```c
/*
 * intervalTimer.h
 *
 *  Created on: Apr 2, 2014
 *      Author: hutch
 */

// Provides an API for accessing the three hardware timers that are installed
// in the ZYNQ fabric.

#ifndef INTERVALTIMER_H_
#define INTERVALTIMER_H_

#include <stdint.h>

// Used to indicate status that can be checked after invoking the function.
typedef uint32_t intervalTimer_status_t;  // Use this type for the return type of a function.

#define INTERVAL_TIMER_STATUS_OK 1        // Return this status if successful.
#define INTERVAL_TIMER_STATUS_FAIL 0      // Return this status if failure.

#define INTERVAL_TIMER_TIMER_0 0
#define INTERVAL_TIMER_TIMER_1 1
#define INTERVAL_TIMER_TIMER_2 2

// You must initialize the timers before you use them.
// timerNumber indicates which timer should be initialized.
// returns INTERVAL_TIMER_STATUS_OK if successful, some other value otherwise.
intervalTimer_status_t intervalTimer_init(uint32_t timerNumber);

// This is a convenience function that initializes all interval timers.
// Simply calls intervalTimer_init() on all timers.
// returns INTERVAL_TIMER_STATUS_OK if successful, some other value otherwise.
intervalTimer_status_t intervalTimer_initAll();

// This function starts the interval timer running.
// timerNumber indicates which timer should start running.
void intervalTimer_start(uint32_t timerNumber);

// This function stops the interval timer running.
// timerNumber indicates which timer should stop running.
void intervalTimer_stop(uint32_t timerNumber);

// This function resets the interval timer.
// timerNumber indicates which timer should reset.
void intervalTimer_reset(uint32_t timerNumber);

// Convenience function for intervalTimer_reset().
// Simply calls intervalTimer_reset() on all timers.
void intervalTimer_resetAll();

// Runs a test on a single timer as indicated by the timerNumber argument.
// Returns INTERVAL_TIMER_STATUS_OK if successful, something else otherwise.
intervalTimer_status_t intervalTimer_test(uint32_t timerNumber);

// Convenience function that invokes test on all interval timers.
// Returns INTERVAL_TIMER_STATUS_OK if successful, something else otherwise.
```

```
58 intervalTimer_status_t intervalTimer_testAll();
59
60 // Once the interval timer has stopped running, use this function to
61 // ascertain how long the timer was running.
62 // The timerNumber argument determines which timer is read.
63 double intervalTimer_getTotalDurationInSeconds(uint32_t timerNumber);
64
65 #endif /* INTERVALTIMER_H_ */
66
```

```
 2 * intervalTimer.c
 7#include "intervalTimer.h"
 8#include <xparameters.h>
 9#include "xil_io.h"
10
11#define TCSR0_init_input 0x000          //set reg to 0
12#define TCSR1_init_input 0b00000000000  //set reg2 to 0
13#define TCSR_SET_CASC_BIT 0x800         //set the cascade bit to one
14#define TCSR_START_ENT0_BIT 0x80        //set only the ent0 bit to 1
15#define TCSR_STOP_ENT0_BIT 0x7f         //use as mask to set ent0 to 0
16
17#define LOAD_CLEAR 0x000 // clear everything to 0
18#define TCSR_LOAD 0b100000 // just enable the 5th bit where the load reg is
19
20#define TCSR0_ADDER 0x00 //Read/Write 0x0 Control/Status Register 0
21#define TLR0_ADDER 0x04 //Read/Write 0x0 Load Register 0
22#define TCR0_ADDER 0x08 //Read 0x0 Timer/Counter Register 0
23#define TCSR1_ADDER 0x10 //Read/Write 0x0 Control/Status Register 1
24#define TLR1_ADDER 0x14 //Read/Write 0x0 Load Register 1
25#define TCR1_ADDER 0x18 //Read 0x0 Timer/Counter Register 1
26
27#define SHIFT_BITS 32
28
29
30
31
32
33//**********Prototype_Functions**********
34int32_t timer_0_read(int32_t offset); // define function read out from timer 0
35int32_t timer_1_read(int32_t offset); // define function read out from timer 1
36int32_t timer_2_read(int32_t offset); // define function read out from timer 2
37void timer_0_write(int32_t offset, int32_t data); //  define function to write to
  timer 0
38void timer_1_write(int32_t offset, int32_t data); // define function to write to timer
  1
39void timer_2_write(int32_t offset, int32_t data); // define function to write to timer
  2
40
41//**********Main_Functions**********
45intervalTimer_status_t intervalTimer_init(uint32_t timerNumber){
46    switch (timerNumber){
47    case INTERVAL_TIMER_TIMER_0:
48        timer_0_write(TCSR0_ADDER, TCSR0_init_input); //write a 0 to the TCSR0
  register.
