```c
 * intervalTimer.c

#include "intervalTimer.h"
#include "xparameters.h"          //needed to access base address of AXIs
#include "xil_io.h"               //includes the low-level Xilinx functions needed for reading and writing to GPIOs
#include "stdio.h"                //needed to make printf work


#define TCSR_0_OFFSET 0x00        //added to base address to access TCSR0
#define TCSR_1_OFFSET 0x10        //added to base address to access TCSR1
#define TLR_0_OFFSET 0x04         //added to base address to access TLR0
#define TLR_1_OFFSET 0x14         //added to base address to access TLR0
#define TCR_0_OFFSET 0x08         //added to base address to access TCR0
#define TCR_1_OFFSET 0x18         //added to base address to access TCR1

#define NUMBER_OF_TIMERS 3
#define TCSR_INIT_BIT 0           //we always initialize the TCSRs with 0
#define CASCADE_BIT 0x00000800    //the cascade bit is bit 11 in the register
#define RESET_BIT 0               //what we pass in when resetting TLR registers
#define LOAD_BIT 0x00000020       //what we pass in when resetting TCSR
#define ENT0_BIT 0x00000080       //what we pass in when starting the timer
#define ENT0_MASK 0xffffff7f      //used to mask ENT0 before passing it in to stop the timer
#define SHIFT_VAL 32              //must shift counter1 in order to properly concatenate

#define RESET_ERROR "Reset error\n\r"

//helper function to read registers
int32_t timers_readRegister(int32_t baseAddr, int32_t offset){
    return Xil_In32(baseAddr + offset); //using low-level Xilinx call
}

//helper function to write to registers
void timers_writeRegister(int32_t baseAddr, int32_t offset, int32_t value){
    Xil_Out32(baseAddr + offset, value); //low-level Xilinx call
}

//helper function to simplify init function
void timers_initTCSRRegisters(int32_t baseAddr){
    timers_writeRegister(baseAddr, TCSR_0_OFFSET, TCSR_INIT_BIT); //initialize TCSR0
    timers_writeRegister(baseAddr, TCSR_1_OFFSET, TCSR_INIT_BIT); //initialize TCSR1

    timers_writeRegister(baseAddr, TCSR_0_OFFSET, CASCADE_BIT);   //set cascade bit to 1 without affecting other bits
}

//function used to initialize each TCSR register for an individual timer
intervalTimer_status_t intervalTimer_init(uint32_t timerNumber){
    if(timerNumber == INTERVAL_TIMER_TIMER_0){    //initialize TCSR registers for timer 0
        timers_initTCSRRegisters(XPAR_AXI_TIMER_0_BASEADDR); //calls helper function to initialize both TCSR registers and set cascade bit

        return INTERVAL_TIMER_STATUS_OK;                    //if we get through those initializations, return that it was successful
    }
```

```
57      else if(timerNumber == INTERVAL_TIMER_TIMER_1){    //initialize TCSR registers for
  timer 1
58          timers_initTCSRRegisters(XPAR_AXI_TIMER_1_BASEADDR); //calls helper function
  to initialize both TCSR registers and set cascade bit
59
60          return INTERVAL_TIMER_STATUS_OK;                    //if we get through those
  initializations, return that it was successful
61      }
62      else if(timerNumber == INTERVAL_TIMER_TIMER_2){    //initialize TCSR registers for
  timer 2
63          timers_initTCSRRegisters(XPAR_AXI_TIMER_2_BASEADDR); //calls helper function
  to initialize both TCSR registers and set cascade bit
64
65          return INTERVAL_TIMER_STATUS_OK;                    //if we get through those
  initializations, return that it was successful
66      }
67
68      return INTERVAL_TIMER_STATUS_FAIL; //if we don't get one of the other return
  statements, it means the initializations failed, so we return that
69
70  }
71
72  //top-level function used to initialize both TCSR registers for all 3 timers
73  intervalTimer_status_t intervalTimer_initAll(){
74      return (intervalTimer_init(INTERVAL_TIMER_TIMER_0) &&
75              intervalTimer_init(INTERVAL_TIMER_TIMER_1) &&
76              intervalTimer_init(INTERVAL_TIMER_TIMER_2)); //ANDs together the results
  from initializing the 3 timers. If any one of them fails, it will return 0
77  }
78
79  //helper function for getting the base address
80  uint32_t timers_baseAddress(uint32_t timerNumber){
81
82      if(timerNumber == INTERVAL_TIMER_TIMER_0) //check if the first timer
