

# CPU\_CTRL\_XMC1

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
- CPU\_CTRL\_XMC1

# CPU\_CTRL\_XMC1

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](mailto:dave@infineon.com)).

---

# CPU\_CTRL\_XMC1

Home

## Abbreviations and Definitions

### Abbreviations and Definitions

<b>Abbreviations:</b>	
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
SCU	System Control Unit

<b>Definitions:</b>	
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) APPs



# CPU\_CTRL\_XMC1

Home

## Overview

### Overview

The **CPU\_CTRL\_XMC1** APP is a system APP. The **CPU\_CTRL\_XMC1** APP provides functionality to configure the behavior of the CPU:

- Debug interface
- Exceptions

### Debug Interface:

The XMC1 series implement various debug interfaces:

- SWD (Serial Wire Debug)
- SPD (Single Pin Debug)

The APP allows reserving the pins needed for the selected interface.

Following pins used for SWD: SWDCLK and SWDIO.  
SWD0 (P0.14, P0.15), SWD1 (P1.3, P1.2)

Following pins used for SPD: SPDIO  
SPD0(P0.14) and SPD1(P1.3)

### Exceptions:

Exceptions in XMC1 series trap illegal memory access and illegal program behavior. The following types of faults are available:

- HardFault: A HardFault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. HardFaults have a fixed priority of -1, meaning they have higher priority than any

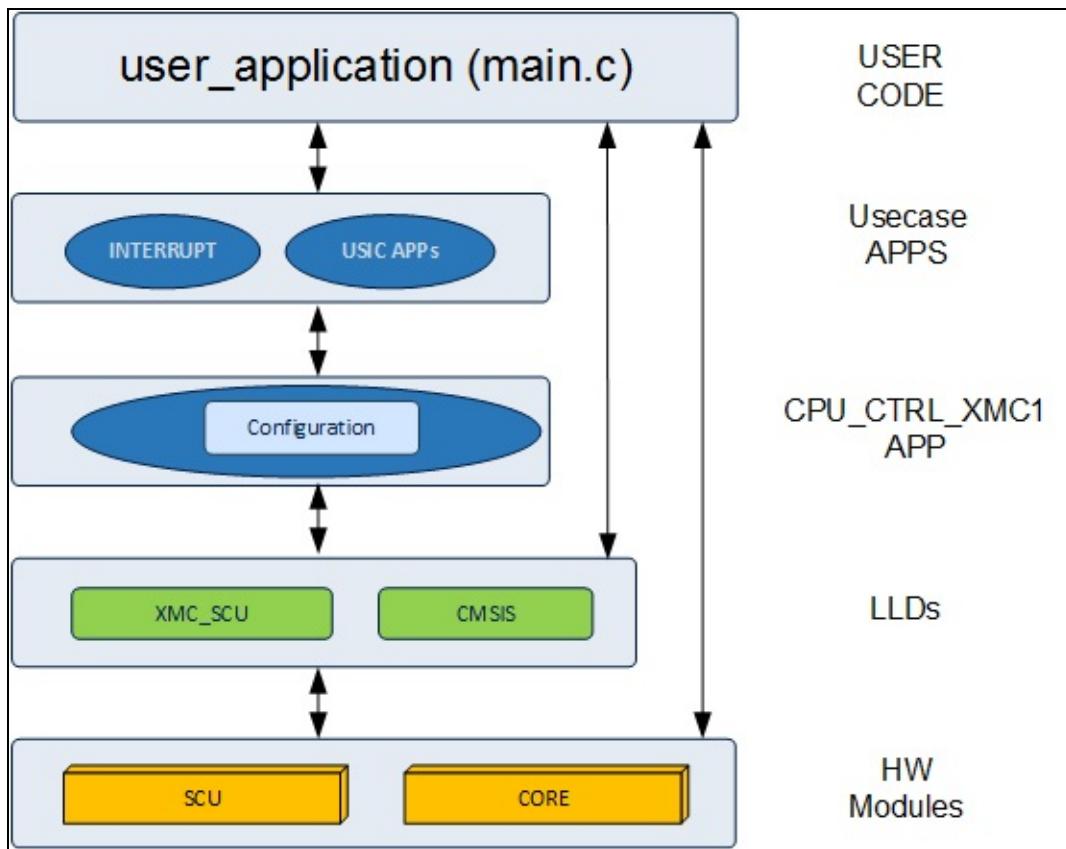
exception with configurable priority.