49        timer_0_write(TCSR1_ADDER, TCSR1_init_input); //write a 0 to the TCSR1
  register.
50        timer_0_write(TCSR0_ADDER, TCSR_SET_CASC_BIT); //set the CASC bit and clear
  the UDT0 bit in the TCSR0 register (cascade mode and up counting).
51        return INTERVAL_TIMER_STATUS_OK;
52    case INTERVAL_TIMER_TIMER_1:
53        timer_1_write(TCSR0_ADDER, TCSR0_init_input); //write a 0 to the TCSR0
  register.
54        timer_1_write(TCSR1_ADDER, TCSR1_init_input); //write a 0 to the TCSR1
  register.
55        timer_1_write(TCSR0_ADDER, TCSR_SET_CASC_BIT); //set the CASC bit and clear
  the UDT0 bit in the TCSR0 register (cascade mode and up counting).
56        return INTERVAL_TIMER_STATUS_OK;
57    case INTERVAL_TIMER_TIMER_2:
```

```
58          timer_2_write(TCSR0_ADDER, TCSR0_init_input); //write a 0 to the TCSR0
  register.
59          timer_2_write(TCSR1_ADDER, TCSR1_init_input); //write a 0 to the TCSR1
  register.
60          timer_2_write(TCSR0_ADDER, TCSR_SET_CASC_BIT); //set the CASC bit and clear
  the UDT0 bit in the TCSR0 register (cascade mode and up counting).
61          return INTERVAL_TIMER_STATUS_OK;
62      default:
63          return INTERVAL_TIMER_STATUS_FAIL;
64      }
65  };
66
67  // This is a convenience function that initializes all interval timers.
70  intervalTimer_status_t intervalTimer_initAll(){
76
77  // This function starts the interval timer running.
78  // timerNumber indicates which timer should start running.
79  void intervalTimer_start(uint32_t timerNumber){
80      switch(timerNumber){
81      case INTERVAL_TIMER_TIMER_0:
82          timer_0_write(TCSR0_ADDER, ( timer_0_read(TCSR0_ADDER) |
  TCSR_START_ENT0_BIT)); // write a 1 into the ENT0 bit in the TCSR0.
83          break; // end start
84      case INTERVAL_TIMER_TIMER_1:
85          timer_1_write(TCSR0_ADDER, ( timer_1_read(TCSR0_ADDER) |
  TCSR_START_ENT0_BIT)); // write a 1 into the ENT0 bit in the TCSR0.
86          break; // end start
87      case INTERVAL_TIMER_TIMER_2:
88          timer_2_write(TCSR0_ADDER, ( timer_2_read(TCSR0_ADDER) |
  TCSR_START_ENT0_BIT)); // write a 1 into the ENT0 bit in the TCSR0.
89          break; // end start
90      default:
91          break; // end start
92      }
93  };
94
95  // This function stops the interval timer running.
96  // timerNumber indicates which timer should stop running.
97  void intervalTimer_stop(uint32_t timerNumber){
98      switch(timerNumber){
99      case INTERVAL_TIMER_TIMER_0:
100         timer_0_write(TCSR0_ADDER, ( timer_0_read(TCSR0_ADDER) & TCSR_STOP_ENT0_BIT));
  // write a 0 into the ENT0 bit in the TCSR0.
101         break; // end stop
102     case INTERVAL_TIMER_TIMER_1:
103         timer_1_write(TCSR0_ADDER, ( timer_1_read(TCSR0_ADDER) & TCSR_STOP_ENT0_BIT));
  // write a 0 into the ENT0 bit in the TCSR0.
104         break; // end stop
105     case INTERVAL_TIMER_TIMER_2:
106         timer_2_write(TCSR0_ADDER, ( timer_2_read(TCSR0_ADDER) & TCSR_STOP_ENT0_BIT));
  // write a 0 into the ENT0 bit in the TCSR0.
107         break; // end stop
108     default:
109         break; // end stop
110     }
111 };
112
113 // This function resets the interval timer.
```

```
114 // timerNumber indicates which timer should reset.
