```c
/* SAM4S4B_tc.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the TC (Timer
 * Counter) peripheral of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_TC_H
#define SAM4S4B_TC_H

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

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define TC0_BASE   (0x40010000U) // TC0 Base Address
#define TC1_BASE   (0x40014000U) // TC1 Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the TC_CCR register
typedef struct {
    volatile uint32_t CLKEN   : 1;
    volatile uint32_t CLKDIS  : 1;
    volatile uint32_t SWTRG   : 1;
    volatile uint32_t         : 29;
} TC_CCR_bits;

// Bit field struct for the TC_CMR register
typedef struct {
    volatile uint32_t TCCLKS  : 3;
    volatile uint32_t CLKI    : 1;
    volatile uint32_t BURST   : 2;
    volatile uint32_t CPCSTOP : 1;
    volatile uint32_t CPCDIS  : 1;
    volatile uint32_t EEVTEDG : 2;
    volatile uint32_t EEVT    : 2;
    volatile uint32_t ENETRG  : 1;
    volatile uint32_t WAVESEL : 2;
    volatile uint32_t WAVE    : 1;
    volatile uint32_t ACPA    : 2;
    volatile uint32_t ACPC    : 2;
    volatile uint32_t AEEVT   : 2;
    volatile uint32_t ASWTRG  : 2;
    volatile uint32_t BCPB    : 2;
    volatile uint32_t BCPC    : 2;
    volatile uint32_t BEEVT   : 2;
    volatile uint32_t BSWTRG  : 2;
} TC_CMR_bits;

// Bit field struct for the TC_SR register
typedef struct {
    volatile uint32_t COVFS   : 1;
    volatile uint32_t LOVRS   : 1;
    volatile uint32_t CPAS    : 1;
    volatile uint32_t CPBS    : 1;
    volatile uint32_t CPCS    : 1;
    volatile uint32_t LDRAS   : 1;
    volatile uint32_t LDRBS   : 1;
    volatile uint32_t ETRGS   : 1;
    volatile uint32_t         : 8;
    volatile uint32_t CLKSTA  : 1;
    volatile uint32_t MTIOA   : 1;
    volatile uint32_t MTIOB   : 1;
    volatile uint32_t         : 13;
} TC_SR_bits;

// Channel struct for each of the 3 TC channels
typedef struct {
    volatile TC_CCR_bits TC_CCR;        // (TcChannel Offset: 0x0) Channel
     Control Register
    volatile TC_CMR_bits TC_CMR;        // (TcChannel Offset: 0x4) Channel Mode
     Register
    volatile uint32_t    TC_SMMR;       // (TcChannel Offset: 0x8) Stepper
     Motor Mode Register
    volatile uint32_t    Reserved1[1];
    volatile uint32_t    TC_CV;         // (TcChannel Offset: 0x10) Counter
     Value
    volatile uint32_t    TC_RA;         // (TcChannel Offset: 0x14) Register A
    volatile uint32_t    TC_RB;         // (TcChannel Offset: 0x18) Register B
    volatile uint32_t    TC_RC;         // (TcChannel Offset: 0x1C) Register C
    volatile TC_SR_bits  TC_SR;         // (TcChannel Offset: 0x20) Status
     Register
    volatile uint32_t    TC_IER;        // (TcChannel Offset: 0x24) Interrupt
     Enable Register
    volatile uint32_t    TC_IDR;        // (TcChannel Offset: 0x28) Interrupt
     Disable Register
    volatile uint32_t    TC_IMR;        // (TcChannel Offset: 0x2C) Interrupt
     Mask Register
    volatile uint32_t    Reserved2[4];
```

```
} TcCh;

#define TC_CH_NUMBER 3 // Number of TC channels
// Peripheral struct for a TC peripheral (either TC0 or TC1)
typedef struct {
    TcCh              TC_CH[TC_CH_NUMBER]; // (Tc Offset: 0x0) channel = 0 .. 2
