```c
/* SAM4S4B_adc.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the ADC
 * (Analog-to-Digital Converter) peripheral of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_ADC_H
#define SAM4S4B_ADC_H

#include <stdint.h>
#include "SAM4S4B_sys.h"
#include "SAM4S4B_pio.h"

////////////////////////////////////////////////////////////////////////////////
 //////////////////////
// ADC Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 //////////////////////

#define ADC_BASE   (0x40038000U) // ADC Base Address


////////////////////////////////////////////////////////////////////////////////
 //////////////////////
// ADC Registers
////////////////////////////////////////////////////////////////////////////////
 //////////////////////

// Bit field struct for the ADC_CR register
typedef struct {
    volatile uint32_t SWRST : 1;
    volatile uint32_t START : 1;
    volatile uint32_t       : 30;
} ADC_CR_bits;

// Bit field struct for the ADC_MR register
typedef struct {
    volatile uint32_t TRGEN : 1;
    volatile uint32_t TRGSEL : 3;
    volatile uint32_t LOWRES : 1;
    volatile uint32_t SLEEP  : 1;
    volatile uint32_t FWUP   : 1;
    volatile uint32_t FREERUN : 1;
    volatile uint32_t PRESCAL : 8;
    volatile uint32_t STARTUP : 4;
    volatile uint32_t SETTLING : 2;
    volatile uint32_t          : 1;
```

```
    volatile uint32_t ANACH    : 1;
    volatile uint32_t TRACKTIM : 4;
    volatile uint32_t TRANSFER : 2;
    volatile uint32_t          : 1;
    volatile uint32_t USEQ     : 1;
} ADC_MR_bits;

// Bit field struct for the ADC_ACR register
typedef struct {
    volatile uint32_t       : 4;
    volatile uint32_t TSON  : 1;
    volatile uint32_t       : 3;
    volatile uint32_t IBCTL : 2;
    volatile uint32_t       : 22;
} ADC_ACR_bits;

// Peripheral struct for the ADC peripheral
typedef struct {
    volatile ADC_CR_bits  ADC_CR;        // (Adc Offset: 0x00) Control Register
    volatile ADC_MR_bits  ADC_MR;        // (Adc Offset: 0x04) Mode Register
    volatile uint32_t     ADC_SEQR1;     // (Adc Offset: 0x08) Channel Sequence
     Register 1
    volatile uint32_t     ADC_SEQR2;     // (Adc Offset: 0x0C) Channel Sequence
     Register 2
    volatile uint32_t     ADC_CHER;      // (Adc Offset: 0x10) Channel Enable
     Register
    volatile uint32_t     ADC_CHDR;      // (Adc Offset: 0x14) Channel Disable
     Register
    volatile uint32_t     ADC_CHSR;      // (Adc Offset: 0x18) Channel Status
     Register
    volatile uint32_t     Reserved1[1];
    volatile uint32_t     ADC_LCDR;      // (Adc Offset: 0x20) Last Converted
     Data Register
    volatile uint32_t     ADC_IER;       // (Adc Offset: 0x24) Interrupt Enable
     Register
    volatile uint32_t     ADC_IDR;       // (Adc Offset: 0x28) Interrupt
     Disable Register
    volatile uint32_t     ADC_IMR;       // (Adc Offset: 0x2C) Interrupt Mask
     Register
    volatile uint32_t     ADC_ISR;       // (Adc Offset: 0x30) Interrupt Status
     Register
    volatile uint32_t     Reserved2[2];
    volatile uint32_t     ADC_OVER;      // (Adc Offset: 0x3C) Overrun Status
     Register
    volatile uint32_t     ADC_EMR;       // (Adc Offset: 0x40) Extended Mode
     Register
    volatile uint32_t     ADC_CWR;       // (Adc Offset: 0x44) Compare Window
     Register
    volatile uint32_t     ADC_CGR;       // (Adc Offset: 0x48) Channel Gain
     Register
```

```
    volatile uint32_t     ADC_COR;       // (Adc Offset: 0x4C) Channel Offset
     Register
    volatile uint32_t     ADC_CDR[15];   // (Adc Offset: 0x50) Channel Data
     Register
    volatile uint32_t     Reserved3[2];
    volatile ADC_ACR_bits ADC_ACR;       // (Adc Offset: 0x94) Analog Control
     Register
    volatile uint32_t     Reserved4[19];
    volatile uint32_t     ADC_WPMR;      // (Adc Offset: 0xE4) Write Protect
     Mode Register
    volatile uint32_t     ADC_WPSR;      // (Adc Offset: 0xE8) Write Protect
     Status Register
    volatile uint32_t     Reserved5[5];
    volatile uint32_t     ADC_RPR;       // (Adc Offset: 0x100) Receive Pointer
     Register
    volatile uint32_t     ADC_RCR;       // (Adc Offset: 0x104) Receive Counter
     Register
    volatile uint32_t     Reserved6[2];
    volatile uint32_t     ADC_RNPR;      // (Adc Offset: 0x110) Receive Next
     Pointer Register
    volatile uint32_t     ADC_RNCR;      // (Adc Offset: 0x114) Receive Next
     Counter Register
    volatile uint32_t     Reserved7[2];
    volatile uint32_t     ADC_PTCR;      // (Adc Offset: 0x120) Transfer
     Control Register
