/\*\* @file mibspi.c

\* @brief MIBSPI Driver Implementation File

\* @date 07-July-2017

\* @version 04.07.00

\*

\*/

/\*

\* Copyright (C) 2009-2016 Texas Instruments Incorporated - www.ti.com

\*

\*

\* 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 Texas Instruments Incorporated 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

\* OWNER 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.

\*

\*/

/\* USER CODE BEGIN (0) \*/

/\* USER CODE END \*/

#include "mibspi.h"

//#include "sys\_vim.h"

/\* USER CODE BEGIN (1) \*/

#include "VLV09\_ValveOutput2SPI.h"

#include "SYSIntlocks\_COMMON.h"

/\* USER CODE END \*/

/\*\* @fn void mibspiInit(void)

\* @brief Initializes the MIBSPI Driver

\*

\* This function initializes the MIBSPI module.

\*/

/\* SourceId : MIBSPI\_SourceId\_001 \*/

/\* DesignId : MIBSPI\_DesignId\_001 \*/

/\* Requirements : HL\_SR153 \*/

void mibspiInit(void)

{

uint32 i ;

/\* USER CODE BEGIN (2) \*/

/\* USER CODE END \*/

 /\*\* @b initialize @b MIBSPI1 \*/

 /\*\* bring MIBSPI out of reset \*/

 mibspiREG1->GCR0 = 0U;

 mibspiREG1->GCR0 = 1U;

 /\*\* enable MIBSPI1 multibuffered mode and enable buffer RAM \*/

 mibspiREG1->MIBSPIE = (mibspiREG1->MIBSPIE & 0xFFFFFFFEU) | 1U;

 /\*\* MIBSPI1 master mode and clock configuration \*/

 mibspiREG1->GCR1 = (mibspiREG1->GCR1 & 0xFFFFFFFCU) | ((uint32)((uint32)1U << 1U) /\* CLOKMOD \*/

 | 1U); /\* MASTER \*/

 /\*\* MIBSPI1 enable pin configuration \*/

 mibspiREG1->INT0 = (mibspiREG1->INT0 & 0xFEFFFFFFU) | (uint32)((uint32)0U << 24U); /\* ENABLE HIGHZ \*/