115 void intervalTimer_reset(uint32_t timerNumber){
116     switch(timerNumber){
117     case INTERVAL_TIMER_TIMER_0:
118         timer_0_write(TLR0_ADDER, LOAD_CLEAR); // write a 0 into the TLR0 register.
119         timer_0_write(TCSR0_ADDER, ( timer_0_read(TCSR0_ADDER) | TCSR_LOAD)); // write
    a 1 into the LOAD0 bit in the TCSR0.
120         timer_0_write(TLR1_ADDER, LOAD_CLEAR); // write a 0 into the TLR1 register.
121         timer_0_write(TCSR1_ADDER, ( timer_0_read(TCSR1_ADDER) | TCSR_LOAD)); // write
    a 1 into the LOAD0 bit in the TCSR1.
122         intervalTimer_init(INTERVAL_TIMER_TIMER_0);// Initialize the load timer reg
    again
123         break;
124
125     case INTERVAL_TIMER_TIMER_1:
126         timer_1_write(TLR0_ADDER, LOAD_CLEAR); // write a 0 into the TLR0 register.
127         timer_1_write(TCSR0_ADDER, ( timer_1_read(TCSR0_ADDER) | TCSR_LOAD)); // write
    a 1 into the LOAD0 bit in the TCSR0.
128         timer_1_write(TLR1_ADDER, LOAD_CLEAR); // write a 0 into the TLR1 register.
129         timer_1_write(TCSR1_ADDER, ( timer_1_read(TCSR1_ADDER) | TCSR_LOAD)); // write
    a 1 into the LOAD0 bit in the TCSR1.
130         intervalTimer_init(INTERVAL_TIMER_TIMER_1);// Initialize the load timer reg
    again
131         break;
132
133     case INTERVAL_TIMER_TIMER_2:
134         timer_2_write(TLR0_ADDER, LOAD_CLEAR); // write a 0 into the TLR0 register.
135         timer_2_write(TCSR0_ADDER, ( timer_2_read(TCSR0_ADDER) | TCSR_LOAD)); // write
    a 1 into the LOAD0 bit in the TCSR0.
136         timer_2_write(TLR1_ADDER, LOAD_CLEAR); // write a 0 into the TLR1 register.
137         timer_2_write(TCSR1_ADDER, ( timer_2_read(TCSR1_ADDER) | TCSR_LOAD)); // write
    a 1 into the LOAD0 bit in the TCSR1.
138         intervalTimer_init(INTERVAL_TIMER_TIMER_2); // Initialize the load timer reg
    again
139         break;
140     default:
141         break;
142     }
143 };
144
145 // Convenience function for intervalTimer_reset().
146 // Simply calls intervalTimer_reset() on all timers.
147 void intervalTimer_resetAll(){
148     intervalTimer_reset(INTERVAL_TIMER_TIMER_0); //reset timer 0
149     intervalTimer_reset(INTERVAL_TIMER_TIMER_1); //reset timer 1
150     intervalTimer_reset(INTERVAL_TIMER_TIMER_2); //reset timer 2
151 };
152
153 // Runs a test on a single timer as indicated by the timerNumber argument.
154 // Returns INTERVAL_TIMER_STATUS_OK if successful, something else otherwise.
155 intervalTimer_status_t intervalTimer_test(uint32_t timerNumber){
156     switch(timerNumber){
157     case INTERVAL_TIMER_TIMER_0: //uses each function for timer 0
158         intervalTimer_init(INTERVAL_TIMER_TIMER_0); //run to see if timer 0 will
    initialize
159         intervalTimer_start(INTERVAL_TIMER_TIMER_0); //run to see if the timer will
    start
160         intervalTimer_stop(INTERVAL_TIMER_TIMER_0);  //run to see if the timer will
```

```
   stop
161         intervalTimer_reset(INTERVAL_TIMER_TIMER_0); //run to see if the timer will
   reset
162         return INTERVAL_TIMER_STATUS_OK;
163     case INTERVAL_TIMER_TIMER_1: //uses each function for timer 1
164         intervalTimer_init(INTERVAL_TIMER_TIMER_1); //run to see if timer 2 will
   initialize
165         intervalTimer_start(INTERVAL_TIMER_TIMER_1); //run to see if the timer will
   start
166         intervalTimer_stop(INTERVAL_TIMER_TIMER_1); //run to see if the timer will
   stop
167         intervalTimer_reset(INTERVAL_TIMER_TIMER_1); //run to see if the timer will
   reset
168         return INTERVAL_TIMER_STATUS_OK;
169     case INTERVAL_TIMER_TIMER_2: //uses each function for timer 2
170         intervalTimer_init(INTERVAL_TIMER_TIMER_2); //run to see if timer 2 will
   initialize
171         intervalTimer_start(INTERVAL_TIMER_TIMER_2); //run to see if the timer will
   start
172         intervalTimer_stop(INTERVAL_TIMER_TIMER_2);  //run to see if the timer will
   stop
173         intervalTimer_reset(INTERVAL_TIMER_TIMER_2); //run to see if the timer will
   reset
174         return INTERVAL_TIMER_STATUS_OK;
175     default:
176         return INTERVAL_TIMER_STATUS_FAIL; // if false input return 0
177     }
178 };
179
180 // Convenience function that invokes test on all interval timers.
