```c
/*
 * intervalTimer.c
 *
 *  Created on: Sep 22, 2014
 *      Author: coltmt
 */

#include "stdio.h"
#include "intervalTimer.h"
#include "xil_io.h"
#include <stdio.h>


// Boolean return values to check valid input parameters
#define INTERVALTIMER_VALID_TIMER_TRUE 1
#define INTERVALTIMER_VALID_TIMER_FALSE 0

// Timer IDs
#define INTERVALTIMER_0_ID 0
#define INTERVALTIMER_1_ID 1
#define INTERVALTIMER_2_ID 2

// Access the cascading set of timers at these addresses
#define INTERVALTIMER_TCSR0_ADDRESS 0X00
#define INTERVALTIMER_TCSR1_ADDRESS 0X10

// All address locations used for the timer initialization
#define INTERVALTIMER_INITIALIZE 0X0

#define INTERVALTIMER_TLR0_ADDRESS 0X04
#define INTERVALTIMER_TLR1_ADDRESS 0X14

// Timer values stored at these addresses
#define INTERVALTIMER_TCR0_ADDRESS 0X08
#define INTERVALTIMER_TCR1_ADDRESS 0X18

// Bit modifying hex values for starting and stopping timer
#define INTERVALTIMER_CASC_BIT 0X0800

#define INTERVALTIMER_LOAD_BIT 0X20
#define INTERVALTIMER_LOAD_CLEAR_BIT 0X00

#define INTERVALTIMER_ENT0_BIT 0X0080
#define INTERVALTIMER_ENT0_CLEAR_BIT 0x00

// Conversion value for TCR1 value to seconds
#define INTERVALTIMER_TCR1_MOD 42

// Delay value used for test function
#define DELAY_COUNT 3

/*
 *  Private functions   ------------------------------
 */


// Starts the timer at given base address
uint32_t intervalTimer_start_timer(uint32_t baseAddress) {

    // This stores the current configuration of bits at given address
    uint32_t tempBit = Xil_In32( baseAddress + INTERVALTIMER_TCSR0_ADDRESS);

    // This modifies the ENT0 bit without modifying the rest
    tempBit = tempBit | INTERVALTIMER_ENT0_BIT;
    Xil_Out32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS , tempBit);

    return 0;
}

// Stops the timer at given base address
uint32_t intervalTimer_stop_timer(uint32_t baseAddress) {

    // Clears the bit at the ENT0 address to stop the timer (doesn't reset)
```

```
    Xil_Out32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS , INTERVALTIMER_ENT0_CLEAR_BIT);
    return 0;
}

// Resets the specific timer using the baseAddress and timerNumber passes in
uint32_t intervalTimer_reset_timer(uint32_t baseAddress, uint32_t timerNumber) {

    // Write a 0 into the TLR0 and TLR1 registers
    Xil_Out32(baseAddress + INTERVALTIMER_TLR0_ADDRESS , INTERVALTIMER_INITIALIZE);
    Xil_Out32(baseAddress + INTERVALTIMER_TLR1_ADDRESS , INTERVALTIMER_INITIALIZE);

    // Write a 1 into the LOAD0 and LOAD1 bits of the TCSR0 and TCSR1 register

    uint32_t startZeroA = Xil_In32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS);
    uint32_t startZeroB = Xil_In32(baseAddress + INTERVALTIMER_TCSR1_ADDRESS);

    startZeroA = startZeroA | INTERVALTIMER_LOAD_BIT;
    startZeroB = startZeroB | INTERVALTIMER_LOAD_BIT;

    Xil_Out32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS , startZeroA);
    Xil_Out32(baseAddress + INTERVALTIMER_TCSR1_ADDRESS , startZeroB);

    // Write a 0 back into the LOAD0 and LOAD1 registers so they stop loading 0 consistently

    uint32_t startOneA = Xil_In32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS);
    uint32_t startOneB = Xil_In32(baseAddress + INTERVALTIMER_TCSR1_ADDRESS);

    startOneA = startOneA | INTERVALTIMER_LOAD_CLEAR_BIT;
    startOneB = startZeroB | INTERVALTIMER_LOAD_CLEAR_BIT;

    Xil_Out32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS , startOneA);
    Xil_Out32(baseAddress + INTERVALTIMER_TCSR1_ADDRESS , startOneB);

    intervalTimer_init(timerNumber);

    return 0;


}

// Inits the timer by writing to TCSR0, TCSR1 and CASC bit