 /\*\* - Delays \*/

 mibspiREG1->DELAY = (uint32)((uint32)4U << 24U) /\* C2TDELAY \*/

 | (uint32)((uint32)5U << 16U) /\* T2CDELAY \*/

 | (uint32)((uint32)0U << 8U) /\* T2EDELAY \*/

 | (uint32)((uint32)0U << 0U); /\* C2EDELAY \*/

 /\*\* - Data Format 0 \*/

 mibspiREG1->FMT0 = (uint32)((uint32)200U << 24U) /\* wdelay \*/

 | (uint32)((uint32)0U << 23U) /\* parity Polarity \*/

 | (uint32)((uint32)0U << 22U) /\* parity enable \*/

 | (uint32)((uint32)0U << 21U) /\* wait on enable \*/

 | (uint32)((uint32)0U << 20U) /\* shift direction \*/

 | (uint32)((uint32)0U << 17U) /\* clock polarity \*/

 | (uint32)((uint32)1U << 16U) /\* clock phase \*/

 | (uint32)((uint32)7U << 15U) /\* baudrate prescale \*/

 | (uint32)((uint32)16U << 0U); /\* data word length \*/

 /\*\* - Data Format 1 \*/

 mibspiREG1->FMT1 = (uint32)((uint32)0U << 24U) /\* wdelay \*/

 | (uint32)((uint32)0U << 23U) /\* parity Polarity \*/

 | (uint32)((uint32)0U << 22U) /\* parity enable \*/

 | (uint32)((uint32)0U << 21U) /\* wait on enable \*/

 | (uint32)((uint32)0U << 20U) /\* shift direction \*/

 | (uint32)((uint32)0U << 17U) /\* clock polarity \*/

 | (uint32)((uint32)0U << 16U) /\* clock phase \*/

 | (uint32)((uint32)79U << 8U) /\* baudrate prescale \*/

 | (uint32)((uint32)16U << 0U); /\* data word length \*/

 /\*\* - Data Format 2 \*/

 mibspiREG1->FMT2 = (uint32)((uint32)0U << 24U) /\* wdelay \*/

 | (uint32)((uint32)0U << 23U) /\* parity Polarity \*/

 | (uint32)((uint32)0U << 22U) /\* parity enable \*/

 | (uint32)((uint32)0U << 21U) /\* wait on enable \*/

 | (uint32)((uint32)0U << 20U) /\* shift direction \*/

 | (uint32)((uint32)0U << 17U) /\* clock polarity \*/

 | (uint32)((uint32)0U << 16U) /\* clock phase \*/

 | (uint32)((uint32)79U << 8U) /\* baudrate prescale \*/

 | (uint32)((uint32)16U << 0U); /\* data word length \*/

 /\*\* - Data Format 3 \*/

 mibspiREG1->FMT3 = (uint32)((uint32)0U << 24U) /\* wdelay \*/

 | (uint32)((uint32)0U << 23U) /\* parity Polarity \*/

 | (uint32)((uint32)0U << 22U) /\* parity enable \*/

 | (uint32)((uint32)0U << 21U) /\* wait on enable \*/

 | (uint32)((uint32)0U << 20U) /\* shift direction \*/

 | (uint32)((uint32)0U << 17U) /\* clock polarity \*/

 | (uint32)((uint32)0U << 16U) /\* clock phase \*/

 | (uint32)((uint32)79U << 8U) /\* baudrate prescale \*/

 | (uint32)((uint32)16U << 0U); /\* data word length \*/

 /\*\* - Default Chip Select \*/

 mibspiREG1->DEF = (uint32)(0xFFU);

 /\*\* - wait for buffer initialization complete before accessing MibSPI registers \*/

 /\*SAFETYMCUSW 28 D MR:NA <APPROVED> "Hardware status bit read check" \*/

 while ((mibspiREG1->FLG & 0x01000000U) != 0U)

 {

 } /\* Wait \*/

 /\*\* enable MIBSPI RAM Parity \*/

 mibspiREG1->UERRCTRL = (mibspiREG1->UERRCTRL & 0xFFFFFFF0U) | (0x00000005U);

 /\*\* - initialize transfer groups \*/

 mibspiREG1->TGCTRL[0U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)0U << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[1U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)8U << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[2U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)(8U+8U) << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[3U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)(8U+8U+4U) << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[4U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)(8U+8U+4U+4U) << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[5U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)(8U+8U+4U+4U+0U) << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[6U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)(8U+8U+4U+4U+0U+0U) << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[7U] = (uint32)((uint32)1U << 30U) /\* oneshot \*/

 | (uint32)((uint32)0U << 29U) /\* pcurrent reset \*/

 | (uint32)((uint32)TRG\_ALWAYS << 20U) /\* trigger event \*/

 | (uint32)((uint32)TRG\_DISABLED << 16U) /\* trigger source \*/

 | (uint32)((uint32)(8U+8U+4U+4U+0U+0U+0U) << 8U); /\* start buffer \*/

 mibspiREG1->TGCTRL[8U] = (uint32)(8U+8U+4U+4U+0U+0U+0U+0U) << 8U;

 mibspiREG1->LTGPEND = (mibspiREG1->LTGPEND & 0xFFFF00FFU) | (uint32)((uint32)((8U+8U+4U+4U+0U+0U+0U+0U)-1U) << 8U);

 /\*\* - initialize buffer ram \*/

 {

 i = 0U;

 while (i < (8U))

 {

 if(0==i%2)

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)1U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 else

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

 while (i < (8U+8U))

 {

 if(0==i%2)

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)1U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 else

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

 while (i < (8U+8U+4U))

 {

 if (0==i%2)

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)1U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 else

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

 while (i < (8U+8U+4U+4U))

 {

 if(0==i%2)

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)1U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 else

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)1U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_0)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#if (0U > 0U)

 {

#if (0U > 1U)

 while (i < ((8U+8U+4U+4U+0U)-1U))

 {

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_4)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_4)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

#if (0U > 0U)

 {

#if (0U > 1U)

 while (i < ((8U+8U+4U+4U+0U+0U)-1U))

 {

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_5)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_5)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

#if (0U > 0U)

 {

#if (0U > 1U)

 while (i < ((8U+8U+4U+4U+0U+0U+0U)-1U))

 {

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_6)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_6)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

#if (0U > 0U)

 {

#if (0U > 1U)

 while (i < ((8U+8U+4U+4U+0U+0U+0U+0U)-1U))

 {

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 11U) /\* lock transmission \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_7)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

 mibspiRAM1->tx[i].control = (uint16)((uint16)4U << 13U) /\* buffer mode \*/

 | (uint16)((uint16)0U << 12U) /\* chip select hold \*/

 | (uint16)((uint16)0U << 10U) /\* enable WDELAY \*/

 | (uint16)((uint16)0U << 8U) /\* data format \*/

 /\*SAFETYMCUSW 334 S MR:10.5 <APPROVED> "LDRA Tool issue" \*/

 | ((uint16)(~((uint16)0xFFU ^ (uint16)CS\_7)) & (uint16)0x00FFU); /\* chip select \*/

 i++;