The APP allows generating code to make easier debugging the cause of hard fault.

### Boot pin modes:

- Only available in XMC1400
- Reserves the P4.6 and P4.7 pins

The figure below shows the APP structure in DAVE™. It uses the CMSIS library for configuring priority groups for interrupts used by the top level APPs.



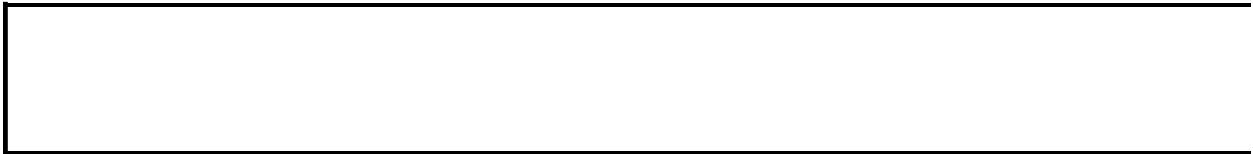
### Supported Devices

1. XMC1400 Series
2. XMC1300 Series

3. XMC1200 Series
4. XMC1100 Series

## **References**

1. XMC1400 Reference Manual
  2. XMC1300 Reference Manual
  3. XMC1200 Reference Manual
  4. XMC1100 Reference Manual
- 

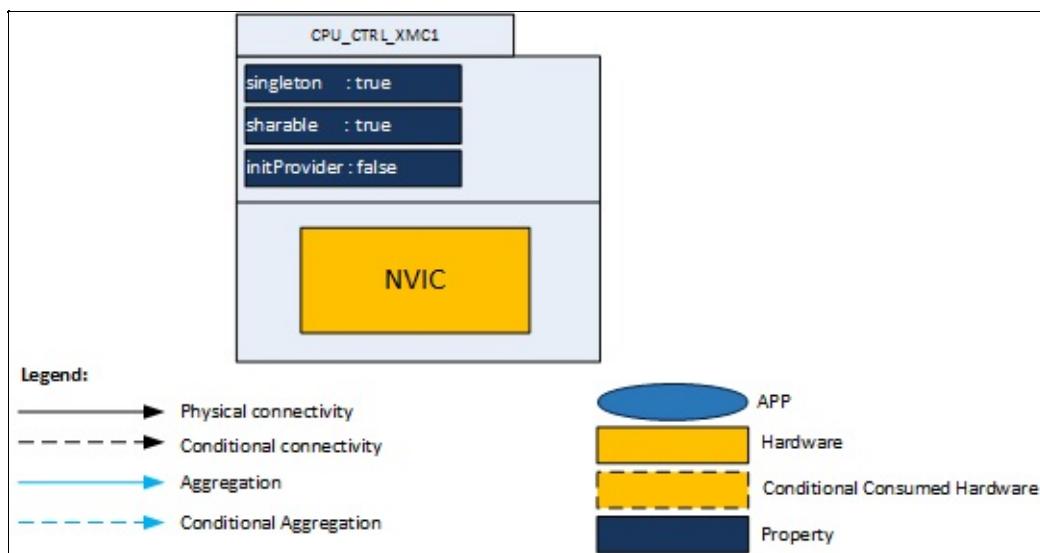


# CPU\_CTRL\_XMC1

Home

## Architecture Description

### Architecture Description



**Figure 1 :** Architecture of **CPU\_CTRL\_XMC1** APP

The above diagram represents the internal software architecture of the **CPU\_CTRL\_XMC1** APP. This APP cannot be used as a stand alone APP. Top level APPs like INTERRUPT and USIC consume this APP.

An instance of the **CPU\_CTRL\_XMC1** APP generates (after code generation) a specific data structure with the GUI configuration. The name of this data structure can be modified by changing the APP instance label (e.g. change label from default CPU\_CTRL\_XMC1\_0 to MY\_CPU).