181 // Returns INTERVAL_TIMER_STATUS_OK if successful, something else otherwise.
182 intervalTimer_status_t intervalTimer_testAll(){
183     intervalTimer_test(INTERVAL_TIMER_TIMER_0); //test timer 0
184     intervalTimer_test(INTERVAL_TIMER_TIMER_1); //test timer 1
185     intervalTimer_test(INTERVAL_TIMER_TIMER_2); //test timer 2
186     return INTERVAL_TIMER_STATUS_OK;
187 };
188
189 // Once the interval timer has stopped running, use this function to
190 // ascertain how long the timer was running.
191 // The timerNumber argument determines which timer is read.
192 double intervalTimer_getTotalDurationInSeconds(uint32_t timerNumber){
193     switch(timerNumber){
194     case INTERVAL_TIMER_TIMER_0:{
195         int64_t upper_64;
196         int32_t lower_32;
197         double frec = XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ; //set to a double so it doesn't
   concatenate it
198         upper_64 = timer_0_read(TCR1_ADDER);
199         lower_32 = timer_0_read(TCR0_ADDER);
200         upper_64 = (upper_64 << SHIFT_BITS) + lower_32; // shift the upper reg over so
   the lower reg can be added
201         return (double)((upper_64)/frec);
202     }
203     case INTERVAL_TIMER_TIMER_1:{
204         uint64_t upper_64;
205         uint32_t lower_32;
206         double frec = XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ;//set to a double so it doesn't
```

```
   concatenate it
207         upper_64 = timer_1_read(TCR1_ADDER);
208         lower_32 = timer_1_read(TCR0_ADDER);
209         upper_64 = (upper_64 << SHIFT_BITS) + lower_32;// shift the upper reg over so
   the lower reg can be added
210         return (double)((upper_64)/frec);
211     }
212     case INTERVAL_TIMER_TIMER_2:{
213         int64_t upper_64;
214         int32_t lower_32;
215         double frec = XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ;//set to a double so it doesn't
   concatenate it
216         upper_64 = timer_2_read(TCR1_ADDER);
217         lower_32 = timer_2_read(TCR0_ADDER);
218         upper_64 = (upper_64 << SHIFT_BITS) + lower_32;// shift the upper reg over so
   the lower reg can be added
219         return (double)((upper_64)/frec);
220     }
221     default:
222         return (double)(0);
223     }
224 }
225
226 //**********HelperFunction**********
227 void timer_0_write(int32_t offset, int32_t data){ //helper function to
228     Xil_Out32( XPAR_AXI_TIMER_0_BASEADDR + offset, data);//Xilinxs function to write
   to the register.
229 }
230 void timer_1_write(int32_t offset, int32_t data){ //helper function to
231     Xil_Out32( XPAR_AXI_TIMER_1_BASEADDR + offset, data);//Xilinxs function to write
   to the register.
232 }
233 void timer_2_write(int32_t offset, int32_t data){ //helper function to
234     Xil_Out32( XPAR_AXI_TIMER_2_BASEADDR + offset, data);//Xilinxs function to write
   to the register.
235 }
236 int32_t timer_0_read(int32_t offset){// helper function that Reads from the register
237     return Xil_In32( XPAR_AXI_TIMER_0_BASEADDR + offset); // returns data from the
   register
238 }
239 int32_t timer_1_read(int32_t offset){// helper function that Reads from the register
240     return Xil_In32( XPAR_AXI_TIMER_1_BASEADDR + offset); // returns data from the
   register
241 }
242 int32_t timer_2_read(int32_t offset){// helper function that Reads from the register
243     return Xil_In32( XPAR_AXI_TIMER_2_BASEADDR + offset); // returns data from the
   register
244 }
245
246
```