 }

#endif

 }

 /\*\* - set interrupt levels \*/

 mibspiREG1->LVL = (uint32)((uint32)0U << 9U) /\* TXINT \*/

 | (uint32)((uint32)0U << 8U) /\* RXINT \*/

 | (uint32)((uint32)0U << 6U) /\* OVRNINT \*/

 | (uint32)((uint32)0U << 4U) /\* BITERR \*/

 | (uint32)((uint32)0U << 3U) /\* DESYNC \*/

 | (uint32)((uint32)0U << 2U) /\* PARERR \*/

 | (uint32)((uint32)0U << 1U) /\* TIMEOUT \*/

 | (uint32)((uint32)0U << 0U); /\* DLENERR \*/

 /\*\* - clear any pending interrupts \*/

 mibspiREG1->FLG |= 0xFFFFU;

 /\*\* - enable interrupts \*/

 mibspiREG1->INT0 = (mibspiREG1->INT0 & 0xFFFF0000U)

 | (uint32)((uint32)0U << 9U) /\* TXINT \*/

 | (uint32)((uint32)0U << 8U) /\* RXINT \*/

 | (uint32)((uint32)0U << 6U) /\* OVRNINT \*/

 | (uint32)((uint32)0U << 4U) /\* BITERR \*/

 | (uint32)((uint32)0U << 3U) /\* DESYNC \*/

 | (uint32)((uint32)0U << 2U) /\* PARERR \*/

 | (uint32)((uint32)0U << 1U) /\* TIMEOUT \*/

 | (uint32)((uint32)0U << 0U); /\* DLENERR \*/

 /\*\* @b initialize @b MIBSPI1 @b Port \*/

 /\*\* - MIBSPI1 Port output values \*/

 mibspiREG1->PC3 = (uint32)((uint32)0U << 0U) /\* SCS[0] \*/

 | (uint32)((uint32)1U << 1U) /\* SCS[1] \*/

 | (uint32)((uint32)1U << 2U) /\* SCS[2] \*/

 | (uint32)((uint32)1U << 3U) /\* SCS[3] \*/

 | (uint32)((uint32)1U << 4U) /\* SCS[4] \*/

 | (uint32)((uint32)1U << 5U) /\* SCS[5] \*/

 | (uint32)((uint32)0U << 8U) /\* ENA \*/

 | (uint32)((uint32)0U << 9U) /\* CLK \*/

 | (uint32)((uint32)0U << 10U) /\* SIMO[0] \*/

 | (uint32)((uint32)0U << 11U) /\* SOMI[0] \*/

 | (uint32)((uint32)0U << 17U) /\* SIMO[1] \*/

 | (uint32)((uint32)0U << 25U); /\* SOMI[1] \*/

 /\*\* - MIBSPI1 Port direction \*/

 mibspiREG1->PC1 = (uint32)((uint32)1U << 0U) /\* SCS[0] \*/

 | (uint32)((uint32)1U << 1U) /\* SCS[1] \*/

 | (uint32)((uint32)1U << 2U) /\* SCS[2] \*/

 | (uint32)((uint32)1U << 3U) /\* SCS[3] \*/

 | (uint32)((uint32)1U << 4U) /\* SCS[4] \*/

 | (uint32)((uint32)1U << 5U) /\* SCS[5] \*/

 | (uint32)((uint32)0U << 8U) /\* ENA \*/

 | (uint32)((uint32)1U << 9U) /\* CLK \*/

 | (uint32)((uint32)1U << 10U) /\* SIMO[0] \*/

 | (uint32)((uint32)0U << 11U) /\* SOMI[0] \*/

 | (uint32)((uint32)0U << 17U) /\* SIMO[1] \*/

 | (uint32)((uint32)0U << 25U); /\* SOMI[1] \*/

 /\*\* - MIBSPI1 Port open drain enable \*/

 mibspiREG1->PC6 = (uint32)((uint32)0U << 0U) /\* SCS[0] \*/

 | (uint32)((uint32)0U << 1U) /\* SCS[1] \*/

 | (uint32)((uint32)0U << 2U) /\* SCS[2] \*/

 | (uint32)((uint32)0U << 3U) /\* SCS[3] \*/

 | (uint32)((uint32)0U << 4U) /\* SCS[4] \*/

 | (uint32)((uint32)0U << 5U) /\* SCS[5] \*/

 | (uint32)((uint32)0U << 8U) /\* ENA \*/

 | (uint32)((uint32)0U << 9U) /\* CLK \*/

 | (uint32)((uint32)0U << 10U) /\* SIMO[0] \*/

 | (uint32)((uint32)0U << 11U) /\* SOMI[0] \*/

 | (uint32)((uint32)0U << 17U) /\* SIMO[1] \*/

 | (uint32)((uint32)0U << 25U); /\* SOMI[1] \*/

 /\*\* - MIBSPI1 Port pullup / pulldown selection \*/

 mibspiREG1->PC8 = (uint32)((uint32)1U << 0U) /\* SCS[0] \*/

 | (uint32)((uint32)1U << 1U) /\* SCS[1] \*/

