```
/*
The following are included in this single document to illustrate a Power Management
chip device driver and simple test code, layered over a CPU-SPI peripheral
driver (not included, besides portions of the interface), with Doxygen documentation tags:

pmic35584.h
pmic35584.c
pmictest.c
periph_pins.h
spicpu.h
spicpu.c

This driver implements init, register read, register write functions for the PMIC.
All file header notation is for demonstration purposes only.
This is prototype, not production code.
*/


/**
 * @file pmic35584.h
 *
 * @brief Power management IC declarations, registers
 *
 * @author Tyler Sheffield
 *
 * @copyright Copyright (c) 2019 MemBeta. All rights reserved.
 * */
#ifndef PMIC35584_H
#define PMIC35584_H

#include <stdint.h>

/*****************************************************************************/
/*-----------------------------------Macros/Enum-----------------------------*/
/*****************************************************************************/
#define PMIC_SPI_MODULE    2
#define PMIC_SPI_BAUDRATE  1000000

/* ******************************************************* */
/* The 16-bit frame for this PMIC is constructed as follows:
/* PMIC write, PMIC read - CMD1|ADDR6|DATA8|P1 *\/ */
/* CMD = 1 for write, 0 for read */
/* ******************************************************* */
#define PMIC_WR_CMD_BIT 0x4000
#define PMIC_PLOADLEN   15
#define PMIC_DATABITS   8
#define PMIC_MASKLO     0x00FF
#define PMIC_MASKHI     0xFF00

/* Define the addresses for the PMIC registers, no local storage for now */
/* MISRA C 2004 requires use of = for only first or all, not some though
 * this was removed for 2012 */
enum PMICReg
{
    PMIC_DEVCFG0 = 0x0,
    PMIC_DEVCFG1,
    PMIC_DEVCFG2,
    PMIC_PROTCFG,
    PMIC_SYSPCFG0,
    PMIC_SYSPCFG1,
    PMIC_WDCFG0,
    PMIC_WDCFG1,
    PMIC_FWDCFG,
    PMIC_WWDCFG0,
    PMIC_WWDCFG1,
    PMIC_RSYSPCFG0,
    PMIC_RSYSPCFG1,
    PMIC_RWDCFG0,
    PMIC_RWDCFG1,
```

```
    PMIC_RFWDCFG,

    // offset 0x10
    PMIC_RWWDCFG0,
    PMIC_RWWDCFG1,
    PMIC_WKTIMCFG0,
    PMIC_WKTIMCFG1,
    PMIC_WKTIMCFG2,
    PMIC_DEVCTRL,
    PMIC_DEVCTRLN,
    PMIC_WWDSCMD,
    PMIC_FWDRSP,
    PMIC_FWDRSPSYNC,
    PMIC_SYSFAIL,
    PMIC_INITERR,
    PMIC_IF,
    PMIC_SYSSF,
    PMIC_WKSF,
    PMIC_SPISF,

    // offset 0x20
    PMIC_MONSF0,
    PMIC_MONSF1,
    PMIC_MONSF2,
    PMIC_MONSF3,
    PMIC_OTFAIL,
    PMIC_OTWRNSF,
    PMIC_VMONSTAT,
    PMIC_DEVSTAT,
    PMIC_PROTSTAT,
    PMIC_WWDSTAT,
    PMIC_FWDSTAT0,
    PMIC_FWDSTAT1,
    PMIC_ABIST_CTRL0,
    PMIC_ABIST_CTRL1,
    PMIC_ABIST_SELECT0,
    PMIC_ABIST_SELECT1,

    // offset 0x30
    PMIC_ABIST_SELECT2,
    PMIC_BCK_FREQ_CHANGE,
    PMIC_BCK_FRE_SPREAD,
    PMIC_BCK_MAIN_CTRL,

    // not contiguous!
    PMIC_GTM = 0x3F
};
typedef enum PMICReg PMICReg_t;

/*****************************************************************************/
/*-------------------------Function Prototypes-------------------------------*/
/*****************************************************************************/
void PMIC_spiChannelInit(uint32_t);
void PMIC_setRegister(PMICReg_t, uint8_t);
uint8_t PMIC_getRegister(PMICReg_t);
#endif /* PMIC_H */



/**
 * @file pmic35584.c
 *
 * @brief Interface implementation for the Power Management IC
 *
 * @author Tyler Sheffield
 *
 * @copyright Copyright (c) 2019 MemBeta. All rights reserved.
