUREMENT_ADV_STATUS_FAILURE
;
00928
00929     if ( (uint32_t)compare_value <= ADC_MEASUR
```

```
EMENT_ADV_10_BIT_MAX_VALUE)
00930    {
00931        XMC_VADC_GROUP_SetResultFastCompareValue
00932        (group_ptrs[handle_ptr->group_index],
00933         (uint32_t) handle_ptr->ch_handle->result_reg_num
00934         ber,
00935         (XMC_VADC_RESULT_SIZE_t)compare_value);
00936        status = ADC_MEASUREMENT_ADV_STATUS_SUCC
00937 ESS;
00938    }
00939 /*-----*
00940 *-----*/
00941 /* Set the Subtraction value */
00942 void ADC_MEASUREMENT_ADV_SetSubtractionValue(
00943 const ADC_MEASUREMENT_ADV_t *const handle_ptr,
00944
00945 ADC_MEASUREMENT_ADV_SUBTRACTION_t subtraction_align
00946 ment,
00947
00948 uint16_t subtraction_value)
00949 {
00950     uint32_t groups;
00951     uint8_t i;
00952     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetSubtrac
00953 tionValue:Invalid handle_ptr", (handle_ptr != NULL
00954 ))
00955
00956 #ifdef ADC_MEASUREMENT_ADV_SYNC_USED
00957     groups = handle_ptr->sync_slaves;
00958 #else
00959     groups = (uint32_t)0;
```



```
00974                                         (uint32_t)ha  
ndle_ptr->ch_handle->input_class);  
00975 }  
00976  
00977 /*-----  
-----*/  
00978 /* Sets the alternate reference for a partic  
ular channel*/  
00979 void ADC_MEASUREMENT_ADV_SetAlternateReferen  
ce(const ADC_MEASUREMENT_ADV_CHANNEL_t *const hand  
le_ptr,  
00980  
    const XMC_VADC_CHANNEL_REF_t reference_select)  
00981 {  
00982     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetAltera  
teReference:Invalid handle_ptr", (handle_ptr != NU  
LL))  
00983  
00984     XMC_VADC_GROUP_ChannelSetInputReference(gr  
oup_ptrs[handle_ptr->group_index], (uint32_t)hand  
le_ptr->ch_num,  
00985                                         re  
ference_select);  
00986  
00987 }  
00988  
00989 #if (XMC_VADC_SHS_AVAILABLE == 1U)  
00990 /*-----  
-----*/  
00991 /* Sets the gain ratio for a particular chan  
nel*/  
00992 void ADC_MEASUREMENT_ADV_SetChannelGain(const  
    ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_ptr,  
00993                                         const  
    ADC_MEASUREMENT_ADV_GAIN_t gain_factor)
```

```

00994 {
00995     uint8_t channel_number;
00996     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetChannel
Gain:Invalid handle_ptr", (handle_ptr != NULL))
00997
00998     channel_number = handle_ptr->ch_num;
00999 #ifdef ADC_MEASUREMENT_ADV_SHS_GAIN_ALIAS
01000     if (handle_ptr->ch_handle->alias_channel !=
= XMC_VADC_CHANNEL_ALIAS_DISABLED)
01001     {
01002         channel_number = (uint8_t) handle_ptr->c
h_handle->alias_channel;
01003     }
01004 #endif
01005     XMC_VADC_GLOBAL_SHS_SetGainFactor(ADC_MEAS
UREMENT_ADV_SHS_PTR,
01006                                         (uint8_t
)gain_factor,
01007                                         (XMC_VAD
C_GROUP_INDEX_t)handle_ptr->group_index,
01008                                         (uint32_
t)channel_number);
01009 }
01010 #endif
01011
01012 /*~~~~~*/~~~~~*/~~~~~*/
01013 /* Select the boundary for the channel*/
01014 void ADC_MEASUREMENT_ADV_SelectBoundary(const
ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_ptr,
01015                                         XMC_
VADC_BOUNDARY_SELECT_t boundary,
01016                                         XMC_
VADC_CHANNEL_BOUNDARY_t boundary_selection)
01017 {
01018     XMC_ASSERT("ADC_MEASUREMENT_ADV_SelectBoun

```