 | (uint32)((uint32)1U << 2U) /\* SCS[2] \*/

 | (uint32)((uint32)1U << 3U) /\* SCS[3] \*/

 | (uint32)((uint32)1U << 4U) /\* SCS[4] \*/

 | (uint32)((uint32)1U << 5U) /\* SCS[5] \*/

 | (uint32)((uint32)1U << 8U) /\* ENA \*/

 | (uint32)((uint32)1U << 9U) /\* CLK \*/

 | (uint32)((uint32)1U << 10U) /\* SIMO[0] \*/

 | (uint32)((uint32)1U << 11U) /\* SOMI[0] \*/

 | (uint32)((uint32)1U << 17U) /\* SIMO[1] \*/

 | (uint32)((uint32)1U << 25U); /\* SOMI[1] \*/

 /\*\* - MIBSPI1 Port pullup / pulldown enable\*/

 mibspiREG1->PC7 = (uint32)((uint32)0U << 0U) /\* SCS[0] \*/

 | (uint32)((uint32)0U << 1U) /\* SCS[1] \*/

 | (uint32)((uint32)0U << 2U) /\* SCS[2] \*/

 | (uint32)((uint32)0U << 3U) /\* SCS[3] \*/

 | (uint32)((uint32)0U << 4U) /\* SCS[4] \*/

 | (uint32)((uint32)0U << 5U) /\* SCS[5] \*/

 | (uint32)((uint32)0U << 8U) /\* ENA \*/

 | (uint32)((uint32)0U << 9U) /\* CLK \*/

 | (uint32)((uint32)0U << 10U) /\* SIMO[0] \*/

 | (uint32)((uint32)0U << 11U) /\* SOMI[0] \*/

 | (uint32)((uint32)0U << 17U) /\* SIMO[1] \*/

 | (uint32)((uint32)0U << 25U); /\* SOMI[1] \*/

 /\* MIBSPI1 set all pins to functional \*/

 mibspiREG1->PC0 = (uint32)((uint32)1U << 0U) /\* SCS[0] \*/

 | (uint32)((uint32)0U << 1U) /\* SCS[1] \*/

 | (uint32)((uint32)0U << 2U) /\* SCS[2] \*/

 | (uint32)((uint32)0U << 3U) /\* SCS[3] \*/

 | (uint32)((uint32)0U << 4U) /\* SCS[4] \*/

 | (uint32)((uint32)0U << 5U) /\* SCS[5] \*/

 | (uint32)((uint32)1U << 8U) /\* ENA \*/

 | (uint32)((uint32)1U << 9U) /\* CLK \*/

 | (uint32)((uint32)1U << 10U) /\* SIMO[0] \*/

 | (uint32)((uint32)1U << 11U) /\* SOMI[0] \*/

 | (uint32)((uint32)1U << 17U) /\* SIMO[1] \*/

 | (uint32)((uint32)1U << 25U); /\* SOMI[1] \*/

 /\*\* - Finally start MIBSPI1 \*/

 mibspiREG1->GCR1 = (mibspiREG1->GCR1 & 0xFEFFFFFFU) | 0x01000000U;

/\* USER CODE BEGIN (3) \*/

/\* USER CODE END \*/

}

/\*\* @fn void mibspiSetFunctional(mibspiBASE\_t \*mibspi, uint32 port)

\* @brief Change functional behavior of pins at runtime.

\* @param[in] mibspi - mibspi module base address

\* @param[in] port - Value to write to PC0 register

\*

\* Change the value of the PC0 register at runtime, this allows to

\* dynamically change the functionality of the MIBSPI pins between functional

\* and GIO mode.

\*/

/\* SourceId : MIBSPI\_SourceId\_002 \*/

/\* DesignId : MIBSPI\_DesignId\_002 \*/

/\* Requirements : HL\_SR154 \*/

void mibspiSetFunctional(mibspiBASE\_t \*mibspi, uint32 port)

{

/\* USER CODE BEGIN (4) \*/

/\* USER CODE END \*/

 mibspi->PC0 = port;

/\* USER CODE BEGIN (5) \*/

/\* USER CODE END \*/

}

/\*\* @fn void mibspiSetData(mibspiBASE\_t \*mibspi, uint32 group, uint16 \* data)

\* @brief Set Buffer Data

\* @param[in] mibspi - Spi module base address

\* @param[in] group - Transfer group (0..7)

\* @param[in] data - new data for transfer group

\*

\* This function updates the data for the specified transfer group,

\* the length of the data must match the length of the transfer group.

