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
- Release History
License Terms and Copyright Information

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

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

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

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

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

## Abbreviations:

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

## Definitions:

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

## Overview

**GLOBAL_ADC** is a basic APP which configures VADC Global Registers. It configures the VADC clocks and other functions. *The **GLOBAL_ADC** APP provides the following functionalities to configure the VADC peripheral:*

1. Configure the Analog clock.
2. Configure the digital clock (arbitration clock).
3. Startup calibration
4. Configures post calibration for each group.
5. Configures arbiter behaviour for each group.
**Figure 1**: Overview of VADC peripheral

Figure 1 shows the overview of the VADC peripheral. In this the **GLOBAL_ADC** APP would configure the clock dividers the entire module. It would also do all the necessary initialization for the module.
Figure 2: Hardware and Software connectivity of GLOBAL_ADC APP

Figure 2, shows how the APP is structured in DAVE. XMC controllers provide the VADC module be used for analog to digital conversion. The LLD layer provides abstraction for these hardware modules. The GLOBAL_ADC APP uses VADC and SCU LLDs and other dependent APPS like CLOCK_XMCx for the functionality.

Supported Devices
The APP supports below devices:

1. XMC4500 Series
2. XMC4400 Series
3. XMC4300 Series
4. XMC4200 / XMC4100 Series
5. XMC1300 Series
6. XMC1200 Series
7. XMC1100 Series

Reference

1. XMC4500 Reference Manual
2. XMC4400 Reference Manual
5. XMC1300 Reference Manual
Architecture Description

Figure 1 explains the architecture of the APP: This would pictorially represent the internal workings of the GLOBAL_ADC. This shows the hardware resources that are consumed, the depended APPs and the various signals that would be exported out.
Figure 1: Architecture of GLOBAL_ADC APP

The diagram above represents the internal software architecture of the GLOBAL_ADC APP. A GLOBAL_ADC APP instance exists in a DAVE-4.0 project with fixed attributes as shown and configures the VADC peripheral. This in addition requires the consumption of the CLOCK APPS for its configuration and functioning. The GLOBAL_ADC
APP also provides output signals for inter-peripheral connections.

**Signals:**

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

**Table 1: APP IO signals**

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Input/Output</th>
<th>Availability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>global_signal</td>
<td>Output</td>
<td>Always</td>
<td>Global connection signal: Should be connected to the APP which consumes GLOBAL_ADC. By default.</td>
</tr>
<tr>
<td>end_of_arbitration_round_group0</td>
<td>Output</td>
<td>Always</td>
<td>End of arbitration round signal from group-0: Can be connected to the VADC module for gating or trigger to various request sources. An end of arbitration.</td>
</tr>
</tbody>
</table>
round raises a signal which can also be given to other peripherals.

end_of_sampling_signal_of_group0  Output  Always

End of sampling phase signal from group 0:
Can be connected internally to the VADC module for gating or trigger for the various request sources.
Generates a signal when the sampling signal is completed.

end_of_arbitration_round_group1  Output  Always

End of arbitration round signal from group 1:
Can be connected to the VADC module for gating or trigger for the various request sources.
An end
an arbitration round raises a signal which can also be given to other peripherals.

End of sampling phase signal from group-1:
Can be connected internally to the VADC module for gating or trigger for the various request sources.
Generates a signal when the sampling phase completes.

end_of_sampling_signal_of_group1  Output  Always

End of arbitration round signal from group-2:
Can be connected to the VADC module for gating or trigger for the various request sources.

end_of_arbitration_round_group2  Output  Always
<table>
<thead>
<tr>
<th>end_of_sampling_signal_of_group2</th>
<th>Output</th>
<th>Always</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of sampling signal of group 2 from group-2: Can be connected internally to the VADC module for gating or trigger for the various request sources. Generates a signal when the sampling of a signal is completed.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>end_of_arbitration_round_group3</th>
<th>Output</th>
<th>Always</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of arbitration round signal from group-3: Can be connected to the VADC module for gating or trigger for the various request sources.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An end of arbitration round raises a signal which can also be given to other peripherals.
An end of the arbitration round raises a signal which can also be given to other peripherals.

