```
 2  * clockMain.c
 7
 8 /***********************************
 9 ************ Flag Method ***********
10 ***********************************/
11 #include <stdio.h>
12 #include "supportFiles/leds.h"
13 #include "supportFiles/globalTimer.h"
14 #include "supportFiles/interrupts.h"
15 #include "supportFiles/utils.h"
16 #include <stdbool.h>
17 #include <stdint.h>
18 #include "clockControl.h"
19 #include "clockDisplay.h"
20 #include "supportFiles/display.h"
21
22 #include "xparameters.h"
23
24 #define TOTAL_SECONDS 60
25 // The formula for computing the load value is based upon the formula from 4.1.1
   (calculating timer intervals)
26 // in the Cortex-A9 MPCore Technical Reference Manual 4-2.
27 // Assuming that the prescaler = 0, the formula for computing the load value based
   upon the desired period is:
28 // load-value = (period * timer-clock) - 1
29 #define TIMER_PERIOD 91.0E-3 // You can change this value to a value that you
   select.10.0E-3
30 #define TIMER_CLOCK_FREQUENCY (XPAR_CPU_CORTEXA9_0_CPU_CLK_FREQ_HZ / 2)
31 #define TIMER_LOAD_VALUE ((TIMER_PERIOD * TIMER_CLOCK_FREQUENCY) - 1.0)
32
33 /*
34 int main() {
35         clockDisplay_init();
36         clockControl_init();
37     while (1) {
38         clockControl_tick();
39         utils_msDelay(100);
40     }
41 }
42 */
43 /*
44 int main()
45 {
46     // Initialize the GPIO LED driver and print out an error message if it fails
   (argument = true).
47        // You need to init the LEDs so that LD4 can function as a heartbeat.
48     leds_init(true);
49     // Init all interrupts (but does not enable the interrupts at the devices).
50     // Prints an error message if an internal failure occurs because the argument =
   true.
51     interrupts_initAll(true);
52     interrupts_setPrivateTimerLoadValue(TIMER_LOAD_VALUE);
53     u32 privateTimerTicksPerSecond = interrupts_getPrivateTimerTicksPerSecond();
54     printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);
55     // Allow the timer to generate interrupts.
56     interrupts_enableTimerGlobalInts();
57     // Initialization of the clock display is not time-dependent, do it outside of the
   state machine.
```

```
58    clockDisplay_init();
59    // Keep track of your personal interrupt count. Want to make sure that you don't
  miss any interrupts.
60     int32_t personalInterruptCount = 0;
61    // Start the private ARM timer running.
62    interrupts_startArmPrivateTimer();
63    // Enable interrupts at the ARM.
64    interrupts_enableArmInts();
65    // interrupts_isrInvocationCount() returns the number of times that the timer ISR
  was invoked.
66    // This value is maintained by the timer ISR. Compare this number with your own
  local
67    // interrupt count to determine if you have missed any interrupts.
68     while (interrupts_isrInvocationCount() < (TOTAL_SECONDS *
  privateTimerTicksPerSecond)) {
69      if (interrupts_isrFlagGlobal) {  // This is a global flag that is set by the
  timer interrupt handler.
70          // Count ticks.
71        personalInterruptCount++;
72        clockControl_tick();
73          interrupts_isrFlagGlobal = 0;
74      }
75   }
76   interrupts_disableArmInts();
77   printf("isr invocation count: %ld\n\r", interrupts_isrInvocationCount());
78   printf("internal interrupt count: %ld\n\r", personalInterruptCount);
79   return 0;
80 }
81 ///
82
83 void isr_function() {
84
85  // Leave blank for the flag method.
86  // I already set the flag for you in another routine.
87 }
88 */
89
90
91 static uint32_t isr_functionCallCount = 0;
92
93 int main()
94 {
95    // Initialize the GPIO LED driver and print out an error message if it fails
  (argument = true).
96    // You need to init the LEDs so that LD4 can function as a heartbeat.
97    leds_init(true);
98    // Init all interrupts (but does not enable the interrupts at the devices).
99    // Prints an error message if an internal failure occurs because the argument =
  true.
100    interrupts_initAll(true);
101    interrupts_setPrivateTimerLoadValue(TIMER_LOAD_VALUE);
102    printf("timer load value:%ld\n\r", (int32_t) TIMER_LOAD_VALUE);
103    u32 privateTimerTicksPerSecond = interrupts_getPrivateTimerTicksPerSecond();
104    printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);
105    interrupts_enableTimerGlobalInts();
106    // Initialization of the clock display is not time-dependent, do it outside of the
  state machine.
107    clockDisplay_init();
```

```
108     // Start the private ARM timer running.
109     interrupts_startArmPrivateTimer();
110     // Enable interrupts at the ARM.
111     interrupts_enableArmInts();
112     // The while-loop just waits until the total number of timer ticks have occurred
    before proceeding.
113     while (interrupts_isrInvocationCount() < (TOTAL_SECONDS *
    privateTimerTicksPerSecond));
114     // All done, now disable interrupts and print out the interrupt counts.
115     interrupts_disableArmInts();
116     printf("isr invocation count: %ld\n\r", interrupts_isrInvocationCount());
117     printf("internal interrupt count: %ld\n\r", isr_functionCallCount);
118     return 0;
119 }
120
121 // The clockControl_tick() function is now called directly by the timer interrupt
    service routine.
122 void isr_function() {
123     clockControl_tick();
124    isr_functionCallCount++;
125     // Add the necessary code here.
126 }
127
128
```