 * */
#include <stdio.h>
```

```c
#include "pmic35584.h"
#include "Ifx_Assert.h"
#include "spicpu.h"
#include "periph_pins.h"

/******************************************************************************/
/*-----------------------------Global variables-------------------------------*/
/******************************************************************************/
/**< @brief Spi global data */
extern App_SpiCpu g_SpiCpu[];

/**< @brief Spi memory mapped module references */
extern Ifx_SPI* IfxSpi_ModuleRefTable[];

/******************************************************************************/
/*-------------------------Function Prototypes--------------------------------*/
/******************************************************************************/
extern void IfxSpi_MasterConfig_ISRinit(uint32_t, IfxSpi_SpiMaster_Config*);


/** @brief Spi master channel initialization
 *
 * This function initialises the given SPI module in master mode for the PMIC device.
 * NOTE: interrupts should be disabled before calling this function, then enabled.
 *
 * @param loopback_ctrlbit :One for loopback enable, zero for no
 */
void PMIC_spiChannelInit(uint32_t loopback_ctrlbit)
{
    uint32_t                          i;
    IfxSpi_SpiMaster_Config        spiMasterConfig;
    IfxSpi_SpiMaster_ChannelConfig spiMasterChannelConfig;

    uint32_t module = PMIC_SPI_MODULE;
    Ifx_Spi* moduleSpi = IfxSpi_ModuleRefTable[module];

    /* spiMasterConfig init */

    /* create module config */
    IfxSpi_SpiMaster_initModuleConfig(&spiMasterConfig, moduleSpi);

    /* set the maximum baudrate */
    spiMasterConfig.base.maximumBaudrate = PMIC_SPI_BAUDRATE;

    /* ISR priorities and interrupt target */
    IfxSpi_MasterConfig_ISRinit( module, &spiMasterConfig );

    /* pin configuration, except CS */
    const IfxSpi_SpiMaster_Pins pins = PMIC_SPI_PINSGROUP;
    spiMasterConfig.pins = &pins;

    /* initialize module */
    IfxSpi_SpiMaster_initModule(&g_SpiCpu[module].drivers.spiMaster, &spiMasterConfig);


    /* spiMasterChannelConfig init */

    /* create channel config */
    IfxSpi_SpiMaster_initChannelConfig(&spiMasterChannelConfig,
        &g_SpiCpu[module].drivers.spiMaster);

    /* Settings shared for all channels on this module */
    spiMasterChannelConfig.mode                 = IfxSpi_SpiMaster_Mode_short;
    spiMasterChannelConfig.channelBasedCs       = IfxSpi_SpiMaster_ChannelBasedCs_disabled;
    spiMasterChannelConfig.base.mode.enabled    = 1;
    spiMasterChannelConfig.base.mode.autoCS     = 1;

    /* * ** NOTE for loopback mode ** */
    /* * In case you want to configure and test a SPI channel in loopback, you have to */
```

```
    /* * select: */
    /* * spiMasterChannelConfig.base.mode.loopback = 1 */
    /* * */
    /* * If an output pin is configured in loopback mode, the  SPI channel number will
    /* * be extracted from the pin map configuration. */
    /* * */
    /* * If an output pin is not configured in loopback, the default SPI channel selected */
    /* * will be 0. */
    spiMasterChannelConfig.base.mode.loopback = loopback_ctrlbit;

    /* Start by setting up the ChannelConfig; in driver initchannel, the channel
     * object copies information from the channel config object */
    /* *** NOTE the peripheral uses this setting to set the slave select line *** */
    const IfxSpi_SpiMaster_Output slsOutput = PMIC_SPI_CSGROUP;

    spiMasterChannelConfig.sls.output.pin    = slsOutput.pin;
    spiMasterChannelConfig.sls.output.mode   = slsOutput.mode;
    spiMasterChannelConfig.sls.output.driver = slsOutput.driver;

    /*PMIC Configuration
     *Short Data Mode is BACON.LAST = 1 and BACON.BYTE = 0.
     *BACON.PARTYP=0 for even parity per PMIC manual
     *BACON.MSB=1
     *BACON.DL=15
     *BACON.CS=1
     *BACON.LEAD=1
     *BACON.TRAIL=1
     *ECON1.CPH=1
     *ECON1.PAREN=1
     *The LAST and BYTE options are written to regs in the driver write function
     *per channel mode setting
     *CS is supposed to be set through the pin selection */
    spiMasterChannelConfig.base.mode.parityMode = 0;   //PARTYP
    spiMasterChannelConfig.base.mode.dataHeading = 1;  //MSB
    spiMasterChannelConfig.base.mode.dataWidth = PMIC_PLOADLEN;   //DL
    spiMasterChannelConfig.base.mode.parityCheck = 1;  //PAREN
    spiMasterChannelConfig.base.mode.csLeadDelay = SpiIf_SlsoTiming_1;  //LEAD
    spiMasterChannelConfig.base.mode.shiftClock = 1;   //CPH
    spiMasterChannelConfig.base.mode.csTrailDelay = 1; //TRAIL

    /* set the baudrate for this channel */
    spiMasterChannelConfig.base.baudrate = PMIC_SPI_BAUDRATE;

    /* initialize channel */
    IfxSpi_SpiMaster_initChannel(&g_SpiCpu[module].drivers.spiMasterChannel,
        &spiMasterChannelConfig);
    /* end spiMasterChannelConfig init */

} // PMIC_SpiChannelInit


/** @brief Sets a PMIC register to the given value
 *
 *  Sets a PMIC control register value.  This is a blocking
 *  call in which SPI transaction completion is awaited.  A timeout needs to be added.
