```c
/*
 * clockMain.c
 *
 *  Created on: May 21, 2015
 *      Author: Taylor Cowley
 */


#include <stdio.h>
#include "supportFiles/leds.h"
#include "supportFiles/globalTimer.h"
#include "supportFiles/interrupts.h"
#include <stdbool.h>
#include <stdint.h>
#include "clockControl.h"
#include "clockDisplay.h"
#include "supportFiles/display.h"

#include "xparameters.h"

#define TOTAL_SECONDS 60
// The formula for computing the load value is based upon the formula from 4.1.1 (calculating
timer intervals)
// in the Cortex-A9 MPCore Technical Reference Manual 4-2.
// Assuming that the prescaler = 0, the formula for computing the load value based upon the
desired period is:
// load-value = (period * timer-clock) - 1
#define TIMER_PERIOD 1.0E-2                  ////this is a good timer period
#define TIMER_CLOCK_FREQUENCY (XPAR_CPU_CORTEXA9_0_CPU_CLK_FREQ_HZ / 2)
#define TIMER_LOAD_VALUE ((TIMER_PERIOD * TIMER_CLOCK_FREQUENCY) - 1.0)

int main()
{
    // Initialize the GPIO LED driver and print out an error message if it fails (argument =
true).
       // You need to init the LEDs so that LD4 can function as a heartbeat.
    leds_init(true);
    // Init all interrupts (but does not enable the interrupts at the devices).
    // Prints an error message if an internal failure occurs because the argument = true.
    interrupts_initAll(true);
    interrupts_setPrivateTimerLoadValue(TIMER_LOAD_VALUE);
    u32 privateTimerTicksPerSecond = interrupts_getPrivateTimerTicksPerSecond();
    printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);
    // Allow the timer to generate interrupts.
    interrupts_enableTimerGlobalInts();
    // Initialization of the clock display is not time-dependent, do it outside of the state
machine.
    clockDisplay_init();                //woo! display init!
    clockControl_tick();                //tick it once to get past the init tick.
    // Keep track of your personal interrupt count. Want to make sure that you don't miss any
interrupts.
     int32_t personalInterruptCount = 0;
    // Start the private ARM timer running.
    interrupts_startArmPrivateTimer();
    // Enable interrupts at the ARM.
    interrupts_enableArmInts();
    // interrupts_isrInvocationCount() returns the number of times that the timer ISR was
```

```
invoked.
    // This value is maintained by the timer ISR. Compare this number with your own local
    // interrupt count to determine if you have missed any interrupts.
     while (interrupts_isrInvocationCount() < (TOTAL_SECONDS * privateTimerTicksPerSecond)) {
      if (interrupts_isrFlagGlobal) {  // This is a global flag that is set by the timer
interrupt handler.
          // Count ticks.
        personalInterruptCount++;
        clockControl_tick();
          interrupts_isrFlagGlobal = 0;
      }
   }
   interrupts_disableArmInts();
   printf("isr invocation count: %ld\n\r", interrupts_isrInvocationCount());
   printf("internal interrupt count: %ld\n\r", personalInterruptCount);
   return 0;
}
```