# CPU\_CTRL\_XMC1

Home

## APP Configuration Parameters

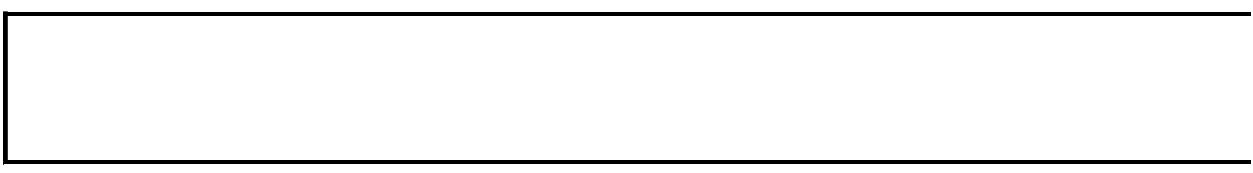
### App Configuration Parameters

General Settings	Exception Settings
Number of priority bits for priority grouping: <input type="text" value="2"/>	
Debug interface:	<input type="button" value="SWD0"/>
<input type="checkbox"/> Enable boot pin modes	

Figure 1: General Settings

General Settings	Exception Settings
<input type="checkbox"/> Enable hard fault debugging support	

Figure 2: Exception Settings



# CPU\_CTRL\_XMC1

Home

## Enumerations

```
CPU_CTRL_XMC1_STATUS {  
enum CPU_CTRL_XMC1_STATUS_SUCCESS = 0U,  
CPU_CTRL_XMC1_STATUS_FAILURE = 1U }
```

## Enumeration Type Documentation

### enum CPU\_CTRL\_XMC1\_STATUS

#### Enumerator:

*CPU\_CTRL\_XMC1\_STATUS\_SUCCESS* APP initialization is success

*CPU\_CTRL\_XMC1\_STATUS\_FAILURE* APP initialization is failure

Definition at line [77](#) of file [CPU\\_CTRL\\_XMC1.h](#).

# CPU\_CTRL\_XMC1

Home

Data Structures

## Data structures

## Data Structures

---

struct **CPU\_CTRL\_XMC1**

Configuration structure for  
**CPU\_CTRL\_XMC1** APP. More...

typedef struct **CPU\_CTRL\_XMC1** **CPU\_CTRL\_XMC1\_t**

Configuration structure for  
**CPU\_CTRL\_XMC1** APP.

---



# CPU\_CTRL\_XMC1

Home

## Methods

```
DAVE_APP_VERSION_t CPU_CTRL_XMC1_GetAppVersion (void)  
Get CPU_CTRL_XMC1 APP version.
```

## Methods

---

## Function Documentation

**DAVE\_APP\_VERSION\_t CPU\_CTRL\_XMC1\_GetAppVersion ( void )**

Get **CPU\_CTRL\_XMC1** 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_APP_VERSION_t version;
    init_status = DAVE_Init();

    version = CPU_CTRL_XMC1_GetAppVersion();
    if (version.major != 4U)
    {
    }
    while(1) {