<table>
<thead>
<tr>
<th>end_of_sampling_signal_of_group3</th>
<th>Output</th>
<th>Always</th>
</tr>
</thead>
</table>

End of sampling phase signal from group-3:
Can be connected internally to the VADC module for gating or trigger for various request sources.
Generates a signal when the sampling of a signal is completed.

<table>
<thead>
<tr>
<th>group0_input_queue</th>
<th>Input</th>
<th>Always</th>
</tr>
</thead>
</table>

Queue selection for Group-0:
If a ADC_QUEUE_APP is consumed by the group to which the Queue belongs.
has to be forced to group-0. If a connection can be made to the ADC_QUEUE APP, a scan
selection for Group-0 can be made.

Group-0 (app_group0)

- **group0_input_scan**: Input, Always
  - Scan selection for Group-0.
  - If an ADC_SCAN is consumed and the group to which the scan belongs has to be forced to group-0, a connection can be made to the ADC_SCAN_APP.

- **sync_request_signal_group0**: Input, Always
  - Sync request signal for Group-0.
  - If a sync operation is needed and if the slave group must be forced to group-0, a connection can be made.
| input_queue                  | Input  | Always | |-----------------------------|-------|--------|
| group1_input_queue          |       |        |
| Input                       |       |        |
| APP                         |       |        |
| selection                   |       |        |
| Group-1                      |       |        |
| If a ADC_QUEUE APP is consumed and the group to which the Queue belongs has to be forced to group-1, a connection can be made to the ADC_QUEUE APP.

| input_scan                  | Input  | Always | |-----------------------------|-------|--------|
| group1_input_scan           |       |        |
| Input                       |       |        |
| APP                         |       |        |
| Scan selection              |       |        |
| for Group-1                 |       |        |
| If a ADC_SCAN APP is consumed and the group to which the scan belongs has to be forced to group-1, a connection can be made to the ADC_SCAN APP.

<p>| Sync request                | Input  | Always | |-----------------------------|-------|--------|
| Sync request                |       |        |
| signal for Group-1           |       |        |
| If a sync operation is selected... |       |        |</p>
<table>
<thead>
<tr>
<th>sync_request_signal_group1</th>
<th>Input</th>
<th>Always</th>
<th>needed. The slave must be forced to group-1 if the connection is made.</th>
</tr>
</thead>
<tbody>
<tr>
<td>group2_input_queue</td>
<td>Input</td>
<td>Always</td>
<td>Queue selection for Group-2. If an ADC_QUEUE APP is consumed and the group to which the Queue belongs has to be forced to group-2, a connection to the ADC_QUEUE APP can be made.</td>
</tr>
<tr>
<td>group2_input_scan</td>
<td>Input</td>
<td>Always</td>
<td>Scan selection for Group-2. If an ADC_SCAN APP is consumed and the group to which the scan belongs has to be forced to group-2, a connection can be made.</td>
</tr>
<tr>
<td>sync_request_signal_group2</td>
<td>Input</td>
<td>Always</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>group3_input_queue</td>
<td>Input</td>
<td>Always</td>
<td></td>
</tr>
</tbody>
</table>

Sync request signal for Group-2

If a sync operation is needed and if the slave group must be forced to group-2, this connection can be made.

Queue selection for Group-3

If an ADC_QUEUE APP is consumed and the group to which the Queue belongs has to be forced to group-3, a connection can be made to the ADC_QUEUE APP.