    volatile uint32_t TC_BCR;              // (Tc Offset: 0xC0) Block Control
     Register
    volatile uint32_t TC_BMR;              // (Tc Offset: 0xC4) Block Mode
     Register
    volatile uint32_t TC_QIER;             // (Tc Offset: 0xC8) QDEC Interrupt
     Enable Register
    volatile uint32_t TC_QIDR;             // (Tc Offset: 0xCC) QDEC Interrupt
     Disable Register
    volatile uint32_t TC_QIMR;             // (Tc Offset: 0xD0) QDEC Interrupt
     Mask Register
    volatile uint32_t TC_QISR;             // (Tc Offset: 0xD4) QDEC Interrupt
     Status Register
    volatile uint32_t TC_FMR;              // (Tc Offset: 0xD8) Fault Mode
     Register
    volatile uint32_t Reserved1[2];
    volatile uint32_t TC_WPMR;             // (Tc Offset: 0xE4) Write Protect
     Mode Register
} Tc;

// Pointers to Tc-sized chunks of memory at each TC peripheral
#define TC0 ((Tc*) TC0_BASE)
#define TC1 ((Tc*) TC1_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Clock speeds for the 5 TC clocks used in generating delays
#define TC_CLK1_SPEED (MCK_FREQ / 2)
#define TC_CLK2_SPEED (MCK_FREQ / 8)
#define TC_CLK3_SPEED (MCK_FREQ / 32)
#define TC_CLK4_SPEED (MCK_FREQ / 128)
#define TC_CLK5_SPEED 32000

// Values which TCCLKS bits can take on in TC_CMR
#define TC_CLK1_ID 0
#define TC_CLK2_ID 1
#define TC_CLK3_ID 2
#define TC_CLK4_ID 3
#define TC_CLK5_ID 4

// Arbitrary block IDs used to easily find a channel's block
```

```
#define TC_BLOCK0_ID 0
#define TC_BLOCK1_ID 1

// Values which "channelID" can take on in several functions
#define TC_CH0_ID 0
#define TC_CH1_ID 1
#define TC_CH2_ID 2
#define TC_CH3_ID 3
#define TC_CH4_ID 4
#define TC_CH5_ID 5

// Values which the WAVESEL bits can take on in TC_CMR
#define TC_MODE_UP        0 // The counter increases then resets low once it
 caps out
#define TC_MODE_UPDOWN    1 // The counter increases then decreases once it
 caps out
#define TC_MODE_UP_RC     2 // The counter increases then resets low when an RC
 match occurs
#define TC_MODE_UPDOWN_RC 3 // The counter increases then decreases when an RC
 match occurs

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define TC_WPMR_WPKEY_PASSWD (0x54494Du << 8)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Initializes the TC peripheral by enabling the Master Clock to TC0 and TC1.
 */
void tcInit() {
    pmcEnablePeriph(PMC_ID_TC0);
    pmcEnablePeriph(PMC_ID_TC1);
}

/* Returns the TC block ID that corresponds to a given channel.
 *    -- channelID: a TC channel ID, e.g. TC_CH3_ID
 *    -- return: a TC block ID, e.g. TC_BLOCK1_ID */
int tcChannelToBlock(int channelID) {
    return channelID / 3;
}

/* Returns a pointer to the given block's base address.
 *    -- block: a TC block ID, e.g. TC_BLOCK1_ID
 *    -- return: a pointer to a Tc-sized block of memory at the block "block"
  */
```

```
Tc* tcBlockToBlockBase(int block) {
    return (block ? TC1 : TC0);
}

/* Given a channel, returns a pointer to the corresponding block's base
 address.
 *    -- channelID: a TC channel ID, e.g. TC_CH3_ID
 *    -- return: a pointer to a Tc-sized block of memory at the block "block"
  */
Tc* tcChannelToBlockBase(int channelID) {
    return tcBlockToBlockBase(tcChannelToBlock(channelID));
}

/* Enables a TC channel and configures it with the desired clock and mode.