uint32_t intervalTimer_init_timer(uint32_t baseAddress) {

    // Write a 0 to the TCSR0 bit
    Xil_Out32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS , INTERVALTIMER_INITIALIZE);

    // Write a 0 to the TCSR1 bit
    Xil_Out32(baseAddress + INTERVALTIMER_TCSR1_ADDRESS , INTERVALTIMER_INITIALIZE);

    // Set CASC bit to 1 in the TCSR0 register
    Xil_Out32(baseAddress + INTERVALTIMER_TCSR0_ADDRESS , INTERVALTIMER_CASC_BIT);

    //Clear UDT0 bit in the TCSR0 register is done by default when 0 is written

    return 0;
}

// Reads the address given and returns the value
u32 readRegister(uint32_t registerOffset) {
    uint32_t address = registerOffset;
    return Xil_In32(address);
}


// Reads the address given for the given timer number and returns the value
u32 readTimerRegister(uint32_t registerOffset , uint32_t timerNumber) {

    uint32_t address;

    // Verifies that timerNumber is valid
    switch (timerNumber) {

    case 0:
```

```
        address = XPAR_AXI_TIMER_0_BASEADDR + registerOffset;
        return Xil_In32(address);
        break;

    case 1:

        address = XPAR_AXI_TIMER_1_BASEADDR + registerOffset;
        return Xil_In32(address);
        break;

    case 2:

        address = XPAR_AXI_TIMER_2_BASEADDR + registerOffset;
        return Xil_In32(address);
        break;

    /*
     *  If the parameter timerNumber fails, it returns the value of timer 0
     *  by default.
     */

    default:

        printf("error; invalid timer access \n\r");             //error message
        return  Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + registerOffset);
        break;
    }
}

// Test function delay; ONLY CALLED IN: intervalTimer_runTest function
void waitALongTime() {
    volatile int32_t a = 0;
    int32_t i, j;
    for (i=0; i<DELAY_COUNT; i++)
        for (j=0; j<INT32_MAX; j++)
            a++;
}

/*
 *  Public functions    -------------------------------
 */

uint32_t intervalTimer_start(uint32_t timerNumber) {

        // Switch statement verifies that timerNumber is valid
    switch (timerNumber) {

    case 0:

        intervalTimer_start_timer(XPAR_AXI_TIMER_0_BASEADDR);   //starts timer 0
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 1:

        intervalTimer_start_timer(XPAR_AXI_TIMER_1_BASEADDR);   //starts timer 1
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 2:

        intervalTimer_start_timer(XPAR_AXI_TIMER_2_BASEADDR);   //starts timer 2
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    default:

        printf("error; invalid timer access \n\r");             //error message
        return INTERVALTIMER_VALID_TIMER_FALSE;
        break;
    }
```

```
    return 0;
}
uint32_t intervalTimer_stop(uint32_t timerNumber){


    // Switch statement verifies that the timerNumber is valid
    switch (timerNumber) {

    case 0:

        intervalTimer_stop_timer(XPAR_AXI_TIMER_0_BASEADDR);    //stops timer 0
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 1:

        intervalTimer_stop_timer(XPAR_AXI_TIMER_1_BASEADDR);    //stops timer 1
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 2:

        intervalTimer_stop_timer(XPAR_AXI_TIMER_2_BASEADDR);    //stops timer 2
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    default:

        printf("error; invalid timer access \n\r");             //error message
        return INTERVALTIMER_VALID_TIMER_FALSE;
        break;
    }

return 0;
}
uint32_t intervalTimer_reset(uint32_t timerNumber) {

    // Switch statement verifies that the timerNumber is valid
    switch (timerNumber) {

    case 0:

