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
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>SCU</td>
<td>System Control Unit</td>
</tr>
</tbody>
</table>

**Definitions:**

<table>
<thead>
<tr>
<th>Definition</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) APPs</td>
</tr>
</tbody>
</table>
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
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

APP Configuration Parameters

App Configuration Parameters

<table>
<thead>
<tr>
<th>General Settings</th>
<th>Exception Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of priority bits for priority grouping:</td>
<td>2</td>
</tr>
<tr>
<td>Debug interface:</td>
<td>SWD0</td>
</tr>
<tr>
<td>Enable boot pin modes</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: General Settings

Figure 2: Exception Settings
CPU_CTRL_XMC1

Enumerations

```c
enum CPU_CTRL_XMC1_STATUS {
    CPU_CTRL_XMC1_STATUS_SUCCESS = 0U,
    CPU_CTRL_XMC1_STATUS_FAILURE = 1U
};
```
Enumeration Type Documentation

defined 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.
<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data structures</td>
</tr>
</tbody>
</table>

CPU_CTRL_XMC1
# Data Structures

<table>
<thead>
<tr>
<th>struct</th>
<th><code>CPU_CTRL_XMC1</code></th>
<th>Configuration structure for <code>CPU_CTRL_XMC1</code> APP. More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct</td>
<td><code>CPU_CTRL_XMC1</code></td>
<td><code>CPU_CTRL_XMC1_t</code></td>
</tr>
</tbody>
</table>
### CPU_CTRL_XMC1

#### Methods

<table>
<thead>
<tr>
<th>DAVE_APP_VERSION_t</th>
<th>CPU_CTRL_XMC1_GetAppVersion (void)</th>
</tr>
</thead>
</table>

Get **CPU_CTRL_XMC1** APP version.
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:

```c
#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

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 of the case find the root cause for the exception.
The code that injects the fault looks like:

```c
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 1**: CPU_CTRL_XMC1 APP UI configuration

**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:

```c
XMC_PAU_DisablePeripheralAccess(XMC_PAU_PERIPHERAL_USIC0_CH0);
USIC0_CH0->KSCFG = 1;
```

The stacked PC points to the offensive code.

---

**Figure 2**: Invalid peripheral access Hardfault debugging
## CPU_CTRL_XMC1

### Home

### Release History

<table>
<thead>
<tr>
<th>Release History</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

### Release History

<table>
<thead>
<tr>
<th>Release History</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

### Release History

<table>
<thead>
<tr>
<th>Release History</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Home</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>CPU_CTRL_XMC1</td>
</tr>
<tr>
<td>CPU_CTRL_XMC1</td>
</tr>
</tbody>
</table>
# Data Structures

```c
struct CPU_CTRL_XMC1
    Configuration structure for CPU_CTRL_XMC1 APP. More...
```
Data Structures

Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>CPU_CTRL_XMC1</th>
<th>Configuration structure for CPU_CTRL_XMC1 APP</th>
</tr>
</thead>
</table>
# CPU_CTRL_XMC1

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

## 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

<table>
<thead>
<tr>
<th>bool</th>
<th>initialized</th>
</tr>
</thead>
</table>

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

## Data Structure Index

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CTRL_XMC1</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

---
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

<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Variables</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- initialized: `CPU_CTRL_XMC1`
Here is a list of all documented files with brief descriptions:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CTRL_XMC1.c</td>
<td></td>
</tr>
<tr>
<td>CPU_CTRL_XMC1.h</td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>File List</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t CPU_CTRL_XMC1_GetAppVersion (void)</td>
<td>Get CPU_CTRL_XMC1 APP version.</td>
</tr>
</tbody>
</table>

Go to the source code of this file.
# CPU_CTRL_XMC1

<table>
<thead>
<tr>
<th>Home</th>
<th>File List</th>
<th>Globals</th>
<th>Data Structures</th>
</tr>
</thead>
</table>

**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_xmc1Extern.h"
Data Structures

```c
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.
Functions

<table>
<thead>
<tr>
<th>DAVE_APP_VERSION_t</th>
<th>CPU_CTRL_XMC1_GetAppVersion (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get CPU_CTRL_XMC1 APP version.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CTRL_XMC1_STATUS {</td>
</tr>
<tr>
<td>CPU_CTRL_XMC1_STATUS_SUCCESS = 0U,</td>
</tr>
<tr>
<td>CPU_CTRL_XMC1_STATUS_FAILURE = 1U }</td>
</tr>
</tbody>
</table>

Go to the source code of this file.
CPU_CTRL_XMC1

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

- **CPU_CTRL_XMC1_GetAppVersion()**: `CPU_CTRL_XMC1.c`, `CPU_CTRL_XMC1.h`
CPU_CTRL_XMC1

- **CPU_CTRL_XMC1_t**: CPU_CTRL_XMC1.h
### CPU_CTRL_XMC1

- **CPU_CTRL_XMC1_STATUS**: `CPU_CTRL_XMC1.h`
CPU_CTRL_XMC1

- CPU_CTRL_XMC1_STATUS_FAILURE : CPU_CTRL_XMC1.h
- CPU_CTRL_XMC1_STATUS_SUCCESS : CPU_CTRL_XMC1.h
```c
#ifndef CPU_CTRL_XMC1_H
#define CPU_CTRL_XMC1_H