    }
    return (0);
}
```

Definition at line **74** of file **CPU\_CTRL\_XMC1.c**.



# CPU\_CTRL\_XMC1

Home

## Usage

### Usage

The below example illustrates how to use the exceptions settings using XMC1400 board.

#### Required Hardware

XMC1400 board

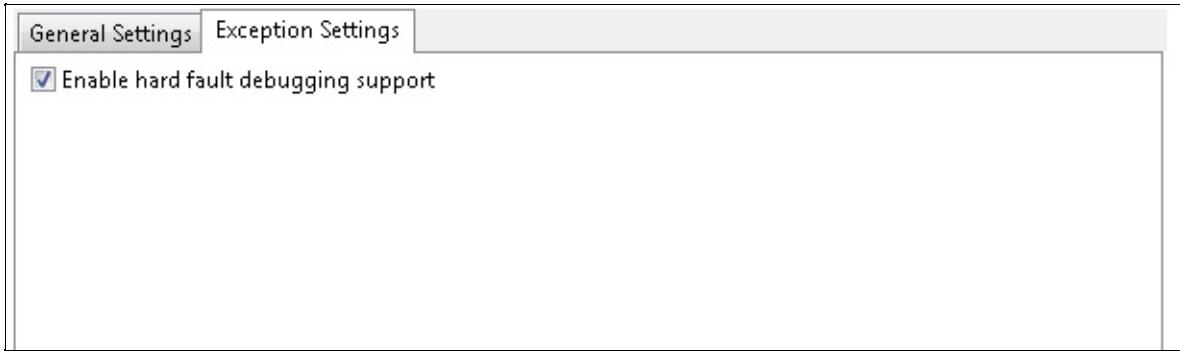
#### Instantiate the required APPs

Drag the **CPU\_CTRL\_XMC1** APP. Update the fields in the GUI of this APP with the following configuration.

Example: Access invalid address Hard fault exceptions have various reasons:

- Execution of unknown opcodes
- Trying to access invalid address
- Trying to read or write to an on-chip peripheral with disabled access using PAU.
- Stack corruption
- Trying to switch to ARM state

In this example, a HardFault exception is triggered due to an invalid address access. To analyze and find the offensive code, the **CPU\_CTRL\_XMC1** is used. The APP is configured to generate a hard fault handler with debugging capabilities. These debugging capabilities allow the user to decode the stack at the moment of the exception and in most cases find the root cause for the exception.

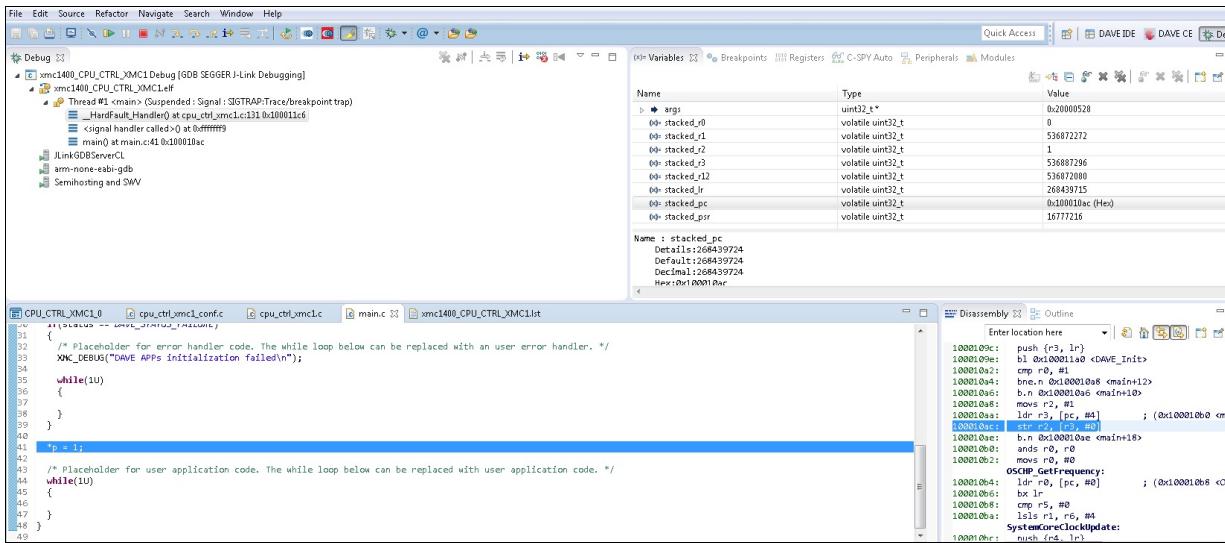


**Figure 1 : CPU\_CTRL\_XMC1 APP UI configuration**

The code that injects the fault looks like:

```
int32_t *p = (int32_t *)0x20004000U;
*p = 1;
```

The code is trying to access via a pointer over the RAM region. This actual code can be due to an overflow in a pointer. After executing the code, the breakpoint in the hardfault handler will stop the execution. The stacked PC can help us to identify the offensive code.



**Figure 2 : Invalid address access Hardfault debugging**

The stacked PC is pointing to the code where the invalid address was used. Example: Access peripheral disabled by PAU In this example we try to access a peripheral which previously has been disabled by the

application.

The following code will trigger a hardfault:

```
XMC_PAU_DisablePeripheralAccess(XMC_PAU_PERIPHERAL_USICO_CH0);  
  