 *
 *  @param addr :register address
 *  @param data :byte payload that is to be sent
 */
void PMIC_setRegister(PMICReg_t addr, uint8_t data)
{
    uint32_t module = PMIC_SPI_MODULE;

    uint16_t txframe = PMIC_WR_CMD_BIT | ((addr << PMIC_DATABITS) & PMIC_MASKHI) | data;

    g_SpiCpu[module].SpiBuffer.spiTxBuffer[0] = txframe;

    while (IfxSpi_SpiMaster_getStatus(&g_SpiCpu[module].drivers.spiMasterChannel)
            == SpiIf_Status_busy)
```

```
    {}
    IfxSpi_SpiMaster_exchange(&g_SpiCpu[module].drivers.spiMasterChannel,
                              &g_SpiCpu[module].SpiBuffer.spiTxBuffer[0],
                              &g_SpiCpu[module].SpiBuffer.spiRxBuffer[0], SPI_BUFFER_SIZE);

    /* wait until received all data */
    while (IfxSpi_SpiMaster_getStatus(&g_SpiCpu[module].drivers.spiMasterChannel)
           == SpiIf_Status_busy)
    {}

    uint16_t* spiReceived = (uint16 *)g_SpiCpu[module].SpiBuffer.spiRxBuffer;
    uint8_t rxdata = (uint8_t) (PMIC_MASKLO & (*spiReceived));

    /* Check that write data was echoed back from device in the return data field */
    if (rxdata != data) {
        //TODO log error
        printf("PMIC SPI write error\n");
    }
} //PMIC_setRegister


/** @brief Gets the value of a register on the PMIC
 *
 *  Reads the PMIC control register to know which bits are being read. This is a blocking
 *  call waiting for proper SPI status.  A timeout needs to be added.
 *
 *  @param addr :register to read
 *  @return the PMIC register value
 */
uint8_t PMIC_getRegister(PMICReg_t addr)
{
    uint32_t module = PMIC_SPI_MODULE;
    uint8_t rxdata;

    uint16_t txframe = (addr << PMIC_DATABITS) & PMIC_MASKHI;

    g_SpiCpu[module].SpiBuffer.spiTxBuffer[0] = txframe;

    while (IfxSpi_SpiMaster_getStatus(&g_SpiCpu[module].drivers.spiMasterChannel)
           == SpiIf_Status_busy)
    {}
    IfxSpi_SpiMaster_exchange(&g_SpiCpu[module].drivers.spiMasterChannel,
                              &g_SpiCpu[module].SpiBuffer.spiTxBuffer[0],
                              &g_SpiCpu[module].SpiBuffer.spiRxBuffer[0], SPI_BUFFER_SIZE);


    /* wait until received all data */
    while (IfxSpi_SpiMaster_getStatus(&g_SpiCpu[module].drivers.spiMasterChannel)
           == SpiIf_Status_busy)
    {}

    uint16_t* spiReceived = (uint16_t *)g_SpiCpu[module].SpiBuffer.spiRxBuffer;
    rxdata = (uint8_t) (PMIC_MASKLO & (*spiReceived));

    return rxdata;
} //PMIC_getRegister




/**
 * @file pmictest.c
 *
 * @brief Init and run a test sequence of PMIC SPI interface
 *
 * @author Tyler Sheffield
 *
```

```
 * @copyright Copyright (c) 2019 MemBeta. All rights reserved.