Scan selection for Group-3

If a scan needs to be made, the ADC_SCAN APP

be made to the ADC_SCAN APP.
<table>
<thead>
<tr>
<th>Function</th>
<th>Input Type</th>
<th>Typ</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>group3_input_scan</td>
<td>Input</td>
<td>Always</td>
<td></td>
</tr>
<tr>
<td>sync_request_signal_group3</td>
<td>Input</td>
<td>Always</td>
<td></td>
</tr>
</tbody>
</table>

APP is consumed and the group to which the scan belongs must be forced to group-3. A connection can be made to ADC_SCAN_APP.

Sync request signal for Group-3. If a sync operation is needed and the slave group must be forced to group-3, a connection can be made.
**Global ADC**

**APP Configuration Parameters**

### General Settings

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral bus clock (MHz):</td>
<td>120</td>
</tr>
<tr>
<td>Desired analog clock (MHz):</td>
<td>30</td>
</tr>
<tr>
<td>Actual analog clock (MHz):</td>
<td>30</td>
</tr>
<tr>
<td>Digital clock:</td>
<td>fADC</td>
</tr>
<tr>
<td>Actual digital clock (MHz):</td>
<td>120</td>
</tr>
</tbody>
</table>

- **Enable startup calibration**

**Figure 1: General Settings**
Figure 2: Advanced Settings
# GLOBAL_ADC

## Enumerations

```c
enum GLOBAL_ADC_STATUS {
    GLOBAL_ADC_SUCCESS = 0,
    GLOBAL_ADC_FAILURE,
    GLOBAL_ADC_UNINITIALIZED
};

typedef enum GLOBAL_ADC_STATUS GLOBAL_ADC_STATUS_t
```

GLOBAL_ADC state information.

More...
### Enumeration Type Documentation

```c
enum GLOBAL_ADC_STATUS
{
    GLOBAL_ADC_SUCCESS,  // APP is in INITIALIZED state after execution of the Init function
    GLOBAL_ADC_FAILURE,  // Initialization failed returns this as status
    GLOBAL_ADC_UNINITIALIZED  // This is the default state after power on reset.
};
```

Definition at line 101 of file GLOBAL_ADC.h.
<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data structures</td>
<td></td>
</tr>
</tbody>
</table>
## Data Structures

<table>
<thead>
<tr>
<th>struct</th>
<th>GLOBAL_ADC_GROUP</th>
<th>Structure to hold the configuration information of a group. More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct</td>
<td>GLOBAL_ADC_GROUP</td>
<td>GLOBAL_ADC_GROUP_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Structure to hold the configuration information of a group.</td>
</tr>
</tbody>
</table>

| typedef struct | GLOBAL_ADC       | GLOBAL_ADC_t                                                        |
|               |                  | Configuration Data structure of GLOBAL_ADC APP.                    |
# GLOBAL_ADC

## Methods

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>GLOBAL_ADC_GetAppVersion (void)</td>
<td>Get GLOBAL_ADC APP version.</td>
</tr>
<tr>
<td>GLOBAL_ADC_STATUS_t</td>
<td>GLOBAL_ADC_Init (GLOBAL_ADC_t *const handle_ptr)</td>
<td>Initializes the ADC global as per user configured values.</td>
</tr>
</tbody>
</table>
Function Documentation

DAVE_APP_VERSION_t GLOBAL_ADC_GetAppVersion (void)

Get GLOBAL_ADC APP version.

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

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

Example Usage:

```c
#include <DAVE.h>

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

    // Initialize GLOBAL_ADC APP:
    // GLOBAL_ADC_Init() is called from within DAVE_Init().
    init_status = DAVE_Init();

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

    // More code here
    while(1) {

    }
```
GLOBAL_ADC_STATUS_t GLOBAL_ADC_Init (GLOBAL_ADC_t *const)

Initializes the ADC global as per user configured values.

**Returns:**
void

**Description:**
Initializes the VADC peripheral. Invokes various VADC LLD drivers to initialize the VADC peripheral. This would invoke The XMC_VADC_GLOBAL_Init(), XMC_VADC_GROUP_Init(). It also invokes XMC_VADC_GROUP_SetPowerMode() to power on available groups.