USICO_CH0->KSCFG = 1;
```

The stacked PC points to the offensive code.

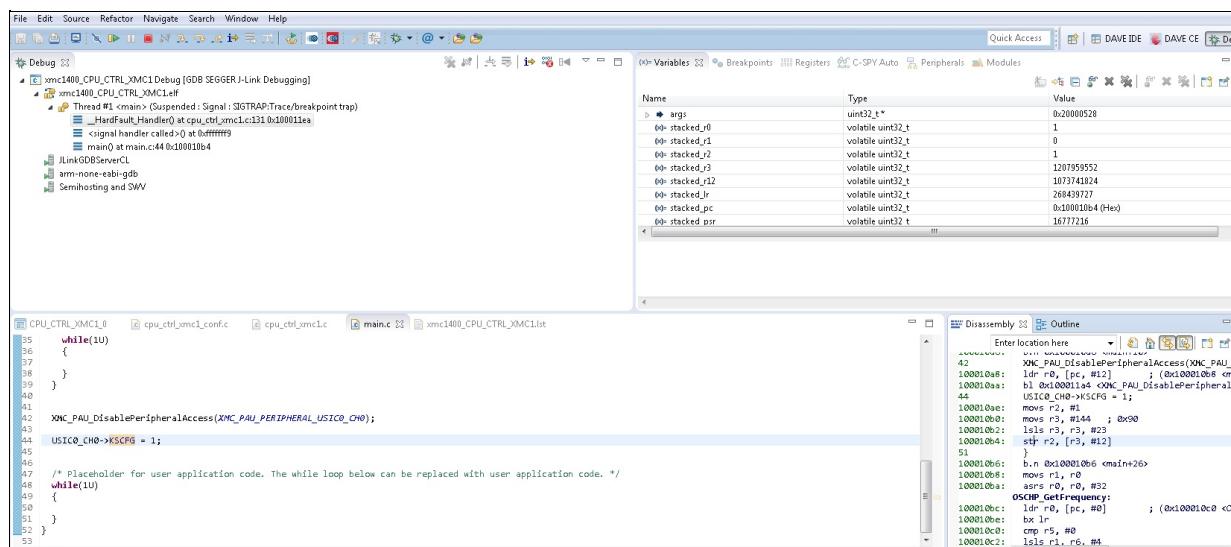


Figure 2 : Invalid peripheral access Hardfault debugging

# CPU\_CTRL\_XMC1

Home

## Release History

### Release History



# CPU\_CTRL\_XMC1

Home

Data Structures

**CPU\_CTRL\_XMC1**

## Data Structures

---

```
struct CPU_CTRL_XMC1
```

Configuration structure for **CPU\_CTRL\_XMC1** APP. More...

---



# CPU\_CTRL\_XMC1

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

## Data Structures

Here are the data structures with brief descriptions:

[CPU\\_CTRL\\_XMC1](#)

Configuration structure for [CPU\\_CTRL\\_XMC1](#)  
APP

---

---

# CPU\_CTRL\_XMC1

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

## CPU\_CTRL\_XMC1

### Struct Reference

[CPU\\_CTRL\\_XMC1 | Data structures](#)

## Detailed Description

Configuration structure for **CPU\_CTRL\_XMC1 APP.**

Definition at line **97** of file **CPU\_CTRL\_XMC1.h**.

```
#include <CPU_CTRL_XMC1.h>
```

## Data Fields

bool **initialized**

---

## Field Documentation

**bool CPU\_CTRL\_XMC1::initialized**

APP is initialized or not.

Definition at line **99** of file [CPU\\_CTRL\\_XMC1.h](#).

---

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

- [CPU\\_CTRL\\_XMC1.h](#)
- 



# CPU\_CTRL\_XMC1

Home

Data Structures

Data Structure Index

Data Fields

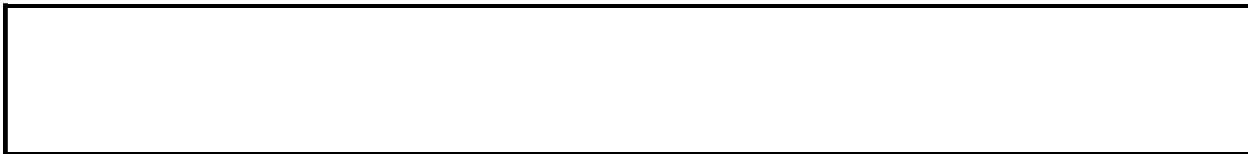
## Data Structure Index

C

C

[CPU\\_CTRL\\_XMC1](#)

C



# CPU\_CTRL\_XMC1

Home		
Data Structures	Data Structure Index	Data Fields
All	Variables	

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

- initialized : [CPU\\_CTRL\\_XMC1](#)
-

# CPU\_CTRL\_XMC1

Home		
Data Structures	Data Structure Index	Data Fields
All	Variables	

- initialized : CPU\_CTRL\_XMC1
-

# CPU\_CTRL\_XMC1

Home

File List

Globals

## File List

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

[CPU\\_CTRL\\_XMC1.c](#) [code] 

[CPU\\_CTRL\\_XMC1.h](#) [code] 

# CPU\_CTRL\_XMC1

[Home](#)

[File List](#)

[Globals](#)

[Functions](#)

## CPU\_CTRL\_XMC1.c File Reference

## Detailed Description

**Date:**

2015-10-14

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 [\*\*CPU\\_CTRL\\_XMC1.c\*\*](#).