 * */

#include <stdio.h>
#include <Cpu/Std/IfxCpu.h>
#include "pmic35584.h"

/******************************************************************************/
/*-------------------------Function Prototypes--------------------------------*/
/******************************************************************************/
extern int IfxSpiCpuDemo_SingleFrameIO(uint32_t, uint16_t);


/** @brief PMIC init for test run
 *
 * This function is called from main during initialization phase
 */
void PMIC_spiInit()
{
    /* disable interrupts */
    boolean interruptState = IfxCpu_disableInterrupts();

    PMIC_spiChannelInit(0);

    /* enable interrupts again */
    IfxCpu_restoreInterrupts(interruptState);
} //PMIC_spiInit


/** @brief PMIC init for test run
 *
 * This function is called from main during initialization phase
 * Inits the channel with PMIC specs, but sets loopback bit so data never leaves chip
 */
void PMIC_spiLoopbackInit()
{
    /* disable interrupts */
    boolean interruptState = IfxCpu_disableInterrupts();

    PMIC_spiChannelInit(1);

    /* enable interrupts again */
    IfxCpu_restoreInterrupts(interruptState);
} //PMIC_spiLoopbackInit


/* @brief PMIC test code sequence
 *
 * UNDER CONSTRUCTION for testing
 * This function is called from main application
*/
void PMIC_spiRun(void)
{
    uint8_t regclear = 0xFF;

    // READ SPI STATUS REG
    printf("PMIC SPI  read reg 0x%x: 0x%x\n", PMIC_SPISF, PMIC_getRegister(PMIC_SPISF) );

    // READ SPI Interrupt Flags
    printf("PMIC SPI  read reg 0x%x: 0x%x\n", PMIC_IF, PMIC_getRegister(PMIC_IF) );

    // WRITE SPI Interrupt Flags to clear
    printf("PMIC SPI write reg 0x%x: 0x%x\n", PMIC_IF, regclear );
    PMIC_setRegister(PMIC_IF, regclear);

    // READ SPI STATUS REG
    printf("PMIC SPI  read reg 0x%x: 0x%x\n", PMIC_SPISF, PMIC_getRegister(PMIC_SPISF) );

    // READ GTM REG
```

```
    printf("PMIC SPI  read reg 0x%x: 0x%x\n", PMIC_GTM, PMIC_getRegister(PMIC_GTM) );

    // READ WWDCFG1 REG
    printf("PMIC SPI  read reg 0x%x: 0x%x\n", PMIC_WWDCFG1, PMIC_getRegister(PMIC_WWDCFG1) );

} //PMIC_spiRun


/* @brief PMIC test code sequence, internal SPI loopback
 *
 * UNDER CONSTRUCTION for testing
 * This function is called from main application
*/
void PMIC_spiLoopbackRun(void)
{
    uint16_t tpattern = 0x7654;

    if ( IfxSpiCpuDemo_SingleFrameIO(PMIC_Spi_MODULE,tpattern) ) {
        printf("PMIC SPI loopback test FAIL on module %d\n", PMIC_Spi_MODULE);
    }
    else {
        printf("PMIC SPI loopback test success on module %d\n", PMIC_Spi_MODULE);
    }

} //PMIC_spiLoopbackRun




/**
 * @file periph_pins.h
 *
 * @brief Macros for setting up pins; dependent upon the peripheral PinMap
 *
 * @author Tyler Sheffield
 *
 * @copyright Copyright (c) 2019 MemBeta. All rights reserved.
 * */

#include <_PinMap/IfxSpi_PinMap.h>

/* ********************************** */
/* PMIC ******************************* */
/* ********************************** */

// Pin Groups
#define PMIC_SPI_PINSGROUP                                                  \
    {&IfxSpi2_SCLK_P15_3_OUT,   /* SCLK */                                  \
     IfxPort_OutputMode_pushPull,                                           \
     &IfxSpi2_MTSR_P15_6_OUT,                                               \
     IfxPort_OutputMode_pushPull, /* MTSR */                                 \
     &IfxSpi2_MRSTB_P15_7_IN,                                               \
     IfxPort_InputMode_pullUp,  /* MRST */                                  \
     IfxPort_PadDriver_cmosAutomotiveSpeed1 /* pad driver mode */            \
    }

#define PMIC_SPI_CSGROUP                        \
    {&IfxSpi2_SLSO1_P14_2_OUT,                  \
     IfxPort_OutputMode_pushPull,               \
     IfxPort_PadDriver_cmosAutomotiveSpeed1     \
    }

// Single Pins
#define PMIC_SPI_SCLK &IfxSpi2_SCLK_P15_3_OUT
#define PMIC_SPI_MOSI &IfxSpi2_MTSR_P15_6_OUT
#define PMIC_SPI_MISO &IfxSpi2_MRSTB_P15_7_IN
#define PMIC_SPI_CS &IfxSpi2_SLSO1_P14_2_OUT
#define PMIC_SPI_SPD IfxPort_PadDriver_cmosAutomotiveSpeed1
#define PMIC_SPI_OUTMODE IfxPort_OutputMode_pushPull
```

```
#define PMIC_SPI_INMODE IfxPort_InputMode_pullUp



/**
 * @file spicpu.h
 * @brief Spi Master header (using cpu)
 *
 * @copyright Copyright (c) 2014 ChipMaker. All rights reserved.