/*******************************************
**************************************************
**************************
* HEADER FILES
**************************************************
*******************************************

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

/*******************************************
**************************************************
**************************
* MACROS
**************************************************
*******************************************

/*******************************************
**************************************************
**************************
* Enums
**************************************************
*******************************************
```
typedef enum CPU_CTRL_XMC1_STATUS
{
    CPU_CTRL_XMC1_STATUS_SUCCESS = 0U,
    CPU_CTRL_XMC1_STATUS_FAILURE = 1U
} CPU_CTRL_XMC1_STATUS_t;

typedef struct CPU_CTRL_XMC1
{
    bool initialized;
} CPU_CTRL_XMC1_t;

#ifdef __cplusplus
extern "C" {
#endif

DAVE_APP_VERSION_t CPU_CTRL_XMC1_GetAppVersion(void);

#ifdef __cplusplus
}
#endif
CPU_CTRL_XMC1_STATUS_t CPU_CTRL_XMC1_Init(CPU_CTRL_XMC1_t *const handler);
#endif __cplusplus
}
#endif /* CPU_CTRL_XMC1_H */
#include "cpu_ctrl_xmc1Extern.h"
Go to the documentation of this file.

```c
00001
00050 /********************************************************************************
00002 */
00003 */*******************************************************************************
00004 */*******************************************************************************
00005 */
00006 */*******************************************************************************
00007 /* HEADER FILES */
00008 */*******************************************************************************
00009 */*******************************************************************************
00010 */
00011 */*******************************************************************************
00012 */
00013 #include "cpu_ctrl_xmc1.h"
00014 */
00015 */*******************************************************************************
00016 */*******************************************************************************
00017 */
00018 /* MACROS */
00019 */*******************************************************************************
00020 */*******************************************************************************
00021 */
00022 */*******************************************************************************
00023 */
00024 /* LOCAL DATA */
00025 */*******************************************************************************
00026 */*******************************************************************************
00027 */
00028 */*******************************************************************************
00029 */
00030 */*******************************************************************************
00031 */
00032 */*******************************************************************************
```
* API IMPLEMENTATION

/* API to retrieve the version of the CPU_CTRL_XMC1 APP */

DAVE_APP_VERSION_t CPU_CTRL_XMC1_GetAppVersion(void)
{
    DAVE_APP_VERSION_t version;
    version.major = CPU_CTRL_XMC1_MAJOR_VERSION;
    version.minor = CPU_CTRL_XMC1_MINOR_VERSION;
    version.patch = CPU_CTRL_XMC1_PATCH_VERSION;
    return (version);
}

/* Dummy Init API to maintain backward compatibility */
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 debugger
00125 }
00126
00127 /*KEIL*/
00128 #if defined(__CC_ARM)
__asm

void HardFault_Handler(void)
{
    EXTERN __HardFault_Handler [CODE]

    movs r0,#4
    /* load bit mask into R0 */
    mov r1, lr
    /* load link register into R1 */
    tst r0, r1
    /* compare with bitmask */
    beq _MSP
    /* if bitmask is set: stack pointer is in PSP. Otherwise in MSP */
    mrs r0, psp
    /* otherwise: stack pointer is in PSP */
    b __HardFault_Handler
    /* go to part which loads the PC */

    _MSP
    /* stack pointer is in MSP register */
    mrs r0, msp
    /* load stack pointer into R0 */
    b __HardFault_Handler
    /* go to part which loads the PC */
}

#endif

/*IAR*/
#if defined(__ICCARM__)
void HardFault_Handler(void)
{
    asm(
        "movs r0,#4 \n"
        /* load bit mask into R0 */
        "mov r1, lr \n"
        /* load link register into R1 */
        "tst r0, r1 \n"
        /* compare with bitmask */
    )
}
/*
  if bitmask is set: stack pointer is in PSP. Otherwise in MSP */
mrs r0, psp \n"  /* otherwise: stack pointer is in PSP */
b __HardFault_Handler \n"  /* go to part which loads the PC */

"_MSP: \n"  /* stack pointer is in MSP register */
mrs r0, msp \n"  /* load stack pointer into R0 */
b __HardFault_Handler \n"  /* decode more information. R0 contains pointer to stack frame */

});

#endif

/*TASKING*/
#if defined(__TASKING__)
void HardFault_Handler(void)
{
  __asm volatile ("
movs r0,#4 \n"  /* load bit mask into R0 */
mov r1, lr \n"  /* load link register into R1 */
tst r0, r1 \n"  /* compare with bitmask */
beq _MSP \n"  /* if bitmask is set: stack pointer is in PSP. Otherwise in MSP */
mrs r0, psp \n"  /* otherwise: stack pointer is in PSP */
b __HardFault_Handler \n"  /* go to part which loads the PC */
"_MSP: \n"  /* stack pointer is in MSP register */
mrs r0, msp \n"  /* load stack pointer
into R0 */
00177 " b __HardFault_Handler \n" /* decode more 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