**Example Usage:**
```
#include <DAVE.h>
int main (void)
{
    DAVE_Init(); //GLOBAL_ADC_Init is called with in DAVE_Init
    while(1);
    return 0;
}
```

This function initializes all instances of the ADC Global APP and low level app.

Definition at line 110 of file GLOBAL_ADC.c.
References  GLOBAL_ADC::enable_start_up_calibration, GLOBAL_ADC_SUCCESS, GLOBAL_ADC_UNINITIALIZED, GLOBAL_ADC::global_config_handle, GLOBAL_ADC::global_shs_ptr, GLOBAL_ADC_GROUP::group_config_handle, GLOBAL_ADC_GROUP::group_handle, GLOBAL_ADC::group_ptrs_array, GLOBAL_ADC::init_state, GLOBAL_ADC::module_ptr, GLOBAL_ADC_GROUP::post_calibration, and GLOBAL_ADC_GROUP::state.
GLOBAL_ADC

Home

Release History

Release History
GLOBAL_ADC

Data Structures

Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_ADC</td>
<td>Configuration Data structure of GLOBAL_ADC APP</td>
</tr>
<tr>
<td>GLOBAL_ADC_GROUP</td>
<td>Structure to hold the configuration information of a group</td>
</tr>
<tr>
<td>Home</td>
<td>Data Structures</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>

GLOBAL_ADC Struct Reference
Detailed Description

Configuration Data structure of `GLOBAL_ADC` APP.

Definition at line 136 of file `GLOBAL_ADC.h`.

#include <GLOBAL_ADC.h>
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GLOBAL_ADC_GROUP_t *const</code></td>
<td><code>group_ptrs_array[XMC_VADC_MAXIMUM_NUM_GROUPS]</code></td>
</tr>
<tr>
<td><code>const XMC_VADC_GLOBAL_CONFIG_t *const</code></td>
<td><code>global_config_handle</code></td>
</tr>
<tr>
<td><code>XMC_VADC_GLOBAL_t *const</code></td>
<td><code>module_ptr</code></td>
</tr>
<tr>
<td><code>XMC_VADC_GLOBAL_SHS_t *const</code></td>
<td><code>global_shs_ptr</code></td>
</tr>
<tr>
<td><code>GLOBAL_ADC_STATUS_t</code></td>
<td><code>init_state</code></td>
</tr>
<tr>
<td><code>const bool</code></td>
<td><code>enable_startup_calibration</code></td>
</tr>
<tr>
<td>Field Documentation</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td><strong>const bool</strong> <code>GLOBAL_ADC::enable_startup_calibration</code></td>
<td></td>
</tr>
<tr>
<td>Enable startup calibration for all the converters</td>
<td></td>
</tr>
<tr>
<td>Definition at line <strong>150</strong> of file <code>GLOBAL_ADC.h</code></td>
<td></td>
</tr>
<tr>
<td>Referenced by <code>GLOBAL_ADC_Init()</code></td>
<td></td>
</tr>
<tr>
<td><strong>const XMC_VADC_GLOBAL_CONFIG_t</strong> <code>GLOBAL_ADC::global_config_handle</code></td>
<td></td>
</tr>
<tr>
<td>This is the pointer to the Global LLD Handle.</td>
<td></td>
</tr>
<tr>
<td>Definition at line <strong>141</strong> of file <code>GLOBAL_ADC.h</code></td>
<td></td>
</tr>
<tr>
<td>Referenced by <code>GLOBAL_ADC_Init()</code></td>
<td></td>
</tr>
<tr>
<td><strong>XMC_VADC_GLOBAL_SHS_t</strong> <code>GLOBAL_ADC::global_shs_ptr</code></td>
<td></td>
</tr>
<tr>
<td>This is the sample and hold structure pointer</td>
<td></td>
</tr>
<tr>
<td>Definition at line <strong>146</strong> of file <code>GLOBAL_ADC.h</code></td>
<td></td>
</tr>
<tr>
<td>Referenced by <code>GLOBAL_ADC_Init()</code></td>
<td></td>
</tr>
<tr>
<td><strong>GLOBAL_ADC_GROUP_t</strong> <code>GLOBAL_ADC::group_ptrs_array[XMC_VADC_MAXIMUM_NUM_GROUPS]</code></td>
<td></td>
</tr>
<tr>
<td>This is an array of pointers to the ADC Groups</td>
<td></td>
</tr>
<tr>
<td>Definition at line <strong>139</strong> of file <code>GLOBAL_ADC.h</code></td>
<td></td>
</tr>
<tr>
<td>Referenced by <code>GLOBAL_ADC_Init()</code></td>
<td></td>
</tr>
</tbody>
</table>
GLOBAL_ADC_STATUS_t GLOBAL_ADC::init_state