```
#include "cpu_ctrl_xmc1.h"
```

## Functions

DAVE\_APP\_VERSION\_t **CPU\_CTRL\_XMC1\_GetAppVersion** (void)  
Get **CPU\_CTRL\_XMC1** APP version.

Go to the source code of this file.

---

# CPU\_CTRL\_XMC1

[Home](#)

[File List](#)

[Globals](#)

[Data Structures](#)

## CPU\_CTRL\_XMC1.h File Reference

## Detailed Description

**Date:**

2015-10-14

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 [\*\*CPU\\_CTRL\\_XMC1.h\*\*](#).

```
#include <xmc_common.h> #include <DAVE_Common.h>
#include "cpu_ctrl_xmc1_conf.h"
#include "cpu_ctrl_xmc1_extern.h"
```

## Data Structures

---

struct **CPU\_CTRL\_XMC1**

Configuration structure for **CPU\_CTRL\_XMC1** APP. More...

## TypeDefs

```
typedef struct CPU_CTRL_XMC1 CPU_CTRL_XMC1_t  
Configuration structure for  
CPU_CTRL_XMC1 APP.
```

## Functions

```
DAVE_APP_VERSION_t CPU_CTRL_XMC1_GetAppVersion (void)
Get CPU_CTRL_XMC1 APP version.

CPU_CTRL_XMC1_STATUS {
CPU_CTRL_XMC1_STATUS_SUCCESS =
enum 0U,
CPU_CTRL_XMC1_STATUS_FAILURE =
1U }
```

Go to the source code of this file.

---

---

# CPU\_CTRL\_XMC1

<a href="#">Home</a>					
<a href="#">File List</a>	<a href="#">Globals</a>				
<a href="#">All</a>	<a href="#">Functions</a>	<a href="#">Typedefs</a>	<a href="#">Enumerations</a>	<a href="#">Enumerator</a>	

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

- CPU\_CTRL\_XMC1\_GetAppVersion() : [CPU\\_CTRL\\_XMC1.c](#) , [CPU\\_CTRL\\_XMC1.h](#)
- CPU\_CTRL\_XMC1\_STATUS : [CPU\\_CTRL\\_XMC1.h](#)
- CPU\_CTRL\_XMC1\_STATUS\_FAILURE : [CPU\\_CTRL\\_XMC1.h](#)
- CPU\_CTRL\_XMC1\_STATUS\_SUCCESS : [CPU\\_CTRL\\_XMC1.h](#)
- CPU\_CTRL\_XMC1\_t : [CPU\\_CTRL\\_XMC1.h](#)

# CPU\_CTRL\_XMC1

<a href="#">Home</a>					
<a href="#">File List</a>	<a href="#">Globals</a>				
All	<a href="#">Functions</a>	<a href="#">Typedefs</a>	<a href="#">Enumerations</a>	<a href="#">Enumerator</a>	

- 
- CPU\_CTRL\_XMC1\_GetAppVersion() : [CPU\\_CTRL\\_XMC1.c](#) , [CPU\\_CTRL\\_XMC1.h](#)
-

# CPU\_CTRL\_XMC1

Home					
File List	Globals				
All	Functions	Typedefs	Enumerations	Enumerator	

- CPU\_CTRL\_XMC1\_t : CPU\_CTRL\_XMC1.h
- 



# CPU\_CTRL\_XMC1

Home					
File List	Globals				
All	Functions	Typedefs	Enumerations	Enumerator	

- CPU\_CTRL\_XMC1\_STATUS : [CPU\\_CTRL\\_XMC1.h](#)
-

# CPU\_CTRL\_XMC1

Home					
File List	Globals				
All	Functions	Typedefs	Enumerations	Enumerator	

- CPU\_CTRL\_XMC1\_STATUS\_FAILURE : [CPU\\_CTRL\\_XMC1.h](#)
  - CPU\_CTRL\_XMC1\_STATUS\_SUCCESS : [CPU\\_CTRL\\_XMC1.h](#)
-

# CPU\_CTRL\_XMC1

Home	
File List	Globals

## CPU\_CTRL\_XMC1.h

[Go to the documentation of this file.](#)

