```c
/*
 * intervalTimer.c
 *
 *  Created on: May 12, 2015
 *      Author: Taylor Cowley
 */

#include "intervalTimer.h"          //my file
#include "xparameters.h"            //Addresses of timers.
#include <stdint.h>                 //weird types.
#include <stdio.h>                  //print to the screen
#include "xil_io.h"                 //read and write the registers

/** For the selected timer, sets the ENT0 bit in TCSR0; starting the timer */
uint32_t intervalTimer_start(uint32_t timerNumber){
    //sets the ENT0 (go) bit in the status register TCSR0
    set_bit_in_address(get_timer_base_address(timerNumber) + TCSR0, ENT0);

    return 1;       //always assume the function succeeds
}

/** For the selected timer, clears the ENT0 bit in TCSR0; stopping the timer */
uint32_t intervalTimer_stop(uint32_t timerNumber){
    //clears the ENT0 bit in the status register TCSR0
    clear_bit_in_address(get_timer_base_address(timerNumber) + TCSR0, ENT0);

    return 1;       //always assume the function succeeds
}

/**This resets the selected timer. It puts 0 in the number register, loads it into the timer,
 * then resets the timer */
uint32_t intervalTimer_reset(uint32_t timerNumber){
    Xil_Out32(get_timer_base_address(timerNumber) + TLR0, 0x0000);          //0 in TLR0
    set_bit_in_address(get_timer_base_address(timerNumber) + TCSR0, LOAD0); //Set Load0 bit in
TCSR0
    Xil_Out32(get_timer_base_address(timerNumber) + TLR1, 0x0000);          //0 in TLR1
    set_bit_in_address(get_timer_base_address(timerNumber) + TCSR1, LOAD1); //Set Load1 bit in
TCSR1
    intervalTimer_init(timerNumber);                                    //Init the counter
    return 1;                           //always say that we succeeded
}

/** inits the selected timer
 * -Writes 0 to TCSR0 (clear status)
 * -Writes 0 to TCSR1 (clear status)
 * -Sets the Cascade bit in TCSR0 so they act as one cascaded timer
 *  */
uint32_t intervalTimer_init(uint32_t timerNumber){
    Xil_Out32(get_timer_base_address(timerNumber) + TCSR0, 0x0000);      //0 in TCSR0
    Xil_Out32(get_timer_base_address(timerNumber) + TCSR1, 0x0000);      //0 in TCSR1
    set_bit_in_address(get_timer_base_address(timerNumber) + TCSR0, CASC); //set the cascade bit
    return 1;                           //always say that we succeeded
}

/** inits all the timers. Simply calls intervalTimer_init for each timer */
uint32_t intervalTimer_initAll(){
    intervalTimer_init(XPAR_AXI_TIMER_2_DEVICE_ID);     //inits timer 2
```

```
    intervalTimer_init(XPAR_AXI_TIMER_1_DEVICE_ID);     //inits timer 1
    intervalTimer_init(XPAR_AXI_TIMER_0_DEVICE_ID);     //inits timer 0
    return 1;                           //and always a success.
}

/** resets all the timers. Simply calls intervalTimer_reset for each timer */
uint32_t intervalTimer_resetAll(){
    intervalTimer_reset(XPAR_AXI_TIMER_2_DEVICE_ID);    //resets timer 2
    intervalTimer_reset(XPAR_AXI_TIMER_1_DEVICE_ID);    //resets timer 1
    intervalTimer_reset(XPAR_AXI_TIMER_0_DEVICE_ID);    //resets timer 0
    return 1;                           //and always a success.
}

/** For the selected timer, this gets the total number the timer counted to
 *  and converts it to seconds. Possible problem - the rollover case is not accounted for */
uint32_t intervalTimer_getTotalDurationInSeconds(uint32_t timerNumber, double *seconds){
    uint64_t total = 0;                 //init our count

    //get the most significant bits and put them in total
    total = Xil_In32(get_timer_base_address(timerNumber) + TCR1);

    //move the most significant bits to where they belong; the most significant places
    total = total << LEFT_SHIFT_32;         //total now looks like 0x####0000

    //add on the least significant bits
    total = total + Xil_In32(get_timer_base_address(timerNumber) + TCR0);

    double result = (double) total;         //gotta convert to double!
    result = result / get_timer_frequency(timerNumber); //divide by the frequency to get seconds

    *seconds = result;
    return 1;
}

/** This has the potential to set many bits; as many bits as are 1 in bit
 * ONLY SETS THE BITS THAT ARE 1 */
uint32_t set_bit_in_address(uint32_t address, uint32_t bit){
    uint32_t value = Xil_In32(address);     //get the old value from that address
    value = value | bit;                    //mask the new bit (1) onto the old value
    Xil_Out32(address, value);              //write the new value to the address
    return 1;                               //always assume this function works.
}

/** This has the potential to clear many bits; as many bits as are 1 in bit
 * ONLY CLEARS THE BITS THAT ARE 1 */
uint32_t clear_bit_in_address(uint32_t address, uint32_t bit){
    uint32_t value = Xil_In32(address);     //get the old value from that address
    value = value & ~bit;                   //mask the new bit (0) onto the old value
    Xil_Out32(address, value);              //write the new value to the address
    return 1;                               //always assume this function works.