This hold the State of the GLOBAL_ADC APP

Definition at line 148 of file GLOBAL_ADC.h.

Referenced by GLOBAL_ADC_Init().

XMC_VADC_GLOBAL_t* const GLOBAL_ADC::module_ptr

This is the register structure pointer to the VADC kernel.

Definition at line 143 of file GLOBAL_ADC.h.

Referenced by GLOBAL_ADC_Init().

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

- GLOBAL_ADC.h
GLOBAL_ADC

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

GLOBAL_ADC_GROUP Struct Reference

Data structures
Detailed Description

Structure to hold the configuration information of a group.

Definition at line 122 of file GLOBAL_ADC.h.

#include <GLOBAL_ADC.h>
## Data Fields

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMC_VADC_GROUP_t *const</td>
<td>group_handle</td>
</tr>
<tr>
<td>const XMC_VADC_GROUP_CONFIG_t *const</td>
<td>group_config_handle</td>
</tr>
<tr>
<td>const bool</td>
<td>post_calibration</td>
</tr>
<tr>
<td>GLOBAL_ADC_STATUS_t</td>
<td>state</td>
</tr>
</tbody>
</table>
Field Documentation

**const XMC_VADC_GROUP_CONFIG_t**

This is the pointer to the Handle of the Group APP.

Definition at line 126 of file `GLOBAL_ADC.h`.

Referenced by `GLOBAL_ADC_Init()`.

**XMC_VADCGROUP_t**

This holds the VADC group Registers.

Definition at line 124 of file `GLOBAL_ADC.h`.

Referenced by `GLOBAL_ADC_Init()`.

**const bool**

This enables the post calibration for a specific group

Definition at line 128 of file `GLOBAL_ADC.h`.

Referenced by `GLOBAL_ADC_Init()`.

**GLOBAL_ADC_STATUS_t**

This enumerates the state of the APP.

Definition at line 130 of file `GLOBAL_ADC.h`.

Referenced by `GLOBAL_ADC_Init()`.
The documentation for this struct was generated from the following file:

- **GLOBAL_ADC.h**
<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
</tr>
</thead>
</table>

### Data Structure Index

<table>
<thead>
<tr>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GLOBAL_ADC</strong></td>
</tr>
<tr>
<td><strong>GLOBAL_ADC_GROUP</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G</th>
</tr>
</thead>
</table>
Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- `enable_startup_calibration` : `GLOBAL_ADC`
- `global_config_handle` : `GLOBAL_ADC`
- `global_shs_ptr` : `GLOBAL_ADC`
- `group_config_handle` : `GLOBAL_ADC_GROUP`
- `group_handle` : `GLOBAL_ADC_GROUP`
- `group_ptrs_array` : `GLOBAL_ADC`
- `init_state` : `GLOBAL_ADC`
- `module_ptr` : `GLOBAL_ADC`
- `post_calibration` : `GLOBAL_ADC_GROUP`
- `state` : `GLOBAL_ADC_GROUP`
GLOBAL_ADC