 *
 */
#ifndef SPICPU_H
#define SPICPU_H

/******************************************************************************/
/*-----------------------------------Includes---------------------------------*/
/******************************************************************************/
#include <Spi/SpiMaster/IfxSpi_SpiMaster.h>

/******************************************************************************/
/*-------------------------------------Macros---------------------------------*/
/******************************************************************************/
#define SPI_BUFFER_SIZE 1   /**< @brief Tx/Rx Buffer size */
#define NUM_SPI_MODULES 6

/******************************************************************************/
/*-----------------------------Data Structures--------------------------------*/
/******************************************************************************/
typedef struct
{
    uint16 spiTxBuffer[SPI_BUFFER_SIZE];      /**< @brief Spi Transmit buffer */
    uint16 spiRxBuffer[SPI_BUFFER_SIZE];      /**< @brief Spi receive buffer */
} AppSpiBuffer;

/** @brief SpiCpu global data */
typedef struct
{
    AppSpiBuffer SpiBuffer;                     /**< @brief Spi buffer */
    struct
    {
        IfxSpi_SpiMaster         spiMaster;        /**< @brief Spi Master handle */
        IfxSpi_SpiMaster_Channel spiMasterChannel; /**< @brief Spi Master Channel handle */
    }drivers;
} App_SpiCpu;

/******************************************************************************/
/*-----------------------------Global variables-------------------------------*/
/******************************************************************************/
extern App_SpiCpu g_SpiCpu[];

#endif


/**
 * @file spicpu.c
 * @brief SPI Peripheral Service Functions (Cpu Spi)
 *
 * @copyright Copyright (c) 2019 MemBeta. All rights reserved.
 * */

/******************************************************************************/
/*-----------------------------------Includes---------------------------------*/
/******************************************************************************/
#include <stdio.h>
#include <stdint.h>
#include "spicpu.h"

/******************************************************************************/
```

```
/*------------------------------Global variables------------------------------*/
/******************************************************************************/
App_SpiCpu g_SpiCpu[NUM_SPI_MODULES]; /**< @brief Spi global data */

Ifx_Spi* IfxSpi_ModuleRefTable[] =
{
    &MODULE_Spi0,
    &MODULE_Spi1,
    &MODULE_Spi2,
    &MODULE_Spi3,
    &MODULE_Spi4,
    &MODULE_Spi5
};  /**< @brief Spi memory mapped module references */

/******************************************************************************/
/*-------------------------Function Implementations---------------------------*/
/******************************************************************************/

/** @brief Function for firing SPI transaction
 *
 * SPI exchange and blocking status code, with rx/tx buffer compare
 *
 * @param module :which Spi module to use
 * @param data :16-bit data to be transmitted
 * @return Number of errors found in received data compared to expected
 */
int IfxSpiCpuDemo_SingleFrameIO(uint32_t module, uint16_t data)
{
    g_SpiCpu[module].SpiBuffer.spiTxBuffer[0] = data;

    {
        /* Master transfer */
        printf("wait until master is free\n");
        while (IfxSpi_SpiMaster_getStatus(&g_SpiCpu[module].drivers.spiMasterChannel)
               == SpiIf_Status_busy)
        {}
        IfxSpi_SpiMaster_exchange(&g_SpiCpu[module].drivers.spiMasterChannel,
        &g_SpiCpu[module].SpiBuffer.spiTxBuffer[0],
                                  &g_SpiCpu[module].SpiBuffer.spiRxBuffer[0], SPI_BUFFER_SIZE);
    }

    /* wait until received all data */
    while (IfxSpi_SpiMaster_getStatus(&g_SpiCpu[module].drivers.spiMasterChannel) ==
    SpiIf_Status_busy)
    {}

    printf("Check received SPI data\n");

    uint32_t errors       = 0;
    uint16_t *spiReceived = (uint16_t *)g_SpiCpu[module].SpiBuffer.spiRxBuffer;
    uint16_t *spiExpected = (uint16_t *)g_SpiCpu[module].SpiBuffer.spiTxBuffer;

    uint32_t i;
    for (i = 0; i < SPI_BUFFER_SIZE; i++) {
        if (*spiReceived++ != *spiExpected++) {
            ++errors;
        }
    } //for

    return errors;

} //IfxSpiCpuDemo_SingleFrameIO
```