```
/* SAM4S4B_spi.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the SPI (Serial
 * Peripheral Interface) peripheral of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_SPI_H
#define SAM4S4B_SPI_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// SPI Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define SPI_BASE   (0x40008000U) // SPI Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// SPI Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the SPI_CR register
typedef struct {
    volatile uint32_t SPIEN    : 1;
    volatile uint32_t SPIDIS   : 1;
    volatile uint32_t          : 5;
    volatile uint32_t SWRST    : 1;
    volatile uint32_t          : 16;
    volatile uint32_t LASTXFER : 1;
    volatile uint32_t          : 7;
} SPI_CR_bits;

// Bit field struct for the SPI_MR register
typedef struct {
    volatile uint32_t MSTR    : 1;
    volatile uint32_t PS      : 1;
    volatile uint32_t PCSDEC  : 1;
    volatile uint32_t         : 1;
    volatile uint32_t MODFDIS : 1;
    volatile uint32_t WDRBT   : 1;
    volatile uint32_t         : 1;
    volatile uint32_t LLB     : 1;
    volatile uint32_t         : 8;
    volatile uint32_t PCS     : 4;
    volatile uint32_t         : 4;
    volatile uint32_t DLYBCS  : 8;
} SPI_MR_bits;

// Bit field struct for the SPI_RDR register
typedef struct {
    volatile uint32_t RD  : 16;
    volatile uint32_t PCS : 4;
    volatile uint32_t     : 12;
} SPI_RDR_bits;

// Bit field struct for the SPI_TDR register
typedef struct {
    volatile uint32_t TD       : 16;
    volatile uint32_t PCS      : 4;
    volatile uint32_t          : 4;
    volatile uint32_t LASTXFER : 1;
    volatile uint32_t          : 7;
} SPI_TDR_bits;

// Bit field struct for the SPI_SR register
typedef struct {
    volatile uint32_t RDRF    : 1;
    volatile uint32_t TDRE    : 1;
    volatile uint32_t MODF    : 1;
    volatile uint32_t OVRES   : 1;
    volatile uint32_t ENDRX   : 1;
    volatile uint32_t ENDTX   : 1;
    volatile uint32_t RXBUFF  : 1;
    volatile uint32_t TXBUFE  : 1;
    volatile uint32_t NSSR    : 1;
    volatile uint32_t TXEMPTY : 1;
    volatile uint32_t UNDES   : 1;
    volatile uint32_t         : 5;
    volatile uint32_t SPIENS  : 1;
    volatile uint32_t         : 15;
} SPI_SR_bits;

// Bit field struct for the SPI_CSR register
typedef struct {
    volatile uint32_t CPOL   : 1;
    volatile uint32_t NCPHA  : 1;
    volatile uint32_t CSNAAT : 1;
    volatile uint32_t CSAAT  : 1;
    volatile uint32_t BITS   : 4;
    volatile uint32_t SCBR   : 8;
    volatile uint32_t DLYBS  : 8;
    volatile uint32_t DLYBCT : 8;
} SPI_CSR_bits;
```

```
// Peripheral struct for the SPI peripheral
typedef struct {
    volatile SPI_CR_bits  SPI_CR;        // (Spi Offset: 0x00) Control Register
    volatile SPI_MR_bits  SPI_MR;        // (Spi Offset: 0x04) Mode Register
    volatile SPI_RDR_bits SPI_RDR;       // (Spi Offset: 0x08) Receive Data
     Register
    volatile SPI_TDR_bits SPI_TDR;       // (Spi Offset: 0x0C) Transmit Data
     Register
    volatile SPI_SR_bits  SPI_SR;        // (Spi Offset: 0x10) Status Register
    volatile uint32_t     SPI_IER;       // (Spi Offset: 0x14) Interrupt Enable
     Register
    volatile uint32_t     SPI_IDR;       // (Spi Offset: 0x18) Interrupt
     Disable Register
    volatile uint32_t     SPI_IMR;       // (Spi Offset: 0x1C) Interrupt Mask
     Register
    volatile uint32_t     Reserved1[4];
    volatile SPI_CSR_bits SPI_CSR0;      // (Spi Offset: 0x30) Chip Select
     Register 0
    volatile SPI_CSR_bits SPI_CSR1;      // (Spi Offset: 0x30) Chip Select
     Register 1
    volatile SPI_CSR_bits SPI_CSR2;      // (Spi Offset: 0x30) Chip Select
     Register 2
    volatile SPI_CSR_bits SPI_CSR3;      // (Spi Offset: 0x30) Chip Select
     Register 3
    volatile uint32_t     Reserved2[41];
    volatile uint32_t     SPI_WPMR;      // (Spi Offset: 0xE4) Write Protection
     Control Register
    volatile uint32_t     SPI_WPSR;      // (Spi Offset: 0xE8) Write Protection
     Status Register