}

/** This receives a timer id and returns that timer's base address */
uint32_t get_timer_base_address(uint32_t timerNumber) {
    switch(timerNumber) {
    case XPAR_AXI_TIMER_2_DEVICE_ID:        //timer 2
        return XPAR_AXI_TIMER_2_BASEADDR;   //return the base address
```

```
    case XPAR_AXI_TIMER_1_DEVICE_ID:        //timer 1
        return XPAR_AXI_TIMER_1_BASEADDR;   //return the base address
    case XPAR_AXI_TIMER_0_DEVICE_ID:    //timer 0
        return XPAR_AXI_TIMER_0_BASEADDR;   //return the base address
    default:        //There has been a terrible error
        printf("Trying to retrieve address of nonexistent timer\n\r"); //Print the error
        return 0;                       //ERROR - return 0
    }
}

/** This receives a timer id and returns that timer's frequency */
uint32_t get_timer_frequency(uint32_t timerNumber) {
    switch(timerNumber) {
        case XPAR_AXI_TIMER_2_DEVICE_ID:        //timer 2
            return XPAR_AXI_TIMER_2_CLOCK_FREQ_HZ;  //return the frequency
        case XPAR_AXI_TIMER_1_DEVICE_ID:        //timer 1
            return XPAR_AXI_TIMER_1_CLOCK_FREQ_HZ;  //return the frequency
        case XPAR_AXI_TIMER_0_DEVICE_ID:    //timer 0
            return XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ;  //return the frequency
        default:        //There has been a terrible error
            printf("Trying to retrieve frequency of nonexistent timer\n\r"); //Print the error
            return 0;                       //ERROR - return 0
        }
}

/** This function tests a timer. There are several printf statements for you to read */
uint32_t intervalTimer_runTest(uint32_t timerNumber){
    double seconds = 0.0;
    printf("Testing timer %d", timerNumber);

    intervalTimer_init(timerNumber);
      intervalTimer_reset(timerNumber);
      // Show that the timer is reset.
      // Loads the least significant bits of the timer and prints them out
      printf("timer_%d TCR0 should be 0 at this point:%ld\n\r", timerNumber,
              Xil_In32(get_timer_base_address(timerNumber)+TCR0));
      //loads the most significant bits of the timer and prints them out
      printf("timer_%d TCR1 should be 0 at this point:%ld\n\r", timerNumber,
              Xil_In32(get_timer_base_address(timerNumber)+TCR1));

      intervalTimer_start(timerNumber);     //starts the timer

      // Show that the timer is running.
      // The following statements just look at the least significant bits to see that they change
      printf("The following register values should be changing while reading them.\n\r");
      printf("timer_%d TCR0 should be changing at this point:%ld\n\r", timerNumber,
              Xil_In32(get_timer_base_address(timerNumber)+TCR0));
      printf("timer_%d TCR0 should be changing at this point:%ld\n\r", timerNumber,
              Xil_In32(get_timer_base_address(timerNumber)+TCR0));
      printf("timer_%d TCR0 should be changing at this point:%ld\n\r", timerNumber,
              Xil_In32(get_timer_base_address(timerNumber)+TCR0));
      printf("timer_%d TCR0 should be changing at this point:%ld\n\r", timerNumber,
              Xil_In32(get_timer_base_address(timerNumber)+TCR0));
      printf("timer_%d TCR0 should be changing at this point:%ld\n\r", timerNumber,
              Xil_In32(get_timer_base_address(timerNumber)+TCR0));

      //calculate time in seconds
```

```
	intervalTimer_getTotalDurationInSeconds(timerNumber, &seconds);
	//this is to time the delay.
	printf("timer time before delay is %f\n\r", seconds);

	//this waits a long time
	volatile int32_t a = 0;           //all
	  int32_t i, j;                   //we do
	  for (i=0; i<DELAY_COUNT; i++)   //is
	      for (j=0; j<INT32_MAX; j++) //count
	          a++;                    //many times

	  //now to make sure that TCR1 has changed; ie the cascade works properly.
	printf("timer_%d TCR0 value after wait:%lx\n\r", timerNumber,
	        Xil_In32(get_timer_base_address(timerNumber)+TCR0));
	printf("timer_%d TCR1 should have changed at this point:%ld\n\r", timerNumber,
	        Xil_In32(get_timer_base_address(timerNumber)+TCR0));

	intervalTimer_getTotalDurationInSeconds(timerNumber, &seconds);
	printf("total timer time was %f\n\r", seconds);

	return 1;     //return that we were successful
}

/** This function tests all the timers. It calls intervalTimer_runTest for each one */
uint32_t intervalTimer_testAll(){
    intervalTimer_runTest(XPAR_AXI_TIMER_2_DEVICE_ID);      //test timer 2
    intervalTimer_runTest(XPAR_AXI_TIMER_1_DEVICE_ID);      //test timer 1
    intervalTimer_runTest(XPAR_AXI_TIMER_0_DEVICE_ID);      //test timer 0
    return 1;
}
```