```
/* SAM4S4B_pwm.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the PWM
 * (Pulse Width Modulation Controller) peripheral of the SAM4S4B
  microcontroller. */

#ifndef SAM4S4B_PWM_H
#define SAM4S4B_PWM_H

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

////////////////////////////////////////////////////////////////////////////////
 ///////////////////
// PWM Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 ///////////////////

#define PWM_BASE (0x40020000U) // PWM Base Address


////////////////////////////////////////////////////////////////////////////////
 ///////////////////
// PWM Registers
////////////////////////////////////////////////////////////////////////////////
 ///////////////////

// Bit field struct for the PWM_CLK register
typedef struct {
    volatile uint32_t DIVA : 8;
    volatile uint32_t PREA : 4;
    volatile uint32_t      : 4;
    volatile uint32_t DIVB : 8;
    volatile uint32_t PREB : 4;
    volatile uint32_t      : 4;
} PWM_CLK_bits;

// Bit field struct for the PWM_CMR register
typedef struct {
    volatile uint32_t CPRE : 4;
    volatile uint32_t      : 4;
    volatile uint32_t CALG : 1;
    volatile uint32_t CPOL : 1;
    volatile uint32_t CES  : 1;
    volatile uint32_t      : 5;
```

```
    volatile uint32_t DTE  : 1;
    volatile uint32_t DTHI : 1;
    volatile uint32_t DTLI : 1;
    volatile uint32_t      : 13;
} PWM_CMR_bits;

// Channel struct for each of the PWM peripheral's 4 channels
typedef struct {
    volatile PWM_CMR_bits PWM_CMR;       // (PwmCh_num Offset: 0x0) PWM Channel
     Mode Register
    volatile uint32_t     PWM_CDTY;      // (PwmCh_num Offset: 0x4) PWM Channel
     Duty Cycle Register
    volatile uint32_t     PWM_CDTYUPD;   // (PwmCh_num Offset: 0x8) PWM Channel
     Duty Cycle Update Register
    volatile uint32_t     PWM_CPRD;      // (PwmCh_num Offset: 0xC) PWM Channel
     Period Register
    volatile uint32_t     PWM_CPRDUPD;   // (PwmCh_num Offset: 0x10) PWM
     Channel Period Update Register
    volatile uint32_t     PWM_CCNT;      // (PwmCh_num Offset: 0x14) PWM
     Channel Counter Register
    volatile uint32_t     PWM_DT;        // (PwmCh_num Offset: 0x18) PWM
     Channel Dead Time Register
    volatile uint32_t     PWM_DTUPD;     // (PwmCh_num Offset: 0x1C) PWM
     Channel Dead Time Update Register
} PwmCh;

// Channel struct for each of the PWM peripheral's 8 comparison options
typedef struct {
    volatile uint32_t PWM_CMPV;      // (PwmCmp Offset: 0x0) PWM Comparison x
     Value Register
    volatile uint32_t PWM_CMPVUPD;   // (PwmCmp Offset: 0x4) PWM Comparison x
     Value Update Register
    volatile uint32_t PWM_CMPM;      // (PwmCmp Offset: 0x8) PWM Comparison x
     Mode Register
    volatile uint32_t PWM_CMPMUPD;   // (PwmCmp Offset: 0xC) PWM Comparison x
     Mode Update Register
} PwmCmp;

#define PWM_CMP_NUMBER 8
#define PWM_CH_NUMBER 4
// Peripheral struct for the PWM peripheral
typedef struct {
    volatile PWM_CLK_bits PWM_CLK;       // (Pwm Offset: 0x00) PWM Clock
     Register
    volatile uint32_t   PWM_ENA;       // (Pwm Offset: 0x04) PWM Enable
     Register
    volatile uint32_t   PWM_DIS;       // (Pwm Offset: 0x08) PWM Disable
     Register
    volatile uint32_t   PWM_SR;        // (Pwm Offset: 0x0C) PWM Status
     Register
```

```
volatile uint32_t   PWM_IER1;      // (Pwm Offset: 0x10) PWM Interrupt
 Enable Register 1