```
dary:Invalid handle_ptr", (handle_ptr != NULL))
01019
01020     XMC_VADC_GROUP_ChannelSetBoundarySelection
(group_ptrs[handle_ptr->group_index], (uint32_t)ha
ndle_ptr->ch_num,
01021
    boundary, boundary_selection);
01022
01023 }
01024
01025 /*~~~~~
~~~~~
~~~~~*/
01026 /* configure the upper boundary for a channe
l.*/
01027 void ADC_MEASUREMENT_ADV_SetBoundaryUpper(co
nst ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_pt
r,
01028
ui
nt32_t boundary_value)
01029 {
01030     uint32_t boundary_select;
01031     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetBoundar
y:Invalid handle_ptr", (handle_ptr != NULL))
01032
01033     boundary_select = handle_ptr->ch_handle->u
pper_boundary_select;
01034
01035     ADC_MEASUREMENT_ADV_lSetBoundary(handle_pt
r, (XMC_VADC_CHANNEL_BOUNDARY_t)boundary_select, b
oundary_value);
01036 }
01037
01038 /*~~~~~
~~~~~
~~~~~*/
01039 /* configure the lower boundary for a channe
```

```
1. */
01040 void ADC_MEASUREMENT_ADV_SetBoundaryLower(co
nstant ADC_MEASUREMENT_ADV_CHANNEL_t *const handle_pt
r,
01041                                         ui
nt32_t boundary_value)
01042 {
01043     uint32_t boundary_select;
01044     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetBoundar
y:Invalid handle_ptr", (handle_ptr != NULL))
01045
01046     boundary_select = handle_ptr->ch_handle->l
ower_boundary_select;
01047
01048     ADC_MEASUREMENT_ADV_lSetBoundary(handle_pt
r, (XMC_VADC_CHANNEL_BOUNDARY_t)boundary_select, b
oundary_value);
01049 }
01050
01051
01052 /*~~~~~*/
01053 /* Aliased channel number is returned if the
   channel has alias enabled */
01054 XMC_VADC_CHANNEL_ALIAS_t ADC_MEASUREMENT_ADV
 _GetAliasValue(const ADC_MEASUREMENT_ADV_CHANNEL_t
 *const handle_ptr)
01055 {
01056     XMC_VADC_CHANNEL_ALIAS_t return_value;
01057     ADC_MEASUREMENT_ADV_ALIAS_t alias_value;
01058
01059     XMC_ASSERT("ADC_MEASUREMENT_ADV_GetAliasVa
lue:Invalid handle_ptr", (handle_ptr != NULL))
01060
01061     alias_value.alias = XMC_VADC_GROUP_GetAlia
s(group_ptrs[handle_ptr->group_index]);
```

```
01062     if ((uint8_t)0 == handle_ptr->ch_num )
01063     {
01064         return_value = (XMC_VADC_CHANNEL_ALIAS_t
01065             )alias_value.alias0;
01066         if ((uint32_t)0 == alias_value.alias0)
01067         {
01068             return_value = XMC_VADC_CHANNEL_ALIAS_
01069             DISABLED;
01070         }
01071     }
01072     else if ((uint8_t)1 == handle_ptr->ch_num )
01073     {
01074         return_value = (XMC_VADC_CHANNEL_ALIAS_t
01075             )alias_value.alias1;
01076         if ((uint32_t)1 == alias_value.alias1)
01077         {
01078             return_value = XMC_VADC_CHANNEL_ALIAS_
01079             DISABLED;
01080         }
01081     }
01082
01083     return(return_value);
01084
01085 }
01086 /*~~~~~
~~~~~
~~~~~*/
01087
01088 #ifdef ADC_MEASUREMENT_ADV_SYNC_USED
01089 /* Enables uniform conversion configurations
01090 across slaves*/
```

```
01090 void ADC_MEASUREMENT_ADV_SetIclass(const ADC
 _MEASUREMENT_ADV_t *const handle_ptr)
01091 {
01092     XMC_VADC_CHANNEL_CONV_t req_iclass;
01093     XMC_VADC_GROUP_CLASS_t conv_class;
01094     XMC_VADC_GLOBAL_CLASS_t conv_class_global;
01095
01096     XMC_ASSERT("ADC_MEASUREMENT_ADV_SetIclass:
Invalid handle_ptr", (handle_ptr != NULL))
01097
01098     req_iclass = ADC_MEASUREMENT_ADV_1GetIclas
s(handle_ptr);
01099     conv_class = XMC_VADC_GROUP_GetInputClass(
group_ptrs[handle_ptr->group_index], req_iclass);
01100     conv_class_global.globiclass = conv_class.
g_iclass0;
01101     XMC_VADC_GLOBAL_InputClassInit(ADC_MEASURE
MENT_ADV_GLOBAL_PTR, conv_class_global,
01102                                         XMC_VADC_GR
01103                                         OUP_CONV_STD, (uint32_t)ADC_MEASUREMENT_ADV_GLOBIC
LASS1);
01104 }
01105 #endif
```