    volatile uint32_t     Reserved3[5];
    volatile uint32_t     SPI_RPR;       // (Spi Offset: 0x100) Receive Pointer
     Register
    volatile uint32_t     SPI_RCR;       // (Spi Offset: 0x104) Receive Counter
     Register
    volatile uint32_t     SPI_TPR;       // (Spi Offset: 0x108) Transmit
     Pointer Register
    volatile uint32_t     SPI_TCR;       // (Spi Offset: 0x10C) Transmit
     Counter Register
    volatile uint32_t     SPI_RNPR;      // (Spi Offset: 0x110) Receive Next
     Pointer Register
    volatile uint32_t     SPI_RNCR;      // (Spi Offset: 0x114) Receive Next
     Counter Register
    volatile uint32_t     SPI_TNPR;      // (Spi Offset: 0x118) Transmit Next
     Pointer Register
    volatile uint32_t     SPI_TNCR;      // (Spi Offset: 0x11C) Transmit Next
     Counter Register
    volatile uint32_t     SPI_PTCR;      // (Spi Offset: 0x120) Transfer
     Control Register
    volatile uint32_t     SPI_PTSR;      // (Spi Offset: 0x124) Transfer Status
     Register
} Spi;

// Pointer to an Spi-sized chunk of memory at the SPI peripheral
#define SPI ((Spi*) SPI_BASE)


///////////////////////////////////////////////////////////////////////////////
 /////////////////////
// SPI Definitions
///////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define SPI_WPMR_WPKEY_PASSWD (0x535049u << 8)


///////////////////////////////////////////////////////////////////////////////
 /////////////////////
// SPI Functions
///////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Enables the SPI peripheral and intializes its clock speed (baud rate),
 polarity, and phase. */
void spiInit(uint32_t clkdivide, uint32_t cpol, uint32_t ncpha) {
    pmcEnablePeriph(PMC_ID_SPI);
    /*Initializes the SPI interface for Chip Select line 0

    clkdivide (0x01 to 0xFF). The SPI clk will be the master clock / clkdivide
    cpol: clock polarity (0: inactive state is logic level 0, 1: inactive state
     is logic level 1)
    ncpha: clock phase (0: data changed on leading edge of clk and captured on
     next edge, 1: data captured on leading edge of clk and changed on next
     edge)
    Please see p585-p586 for cpol/ncpha timing diagrams

    This implements only: (p601/p610)
        1) SPI Master Mode
        2) Fixed Peripheral Select
        3) Mode Fault Detection Enabled
        4) Local Loopback Disabled
        5) 8 Bits Per Transfer
    Please read the SPI User Interface section of the datasheet for more
     advanced configuration features
    */

    //Initially assigning SPI pins (PA11-PA14) to peripheral A (SPI). Pin
     mapping given in p38-p39
    pioPinMode(PIO_PA11, PIO_PERIPH_A);
```

```c
    pioPinMode(PIO_PA12, PIO_PERIPH_A);
    pioPinMode(PIO_PA13, PIO_PERIPH_A);
    pioPinMode(PIO_PA14, PIO_PERIPH_A);

    //next setting the SPI control register (p600). Set to 1 to enable SPI
    SPI->SPI_CR.SPIEN = 1;

    //next setting the SPI mode register (p601) with the following:
    //master mode
    //fixed peripheral select
    //chip select lines directly connected to peripheral device
    //mode fault detection enabled
    //WDRBT disabled
    //LLB disabled
    //PCS = 0000 (Peripheral 0 selected), means NPCS[3:0] = 1110
    SPI->SPI_MR.MSTR = 1;

    //next setting the chip select register for peripheral 0 (p610)
    //ignoring delays
    SPI->SPI_CSR0.SCBR = (cpol<<0) | (ncpha<<1) | (clkdivide << 16);
}

char spiSendReceive(char send) {
    //Sends one byte over SPI and returns the received character
    SPI->SPI_TDR.TD = send;
    //Wait until Receive Data Register Full (RDRF, bit 0) and TXEMPTY (bit )
    while (!(SPI->SPI_SR.RDRF) || (SPI->SPI_SR.TXEMPTY));
    //After these status bits have gone high, the transaction is complete
    return (char) (SPI->SPI_RDR.RD);
}

short spiSendReceive16(uint16_t send) {
    //sends one 16-bit short over SPI and returns the received short
    short rec;
    rec = spiSendReceive((send & 0xFF00) >> 8); // send data MSB first
    rec = (rec << 8) | spiSendReceive(send & 0xFF);
    return rec;
}


#endif
```