```
 2  * clockControl.c
 7
 8 #include "clockControl.h"         //.h file corresponding to this .c file
 9 #include "clockDisplay.h"         //need access to display functions in order to
   inc/dec and advance time one second on the display
10 #include "stdint.h"               //needed to be able to make variables of type int
11 #include "supportFiles/display.h" //needed to access the functions to interact with
   display
12 #include "stdio.h"                //needed to make debugging printf's work
13
14 #define COUNTER_MIN 0             //counter reset value
15 #define ADC_COUNTER_MAX_VALUE 1   //how long we want to register the touch before
   moving to the auto counting state
16 #define AUTO_COUNTER_MAX_VALUE 10 //how long we want to register the touch before
   initiating auto-update
17 #define RATE_COUNTER_MAX_VALUE 1  //how long between increments when auto-updating
18 #define ADVANCE_TIME 20           //how many ticks before we advance a second
19
20 static uint16_t adcCounter;              //used to track when the first touch was
   registered to when it goes to auto-update
21 static uint16_t autoCounter;             //used to track how long it's been touched
   before starting to auto-update
22 static uint16_t rateCounter;             //used to track how often we should increment
   when in auto-update mode
23 static uint16_t msCounter = COUNTER_MIN; //used to track the amount of ticks we've
   done
24
25 // States for the controller state machine.
26 enum clockControl_st_t {
37
38 // This is a debug state print routine. It will print the names of the states each
41 void debugStatePrint() {
78
79
80 //SM init function
81 void clockControl_init() {
84
85
86 //SM tick function
87 void clockControl_tick() {
88     debugStatePrint(); //used to know which state the program is in at any given
   moment
89   // Perform state update first.
90   switch(currentState) {
91     case init_st:
92         currentState = never_touched_st; //from the init_st we go immediately to the
   never touched state
93         break;
94     case never_touched_st: //this state is to account for the display never been
   touched. We only want the clock to start keeping time if the display is touched
95         if(display_isTouched()) //we only go to the next state if the display is
   touched
96             currentState = waiting_for_touch_st;
97         break;
98     case waiting_for_touch_st: //this state acts as a kind of base state where the SM
   will continually return to in order to wait for next input
99         adcCounter = COUNTER_MIN;  //reset these 3 counters
100        autoCounter = COUNTER_MIN;
```

```
101         rateCounter = COUNTER_MIN;
102         msCounter++;                //increment because coming back to waiting state means we're getting closer to incrementing another second
103         if(display_isTouched()){ //we only want to move to a timer counting state if the display is touched
104             currentState = ad_timer_running_st; //we specifically want to always move from here to the adc timer being counted before moving to the other timer counting states
105             display_clearOldTouchData();        //need to clear previous touch data to get new touches
106         }
107         else if(msCounter >= ADVANCE_TIME)      //if the display wasn't touched and we've incremented the msCounter enough, we want the add second state
108             currentState = add_second_to_clock_st; //we want to go to this state since it's where we increment seconds
109         break;
110     case ad_timer_running_st:
111         if(!display_isTouched() && (adcCounter == ADC_COUNTER_MAX_VALUE)){ //if the display isn't being touched anymore and our adc value is at its max, then we go back to waiting state
112             currentState = waiting_for_touch_st;
113             clockDisplay_performIncDec(); //this state means we touched one of the incDec arrows for a short time and only want to inc/dec by 1, so we call that function
114         }
115         else if(display_isTouched() && (adcCounter == ADC_COUNTER_MAX_VALUE)) //but if the display is still being touched and adc is at its max, we need to go to auto counting
116             currentState = auto_timer_running_st;
117         break;
118     case auto_timer_running_st: //this state is for sensing if we meet the 500ms threshold to start auto-updating
119         if(!display_isTouched() && (autoCounter != AUTO_COUNTER_MAX_VALUE)){ //if the display is no longer being touched and we still had not reached the auto max, then we dont need to auto-update
120                                                                             //and we go back to the waiting state
121             currentState = waiting_for_touch_st;
122             clockDisplay_performIncDec();         //even though we don't want to update, we still got a touch to inc/dec, so we do that by 1
123         }
124         else if(display_isTouched() && (autoCounter == AUTO_COUNTER_MAX_VALUE)){ //if the display is still being touched after we hit the max auto value,
125 
126 //that's when we want to actually start auto-updating
127             currentState = rate_timer_running_st;
128             clockDisplay_performIncDec();           //we begin the auto-update process with a single inc/dec. The rest is taken care of in the next state
129         }
130         break;
131     case rate_timer_running_st: //this is one of the two states that actually handles the auto-updating
132                                 //this first part is to check to see if the display is still being continuously touched
133         if(!display_isTouched() && (rateCounter != RATE_COUNTER_MAX_VALUE)){ //if the display is no longer being touched, we dont inc/dec anymore and instead go back to waiting state
134             currentState = waiting_for_touch_st;
135         }
```

```
135         else if(display_isTouched() && (rateCounter == RATE_COUNTER_MAX_VALUE)){ //if the display is still being touched and the rate counter is at its max, then go to the expired state
136             currentState = rate_timer_expired_st;
137         }
138         break;
139     case rate_timer_expired_st: //this is the second part of the actual auto-updating
140         if(display_isTouched()){ //if it's still being touched, inc/dec AND go back to the rate timer counting state, thus extending the auto-update
141             currentState = rate_timer_running_st;
142             clockDisplay_performIncDec(); //function call to inc/dec by 1
143         }
144         else if(!display_isTouched()) //if the used is no longer touching the display, no more inc/dec needs to be done and we return to the waiting state to await the next user touch
145             currentState = waiting_for_touch_st;
146         break;
147     case add_second_to_clock_st:  //this is the state in charge of the natural timekeeping of the clock
148         //these are transition actions
149         clockDisplay_advanceTimeOneSecond(); //function call for incrementing seconds by 1. Only done when enough ticks have happened
150         currentState = waiting_for_touch_st; //we always go right back to the waiting state after this one to await further user input
151         msCounter = COUNTER_MIN;             //very important to reset the counter in order to prevent the seconds being incremented too quickly
152         break;
153     default:
154         printf("clockControl_tick state update: hit default\n\r"); //simple printf to tell us we didn't hit any of the states
155         break;
156     }
157
158   // Perform state action next.
159   switch(currentState) {
160     case ad_timer_running_st:
161         adcCounter++; //state action is to increment the adc counter.
162                       //this is for knowing if the touch has been long enough to inc/dec
163         break;
164     case auto_timer_running_st:
165         autoCounter++; //need to increment auto counter
166                        //this is to track how long the display is being pressed and if we reach the value needed to start auto-updating
167         break;
168     case rate_timer_running_st:
169         rateCounter++; //start incrementing rate counter
170                        //this it to control how fast we inc/dec when in auto-update mode
171         break;
172     case rate_timer_expired_st:
173         rateCounter = COUNTER_MIN; //important to reset this counter in order to not accidentally keep inc/decing when it should have already stopped
174         break;
175     default:
176         printf("clockControl_tick state action: hit default\n\r"); //simple printf to tell us we didn't do any state actions
177                                                                    //this would
```

```
   indicate out currentState doesn't have any state actions
178         break;
179     }
180 }
181
```