- enable_startup_calibration: GLOBAL_ADC
- global_config_handle: GLOBAL_ADC
- global_shs_ptr: GLOBAL_ADC
- group_config_handle: GLOBAL_ADC_GROUP
- group_handle: GLOBAL_ADC_GROUP
- group_ptrs_array: GLOBAL_ADC
- init_state: GLOBAL_ADC
- module_ptr: GLOBAL_ADC
- post_calibration: GLOBAL_ADC_GROUP
- state: GLOBAL_ADC_GROUP
GLOBAL_ADC

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

- GLOBAL_ADC.c [code]
- GLOBAL_ADC.h [code]
## GLOBAL_ADC

<table>
<thead>
<tr>
<th>File List</th>
<th>Globals</th>
<th>Functions</th>
</tr>
</thead>
</table>

**GLOBAL_ADC.c File**  
Reference
Detailed Description

Date:
2015-03-18

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

Definition in file GLOBAL_ADC.c.
### Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>GLOBAL_ADC_GetAppVersion (void)</td>
<td>Get GLOBAL_ADC APP version.</td>
</tr>
<tr>
<td>GLOBAL_ADC_STATUS_t</td>
<td>GLOBAL_ADC_Init (GLOBAL_ADC_t* const handle_ptr)</td>
<td>Initializes the ADC global as per user configured values.</td>
</tr>
</tbody>
</table>
Function Documentation

GLOBAL_ADC_STATUS_t GLOBAL_ADC_Init (GLOBAL_ADC_t *const

Initializes the ADC global as per user configured values.

This function initializes all instances of the ADC Global APP and low level app.

Definition at line 110 of file GLOBAL_ADC.c.

References GLOBAL_ADC::enable_startup_calibration,
GLOBAL_ADC_SUCCESS, GLOBAL_ADC_UNINITIALIZED,
GLOBAL_ADC::global_config_handle,
GLOBAL_ADC::global_shs_ptr,
GLOBAL_ADC_GROUP::group_config_handle,
GLOBAL_ADC_GROUP::group_handle,
GLOBAL_ADC::group_ptrs_array, GLOBAL_ADC::init_state,
GLOBAL_ADC::module_ptr,
GLOBAL_ADC_GROUP::post_calibration, and
GLOBAL_ADC_GROUP::state.

Go to the source code of this file.
GLOBAL_ADC

GLOBAL_ADC.h File Reference
Detailed Description

Date:
2015-03-18

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

Definition in file `GLOBAL_ADC.h`.
## Data Structures

<table>
<thead>
<tr>
<th>struct</th>
<th>GLOBAL_ADC_GROUP</th>
<th>Structure to hold the configuration information of a group. More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>GLOBAL_ADC</td>
<td>Configuration Data structure of <strong>GLOBAL_ADC</strong> APP. More...</td>
</tr>
</tbody>
</table>
Typedefs

<table>
<thead>
<tr>
<th>Type Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef struct GLOBAL_ADC_GROUP</code></td>
<td><code>GLOBAL_ADC_GROUP_t</code> Structure to hold the configuration information of a group.</td>
</tr>
<tr>
<td><code>typedef struct GLOBAL_ADC</code></td>
<td><code>GLOBAL_ADC_t</code> Configuration Data structure of <code>GLOBAL_ADC</code> APP.</td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DAVE_APP_VERSION_t</code></td>
<td><code>GLOBAL_ADC_GetAppVersion(void)</code>&lt;br&gt;Get <code>GLOBAL_ADC APP</code> version.</td>
</tr>
<tr>
<td><code>GLOBAL_ADC_STATUS_t</code></td>
<td><code>GLOBAL_ADC_Init(GLOBAL_ADC_t *const handle_ptr)</code>&lt;br&gt;Initializes the ADC global as per user configured values.</td>
</tr>
<tr>
<td><code>enum</code></td>
<td><code>GLOBAL_ADC_STATUS {</code>&lt;br&gt;<code>GLOBAL_ADC_SUCCESS =</code>&lt;br&gt;<code>GLOBAL_ADC_FAILURE,</code>&lt;br&gt;<code>GLOBAL_ADC_UNINITIALIZED</code>&lt;br&gt;<code>}</code>&lt;br&gt;<code>GLOBAL_ADC state information. More...</code></td>
</tr>
<tr>
<td><code>typedef enum</code></td>
<td><code>GLOBAL_ADC_STATUS</code>&lt;br&gt;<code>GLOBAL_ADC_STATUS_t</code>&lt;br&gt;<code>GLOBAL_ADC state information.</code></td>
</tr>
</tbody>
</table>
GLOBAL_ADC