volatile uint32_t   PWM_IDR1;      // (Pwm Offset: 0x14) PWM Interrupt
 Disable Register 1
volatile uint32_t   PWM_IMR1;      // (Pwm Offset: 0x18) PWM Interrupt Mask
 Register 1
volatile uint32_t   PWM_ISR1;      // (Pwm Offset: 0x1C) PWM Interrupt
 Status Register 1
volatile uint32_t   PWM_SCM;       // (Pwm Offset: 0x20) PWM Sync Channels
 Mode Register
volatile uint32_t   Reserved1[1];
volatile uint32_t   PWM_SCUC;      // (Pwm Offset: 0x28) PWM Sync Channels
 Update Control Register
volatile uint32_t   PWM_SCUP;      // (Pwm Offset: 0x2C) PWM Sync Channels
 Update Period Register
volatile uint32_t   PWM_SCUPUPD;   // (Pwm Offset: 0x30) PWM Sync Channels
 Update Period Update Register
volatile uint32_t   PWM_IER2;      // (Pwm Offset: 0x34) PWM Interrupt
 Enable Register 2
volatile uint32_t   PWM_IDR2;      // (Pwm Offset: 0x38) PWM Interrupt
 Disable Register 2
volatile uint32_t   PWM_IMR2;      // (Pwm Offset: 0x3C) PWM Interrupt Mask
 Register 2
volatile uint32_t   PWM_ISR2;      // (Pwm Offset: 0x40) PWM Interrupt
 Status Register 2
volatile uint32_t   PWM_OOV;       // (Pwm Offset: 0x44) PWM Output
 Override Value Register
volatile uint32_t   PWM_OS;        // (Pwm Offset: 0x48) PWM Output
 Selection Register
volatile uint32_t   PWM_OSS;       // (Pwm Offset: 0x4C) PWM Output
 Selection Set Register
volatile uint32_t   PWM_OSC;       // (Pwm Offset: 0x50) PWM Output
 Selection Clear Register
volatile uint32_t   PWM_OSSUPD;    // (Pwm Offset: 0x54) PWM Output
 Selection Set Update Register
volatile uint32_t   PWM_OSCUPD;    // (Pwm Offset: 0x58) PWM Output
 Selection Clear Update Register
volatile uint32_t   PWM_FMR;       // (Pwm Offset: 0x5C) PWM Fault Mode
 Register
volatile uint32_t   PWM_FSR;       // (Pwm Offset: 0x60) PWM Fault Status
 Register
volatile uint32_t   PWM_FCR;       // (Pwm Offset: 0x64) PWM Fault Clear
 Register
volatile uint32_t   PWM_FPV;       // (Pwm Offset: 0x68) PWM Fault
 Protection Value Register
volatile uint32_t   PWM_FPE;       // (Pwm Offset: 0x6C) PWM Fault
 Protection Enable Register
volatile uint32_t   Reserved2[3];
volatile uint32_t   PWM_ELMR[2];   // (Pwm Offset: 0x7C) PWM Event Line 0
 Mode Register
volatile uint32_t   Reserved3[11];
```

```
    volatile uint32_t   PWM_SMMR;      // (Pwm Offset: 0xB0) PWM Stepper Motor
     Mode Register
    volatile uint32_t   Reserved4[12];
    volatile uint32_t   PWM_WPCR;      // (Pwm Offset: 0xE4) PWM Write Protect
     Control Register
    volatile uint32_t   PWM_WPSR;      // (Pwm Offset: 0xE8) PWM Write Protect
     Status Register
    volatile uint32_t   Reserved5[7];
    volatile uint32_t   PWM_TPR;       // (Pwm Offset: 0x108) Transmit Pointer
     Register
    volatile uint32_t   PWM_TCR;       // (Pwm Offset: 0x10C) Transmit Counter
     Register
    volatile uint32_t   Reserved6[2];
    volatile uint32_t   PWM_TNPR;      // (Pwm Offset: 0x118) Transmit Next
     Pointer Register
    volatile uint32_t   PWM_TNCR;      // (Pwm Offset: 0x11C) Transmit Next
     Counter Register
    volatile uint32_t   PWM_PTCR;      // (Pwm Offset: 0x120) Transfer Control
     Register
    volatile uint32_t   PWM_PTSR;      // (Pwm Offset: 0x124) Transfer Status
     Register
    volatile uint32_t   Reserved7[2];
    volatile PwmCmp     PWM_CMP[PWM_CMP_NUMBER]; // (Pwm Offset: 0x130) 0 .. 7
    volatile uint32_t   Reserved8[20];
    volatile PwmCh      PWM_CH[PWM_CH_NUMBER]; // (Pwm Offset: 0x200) ch = 0 ..