        intervalTimer_reset_timer(XPAR_AXI_TIMER_0_BASEADDR , INTERVALTIMER_0_ID);  //resets timer 0
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 1:

        intervalTimer_reset_timer(XPAR_AXI_TIMER_1_BASEADDR ,INTERVALTIMER_1_ID);   //resets timer 1
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 2:

        intervalTimer_reset_timer(XPAR_AXI_TIMER_2_BASEADDR , INTERVALTIMER_2_ID);  //resets timer 2
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    default:

        printf("error; invalid timer access \n\r");                                 //error message
        return INTERVALTIMER_VALID_TIMER_FALSE;
        break;
    }

    return 0;
}
uint32_t intervalTimer_init(uint32_t timerNumber) {

    // Switch statement verifies that the timerNumber is valid
    switch (timerNumber) {

    case 0:
```

```
        intervalTimer_init_timer(XPAR_AXI_TIMER_0_BASEADDR);    //initializes timer 0
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 1:

        intervalTimer_init_timer(XPAR_AXI_TIMER_1_BASEADDR);    //initializes timer 1
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 2:

        intervalTimer_init_timer(XPAR_AXI_TIMER_2_BASEADDR);    //initializes timer 2
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    default:

        printf("error; invalid timer access \n\r");             //error message
        return INTERVALTIMER_VALID_TIMER_FALSE;
        break;

    }

    return 0;
}



uint32_t intervalTimer_initAll() {

    // Returns 1 if timer0 and timer1 and timer2 are all initialized.  Returns 0 if one isn't initialized
    return intervalTimer_init(INTERVALTIMER_2_ID) & intervalTimer_init(INTERVALTIMER_1_ID) &
intervalTimer_init(INTERVALTIMER_0_ID);
}



uint32_t intervalTimer_resetAll() {

    // Returns 1 if timer0 and timer1 and timer2 are all reset.  Returns 0 if one isn't reset
    return intervalTimer_reset(INTERVALTIMER_2_ID) & intervalTimer_reset(INTERVALTIMER_1_ID) &
intervalTimer_reset(INTERVALTIMER_0_ID);
}



uint32_t intervalTimer_testAll() {

    // Returns 1 if timer0 and timer1 and timer2 pass test.  Returns 0 if one fails test

    if(intervalTimer_runTest(INTERVALTIMER_0_ID) == INTERVALTIMER_VALID_TIMER_FALSE) {
        printf("timer_0 failed");
        return INTERVALTIMER_VALID_TIMER_FALSE;
    }

    if(intervalTimer_runTest(INTERVALTIMER_1_ID) == INTERVALTIMER_VALID_TIMER_FALSE) {
        printf("timer_1 failed");
        return INTERVALTIMER_VALID_TIMER_FALSE;
    }

    if(intervalTimer_runTest(INTERVALTIMER_2_ID) == INTERVALTIMER_VALID_TIMER_FALSE) {
        printf("timer_2 failed");
        return INTERVALTIMER_VALID_TIMER_FALSE;
    }

    return INTERVALTIMER_VALID_TIMER_TRUE;
}



uint32_t intervalTimer_runTest(uint32_t timerNumber) {
```

```
      intervalTimer_init(timerNumber);
      intervalTimer_reset(timerNumber);
      // Show that the timer is reset.
      printf("timer %ld TCR0 should be 0 at this point:%ld\n\r", timerNumber ,
readTimerRegister(INTERVALTIMER_TCR0_ADDRESS, timerNumber));
      printf("timer %ld TCR1 should be 0 at this point:%ld\n\r", timerNumber ,
readTimerRegister(INTERVALTIMER_TCR0_ADDRESS, timerNumber));

      // Statements returns false if TCR0 and TCR1 aren't 0
      if (readTimerRegister(INTERVALTIMER_TCR0_ADDRESS , timerNumber) !=INTERVALTIMER_INITIALIZE) {
          return INTERVALTIMER_VALID_TIMER_FALSE;
      }

      if (readTimerRegister(INTERVALTIMER_TCR1_ADDRESS , timerNumber) !=INTERVALTIMER_INITIALIZE) {
              return INTERVALTIMER_VALID_TIMER_FALSE;
          }

      intervalTimer_start(timerNumber);
      // Show that the timer is running.