 *    -- channelID: a TC channel ID, e.g. TC_CH3_ID
 *    -- clock: a TC clock ID, e.g. TC_CLK3_ID
 *    -- mode: a TC mode ID, e.g. TC_MODE_UP_RC */
void tcChannelInit(int channelID, uint32_t clock, uint32_t mode) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_CCR.CLKEN   = 1;     // Enable clock
    block->TC_CH[chInd].TC_CMR.TCCLKS  = clock; // Set clock to desired clock
    block->TC_CH[chInd].TC_CMR.WAVE    = 1;     // Waveform mode
    block->TC_CH[chInd].TC_CMR.WAVESEL = mode;  // Set counting mode to desired
     mode
}

/* Configures TC Channel 0 to perform delays using the fastest clock and RC
 compares. */
void tcDelayInit() {
    tcChannelInit(TC_CH0_ID, TC_CLK1_ID, TC_MODE_UP_RC);
}

/* Reads the current value of the counter of a given channel.
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_ID
 *    -- return: the value (32-bit unsigned integer) in channel "channelID"'s
  counter */
uint32_t tcReadChannel(int channelID) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    return block->TC_CH[chInd].TC_CV;
}

/* Resets the counter of a given channel to zero, at which point it continues
 counting.
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_ID */
void tcResetChannel(int channelID) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_CCR.SWTRG = 1;
}
```

```
/* Sets the value of the RC compare register for a given channel, relevant to
 certain TC modes.
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_ID
 *    -- val: the value (32-bit unsigned integer) to write to the RC register
  */
void tcSetRC_compare(int channelID, uint32_t val) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_RC = val;
}

/* Checks whether an RC match has occurred since the last call to
 tcCheckRC_compare().
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_IC
 *    -- return: 1 if an RC match has occurred since the last read; 0 if it
  hasn't */
int tcCheckRC_compare(int channelID) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    return block->TC_CH[chInd].TC_SR.CPCS;
}

/* Delays the system by a specified number of microseconds
 *    -- duration: the number of microseconds to delay
 * Note: This works up to (2^16 - 1 = 65535) us. Using the fastest available
  clock, TC_CLCK1_ID,
 * we achieve a resolution of 0.5 us. Also note that the doesn't use the above
  functions to optimize
 * speed; ideally, it would be written in assembly language for further
  optimization. Requires that
 * tcDelayInit() be called previously. Has not been tested rigorously for
  accuracy. */
void tcDelayMicros(uint32_t duration) {
    Tc* block = tcChannelToBlockBase(TC_CH0_ID);
    int chInd = TC_CH0_ID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_CCR.SWTRG = 1; // Reset counter
    block->TC_CH[chInd].TC_RC = duration * (TC_CLK1_SPEED / 1e6); // Set
     compare value
    while(!(block->TC_CH[chInd].TC_SR.CPCS)); // Wait until an RC Compare has
     occurred
}

/* Delays the system by a specified number of milliseconds
 *    -- duration: the number of milliseconds to delay
 * Note: The dependence on a "for" loop makes this code less efficient than
  tcDelayMicros(), and
 * so should be avoided for durations shorter than 65 milliseconds, in which
  case tcDelayMicros()
 * is the better option. Requires that tcDelayInit() be called previously. Has
  not been tested
 * rigorously for accuracy. */
void tcDelayMillis(int duration) {
    for (int i = 0; i < duration; i++) {
        tcDelayMicros(1000);
    }
}

/* Delays the system by a specified number of seconds
 *    -- duration: the number of seconds to delay
 * Note: the dependence on nested "for" loops and function calls makes this
  code extremely
 * inefficient, and so should be avoided for durations shorter than a minute.
  Requries that
 * tcDelayInit be called previously. Has not been tested rigorously for
  accuracy. */
void tcDelaySeconds(int duration) {
    for (int i = 0; i < duration; i++) {
        tcDelayMillis(1000);
    }
}


#endif
```