\*/

/\* SourceId : MIBSPI\_SourceId\_003 \*/

/\* DesignId : MIBSPI\_DesignId\_003 \*/

/\* Requirements : HL\_SR155 \*/

void mibspiSetData(mibspiBASE\_t \*mibspi, uint32 group, uint16 \* data)

{

/\* USER CODE BEGIN (6) \*/

/\* USER CODE END \*/

 mibspiRAM\_t \*ram = (mibspi == mibspiREG1) ? mibspiRAM1 : ((mibspi == mibspiREG3) ? mibspiRAM3 : mibspiRAM5);

 uint32 start = (mibspi->TGCTRL[group] >> 8U) & 0xFFU;

 uint32 end = (group == 7U) ? (((mibspi->LTGPEND & 0x00007F00U) >> 8U) + 1U) : ((mibspi->TGCTRL[group+1U] >> 8U) & 0xFFU);

 if (end == 0U) {end = 128U;}

 while (start < end)

 {

 /\*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" \*/

 ram->tx[start].data = \*data;

 /\*SAFETYMCUSW 567 S MR:17.1,17.4 <APPROVED> "Pointer increment needed" \*/

 data++;

 start++;

 }

/\* USER CODE BEGIN (7) \*/

/\* USER CODE END \*/

}

/\*\* @fn void mibspiGetData(mibspiBASE\_t \*mibspi, uint32 group, uint16 \* data)

\* @brief Retrieves Buffer Data from receive buffer

\* @param[in] mibspi - Spi module base address

\* @param[in] group - Transfer group (0..7)

\* @param[out] data - pointer to data array

\*

\* @return error flags from data buffer, if there was a receive error on

\* one of the buffers this will be reflected in the return value.

\*

\* This function transfers the data from the specified transfer group receive

\* buffers to the data array, the length of the data must match the length

\* of the transfer group.

\*/

/\* SourceId : MIBSPI\_SourceId\_004 \*/

/\* DesignId : MIBSPI\_DesignId\_004 \*/

/\* Requirements : HL\_SR156 \*/

uint32 mibspiGetData(mibspiBASE\_t \*mibspi, uint32 group, uint16 \* data)

{

/\* USER CODE BEGIN (8) \*/

/\* USER CODE END \*/

 mibspiRAM\_t \*ram = (mibspi == mibspiREG1) ? mibspiRAM1 : ((mibspi == mibspiREG3) ? mibspiRAM3 : mibspiRAM5);

 uint32 start = (mibspi->TGCTRL[group] >> 8U) & 0xFFU;

 uint32 end = (group == 7U) ? (((mibspi->LTGPEND & 0x00007F00U) >> 8U) + 1U) : ((mibspi->TGCTRL[group+1U] >> 8U) & 0xFFU);

 uint16 mibspiFlags = 0U;

 uint32 ret;

 if (end == 0U) {end = 128U;}

 while (start < end)

 {

 mibspiFlags |= ram->rx[start].flags;

 /\*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" \*/

 \*data = ram->rx[start].data;

 /\*SAFETYMCUSW 567 S MR:17.1,17.4 <APPROVED> "Pointer increment needed" \*/

 data++;

 start++;

 }

 ret = ((uint32)mibspiFlags >> 8U) & 0x5FU;

/\* USER CODE BEGIN (9) \*/

/\* USER CODE END \*/

 return ret;

}

/\*\* @fn void mibspiTransfer(mibspiBASE\_t \*mibspi, uint32 group)

\* @brief Transmit Transfer Group

\* @param[in] mibspi - Spi module base address

\* @param[in] group - Transfer group (0..7)

\*

\* Initiates a transfer for the specified transfer group.

\*/

/\* SourceId : MIBSPI\_SourceId\_005 \*/

/\* DesignId : MIBSPI\_DesignId\_005 \*/

/\* Requirements : HL\_SR157 \*/

void mibspiTransfer(mibspiBASE\_t \*mibspi, uint32 group)

{

/\* USER CODE BEGIN (10) \*/

/\* USER CODE END \*/

 mibspi->TGCTRL[group] |= 0x80000000U;

/\* USER CODE BEGIN (11) \*/

/\* USER CODE END \*/

}

/\*\* @fn boolean mibspiIsTransferComplete(mibspiBASE\_t \*mibspi, uint32 group)

\* @brief Check for Transfer Group Ready

\* @param[in] mibspi - Spi module base address

\* @param[in] group - Transfer group (0..7)

\*

\* @return TRUE is transfer complete, otherwise FALSE.

\*

\* Checks to see if the transfer for the specified transfer group

\* has finished.