    volatile uint32_t     ADC_PTSR;      // (Adc Offset: 0x124) Transfer Status
     Register
} Adc;

// Pointer to an Adc-sized chunk of memory at the ADC peripheral
#define ADC ((Adc*) ADC_BASE)


////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// ADC Definitions
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

// Values which "channel" can take on in several functions
#define ADC_CH0  0
#define ADC_CH1  1
#define ADC_CH2  2
#define ADC_CH3  3
#define ADC_CH4  4
#define ADC_CH5  5
#define ADC_CH6  6
#define ADC_CH7  7
#define ADC_CH8  8
#define ADC_CH9  9
#define ADC_CH15 15
```

```
// Values which the LOWRES bit can take on in the ADC_MR register
#define ADC_MR_LOWRES_BITS_12 0 // 12-bit resolution
#define ADC_MR_LOWRES_BITS_10 1 // 10-bit resolution

// Values for each channel's gain in ADC_CGR
#define ADC_CGR_GAIN_X1 0 // Unity gain
#define ADC_CGR_GAIN_X2 2 // Gain times 2
#define ADC_CGR_GAIN_X4 3 // Gain times 4

// Values for each channel's offset in ADC_COR
#define ADC_COR_OFFSET_ON  1 // Centers the analog signal on (G-1)Vrefin/2
 prior to gain
#define ADC_COR_OFFSET_OFF 0 // No offset

// The specific PIO pins and peripheral function which ADC uses, set in
 adcInit()
#define ADC_CH0_PIN PIO_PA17
#define ADC_CH1_PIN PIO_PA18
#define ADC_CH2_PIN PIO_PA19
#define ADC_CH3_PIN PIO_PA20
#define ADC_CH4_PIN PIO_PB0
#define ADC_CH5_PIN PIO_PB1
#define ADC_CH6_PIN PIO_PB2
#define ADC_CH7_PIN PIO_PB3
#define ADC_CH8_PIN PIO_PA21
#define ADC_CH9_PIN PIO_PA22
#define ADC_FUNC    PIO_PERIPH_D

// The maximum value the ADC will record (dependent on the resolution)
#define ADC_DMAX_10 1023 // 2^10 - 1
#define ADC_DMAX_12 4095 // 2^12 - 1

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define ADC_WPMR_WPKEY_PASSWD (0x414443U << 8)


////////////////////////////////////////////////////////////////////////////
 ////////////////////
// ADC Functions
////////////////////////////////////////////////////////////////////////////
 ////////////////////

/* Enables the ADC peripheral and initializes its resolution
 *    -- resolution: an ADC resolution ID, e.g. ADC_MR_LOWRES_BITS_10
 * Note: the ADC clock defaults to MCK_FREQ / 2 = 2 MHz; 1 MHz to 20 MHz is
  allowed. */
void adcInit(uint32_t resolution) {
    pmcEnablePeriph(PMC_ID_ADC);
```

```
    ADC->ADC_MR.LOWRES = resolution; // Set resolution
    ADC->ADC_MR.ANACH = 1; // Allow channels to have independent settings
}

/* Enables an ADC channel and initializes its gain and offset
 *    -- channel: an ADC channel ID, e.g. ADC_CH3
 *    -- gain: an ADC gain ID, e.g. ADC_CGR_GAIN_X2
 *    -- offset: an ADC offset ID, e.g. ADC_COR_OFFSET_ON. Set the offset to 1
  to center the analog
 *       signal on (Gain - 1)*Vref/2 prior to gain */
void adcChannelInit(int channel, int gain, int offset) {
    // Set the channel's PIO pin to perform its ADC function
    switch (channel) {
        case ADC_CH0: pioPinMode(ADC_CH0_PIN, ADC_FUNC); break;
        case ADC_CH1: pioPinMode(ADC_CH1_PIN, ADC_FUNC); break;
        case ADC_CH2: pioPinMode(ADC_CH2_PIN, ADC_FUNC); break;
        case ADC_CH3: pioPinMode(ADC_CH3_PIN, ADC_FUNC); break;
        case ADC_CH4: pioPinMode(ADC_CH4_PIN, ADC_FUNC); break;
        case ADC_CH5: pioPinMode(ADC_CH5_PIN, ADC_FUNC); break;
        case ADC_CH6: pioPinMode(ADC_CH6_PIN, ADC_FUNC); break;
        case ADC_CH7: pioPinMode(ADC_CH7_PIN, ADC_FUNC); break;
        case ADC_CH8: pioPinMode(ADC_CH8_PIN, ADC_FUNC); break;
        case ADC_CH9: pioPinMode(ADC_CH9_PIN, ADC_FUNC); break;
        case ADC_CH15:                                   break;
    }
    ADC->ADC_CHER |= (1 << channel); // Enable the ADC channel

    // Set the gain
    ADC->ADC_CGR |= (gain << (2*channel));
    ADC->ADC_CGR &= ~((~gain & 0b11) << (2*channel));

    // Set the offset
    ADC->ADC_COR |= (offset << channel);
    ADC->ADC_COR &= ~((~offset & 0b1) << channel);
}

/* Reads the analog voltage reported by an ADC channel
 *    -- channel: an ADC channel ID, e.g. ADC_CH3
 *    -- return: the analog voltage represented by the ADC's report (in V)
 * Note: it is important to measure the voltage at the ADC's Vref pin and
  record it in
 * SAM4S4B_sys.h for accurate results. */
float adcRead(int channel) {
    ADC->ADC_CR.START = 1; // Start conversion
    while (!((ADC->ADC_ISR >> channel) & 1)); // Wait for conversion
    int d = ADC->ADC_CDR[channel]; // Received digital value
    int dMax = (ADC->ADC_MR.LOWRES) ? ADC_DMAX_10 : ADC_DMAX_12; // Maximum
     possible value
    return (((float) d) / dMax) * ADC_VREF;
}
```

```
#endif
```