```
/* SAM4S4B_uart.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the UART
 * (Universal Asynchronous Receiver-Transmitter) peripheral of the SAM4S4B
  microcontroller. */

#ifndef SAM4S4B_UART_H
#define SAM4S4B_UART_H

#include <stdint.h>
#include "SAM4S4B_pio.h"

////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// UART Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

#define UART0_BASE (0x400E0600U) // UART0 Base Address


////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// UART Registers
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

// Bit field struct for the UART_CR register
typedef struct {
    volatile uint32_t        : 2;
    volatile uint32_t RSTRX  : 1;
    volatile uint32_t RSTTX  : 1;
    volatile uint32_t RXEN   : 1;
    volatile uint32_t RXDIS  : 1;
    volatile uint32_t TXEN   : 1;
    volatile uint32_t TXDIS  : 1;
    volatile uint32_t RSTSTA : 1;
    volatile uint32_t        : 23;
} UART_CR_bits;

// Bit field struct for the UART_MR register
typedef struct {
    volatile uint32_t        : 9;
    volatile uint32_t PAR    : 3;
    volatile uint32_t        : 2;
    volatile uint32_t CHMODE : 2;
```

```
    volatile uint32_t        : 16;
} UART_MR_bits;

// Bit field struct for the UART_SR register
typedef struct {
    volatile uint32_t RXRDY   : 1;
    volatile uint32_t TXRDY   : 1;
    volatile uint32_t         : 1;
    volatile uint32_t ENDRX   : 1;
    volatile uint32_t ENDTX   : 1;
    volatile uint32_t OVRE    : 1;
    volatile uint32_t FRAME   : 1;
    volatile uint32_t PARE    : 1;
    volatile uint32_t         : 1;
    volatile uint32_t TXEMPTY : 1;
    volatile uint32_t         : 1;
    volatile uint32_t TXBUFE  : 1;
    volatile uint32_t RXBUFE  : 1;
    volatile uint32_t         : 19;
} UART_SR_bits;

// Peripheral struct for the UART peripheral
typedef struct {
    volatile UART_CR_bits UART_CR;       // (Uart Offset: 0x0000) Control
     Register
    volatile UART_MR_bits UART_MR;       // (Uart Offset: 0x0004) Mode Register
    volatile uint32_t     UART_IER;      // (Uart Offset: 0x0008) Interrupt
     Enable Register
    volatile uint32_t     UART_IDR;      // (Uart Offset: 0x000C) Interrupt
     Disable Register
    volatile uint32_t     UART_IMR;      // (Uart Offset: 0x0010) Interrupt
     Mask Register
    volatile UART_SR_bits UART_SR;       // (Uart Offset: 0x0014) Status
     Register
    volatile uint32_t     UART_RHR;      // (Uart Offset: 0x0018) Receive
     Holding Register
    volatile uint32_t     UART_THR;      // (Uart Offset: 0x001C) Transmit
     Holding Register
    volatile uint32_t     UART_BRGR;     // (Uart Offset: 0x0020) Baud Rate
     Generator Register
    volatile uint32_t     Reserved1[55];
    volatile uint32_t     UART_RPR;      // (Uart Offset: 0x100) Receive
     Pointer Register
    volatile uint32_t     UART_RCR;      // (Uart Offset: 0x104) Receive
     Counter Register
    volatile uint32_t     UART_TPR;      // (Uart Offset: 0x108) Transmit
     Pointer Register
    volatile uint32_t     UART_TCR;      // (Uart Offset: 0x10C) Transmit
     Counter Register
    volatile uint32_t     UART_RNPR;     // (Uart Offset: 0x110) Receive Next
     Pointer Register
```

```
    volatile uint32_t     UART_RNCR;     // (Uart Offset: 0x114) Receive Next
     Counter Register
    volatile uint32_t     UART_TNPR;     // (Uart Offset: 0x118) Transmit Next
     Pointer Register
    volatile uint32_t     UART_TNCR;     // (Uart Offset: 0x11C) Transmit Next
     Counter Register
    volatile uint32_t     UART_PTCR;     // (Uart Offset: 0x120) Transfer
     Control Register
    volatile uint32_t     UART_PTSR;     // (Uart Offset: 0x124) Transfer
     Status Register
} Uart;

// Pointer to a Uart-sized chunk of memory at the UART peripheral
#define UART ((Uart*) UART0_BASE)


////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// UART Definitions
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

// Values which the PAR bits in the UART_MR register can take on
#define UART_MR_PAR_EVEN  0 // Even parity
#define UART_MR_PAR_ODD   1 // Odd parity
#define UART_MR_PAR_SPACE 2 // Parity forced to 0
#define UART_MR_PAR_MARK  3 // Parity forced to 1
#define UART_MR_PAR_NO    4 // No parity

// The specific PIO pins and peripheral function which UART uses, set in
 uartInit()
#define UART_URXD0_PIN PIO_PA9
#define UART_ITXD0_PIN PIO_PA10
#define UART_FUNC      PIO_PERIPH_A

// (UART does not have write protection.)


////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// UART Functions
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

/* Enables the UART peripheral and initializes its parity and baut rate.
 *    -- parity:  A UART parity ID, e.g. UART_MR_PAR_SPACE
 *    -- CD: a 16-bit unsigned integer which determines the baud rate as
  follows:
 *       Baud Rate = MCK_FREQ/(16*CD)
 * Note that pin PA9 is used as receive and pin PA10 is used as transmit.
  pioInit() must be called
```

```
 * first. */
void uartInit(uint32_t parity, uint16_t CD) {
    pmcEnablePeriph(PMC_ID_UART0);

    pioPinMode(UART_URXD0_PIN, UART_FUNC); // Set URXD0 pin mode
    pioPinMode(UART_ITXD0_PIN, UART_FUNC); // Set ITXD0 pin mode

    UART->UART_CR.TXEN = 1; // Enable transmitter
    UART->UART_CR.RXEN = 1; // Enable receiver

    UART->UART_MR.PAR = parity; // Set parity
    UART->UART_BRGR   = CD; // Set baud rate divisor
}

/* Transmits a character (1 byte) over UART
 *    -- data: the character to send over UART */
void uartTx(char data) {
    while (!(UART->UART_SR.TXRDY)); // Wait until previous data has been
     transmitted
    UART->UART_THR = data; // Write data into holding register for transmit
}

/* Receives a character (1 byte) over UART
 *    -- return: the character received over UART */
char uartRx() {
    while (!(UART->UART_SR.RXRDY)); // Wait until data has been received
    return (char) UART->UART_RHR; // Return received data in holding register
}


#endif
```