\*/

/\* SourceId : MIBSPI\_SourceId\_006 \*/

/\* DesignId : MIBSPI\_DesignId\_006 \*/

/\* Requirements : HL\_SR158 \*/

boolean mibspiIsTransferComplete(mibspiBASE\_t \*mibspi, uint32 group)

{

 boolean status;

/\* USER CODE BEGIN (12) \*/

/\* USER CODE END \*/

 if(((((mibspi->TGINTFLG & 0xFFFF0000U) >> 16U)>> group) & 1U) == 1U)

 {

 mibspi->TGINTFLG = (mibspi->TGINTFLG & 0x0000FFFFU) | ((uint32)((uint32)1U << group) << 16U);

 status = TRUE;

 }

 else

 {

 status = FALSE;

 }

/\* USER CODE BEGIN (13) \*/

/\* USER CODE END \*/

 return (status);

}

/\*\* @fn void mibspiEnableLoopback(mibspiBASE\_t \*mibspi, loopBackType\_t Loopbacktype)

\* @brief Enable Loopback mode for self test

\* @param[in] mibspi - Mibspi module base address

\* @param[in] Loopbacktype - Digital or Analog

\*

\* This function enables the Loopback mode for self test.

\*/

/\* SourceId : MIBSPI\_SourceId\_007 \*/

/\* DesignId : MIBSPI\_DesignId\_009 \*/

/\* Requirements : HL\_SR161 \*/

//void mibspiEnableLoopback(mibspiBASE\_t \*mibspi, loopBackType\_t Loopbacktype)

//{

///\* USER CODE BEGIN (14) \*/

///\* USER CODE END \*/

//

// /\* Clear Loopback incase enabled already \*/

// mibspi->IOLPKTSTCR = 0U;

//

// /\* Enable Loopback either in Analog or Digital Mode \*/

// mibspi->IOLPKTSTCR = (uint32)0x00000A00U

// | (uint32)((uint32)Loopbacktype << 1U);

//

///\* USER CODE BEGIN (15) \*/

///\* USER CODE END \*/

//}

/\*\* @fn void mibspiDisableLoopback(mibspiBASE\_t \*mibspi)

\* @brief Enable Loopback mode for self test

\* @param[in] mibspi - Mibspi module base address

\*

\* This function disable the Loopback mode.

\*/

/\* SourceId : MIBSPI\_SourceId\_008 \*/

/\* DesignId : MIBSPI\_DesignId\_010 \*/

/\* Requirements : HL\_SR162 \*/

//void mibspiDisableLoopback(mibspiBASE\_t \*mibspi)

//{

///\* USER CODE BEGIN (16) \*/

///\* USER CODE END \*/

//

// /\* Disable Loopback Mode \*/

// mibspi->IOLPKTSTCR = 0x00000500U;

//

///\* USER CODE BEGIN (17) \*/

///\* USER CODE END \*/

//}

//

///\*\* @fn void mibspiPmodeSet(mibspiBASE\_t \*mibspi, mibspiPmode\_t Pmode, mibspiDFMT\_t DFMT)

//\* @brief Set the Pmode for the selected Data Format register

//\* @param[in] mibspi - Mibspi module base address

//\* @param[in] Pmode - Mibspi Parellel mode

//\* PMODE\_NORMAL

//\* PMODE\_2\_DATALINE

//\* PMODE\_4\_DATALINE

//\* PMODE\_8\_DATALINE

//\* @param[in] DFMT - Mibspi Data Format register

//\* DATA\_FORMAT0

//\* DATA\_FORMAT1

//\* DATA\_FORMAT2

//\* DATA\_FORMAT3

//\*

//\* This function sets the Pmode for the selected Data Format register.

//\*/

///\* SourceId : MIBSPI\_SourceId\_009 \*/

///\* DesignId : MIBSPI\_DesignId\_011 \*/

///\* Requirements : HL\_SR524 \*/

//void mibspiPmodeSet(mibspiBASE\_t \*mibspi, mibspiPmode\_t Pmode, mibspiDFMT\_t DFMT)

//{

// uint32 pmctrl\_reg;

// /\* Set the Pmode for the selected Data Format register \*/

// pmctrl\_reg = (mibspi->PMCTRL & (~(uint32)((uint32)0xFFU << (8U \* DFMT))));

// mibspi->PMCTRL = (pmctrl\_reg | (uint32)((uint32)Pmode << ((8U \* DFMT))));

//

//}

//

///\*\* @fn void mibspiEnableGroupNotification(mibspiBASE\_t \*mibspi, uint32 group, uint32 level)

//\* @brief Enable Transfer Group interrupt

//\* @param[in] mibspi - Spi module base address

//\* @param[in] group - Transfer group (0..7)

//\* @param[in] level - Interrupt level

//\*

//\* This function enables the transfer group finished interrupt.