      // Test points for comparison
      u32 TP_One;
      u32 TP_Two;

      printf("The following register values should be changing while reading them.\n\r");
      printf("timer %ld TCR0 should be changing at this point:%ld\n\r", timerNumber ,
readTimerRegister(INTERVALTIMER_TCR0_ADDRESS, timerNumber));
      TP_One = readTimerRegister(INTERVALTIMER_TCR0_ADDRESS , timerNumber);
      printf("timer %ld TCR0 should be changing at this point:%ld\n\r", timerNumber ,
readTimerRegister(INTERVALTIMER_TCR0_ADDRESS, timerNumber));
      printf("timer %ld TCR0 should be changing at this point:%ld\n\r", timerNumber ,
readTimerRegister(INTERVALTIMER_TCR0_ADDRESS, timerNumber));
      printf("timer %ld TCR0 should be changing at this point:%ld\n\r", timerNumber ,
readTimerRegister(INTERVALTIMER_TCR0_ADDRESS, timerNumber));
      TP_Two = readTimerRegister(INTERVALTIMER_TCR0_ADDRESS , timerNumber);
      printf("timer %ld TCR0 should be changing at this point:%ld\n\r", timerNumber ,
readTimerRegister(INTERVALTIMER_TCR0_ADDRESS, timerNumber));

      // Statement returns false if TCR0 isn't changing
      if (TP_One == TP_Two) {
          return INTERVALTIMER_VALID_TIMER_FALSE;
      }

      // Wait about 2 minutes so that you roll over to TCR1.
      // If you don't see a '1' in TCR1 after this long wait you probably haven't programmed the timer correctly.
      waitALongTime();
      printf("timer %ld TCR0 value after wait:%lx\n\r", timerNumber , readTimerRegister(INTERVALTIMER_TCR0_ADDRESS,
timerNumber));
      printf("timer %ld TCR1 should have changed at this point:%ld\n\r", timerNumber
,readTimerRegister(INTERVALTIMER_TCR1_ADDRESS, timerNumber));

      // Statement returns false if TCR1 didn't change
      if (readTimerRegister(INTERVALTIMER_TCR1_ADDRESS,timerNumber) == INTERVALTIMER_INITIALIZE) {
          return INTERVALTIMER_VALID_TIMER_FALSE;
      }

    return INTERVALTIMER_VALID_TIMER_TRUE;
}



uint32_t intervalTimer_getTotalDurationInSeconds(uint32_t timerNumber, double *seconds) {

    double returnTime;  // Modifies the *seconds parameter

    // Accesses the given timerNumber
    switch (timerNumber) {

    case 0:

        // Sets returnTime to current value of the TCR0 address for the time in seconds
        returnTime = (readRegister(XPAR_AXI_TIMER_0_BASEADDR + INTERVALTIMER_TCR0_ADDRESS) / (double)
```

```
XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ);

        // Adds the TCR1 address to the returnTime value in seconds
        returnTime = returnTime + (readRegister(XPAR_AXI_TIMER_0_BASEADDR + INTERVALTIMER_TCR1_ADDRESS) *
INTERVALTIMER_TCR1_MOD);
        *seconds =  returnTime ;
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 1:

        // Sets returnTime to current value of the TCR0 address for the time in seconds
        returnTime = (readRegister(XPAR_AXI_TIMER_1_BASEADDR + INTERVALTIMER_TCR0_ADDRESS) / (double)
XPAR_AXI_TIMER_1_CLOCK_FREQ_HZ);

        // Adds the TCR1 address to the returnTime value in seconds
        returnTime = returnTime + (readRegister(XPAR_AXI_TIMER_1_BASEADDR + INTERVALTIMER_TCR1_ADDRESS) *
INTERVALTIMER_TCR1_MOD);

        *seconds =  returnTime ;
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    case 2:

        // Sets returnTime to current value of the TCR0 address for the time in seconds
        returnTime = (readRegister(XPAR_AXI_TIMER_2_BASEADDR + INTERVALTIMER_TCR0_ADDRESS) / (double)
XPAR_AXI_TIMER_1_CLOCK_FREQ_HZ);

        // Adds the TCR1 address to the returnTime value in seconds
        returnTime = returnTime + (readRegister(XPAR_AXI_TIMER_2_BASEADDR + INTERVALTIMER_TCR1_ADDRESS) *
INTERVALTIMER_TCR1_MOD);

        *seconds =  returnTime ;
        return INTERVALTIMER_VALID_TIMER_TRUE;
        break;

    default:
        printf("error; invalid timer access \n\r");
        return INTERVALTIMER_VALID_TIMER_FALSE;
        break;

    }


    return 0;
}
```