     3
} Pwm;

// Pointer to a Pwm-sized chunk of memory at the PWM peripheral
#define PWM ((Pwm*) PWM_BASE)


////////////////////////////////////////////////////////////////////////////
 ////////////////////
// PWM Definitions
////////////////////////////////////////////////////////////////////////////
 ////////////////////

// Constants relating to clock speed tuning in pwmInit()
#define PWM_CLK_PRE_MAX 11
#define PWM_CLK_DIV_MAX 255

// The specific PIO pins and peripheral function which PWM uses, set in
 pwmInit()
#define PWM_CH0_PIN PIO_PA11
#define PWM_CH1_PIN PIO_PA12
#define PWM_CH2_PIN PIO_PA13
#define PWM_CH3_PIN PIO_PA14
#define PWM_FUNC    PIO_PERIPH_B

// Values which "channelID" can take on in several functions
```

```
#define PWM_CH0 0
#define PWM_CH1 1
#define PWM_CH2 2
#define PWM_CH3 3

// Values which the CPOL bit in the PWM_CMR register can take on
#define PWM_CMR_CPOL_LOW  0 // Output waveform starts at a low level
#define PWM_CMR_CPOL_HIGH 1 // Output waveform starts at a high level

// Values which the CPRE bits in the PWM_CMR register can take on
#define PWM_CMR_CPRE_MCK     0
#define PWM_CMR_CPRE_MCK2    1
#define PWM_CMR_CPRE_MCK4    2
#define PWM_CMR_CPRE_MCK8    3
#define PWM_CMR_CPRE_MCK16   4
#define PWM_CMR_CPRE_MCK32   5
#define PWM_CMR_CPRE_MCK64   6
#define PWM_CMR_CPRE_MCK128  7
#define PWM_CMR_CPRE_MCK256  8
#define PWM_CMR_CPRE_MCK512  9
#define PWM_CMR_CPRE_MCK1024 10
#define PWM_CMR_CPRE_CLKA    11
#define PWM_CMR_CPRE_CLKB    12

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define PWM_WPCR_WPKEY_PASSWD (0x50574DU << 8)


////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// PWM Functions
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

/* Enables the PWM peripheral and initializes its frequency, period, and duty
 cycle.
 * Requires pioInit().
 *    -- freq: the desired frequency of the PWM clock in Hz
 *    -- period: the desired frequency of the PWM waveform in number of clock
  periods
 *    -- dutyCycle: the desired duty cycle of the PWM waveform in number of
  waveform periods
 * Note: the actual frequency of the PWM waveform is given by freq / period,
  where
 * 0 < period < (2^16 = 65536). The higher the period,the more resolution for
  the duty cycle,
 * which is given by Duty Cycle = dutyCycle / period, where 0 < period < (2^16
  = 65536). Note that
```

```
 * 15.319 Hz <= freq <= 4 MHz based on allowable clock divisions. The alignment
  defaults to
 * left-aligned, and so is not set. Requires pioInit(). */
void pwmInit(int channelID, int freq, uint16_t period, uint16_t dutyCycle) {
    pmcEnablePeriph(PMC_ID_PWM);

    switch (channelID) {
        case PWM_CH0: pioPinMode(PWM_CH0_PIN, PWM_FUNC); break;
        case PWM_CH1: pioPinMode(PWM_CH1_PIN, PWM_FUNC); break;
        case PWM_CH2: pioPinMode(PWM_CH2_PIN, PWM_FUNC); break;
        case PWM_CH3: pioPinMode(PWM_CH3_PIN, PWM_FUNC); break;
    }

    PWM->PWM_DIS |= (1 << channelID); // Disables PWM while setting values

    uint32_t preScl[PWM_CLK_PRE_MAX] =
        {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024};
    uint32_t preSclIndex = 0;
    uint32_t linDiv;

    // Finds prescaler and linear divider values
    while (preSclIndex < PWM_CLK_PRE_MAX) {
        linDiv = MCK_FREQ / preScl[preSclIndex] / freq;
        if (linDiv <= PWM_CLK_DIV_MAX) break;
        preSclIndex++;
    }

    // Sets the clock if a configuration can be found. Otherwise, disables the
     clock.
    if (preSclIndex < PWM_CLK_PRE_MAX) {
        PWM->PWM_CLK.PREA = preSclIndex;
        PWM->PWM_CLK.DIVA = linDiv;
    } else {
        PWM->PWM_CLK.DIVA = 0;
    }

    PWM->PWM_CH[channelID].PWM_CMR.CPRE = PWM_CMR_CPRE_CLKA; // Set clock speed
    PWM->PWM_CH[channelID].PWM_CMR.CPOL = PWM_CMR_CPOL_HIGH; // Set waveform
     polarity

    PWM->PWM_CH[channelID].PWM_CPRD = period; // Set period
    PWM->PWM_CH[channelID].PWM_CDTY = dutyCycle; // Set duty cycle

    PWM->PWM_ENA |= (1 << channelID); // Enable PWM after setting values
}


#endif
```