//\*/

///\* SourceId : MIBSPI\_SourceId\_010 \*/

///\* DesignId : MIBSPI\_DesignId\_007 \*/

///\* Requirements : HL\_SR159 \*/

//void mibspiEnableGroupNotification(mibspiBASE\_t \*mibspi, uint32 group, uint32 level)

//{

///\* USER CODE BEGIN (18) \*/

///\* USER CODE END \*/

//

// if (level != 0U)

// {

// mibspi->TGITLVST = (mibspi->TGITLVST & 0x0000FFFFU) | (uint32)((uint32)((uint32)1U << group) << 16U);

// }

// else

// {

// mibspi->TGITLVCR = (mibspi->TGITLVCR & 0x0000FFFFU) | (uint32)((uint32)((uint32)1U << group) << 16U);

// }

// mibspi->TGITENST = (mibspi->TGITENST & 0x0000FFFFU) | (uint32)((uint32)((uint32)1U << group) << 16U);

//

///\* USER CODE BEGIN (19) \*/

///\* USER CODE END \*/

//}

//

//

///\*\* @fn void mibspiDisableGroupNotification(mibspiBASE\_t \*mibspi, uint32 group)

//\* @brief Disable Transfer Group interrupt

//\* @param[in] mibspi - Spi module base address

//\* @param[in] group - Transfer group (0..7)

//\*

//\* This function disables the transfer group finished interrupt.

//\*/

///\* SourceId : MIBSPI\_SourceId\_011 \*/

///\* DesignId : MIBSPI\_DesignId\_008 \*/

///\* Requirements : HL\_SR160 \*/

//void mibspiDisableGroupNotification(mibspiBASE\_t \*mibspi, uint32 group)

//{

///\* USER CODE BEGIN (20) \*/

///\* USER CODE END \*/

//

// mibspi->TGITENCR = (mibspi->TGITENCR & 0x0000FFFFU) | (uint32)((uint32)((uint32)1U << group) << 16U);

//

///\* USER CODE BEGIN (21) \*/

///\* USER CODE END \*/

//}

/\*\* @fn void mibspi1GetConfigValue(mibspi\_config\_reg\_t \*config\_reg, config\_value\_type\_t type)

\* @brief Get the initial or current values of the configuration registers

\*

\* @param[in] \*config\_reg: pointer to the struct to which the initial or current

\* value of the configuration registers need to be stored

\* @param[in] type: whether initial or current value of the configuration registers need to be stored

\* - InitialValue: initial value of the configuration registers will be stored

\* in the struct pointed by config\_reg

\* - CurrentValue: initial value of the configuration registers will be stored

\* in the struct pointed by config\_reg

\*

\* This function will copy the initial or current value (depending on the parameter 'type')

\* of the configuration registers to the struct pointed by config\_reg

\*

\*/

/\* SourceId : MIBSPI\_SourceId\_012 \*/

/\* DesignId : MIBSPI\_DesignId\_012 \*/

/\* Requirements : HL\_SR166 \*/

//void mibspi1GetConfigValue(mibspi\_config\_reg\_t \*config\_reg, config\_value\_type\_t type)

//{

// if (type == InitialValue)

// {

// config\_reg->CONFIG\_GCR1 = MIBSPI1\_GCR1\_CONFIGVALUE;

// config\_reg->CONFIG\_INT0 = MIBSPI1\_INT0\_CONFIGVALUE;

// config\_reg->CONFIG\_LVL = MIBSPI1\_LVL\_CONFIGVALUE;

// config\_reg->CONFIG\_PCFUN = MIBSPI1\_PCFUN\_CONFIGVALUE;

// config\_reg->CONFIG\_PCDIR = MIBSPI1\_PCDIR\_CONFIGVALUE;

// config\_reg->CONFIG\_PCPDR = MIBSPI1\_PCPDR\_CONFIGVALUE;

// config\_reg->CONFIG\_PCDIS = MIBSPI1\_PCDIS\_CONFIGVALUE;

// config\_reg->CONFIG\_PCPSL = MIBSPI1\_PCPSL\_CONFIGVALUE;

// config\_reg->CONFIG\_DELAY = MIBSPI1\_DELAY\_CONFIGVALUE;

// config\_reg->CONFIG\_FMT0 = MIBSPI1\_FMT0\_CONFIGVALUE;

// config\_reg->CONFIG\_FMT1 = MIBSPI1\_FMT1\_CONFIGVALUE;

// config\_reg->CONFIG\_FMT2 = MIBSPI1\_FMT2\_CONFIGVALUE;

// config\_reg->CONFIG\_FMT3 = MIBSPI1\_FMT3\_CONFIGVALUE;

// config\_reg->CONFIG\_MIBSPIE = MIBSPI1\_MIBSPIE\_CONFIGVALUE;