```
00001
00054 #ifndef CPU_CTRL_XMC1_H
00055 #define CPU_CTRL_XMC1_H
00056
00057 /*****
00058 * HEADER FILES
00059 ****/
00060 #include <xmc_common.h>
00061 #include <DAVE_Common.h>
00062 #include "cpu_ctrl_xmc1_conf.h"
00063 ****/
00064 * MACROS
00065 ****/
00066
00067 ****/
00068 * ENUMS
00069 ****/
```

```
*****
***** */
00074 /*
00075  * @brief enumeration for CPU_CTRL_XMC1 APP
00076 */
00077 typedef enum CPU_CTRL_XMC1_STATUS
00078 {
00079     CPU_CTRL_XMC1_STATUS_SUCCESS = 0U,
00080     CPU_CTRL_XMC1_STATUS_FAILURE = 1U
00081 } CPU_CTRL_XMC1_STATUS_t;
00082
00087 ****
***** */
00088 * DATA STRUCTURES
00089 ****
***** */
00097 typedef struct CPU_CTRL_XMC1
00098 {
00099     bool initialized;
00100 } CPU_CTRL_XMC1_t;
00101
00106 ****
***** */
00107 * API PROTOTYPES
00108 ****
***** */
00109
00110 #ifdef __cplusplus
00111 extern "C" {
00112 #endif
00113
00149 DAVE_APP_VERSION_t CPU_CTRL_XMC1_GetAppVersion(void);
```

```
00150
00151 CPU_CTRL_XMC1_STATUS_t CPU_CTRL_XMC1_Init(CP
U_CTRL_XMC1_t *const handler);
00156 #ifdef __cplusplus
00157 }
00158 #endif
00159
00160
00161 #include "cpu_ctrl_xmc1_extern.h"
00162
00163
00164 #endif /* CPU_CTRL_XMC1_H */
00165
```

---

# CPU\_CTRL\_XMC1

Home

File List

Globals

## CPU\_CTRL\_XMC1.c

Go to the documentation of this file.

```
00001
00050 /*****
***** * HEADER FILES *
00051 * HEADER FILES
00052 ****
***** * ****
***** */
00053 #include "cpu_ctrl_xmc1.h"
00054
00055 /*****
***** * ****
***** */
00056 * MACROS
00057 ****
***** */
00058
00059 /*****
***** * ****
***** */
00060 * LOCAL DATA
00061 ****
***** */
00062
00063 /*****
```

```
*****
*****
00064 * LOCAL ROUTINES
00065 ****
*****
***** */
00066
00067 /* ****
*****
***** */
00068 * API IMPLEMENTATION
00069 ****
*****
***** */
00070
00071 /*
00072 * API to retrieve the version of the CPU_CTRL_XMC1_APP
00073 */
00074 DAVE_APP_VERSION_t CPU_CTRL_XMC1_GetAppVersion(void)
00075 {
00076     DAVE_APP_VERSION_t version;
00077
00078     version.major = CPU_CTRL_XMC1_MAJOR_VERSION;
00079     version.minor = CPU_CTRL_XMC1_MINOR_VERSION;
00080     version.patch = CPU_CTRL_XMC1_PATCH_VERSION;
00081
00082     return (version);
00083 }
00084
00085 /* Dummy Init API to maintain backward compatibility */
00086 CPU_CTRL_XMC1_STATUS_t CPU_CTRL_XMC1_Init(CP
```

```
U_CTRL_XMC1_t *const handler)
00087 {
00088     return CPU_CTRL_XMC1_STATUS_SUCCESS;
00089 }
00090
00091 #if (HARDFAULT_ENABLED == 1)
00092
00101 #if defined(__GNUC__)
00102 #pragma GCC diagnostic ignored "-Wunused-but-
-set-variable"
00103 #endif
00104 void __HardFault_Handler(uint32_t args[])
00105 {
00106     volatile uint32_t stacked_r0 ;
00107     volatile uint32_t stacked_r1 ;
00108     volatile uint32_t stacked_r2 ;
00109     volatile uint32_t stacked_r3 ;
00110     volatile uint32_t stacked_r12 ;
00111     volatile uint32_t stacked_lr ;
00112     volatile uint32_t stacked_pc ;
00113     volatile uint32_t stacked_psr ;
00114
00115     stacked_r0 = ((uint32_t)args[0]) ;
00116     stacked_r1 = ((uint32_t)args[1]) ;
00117     stacked_r2 = ((uint32_t)args[2]) ;
00118     stacked_r3 = ((uint32_t)args[3]) ;
00119     stacked_r12 = ((uint32_t)args[4]) ;
00120     stacked_lr = ((uint32_t)args[5]) ;
00121     stacked_pc = ((uint32_t)args[6]) ;
00122     stacked_psr = ((uint32_t)args[7]) ;
00123
00124     __asm("BKPT 0\n") ; // Break into the debu
gger
00125 }
00126
00127 /*KEIL*/
00128 #if defined(__CC_ARM)
```