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

- `GLOBAL_ADC_FAILURE` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_GetAppVersion()` : `GLOBAL_ADC.c` , `GLOBAL_ADC.h`
- `GLOBAL_ADC_GROUP_t` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_Init()` : `GLOBAL_ADC.c` , `GLOBAL_ADC.h`
- `GLOBAL_ADC_STATUS` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_STATUS_t` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_SUCCESS` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_t` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_UNINITIALIZED` : `GLOBAL_ADC.h`
GLOBAL_ADC

- `GLOBAL_ADC_GetAppVersion()` : `GLOBAL_ADC.c`, `GLOBAL_ADC.h`
- `GLOBAL_ADC_Init()` : `GLOBAL_ADC.c`, `GLOBAL_ADC.h`
GLOBAL_ADC

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

- `GLOBAL_ADC_GROUP_t` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_STATUS_t` : `GLOBAL_ADC.h`
- `GLOBAL_ADC_t` : `GLOBAL_ADC.h`
GLOBAL_ADC

- GLOBAL_ADC_STATUS : GLOBAL_ADC.h
GLOBAL_ADC

- GLOBAL_ADC_FAILURE : GLOBAL_ADC.h
- GLOBAL_ADC_SUCCESS : GLOBAL_ADC.h
- GLOBAL_ADC_UNINITIALIZED : GLOBAL_ADC.h
GLOBAL_ADC

Go to the documentation of this file.

```plaintext
#ifndef GLOBAL_ADC_H
#define GLOBAL_ADC_H

/*******************************************
**************************************************
**************************
* HEADER FILES
*******************************************
**************************************************
**************************
#include <xmc_vadc.h>
#include "global_adc_conf.h"
#include <DAVE_common.h>

/******************************************
**************************************************
**************************
* MACROS
*******************************************
**************************************************
**************************
#if (!((XMC_LIB_MAJOR_VERSION == 2U) && 
     (XMC_LIB_MINOR_VERSION >= 0U) && 
     (XMC_LIB_PATCH_VERSION >= 0U)))
#error "GLOBAL_ADC requires XMC Peripheral Library v2.0.0 or higher"
```
typedef enum GLOBAL_ADC_STATUS {
GLOBAL_ADC_SUCCESS = 0,
GLOBAL_ADC_FAILURE,
GLOBAL_ADC_UNINITIALIZED
} GLOBAL_ADC_STATUS_t;

#if  XMC_VADC_GROUP_AVAILABLE  ==  1U
typedef struct GLOBAL_ADC_GROUP {
    XMC_VADC_GROUP_t *const group_handle;
    const XMC_VADC_GROUP_CONFIG_t* const group_config_handle;
    const bool post_calibration;
    GLOBAL_ADC_STATUS_t state;
} GLOBAL_ADC_GROUP_t;
#elif
#define GLOBAL_ADC_GROUP
#endif

typedef struct GLOBAL_ADC
{
#if XMC_VADC_GROUP_AVAILABLE == 1U
    GLOBAL_ADC_GROUP_t* const group_ptrs_array [XMC_VADC_MAXIMUM_NUM_GROUPS];
#endif
    const XMC_VADC_GLOBAL_CONFIG_t* const global_config_handle;
    XMC_VADC_GLOBAL_t* const module_ptr;
#if(XMC_VADC_SHS_AVAILABLE == 1U)
    XMC_VADC_GLOBAL_SHS_t* const global_shs_ptr;
#endif
    GLOBAL_ADC_STATUS_t init_state;
    const bool enable_startup_calibration;
} GLOBAL_ADC_t;