// config\_reg->CONFIG\_LTGPEND = MIBSPI1\_LTGPEND\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[0U] = MIBSPI1\_TGCTRL0\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[1U] = MIBSPI1\_TGCTRL1\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[2U] = MIBSPI1\_TGCTRL2\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[3U] = MIBSPI1\_TGCTRL3\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[4U] = MIBSPI1\_TGCTRL4\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[5U] = MIBSPI1\_TGCTRL5\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[6U] = MIBSPI1\_TGCTRL6\_CONFIGVALUE;

// config\_reg->CONFIG\_TGCTRL[7U] = MIBSPI1\_TGCTRL7\_CONFIGVALUE;

// config\_reg->CONFIG\_UERRCTRL = MIBSPI1\_UERRCTRL\_CONFIGVALUE;

// }

// else

// {

// /\*SAFETYMCUSW 134 S MR:12.2 <APPROVED> "LDRA Tool issue" \*/

// config\_reg->CONFIG\_GCR1 = mibspiREG1->GCR1;

// config\_reg->CONFIG\_INT0 = mibspiREG1->INT0;

// config\_reg->CONFIG\_LVL = mibspiREG1->LVL;

// config\_reg->CONFIG\_PCFUN = mibspiREG1->PC0;

// config\_reg->CONFIG\_PCDIR = mibspiREG1->PC1;

// config\_reg->CONFIG\_PCPDR = mibspiREG1->PC6;

// config\_reg->CONFIG\_PCDIS = mibspiREG1->PC7;

// config\_reg->CONFIG\_PCPSL = mibspiREG1->PC8;

// config\_reg->CONFIG\_DELAY = mibspiREG1->DELAY;

// config\_reg->CONFIG\_FMT0 = mibspiREG1->FMT0;

// config\_reg->CONFIG\_FMT1 = mibspiREG1->FMT1;

// config\_reg->CONFIG\_FMT2 = mibspiREG1->FMT2;

// config\_reg->CONFIG\_FMT3 = mibspiREG1->FMT3;

// config\_reg->CONFIG\_MIBSPIE = mibspiREG1->MIBSPIE;

// config\_reg->CONFIG\_LTGPEND = mibspiREG1->LTGPEND;

// config\_reg->CONFIG\_TGCTRL[0U] = mibspiREG1->TGCTRL[0U];

// config\_reg->CONFIG\_TGCTRL[1U] = mibspiREG1->TGCTRL[1U];

// config\_reg->CONFIG\_TGCTRL[2U] = mibspiREG1->TGCTRL[2U];

// config\_reg->CONFIG\_TGCTRL[3U] = mibspiREG1->TGCTRL[3U];

// config\_reg->CONFIG\_TGCTRL[4U] = mibspiREG1->TGCTRL[4U];

// config\_reg->CONFIG\_TGCTRL[5U] = mibspiREG1->TGCTRL[5U];

// config\_reg->CONFIG\_TGCTRL[6U] = mibspiREG1->TGCTRL[6U];

// config\_reg->CONFIG\_TGCTRL[7U] = mibspiREG1->TGCTRL[7U];

// config\_reg->CONFIG\_UERRCTRL = mibspiREG1->UERRCTRL;

// }

//}

extern void FUN\_BE13WDOff\_Init(void);//turn off the wd for be13 chip

void PRC\_MibSPIInit(void)

{

 mibspiInit();

// FUN\_BE13WDOff\_Init();

}

void PRC\_MibSPI1ms(void)

{

 FUN\_FillAVData1ms();

 FUN\_FillEVData1ms();

 mibspiSetData(mibspiREG1, 0, &VLV\_BE13TransData[0]);

 mibspiTransfer(mibspiREG1,0);

 mibspiSetData(mibspiREG1, 1, &VLV\_BE13TransData[8]);

 mibspiTransfer(mibspiREG1,1);

}

extern void FUN\_BE13IVLV\_PumMo\_Init(void);

void PRC\_MibSPI5ms(void)

{

 FUN\_BE13WDOff\_Init(); //if initial one time, the transfer have some problems, so do it here

 FUN\_BE13IVLV\_PumMo\_Init();//if initial one time, the transfer have some problems, so do it here

 FUN\_SYSSuspendIRQs\_V();

 FUN\_SetAVDurCntr\_5ms();

 FUN\_SetDDDCntr\_5ms();

 FUN\_SYSResumeIRQs\_V();

 FUN\_FillHSVData5ms();

 FUN\_FillUSVData5ms();

 mibspiSetData(mibspiREG1, 2, &VLV\_BE13TransData[0]);

 mibspiTransfer(mibspiREG1,2);

 mibspiSetData(mibspiREG1, 3, &VLV\_BE13TransData[8]);

 mibspiTransfer(mibspiREG1,3);

}