83          return XPAR_AXI_TIMER_0_BASEADDR; //if timer 0 is passed in, return its base
  address
84      else if(timerNumber == INTERVAL_TIMER_TIMER_1) //check if the second timer
85          return XPAR_AXI_TIMER_1_BASEADDR; //if timer 1 is passed in, return its base
  address
86      else
87          return XPAR_AXI_TIMER_2_BASEADDR; //if timer 2 is passed in, return its base
  address
88     }
89
90  //timer start function
91  void intervalTimer_start(uint32_t timerNumber){
92      uint32_t baseAddr = timers_baseAddress(timerNumber); //retrieve base address for
  timer we are starting
93
94      timers_writeRegister(baseAddr, TCSR_0_OFFSET, timers_readRegister(baseAddr,
  TCSR_0_OFFSET) | ENT0_BIT); //must bitwise OR the current value in TCSR0 with ENT0
95
  //in order to write a 1 to ENT0 without disturbing other bits
96  }
97
98
99  //timer stop function
100 void intervalTimer_stop(uint32_t timerNumber){
```

```
101     uint32_t baseAddr = timers_baseAddress(timerNumber); //retrieve base address for
    timer we are stopping
102
103     timers_writeRegister(baseAddr, TCSR_0_OFFSET, timers_readRegister(baseAddr,
    TCSR_0_OFFSET) & ENT0_MASK); //similar to what's done in start function. But we have
    to mask ENT0 first
104
    //since we are writing it to a 0 without disturbing other bits
105 }
106
107 //helper function to reset the TLR registers and set the LOAD bit to 1 on both TCSR
    registers
108 void timers_resetIndTimer(uint32_t baseAddr){
109     timers_writeRegister(baseAddr, TLR_0_OFFSET, RESET_BIT);
110     timers_writeRegister(baseAddr, TCSR_0_OFFSET, (timers_readRegister(baseAddr,
    TCSR_0_OFFSET) | LOAD_BIT)); //use | to mask the LOAD bit and not affect the other
    TCSR0 bits
111
112     timers_writeRegister(baseAddr, TLR_1_OFFSET, RESET_BIT);
113     timers_writeRegister(baseAddr, TCSR_1_OFFSET, (timers_readRegister(baseAddr,
    TCSR_1_OFFSET) | LOAD_BIT)); //use | to mask the LOAD bit and not affect the other
    TCSR1 bits
114 }
115
116 //determines which timer to reset, then calls helper function
117 void intervalTimer_reset(uint32_t timerNumber){
118     if(timerNumber == INTERVAL_TIMER_TIMER_0) //if it's the first timer, call the
    helper function for that one
119         timers_resetIndTimer(XPAR_AXI_TIMER_0_BASEADDR);
120     else if(timerNumber == INTERVAL_TIMER_TIMER_1) //if it's the second timer, call
    the helper function for that one
121         timers_resetIndTimer(XPAR_AXI_TIMER_1_BASEADDR);
122     else if(timerNumber == INTERVAL_TIMER_TIMER_2) //if it's the third timer, call the
    helper function for that one
123         timers_resetIndTimer(XPAR_AXI_TIMER_2_BASEADDR);
124     else
125         printf(RESET_ERROR);
126
127     intervalTimer_initAll();
128 }
129
130
131 //resets both counters on all three timers
132 void intervalTimer_resetAll(){
133     for(uint32_t i = 0; i < NUMBER_OF_TIMERS; ++i){ //use a for loop to iterate
    through and reset each timer
134         intervalTimer_reset(i); //i will correspond to the timer number that needs to
    be reset
135     }
136 }
137
138 //helper function to actually retrieve value of timer
139 double timers_getDuration(uint32_t baseAddr){
140
141     uint64_t counter0 = timers_readRegister(baseAddr, TCR_0_OFFSET); //get value from
    TCR0
142     uint64_t counter1 = timers_readRegister(baseAddr, TCR_1_OFFSET); //get value from
    TCR1
```

```
143
144     uint64_t totDuration = (counter1 << SHIFT_VAL) | counter0; //concatenate values to
    get full 64 bit value.
145                                                               //counter1 needs to be
    to the left which is why it is shifted
146     return (double)totDuration; //needs to be casted to double in order to avoid int
    division in parent function
147 }
148
149 //takes data from helper function and divides by timer frequency to get seconds
150 double intervalTimer_getTotalDurationInSeconds(uint32_t timerNumber){
151     uint32_t baseAddr = timers_baseAddress(timerNumber); //retrieve base address for
    timer we want to get the value from
152
153     return (timers_getDuration(baseAddr) / XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ); //in order
    to get seconds, we need to divide that value by the timer's frequency
154 }
155
156
```