#ifndef __cplusplus
extern "C" {
#endif
/********************
* API Prototypes
********************

DAVE_APP_VERSION_t GLOBAL_ADC_GetAppVersion( void);
GLOBAL_ADC_STATUS_t GLOBAL_ADC_Init(GLOBAL_ADC_t *const handle_ptr);
#include "global_adcExtern.h"
```c
#ifdef __cplusplus
#endif
#endif /* GLOBAL_ADC_H_ */
```
GLOBAL_ADC

Go to the documentation of this file.

```c
#include "global_adc.h"
```

---

GLOBAL_ADC.c
/*This function returns the version of the GLOBAL_ADC APP*/

DAVE_APP_VERSION_t GLOBAL_ADC_GetAppVersion(void)
{
    DAVE_APP_VERSION_t version;
    version.major = (uint8_t) GLOBAL_ADC_MAJOR_VERSION;
    version.minor = (uint8_t) GLOBAL_ADC_MINOR_VERSION;
    version.patch = (uint8_t) GLOBAL_ADC_PATCH_VERSION;
    return version;
}

GLOBAL_ADC_STATUS_t GLOBAL_ADC_Init(GLOBAL_ADC_t *const handle_ptr)
{  
XMC_ASSERT("GLOBAL_ADC_Init:Invalid handle_ptr", (handle_ptr != NULL))
#if (XMC_VADC_GROUP_AVAILABLE == 1U)
  
  uint32_t group_index;
#endif

if (GLOBAL_ADC_UNINITIALIZED == handle_ptr->init_state) {
  /* Initialize an instance of Global hardware */
  XMC_VADC_GLOBAL_Init(handle_ptr->module_ptr, handle_ptr->global_config_handle);

  /* Initialize all the Groups */
  #if (XMC_VADC_GROUP_AVAILABLE == 1U)
  for(group_index = (uint32_t)0; group_index < XMC_VADC_MAXIMUM_NUM_GROUPS; group_index++)
  {
    /*Initialize Group*/
    XMC_VADC_GROUP_Init(handle_ptr->groupptrs_array[group_index]->group_handle,
                        handle_ptr->groupptrs_array[group_index]->group_config_handle);
    
    /* Switch on the converter of the Group[group_index]*/
    XMC_VADC_GROUP_SetPowerMode(handle_ptr->groupptrs_array[group_index]->group_handle,
                               XMC_VADC_GROUP_POWERMODE_NORMAL);
    
    /* Disable the post calibration option for the respective group*/
    if (!true == handle_ptr->groupptrs_array[group_index]->post_calibration)
      
    } #endif
  } /* End of if initialized */
#endif
}
{ 
XMC_VADC_GLOBAL_DisablePostCalibration(handle_ptr->module_ptr, group_index);
}

#if(XMC_VADC_SHSAVAILABLE == 1U)
XMC_VADC_GLOBAL_SHS_EnableAcceleratedMode(handle_ptr->global_shs_ptr, (XMC_VADC_GROUP_INDEX_t)group_index);
#endif

handle_ptr->group_ptrs_array[group_index]->state = GLOBAL_ADC_SUCCESS;

#endif /* _XMC_VADC_GROUP_AVAILABLE */

if((bool)true == handle_ptr->enable_startuption)
{
XMC_VADC_GLOBAL_StartupCalibration(handle_ptr->module_ptr);
}

handle_ptr->init_state = GLOBAL_ADC_SUCCESS;

return (handle_ptr->init_state);