```
00129 __asm void HardFault_Handler(void)
00130 {
00131     EXTERN __HardFault_Handler [CODE]
00132
00133     movs r0,#4          /* load bit mask into R0
   */
00134     mov r1, lr           /* load link register in
to R1 */
00135     tst r0, r1           /* compare with bitmask
   */
00136     beq _MSP             /* if bitmask is set: st
ack pointer is in PSP. Otherwise in MSP */
00137     mrs r0, psp          /* otherwise: stack poin
ter is in PSP */
00138     b __HardFault_Handler      /* go to pa
rt which loads the PC */
00139 _MSP                  /* stack pointer is in M
SP register */
00140     mrs r0, msp          /* load stack pointer in
to R0 */
00141     b __HardFault_Handler      /* go to pa
rt which loads the PC */
00142 }
00143
00144 #endif
00145
00146 /*IAR*/
00147 #if defined(__ICCARM__)
00148 void HardFault_Handler(void)
00149 {
00150     asm(
00151     "movs r0,#4 \n"        /* load bit mask int
o R0 */
00152     "mov r1, lr \n"        /* load link registe
r into R1 */
00153     "tst r0, r1 \n"        /* compare with bitm
ask */
```

```
00154 "beq _MSP \n"          /* if bitmask is set:  
    stack pointer is in PSP. Otherwise in MSP */  
00155 "mrs r0, psp \n"      /* otherwise: stack  
pointer is in PSP */  
00156 "b __HardFault_Handler \n"        /* go t  
o part which loads the PC */  
00157 "_MSP: \n"            /* stack pointer is  
in MSP register */  
00158 "mrs r0, msp \n"      /* load stack pointe  
r into R0 */  
00159 "b __HardFault_Handler \n" /* decode more  
information. R0 contains pointer to stack frame */  
  
00160 );  
00161 }  
00162 #endif  
00163  
00164 /*TASKING*/  
00165 #if defined(__TASKING__)  
00166 void HardFault_Handler(void)  
00167 {  
00168     __asm volatile (  
00169 " movs r0,#4      \n" /* load bit mask into  
    R0 */  
00170 " mov r1, lr      \n" /* load link register  
    into R1 */  
00171 " tst r0, r1      \n" /* compare with bitma  
sk */  
00172 " beq _MSP      \n" /* if bitmask is set:  
    stack pointer is in PSP. Otherwise in MSP */  
00173 " mrs r0, psp      \n" /* otherwise: stack p  
ointer is in PSP */  
00174 " b __HardFault_Handler      \n" /* go to  
part which loads the PC */  
00175 "_MSP: \n"            /* stack pointer is  
in MSP register */  
00176 " mrs r0, msp      \n" /* load stack pointer
```

```
    into R0 */
00177 " b __HardFault_Handler \n" /* decode more
e information. R0 contains pointer to stack frame
*/
00178 );
00179 }
00180 #endif
00181
00182 /*GCC*/
00183 #if defined(__GNUC__)
00184 __attribute__((naked))
00185 void HardFault_Handler(void)
00186 {
00187     __asm volatile (
00188 " movs r0,#4      \n" /* load bit mask into
R0 */
00189 " mov r1, lr      \n" /* load link register
into R1 */
00190 " tst r0, r1      \n" /* compare with bitmask */
00191 " beq _MSP        \n" /* if bitmask is set:
stack pointer is in PSP. Otherwise in MSP */
00192 " mrs r0, psp     \n" /* otherwise: stack pointer
is in PSP */
00193 " b __HardFault_Handler          \n" /* go to
part which loads the PC */
00194 "_MSP:             \n" /* stack pointer is
in MSP register */
00195 " mrs r0, msp     \n" /* load stack pointer
into R0 */
00196 " b __HardFault_Handler \n" /* decode more
information. R0 contains pointer to stack frame
*/
00197 );
00198 }
00199 #endif
00200 #endif
```



