```
/*
 *  Steampunk hydraulic clock
 *
 *  Version 1.0    11/01/2019
 *
 *  This software drives an electric pump that operates a 1-minute siphon, which in
 *  turn operates a clock movement.  The pump continuously fills a cylindrical
 *  container that, once filled, drains via a siphon.  A float in the cylinder
 *  connected to a pawl advances a ratchet wheel 1/60th of a revolution to drive
 *  the clock movement.  A Hall effect sensor connected to the microcontroller
 *  generates an interrupt each time the siphon empties. The software monitors
 *  the cycle time and adjusts the duration of a pulse of current to the pump to
 *  maintain a precise 1 minute cycle time for the siphon.
 *
 *  Timer A2 runs continuously and generates an interrupt every 100 mS. This is used
 *  as a timebase to regulate the pump speed and the accuracy of the clock. The timer
 *  also generates a modulated pulse to the pump to regulate its speed to ensure that
 *  the cycle time to fill and drain the siphon is exactly one minute.
 *
 *
 *  Dick Bipes
 *  dick@carveshop.com
 *
 *
 *  (c) Copyright 2019 by Dick Bipes  All rights reserved
 *
 *
 */
/*
 * ======== Standard MSP430 includes ========
 */
#include <msp430.h>

/*
 * ======== Grace related includes ========
 */
#include <ti/mcu/msp430/csl/CSL.h>
//  Port 1 I/O
#define RedLED          (BIT0)      // Red LED on the LaunchPad
#define GreenLED        (BIT6)      // Green LED on the LaunchPad
#define RedLED2         (BIT3)      // Red LED
#define GreenLED2       (BIT4)      // Green LED
//                                  // P1.2 is the timer PWM drive to the MOSFET pump
#define HallSensor      (BIT1)      // Hall sensor that triggers when the float is low

#define TicksPerCycle   600     // number of timer ticks per desired 1-minute cycle

// These constants are determined by pump characteristics, plumbing, and cylinder size
#define NominalPWM  1500    // Nominal PWM comparator value - 28 mS 10 Hz
#define MinPWM      800     // Minimum PWM value to ensure that the pump fills
                            // the upper cylinder
#define MaxPWM      3000    // Maximum PWM value to ensure that the pump does not
                            // overpower the siphon
#define Kp  .8              // Proportional constant, empirically derived
#define Ki  .4              // Integral constant, empirically derived

unsigned int    ticktock = 0;       // Timer tick - one count per 100 ms
```

```
58 unsigned int    timer_capture;      // current value of ticktock
59 unsigned int    last_capture = 0;   // previous value of ticktock
60
61 int error = 0;      // error, the difference between reference time base and
62                      //the siphon (clock)
63 int i_error = 0;    // integral of the error
64
65 #define ArraySize 15
66 int errorhist[ArraySize] = {0};     // log the error history for development/debug
67 int i = 0;                          // pointer to next entry in the error log
68 int error_counter = 0;              // number of seconds the last cycle was in error
69 int i_error_counter = 0;            // integral of error - the total number of seconds
70                                     // the clock is in error since it started running
71
72 int seconds10x = 0;
73 int minutes = 0;
74 int hours = 0;
75
76
77 int pumpPWM = NominalPWM;   // duration of the power pulse to the motor
78                             //  (sets the pump speed)
79
80 /*
81  * Routine to wait a bit while an LED is turned on or off
82  */
83   void WaitABit(void)
84   {
85       volatile unsigned int i;     // volatile to prevent optimization
86       for (i=0; i<30000; i++);     // wait a while
87   }
88
89 /*
90  * Interrupt service routine called when the Hall Sensor indicates that the float
91  * is low.
92  */
93   void FloatLow(void)
94   {
95        P1IFG &= ~HallSensor;               // Clear the interrupt
96        timer_capture = ticktock;           // capture the time for the last fill/drain cycle
97                                            // compute the error for this last cycle
98        error = TicksPerCycle - (timer_capture - last_capture);
99        last_capture = timer_capture;       // remember the timer count for next cycle
100       i_error += error;                   // integral of the error
101       error_counter = error/10;           // copy error to count LED flashes
102       i_error_counter = i_error/10;       // copy i_error to count LED flashes
103
104       errorhist[i++] = error;             // keep a log of error
105       if (i >= ArraySize)
106           i = 0;
107
108       pumpPWM = pumpPWM - (Kp*error + Ki*i_error);    // Standard PID algorithm, but with no
109                                                       // differential term
110
111       if (pumpPWM < MinPWM)           // keep the PWM in bounds
112           pumpPWM = MinPWM;           // long enough to ensure some water is pumped
113       if (pumpPWM > MaxPWM)
114           pumpPWM = MaxPWM;           // short enough to prevent overpowering the siphon
```

```
115
116         TACCR1 = pumpPWM;               // set the timer to enable the new PWM time
117
118     }
119 /*
120 * Interrupt service routine called when the timer reaches max value - once
121 * every 100 mS
122 */
123     void TimerTick(void)
124     {
125         ticktock++;
126         // For debug, keep track of how long the clock was running
127         seconds10x++;
128         if (seconds10x >= 600)
129         {
130             minutes++;
131             seconds10x = 0;
132         }
133         if (minutes >= 60)
134         {
135             hours++;
136             minutes = 0;
137         }
138     }
139
140 /*
141  *      Flash LEDs to indicate the number of seconds that the cycle deviates
142  *      from the ideal
143  */
144     void FlashI_error(void)
145     {
146         while ((error_counter != 0) || (i_error_counter !=0) )
147         {
148             WaitABit();
149
150             if (error_counter > 0)       // seconds that the last cycle differs from 60 seconds
151             {
152                 P1OUT |= GreenLED;       // turn the LED on
153                 error_counter--;         // count down the number of seconds to zero
154             }
155             if (error_counter < 0)
156             {
157                 P1OUT |= RedLED;         // turn the LED on
158                 error_counter++;         // count up the number of seconds to zero
159             }
160
161             if (i_error_counter > 0)     // seconds that the clock deviates from ideal time
162             {
163                 P1OUT |= GreenLED2;      // turn the LED on
164                 i_error_counter--;       // count down the number of seconds to zero
165             }
166             if (i_error_counter < 0)
167             {
168                 P1OUT |= RedLED2;        // turn the LED on
169                 i_error_counter++;       // count up the number of seconds to zero
170             }
171
```

```
172             WaitABit();
173             P1OUT &= ~(GreenLED + RedLED + GreenLED2 + RedLED2); // turn the LEDs off
174         }
175     }
176
177 /*
178  *  ======== main ========
179  */
180 int main(int argc, char *argv[])
181 {
182     CSL_init();             // Activate Grace-generated configuration
183
184     // >>>>> Fill-in user code here <<<<<
185
186     TACCR1 = NominalPWM;    // set the starting pump PWM duration
187
188
189     for ( ; ; )
190     {
191         FlashI_error();     // display the error times
192     }
193
194 }
195
```